DoubleRatchet

Release 1.0.3-stable

Tim Henkes (Syndace)

Jan 30, 2023

CONTENTS

CHAPTER

ONE

INSTALLATION

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

CHAPTER

TWO

GETTING STARTED

This quick start guide assumes knowledge of the Double Ratchet algorithm.

Next to a few container classes and interfaces, the four major units of this library are *DoubleRatchet*, *DiffieHellmanRatchet*, *SymmetricKeyRatchet* and *KDFChain*. These classes are structured roughly the same:

- Instances can be created through factory methods (called e.g. `create`), **NOT** by calling the constructor/`__init__`.
- Instances can be serialized into JSON-friendly data structures.
- When creating a new instance or deserializing an old instance, a set of configuration options has to be passed. Note that it is your responsibility to pass the same configuration when deserializing as you passed when creating the instance.
- Some of the classes are abstract, requiring you to subclass them and to implement one or two abstract methods.
- For some of the interfaces and abstract classes, implementations using recommended cryptographic primitives are available in the *doubleratchet.recommended* package.

The *DoubleRatchet* class offers a thin and simple message en-/decryption API, using and combining all of the other classes under the hood. For details on the configuration, refer to the *encrypt_initial_message()* or *decrypt_initial_message()* methods of the *DoubleRatchet* class. Take a look at the Double Ratchet Chat example in the python-doubleratchet repository for an example of a full configuration, including the required subclassing and using some of the recommended implementations.

This library implements the core of the Double Ratchet specification and includes a few of the recommended algorithms. This library does currently *not* offer sophisticated decision mechanisms for the deletion of skipped message keys. Skipped message keys are only deleted when the maximum amount is reached and old keys are deleted from the storage in FIFO order. There is no time-based or event-based deletion.

CHAPTER

THREE

SERIALIZATION AND MIGRATION

python-doubleratchet uses pydantic for serialization internally. All classes that support serialization offer a property called `model` which returns the internal state of the instance as a pydantic model, and a method called `from_model` to restore the instance from said model. However, while these properties/methods are available for public access, migrations can't automatically be performed when working with models directly. Instead, the property `json` is provided, which returns the internal state of the instance a JSON-friendly Python dictionary, and the method `from_json`, which restores the instance *after* performing required migrations on the data. Unless you have a good reason to work with the models directly, stick to the JSON serialization APIs.

3.1 Migration from pre-stable

Migration from pre-stable is provided, however, since the class hierarchy and serialization concept has changed, only whole Double Ratchet objects can be migrated to stable. Use the `from_json` method as usual.

CHAPTER

FOUR

PACKAGE: DOUBLERATCHET

4.1 Module: aead

class doubleratchet.aead.**AEAD**

Bases: ABC

Authenticated Encryption with Associated Data (AEAD).

abstract async static 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.

abstract async static 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).

exception doubleratchet.aead.**AuthenticationFailedException**

Bases: Exception

Raised by *AEAD.decrypt()* in case of authentication failure.

exception doubleratchet.aead.**DecryptionFailedException**

Bases: Exception

Raised by *AEAD.decrypt()* in case of decryption failure.

4.2 Module: diffie_hellman_ratchet

class doubleratchet.diffie_hellman_ratchet.**DiffieHellmanRatchet**

Bases: ABC

As communication partners exchange messages they also exchange new Diffie-Hellman public keys, and the Diffie-Hellman output secrets become the inputs to the root chain. The output keys from the root chain become new KDF keys for the sending and receiving chains. This is called the Diffie-Hellman ratchet.

https://signal.org/docs/specifications/doubleratchet/#diffie-hellman-ratchet

---

Note: The specification introduces the symmetric-key ratchet and the Diffie-Hellman ratchet as independent units and links them together in the Double Ratchet. This implementation does not follow that separation, instead the Diffie-Hellman ratchet manages the symmetric-key ratchet internally, which makes the code a little less complicated, as the Double Ratchet doesn't have to forward keys generated by the Diffie-Hellman ratchet to the symmetric-key ratchet.

---

async classmethod create(*own_ratchet_priv*, *other_ratchet_pub*, *root_chain_kdf*, *root_chain_key*, *message_chain_kdf*, *message_chain_constant*, *dos_protection_threshold*)

Create and configure a Diffie-Hellman ratchet.

Parameters

- **own_ratchet_priv** (Optional[bytes]) – The ratchet private key to use initially with this instance.
- **other_ratchet_pub** (bytes) – The ratchet public key of the other party.
- **root_chain_kdf** (Type[*KDF*]) – The KDF to use for the root chain. The KDF must be capable of deriving 64 bytes.
- **root_chain_key** (bytes) – The key to initialize the root chain with, consisting of 32 bytes.
- **message_chain_kdf** (Type[*KDF*]) – The KDF to use for the sending and receiving chains. The KDF must be capable of deriving 64 bytes.
- **message_chain_constant** (bytes) – The constant to feed into the sending and receiving KDF chains on each step.
- **dos_protection_threshold** (int) – The maximum number of skipped message keys to calculate. If more than that number of message keys are skipped, the keys are not calculated to prevent being DoSed.

Return type
TypeVar(DiffieHellmanRatchetTypeT, bound= DiffieHellmanRatchet)

Returns
A configured instance of *DiffieHellmanRatchet*, capable of sending and receiving messages.

property sending_chain_length: int

Returns: The length of the sending chain of the internal symmetric-key ratchet.

property receiving_chain_length: Optional[int]

Returns: The length of the receiving chain of the internal symmetric-key ratchet, if it exists.

abstract static _generate_priv()

Return type
bytes

Returns
A freshly generated private key, capable of performing Diffie-Hellman key exchanges with the public key of another party.

Note: This function is recommended to generate a key pair based on the Curve25519 or Curve448 elliptic curves (https://signal.org/docs/specifications/doubleratchet/#recommended-cryptographic-algorithms).

---

abstract static _derive_pub(*priv*)

Derive the public key from a private key as generated by *_generate_priv()*.

Parameters
priv (bytes) – The private key as returned by *_generate_priv()*.

Return type
bytes

Returns
The public key corresponding to the private key.

abstract static _perform_diffie_hellman(*own_priv*, *other_pub*)

Parameters

- **own_priv** (bytes) – The own ratchet private key.
- **other_pub** (bytes) – The ratchet public key of the other party.

Return type
bytes

Returns
A shared secret agreed on via Diffie-Hellman. This method is recommended to perform X25519 or X448. There is no need to check for invalid public keys (https://signal.org/docs/specifications/doubleratchet/#recommended-cryptographic-algorithms).

property model: *DiffieHellmanRatchetModel*

Returns: The internal state of this *DiffieHellmanRatchet* as a pydantic model.

property json: JSONObject

Returns: The internal state of this *DiffieHellmanRatchet* as a JSON-serializable Python object.

classmethod from_model(*model*, *root_chain_kdf*, *message_chain_kdf*, *message_chain_constant*, *dos_protection_threshold*)

Parameters

- **model** (*DiffieHellmanRatchetModel*) – The pydantic model holding the internal state of a *DiffieHellmanRatchet*, as produced by *model*.
- **root_chain_kdf** (Type[*KDF*]) – The KDF to use for the root chain. The KDF must be capable of deriving 64 bytes.
- **message_chain_kdf** (Type[*KDF*]) – The KDF to use for the sending and receiving chains. The KDF must be capable of deriving 64 bytes.
- **message_chain_constant** (bytes) – The constant to feed into the sending and receiving KDF chains on each step.
- **dos_protection_threshold** (int) – The maximum number of skipped message keys to calculate. If more than that number of message keys are skipped, the keys are not calculated to prevent being DoSed.

Return type
TypeVar(DiffieHellmanRatchetTypeT, bound= DiffieHellmanRatchet)

Returns
A configured instance of *DiffieHellmanRatchet*, with internal state restored from the model.

Raises
InconsistentSerializationException – if the serialized data is structurally correct, but incomplete. This can only happen when migrating a *DoubleRatchet* instance from pre-stable data that was serialized before sending or receiving a single message. In this case, the serialized instance is basically uninitialized and can be discarded/replaced with a new instance without losing information.

Warning: Migrations are not provided via the *model*/*from_model()* API. Use *json*/*from_json()* instead. Refer to *Serialization and Migration* in the documentation for details.

classmethod from_json(*serialized*, *root_chain_kdf*, *message_chain_kdf*, *message_chain_constant*, *dos_protection_threshold*)

Parameters

- **serialized** (Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]], Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]]]]) – A JSON-serializable Python object holding the internal state of a *DiffieHellmanRatchet*, as produced by *json*.
- **root_chain_kdf** (Type[*KDF*]) – The KDF to use for the root chain. The KDF must be capable of deriving 64 bytes.
- **message_chain_kdf** (Type[*KDF*]) – The KDF to use for the sending and receiving chains. The KDF must be capable of deriving 64 bytes.
- **message_chain_constant** (bytes) – The constant to feed into the sending and receiving KDF chains on each step.

- **dos_protection_threshold** (int) – The maximum number of skipped message keys to calculate. If more than that number of message keys are skipped, the keys are not calculated to prevent being DoSed.

Return type
TypeVar(DiffieHellmanRatchetTypeT, bound= DiffieHellmanRatchet)

Returns
A configured instance of *DiffieHellmanRatchet*, with internal state restored from the serialized data.

Raises
InconsistentSerializationException – if the serialized data is structurally correct, but incomplete. This can only happen when migrating a *DoubleRatchet* instance from pre-stable data that was serialized before sending or receiving a single message. In this case, the serialized instance is basically uninitialized and can be discarded/replaced with a new instance without losing information.

Warning: Migrations are not provided via the *model*/*from_model()* API. Use *json*/*from_json()* instead. Refer to *Serialization and Migration* in the documentation for details.

async next_encryption_key()

Return type
Tuple[bytes, *Header*]

Returns
The next (32 bytes) encryption key derived from the sending chain and the corresponding Diffie-Hellman ratchet header.

async next_decryption_key(*header*)

Parameters
header (*Header*) – The Diffie-Hellman ratchet header,

Return type
Tuple[bytes, OrderedDict[Tuple[bytes, int], bytes]]

Returns
The next (32 bytes) decryption key derived from the receiving chain and message keys that were skipped while deriving the new decryption key.

Raises

- *DoSProtectionException* – if a huge number of message keys were skipped that have to be calculated first before decrypting the next message.
- *DuplicateMessageException* – if this message appears to be a duplicate.

exception doubleratchet.diffie_hellman_ratchet.**DoSProtectionException**

Bases: Exception

Raised by *DiffieHellmanRatchet.next_decryption_key()* in case the number of skipped message keys to calculate crosses the DoS protection threshold.

exception doubleratchet.diffie_hellman_ratchet.**DuplicateMessageException**

Bases: Exception

Raised by *DiffieHellmanRatchet.next_decryption_key()* in case is seems the message was processed before.

4.3 Module: double_ratchet

class doubleratchet.double_ratchet.**DoubleRatchet**

Bases: ABC

Combining the symmetric-key ratchet and the Diffie-Hellman ratchet gives the Double Ratchet.

https://signal.org/docs/specifications/doubleratchet/#double-ratchet

---

Note: In this implementation, the Diffie-Hellman ratchet already manages the symmetric-key ratchet internally, see *DiffieHellmanRatchet* for details. The Double Ratchet class adds message en-/decryption and offers a more convenient public API that handles lost and out-of-order messages.

---

async classmethod encrypt_initial_message(*diffie_hellman_ratchet_class*, *root_chain_kdf*, *message_chain_kdf*, *message_chain_constant*, *dos_protection_threshold*, *max_num_skipped_message_keys*, *aead*, *shared_secret*, *recipient_ratchet_pub*, *message*, *associated_data*)

Parameters

- **diffie_hellman_ratchet_class** (Type[*DiffieHellmanRatchet*]) – A non-abstract subclass of *DiffieHellmanRatchet*.
- **root_chain_kdf** (Type[*KDF*]) – The KDF to use for the root chain. The KDF must be capable of deriving 64 bytes.
- **message_chain_kdf** (Type[*KDF*]) – The KDF to use for the sending and receiving chains. The KDF must be capable of deriving 64 bytes.
- **message_chain_constant** (bytes) – The constant to feed into the sending and receiving KDF chains on each step.
- **dos_protection_threshold** (int) – The maximum number of skipped message keys to calculate. If more than that number of message keys are skipped, the keys are not calculated to prevent being DoSed.
- **max_num_skipped_message_keys** (int) – The maximum number of skipped message keys to store in case the lost or out-of-order message comes in later. Older keys are discarded to make space for newer keys.
- **aead** (Type[*AEAD*]) – The AEAD implementation to use for message en- and decryption.
- **shared_secret** (bytes) – A shared secret consisting of 32 bytes that was agreed on by means external to this protocol.
- **recipient_ratchet_pub** (bytes) – The ratchet public key of the recipient.
- **message** (bytes) – The initial message.
- **associated_data** (bytes) – Additional data to authenticate without including it in the ciphertext.

Return type

Tuple[TypeVar(DoubleRatchetTypeT, bound= DoubleRatchet), *EncryptedMessage*]

Returns
A configured instance of *DoubleRatchet* ready to send and receive messages together with the initial message.

async classmethod decrypt_initial_message(*diffie_hellman_ratchet_class*, *root_chain_kdf*, *message_chain_kdf*, *message_chain_constant*, *dos_protection_threshold*, *max_num_skipped_message_keys*, *aead*, *shared_secret*, *own_ratchet_priv*, *message*, *associated_data*)

Parameters

- **diffie_hellman_ratchet_class** (Type[*DiffieHellmanRatchet*]) – A non-abstract subclass of *DiffieHellmanRatchet*.
- **root_chain_kdf** (Type[*KDF*]) – The KDF to use for the root chain. The KDF must be capable of deriving 64 bytes.
- **message_chain_kdf** (Type[*KDF*]) – The KDF to use for the sending and receiving chains. The KDF must be capable of deriving 64 bytes.
- **message_chain_constant** (bytes) – The constant to feed into the sending and receiving KDF chains on each step.
- **dos_protection_threshold** (int) – The maximum number of skipped message keys to calculate. If more than that number of message keys are skipped, the keys are not calculated to prevent being DoSed.
- **max_num_skipped_message_keys** (int) – The maximum number of skipped message keys to store in case the lost or out-of-order message comes in later. Older keys are discarded to make space for newer keys.
- **aead** (Type[*AEAD*]) – The AEAD implementation to use for message en- and decryption.
- **shared_secret** (bytes) – A shared secret that was agreed on by means external to this protocol.
- **own_ratchet_priv** (bytes) – The ratchet private key to use initially.
- **message** (*EncryptedMessage*) – The encrypted initial message.
- **associated_data** (bytes) – Additional data to authenticate without including it in the ciphertext.

Return type
Tuple[TypeVar(DoubleRatchetTypeT, bound= DoubleRatchet), bytes]

Returns
A configured instance of *DoubleRatchet* ready to send and receive messages together with the decrypted initial message.

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).
- ***DoSProtectionException*** – if a huge number of message keys were skipped that have to be calculated first before decrypting the message.

property sending_chain_length: int

Returns: The length of the sending chain of the internal symmetric-key ratchet, as exposed by the internal Diffie-Hellman ratchet.

property receiving_chain_length: Optional[int]

Returns: The length of the receiving chain of the internal symmetric-key ratchet, if it exists, as exposed by the internal Diffie-Hellman ratchet.

abstract 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.

property model: *DoubleRatchetModel*

Returns: The internal state of this *DoubleRatchet* as a pydantic model.

property json: JSONObject

Returns: The internal state of this *DoubleRatchet* as a JSON-serializable Python object.

classmethod from_model(*model*, *diffie_hellman_ratchet_class*, *root_chain_kdf*, *message_chain_kdf*, *message_chain_constant*, *dos_protection_threshold*, *max_num_skipped_message_keys*, *aead*)

Parameters

- **model** (*DoubleRatchetModel*) – The pydantic model holding the internal state of a *DoubleRatchet*, as produced by *model*.
- **diffie_hellman_ratchet_class** (Type[*DiffieHellmanRatchet*]) – A non-abstract subclass of *DiffieHellmanRatchet*.
- **root_chain_kdf** (Type[*KDF*]) – The KDF to use for the root chain. The KDF must be capable of deriving 64 bytes.
- **message_chain_kdf** (Type[*KDF*]) – The KDF to use for the sending and receiving chains. The KDF must be capable of deriving 64 bytes.
- **message_chain_constant** (bytes) – The constant to feed into the sending and receiving KDF chains on each step.
- **dos_protection_threshold** (int) – The maximum number of skipped message keys to calculate. If more than that number of message keys are skipped, the keys are not calculated to prevent being DoSed.
- **max_num_skipped_message_keys** (int) – The maximum number of skipped message keys to store in case the lost or out-of-order message comes in later. Older keys are discarded to make space for newer keys.
- **aead** (Type[*AEAD*]) – The AEAD implementation to use for message en- and decryption.

Return type
TypeVar(DoubleRatchetTypeT, bound= DoubleRatchet)

Returns
A configured instance of *DoubleRatchet*, with internal state restored from the model.

Raises
InconsistentSerializationException – if the serialized data is structurally correct, but incomplete. This can only happen when migrating an instance from pre-stable data that was serialized before sending or receiving a single message. In this case, the serialized instance is basically uninitialized and can be discarded/replaced with a new instance using *encrypt_initial_message()* or *decrypt_initial_message()* without losing information.

Warning: Migrations are not provided via the *model*/*from_model()* API. Use *json*/*from_json()* instead. Refer to *Serialization and Migration* in the documentation for details.

classmethod from_json(*serialized*, *diffie_hellman_ratchet_class*, *root_chain_kdf*, *message_chain_kdf*, *message_chain_constant*, *dos_protection_threshold*, *max_num_skipped_message_keys*, *aead*)

Parameters

- **serialized** (Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]], Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]]]]) – A JSON-serializable Python object holding the internal state of a *DoubleRatchet*, as produced by *json*.
- **diffie_hellman_ratchet_class** (Type[*DiffieHellmanRatchet*]) – A non-abstract subclass of *DiffieHellmanRatchet*.
- **root_chain_kdf** (Type[*KDF*]) – The KDF to use for the root chain. The KDF must be capable of deriving 64 bytes.
- **message_chain_kdf** (Type[*KDF*]) – The KDF to use for the sending and receiving chains. The KDF must be capable of deriving 64 bytes.
- **message_chain_constant** (bytes) – The constant to feed into the sending and receiving KDF chains on each step.
- **dos_protection_threshold** (int) – The maximum number of skipped message keys to calculate. If more than that number of message keys are skipped, the keys are not calculated to prevent being DoSed.
- **max_num_skipped_message_keys** (int) – The maximum number of skipped message keys to store in case the lost or out-of-order message comes in later. Older keys are discarded to make space for newer keys.
- **aead** (Type[*AEAD*]) – The AEAD implementation to use for message en- and decryption.

Return type
TypeVar(DoubleRatchetTypeT, bound= DoubleRatchet)

Returns
A configured instance of *DoubleRatchet*, with internal state restored from the serialized data.

Raises
InconsistentSerializationException – if the serialized data is structurally correct, but incomplete. This can only happen when migrating an instance from pre-stable data

that was serialized before sending or receiving a single message. In this case, the serialized instance is basically uninitialized and can be discarded/replaced with a new instance using *encrypt_initial_message()* or *decrypt_initial_message()* without losing information.

async encrypt_message(*message*, *associated_data*)

Parameters

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

Return type
EncryptedMessage

Returns
The encrypted message including the header to send to the recipient.

async decrypt_message(*message*, *associated_data*)

Parameters

- **message** (*EncryptedMessage*) – The encrypted message.
- **associated_data** (bytes) – Additional data to authenticate without including it in the ciphertext.

Return type
bytes

Returns
The message plaintext, after decrypting and authenticating the ciphertext.

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).
- ***DoSProtectionException*** – if a huge number of message keys were skipped that have to be calculated first before decrypting the message.
- ***DuplicateMessageException*** – if this message appears to be a duplicate.

4.4 Module: kdf_chain

class doubleratchet.kdf_chain.**KDFChain**

Bases: object

The term KDF chain is used when some of the output from a KDF is used as an output key and some is used to replace the KDF key, which can then be used with another input.

https://signal.org/docs/specifications/doubleratchet/#kdf-chains

classmethod create(*kdf*, *key*)

Parameters

- **kdf** (Type[*KDF*]) – The KDF to use for the derivation step.
- **key** (bytes) – The initial chain key.

Return type
TypeVar(KDFChainTypeT, bound= KDFChain)

Returns
A configured instance of *KDFChain*.

property model: *KDFChainModel*

Returns: The internal state of this *KDFChain* as a pydantic model.

property json: JSONObject

Returns: The internal state of this *KDFChain* as a JSON-serializable Python object.

classmethod from_model(*model*, *kdf*)

Parameters

- **model** (*KDFChainModel*) – The pydantic model holding the internal state of a *KDFChain*, as produced by *model*.
- **kdf** (Type[*KDF*]) – The KDF to use for the derivation step.

Return type
TypeVar(KDFChainTypeT, bound= KDFChain)

Returns
A configured instance of *KDFChain*, with internal state restored from the model.

> **Warning:** Migrations are not provided via the *model*/*from_model()* API. Use *json*/*from_json()* instead. Refer to *Serialization and Migration* in the documentation for details.

classmethod from_json(*serialized*, *kdf*)

Parameters

- **serialized** (Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]], Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]]]]) – A JSON-serializable Python object holding the internal state of a *KDFChain*, as produced by *json*.
- **kdf** (Type[*KDF*]) – The KDF to use for the derivation step.

Return type
TypeVar(KDFChainTypeT, bound= KDFChain)

Returns
A configured instance of *KDFChain*, with internal state restored from the serialized data.

async step(*data*, *length*)

Perform a ratchet step of this KDF chain.

Parameters

- **data** (bytes) – The input data.
- **length** (int) – The desired size of the output data, in bytes.

Return type
bytes

Returns
length bytes of output data, derived from the internal KDF key and the input data.

property length: int
Returns: The length of this KDF chain, i.e. the number of steps that have been performed.

4.5 Module: kdf

class doubleratchet.kdf.**KDF**

Bases: ABC

A KDF is defined as a cryptographic function that takes a secret and random KDF key and some input data and returns output data. The output data is indistinguishable from random provided the key isn't known (i.e. a KDF satisfies the requirements of a cryptographic "PRF"). If the key is not secret and random, the KDF should still provide a secure cryptographic hash of its key and input data.

https://signal.org/docs/specifications/doubleratchet/#kdf-chains

abstract async static derive(*key*, *data*, *length*)

Parameters
- **key** (bytes) – The KDF key.
- **data** (bytes) – The input data.
- **length** (int) – The desired size of the output data, in bytes.

Return type
bytes

Returns
length bytes of output data, derived from the KDF key and the input data.

4.6 Module: migrations

Migrations between pydantic model versions, refer to *Serialization and Migration*

exception doubleratchet.migrations.**InconsistentSerializationException**

Bases: Exception

Raised by *parse_double_ratchet_model()* in case data migration from pre-stable serialization format is performed, and the data is structurally correct, but incomplete.

doubleratchet.migrations.**parse_diffie_hellman_ratchet_model**(*serialized*)

Parse a serialized *DiffieHellmanRatchet* instance, as returned by *json*, into the most recent pydantic model available for the class. Perform migrations in case the pydantic models were updated.

Parameters
serialized (Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]], Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]]]) – The serialized instance.

Return type
DiffieHellmanRatchetModel

Returns
The model, which can be used to restore the instance using *from_model()*.

Note: Pre-stable data can only be migrated as a whole using *parse_double_ratchet_model()*.

---

doubleratchet.migrations.**parse_double_ratchet_model**(*serialized*)

Parse a serialized *DoubleRatchet* instance, as returned by *json*, into the most recent pydantic model available for the class. Perform migrations in case the pydantic models were updated. Supports migration of pre-stable data.

Parameters
serialized (Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]], Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]]]) – The serialized instance.

Return type
DoubleRatchetModel

Returns
The model, which can be used to restore the instance using *from_model()*.

Raises
InconsistentSerializationException – if migration from pre-stable serialization format is performed, and the data is structurally correct, but incomplete. In pre-stable, it was possible to serialize instances which were not fully initialized yet. Those instances can be treated as non-existent and be replaced without losing information/messages.

Note: The pre-stable serialization format left it up to the user to implement serialization of key pairs. The migration code assumes the format used by pre-stable python-omemo and will raise an exception if a different format was used. In that case, the custom format has to be migrated first by the user.

---

doubleratchet.migrations.**parse_kdf_chain_model**(*serialized*)

Parse a serialized *KDFChain* instance, as returned by *json*, into the most recent pydantic model available for the class. Perform migrations in case the pydantic models were updated.

Parameters
serialized (Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]], Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]]]) – The serialized instance.

Return type
KDFChainModel

Returns
The model, which can be used to restore the instance using *from_model()*.

Note: Pre-stable data can only be migrated as a whole using *parse_double_ratchet_model()*.

doubleratchet.migrations.**parse_symmetric_key_ratchet_model**(*serialized*)

Parse a serialized *SymmetricKeyRatchet* instance, as returned by *json*, into the most recent pydantic model available for the class. Perform migrations in case the pydantic models were updated.

Parameters
serialized (Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]]], Mapping[str, Union[None, float, int, str, bool]]]], Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]]]) – The serialized instance.

Return type
SymmetricKeyRatchetModel

Returns
The model, which can be used to restore the instance using *from_model()*.

Note: Pre-stable data can only be migrated as a whole using *parse_double_ratchet_model()*.

4.7 Module: models

class doubleratchet.models.**DiffieHellmanRatchetModel**(***data*)

Bases: BaseModel

The model representing the internal state of a *DiffieHellmanRatchet*.

Parameters
- **version** (*str*) –
- **own_ratchet_priv** (*bytes*) –
- **other_ratchet_pub** (*bytes*) –
- **root_chain** (KDFChainModel) –
- **symmetric_key_ratchet** (SymmetricKeyRatchetModel) –

version: str

own_ratchet_priv: bytes

other_ratchet_pub: bytes

root_chain: *KDFChainModel*

symmetric_key_ratchet: *SymmetricKeyRatchetModel*

class Config

Bases: object

json_encoders = {<class 'bytes'>: <function _json_bytes_encoder>}

class doubleratchet.models.**DoubleRatchetModel**(***data*)

Bases: BaseModel

The model representing the internal state of a *DoubleRatchet*.

Parameters

- **version** (*str*) –
- **diffie_hellman_ratchet** (DiffieHellmanRatchetModel) –
- **skipped_message_keys** (*List[*SkippedMessageKeyModel*]*) –

version: str

diffie_hellman_ratchet: *DiffieHellmanRatchetModel*

skipped_message_keys: List[*SkippedMessageKeyModel*]

class Config

Bases: object

json_encoders = {<class 'bytes'>: <function _json_bytes_encoder>}

class doubleratchet.models.**KDFChainModel**(***data*)

Bases: BaseModel

The model representing the internal state of a *KDFChain*.

Parameters

- **version** (*str*) –
- **length** (*int*) –
- **key** (*bytes*) –

version: str

length: int

key: bytes

class Config

Bases: object

json_encoders = {<class 'bytes'>: <function _json_bytes_encoder>}

class doubleratchet.models.**SkippedMessageKeyModel**(***data*)

Bases: BaseModel

The model used as part of the *DoubleRatchetModel*, representing a single skipped message key with meta data.

Parameters

- **ratchet_pub** (*bytes*) –
- **index** (*int*) –
- **message_key** (*bytes*) –

ratchet_pub: bytes

index: int

message_key: bytes

class Config

Bases: object

json_encoders = {<class 'bytes'>: <function _json_bytes_encoder>}

class doubleratchet.models.**SymmetricKeyRatchetModel**(***data*)

Bases: BaseModel

The model representing the internal state of a *SymmetricKeyRatchet*.

Parameters

- **version** (*str*) –
- **receiving_chain** (*Optional[KDFChainModel]*) –
- **sending_chain** (*Optional[KDFChainModel]*) –
- **previous_sending_chain_length** (*Optional[int]*) –

version: str

receiving_chain: Optional[*KDFChainModel*]

sending_chain: Optional[*KDFChainModel*]

previous_sending_chain_length: Optional[int]

4.8 Module: symmetric_key_ratchet

class doubleratchet.symmetric_key_ratchet.**Chain**(*value*)

Bases: Enum

Enumeration identifying the chain to replace by *SymmetricKeyRatchet.replace_chain()*.

SENDING: str = 'SENDING'

RECEIVING: str = 'RECEIVING'

exception doubleratchet.symmetric_key_ratchet.**ChainNotAvailableException**

Bases: Exception

Raised by *SymmetricKeyRatchet.next_encryption_key()* and *SymmetricKeyRatchet.next_decryption_key()* in case the required chain has not been initialized yet.

class doubleratchet.symmetric_key_ratchet.**SymmetricKeyRatchet**

Bases: object

The sending and receiving chains advance as each message is sent and received. Their output keys are used to encrypt and decrypt messages. This is called the symmetric-key ratchet.

https://signal.org/docs/specifications/doubleratchet/#symmetric-key-ratchet

classmethod create(*chain_kdf*, *constant*)

Parameters

- **chain_kdf** (Type[*KDF*]) – The KDF to use for the sending and receiving chains. The KDF must be capable of deriving 64 bytes.
- **constant** (bytes) – The constant to feed into the sending and receiving KDF chains on each step.

Return type
TypeVar(SymmetricKeyRatchetTypeT, bound= SymmetricKeyRatchet)

Returns
A configured instance of *SymmetricKeyRatchet*.

property model: *SymmetricKeyRatchetModel*
Returns: The internal state of this *SymmetricKeyRatchet* as a pydantic model.

property json: JSONObject
Returns: The internal state of this *SymmetricKeyRatchet* as a JSON-serializable Python object.

classmethod from_model(*model*, *chain_kdf*, *constant*)

Parameters

- **model** (*SymmetricKeyRatchetModel*) – The pydantic model holding the internal state of a *SymmetricKeyRatchet*, as produced by *model*.
- **chain_kdf** (Type[*KDF*]) – The KDF to use for the sending and receiving chains. The KDF must be capable of deriving 64 bytes.
- **constant** (bytes) – The constant to feed into the sending and receiving KDF chains on each step.

Return type
TypeVar(SymmetricKeyRatchetTypeT, bound= SymmetricKeyRatchet)

Returns
A configured instance of *SymmetricKeyRatchet*, with internal state restored from the model.

Warning: Migrations are not provided via the *model*/*from_model()* API. Use *json*/*from_json()* instead. Refer to *Serialization and Migration* in the documentation for details.

classmethod from_json(*serialized*, *chain_kdf*, *constant*)

Parameters

- **serialized** (Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]], Mapping[str, Union[None, float, int, str, bool, List[Union[None, float, int, str, bool]], Mapping[str, Union[None, float, int, str, bool]]]]]]) – A JSON-serializable Python object holding the internal state of a *SymmetricKeyRatchet*, as produced by *json*.
- **chain_kdf** (Type[*KDF*]) – The KDF to use for the sending and receiving chains. The KDF must be capable of deriving 64 bytes.
- **constant** (bytes) – The constant to feed into the sending and receiving KDF chains on each step.

Return type
TypeVar(SymmetricKeyRatchetTypeT, bound= SymmetricKeyRatchet)

Returns
A configured instance of *SymmetricKeyRatchet*, with internal state restored from the serialized data.

replace_chain(*chain*, *key*)

Replace either the sending or the receiving chain with a new KDF chain.

Parameters

- **chain** (*Chain*) – The chain to replace.
- **key** (bytes) – The initial chain key for the new KDF chain.

Return type
None

property previous_sending_chain_length: Optional[int]

Returns: The length of the previous sending chain, if it exists.

property sending_chain_length: Optional[int]

Returns: The length of the sending chain, if it exists.

property receiving_chain_length: Optional[int]

Returns: The length of the receiving chain, if it exists.

async next_encryption_key()

Return type
bytes

Returns
The next (32 bytes) encryption key derived from the sending chain.

Raises
ChainNotAvailableException – if the sending chain was never initialized.

async next_decryption_key()

Return type
bytes

Returns
The next (32 bytes) decryption key derived from the receiving chain.

Raises
ChainNotAvailableException – if the receiving chain was never initialized.

4.9 Module: types

class doubleratchet.types.**EncryptedMessage**(*header*, *ciphertext*)

Bases: tuple

A Double Ratchet-encrypted message, consisting of the header and ciphertext.

Parameters

- **header** (Header) –

- **ciphertext** (*bytes*) –

property header

Alias for field number 0

property ciphertext

Alias for field number 1

class doubleratchet.types.**Header**(*ratchet_pub*, *previous_sending_chain_length*, *sending_chain_length*)

Bases: tuple

The header structure sent with each Double Ratchet-encrypted message, containing the metadata to keep the ratchets synchronized.

Parameters

- **ratchet_pub** (*bytes*) –
- **previous_sending_chain_length** (*int*) –
- **sending_chain_length** (*int*) –

property ratchet_pub

Alias for field number 0

property previous_sending_chain_length

Alias for field number 1

property sending_chain_length

Alias for field number 2

4.10 x3dh.recommended

4.10.1 Module: aead_aes_hmac

class doubleratchet.recommended.aead_aes_hmac.**AEAD**

Bases: *AEAD*

An implementation of Authenticated Encryption with Associated Data using AES-256 in CBC mode, HKDF and HMAC with SHA-256 or SHA-512:

HKDF is used with SHA-256 or SHA-512 to generate 80 bytes of output. The HKDF salt is set to a zero-filled byte sequence equal to the digest size of the hash function. HKDF input key material is set to AEAD key. HKDF info is set to an application-specific byte sequence distinct from other uses of HKDF in the application.

The HKDF output is divided into a 32-byte encryption key, a 32-byte authentication key, and a 16-byte IV.

The plaintext is encrypted using AES-256 in CBC mode with PKCS#7 padding, using the encryption key and IV from the previous step.

HMAC is calculated using the authentication key and the same hash function as above. The HMAC input is the associated_data prepended to the ciphertext. The HMAC output is appended to the ciphertext.

abstract static _get_hash_function()

Return type
HashFunction

abstract 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).

4.10.2 Module: diffie_hellman_ratchet_curve25519

class doubleratchet.recommended.diffie_hellman_ratchet_curve25519.**DiffieHellmanRatchet**

Bases: *DiffieHellmanRatchet*

An implementation of *DiffieHellmanRatchet* using Curve25519 keys and performing X25519 key exchanges.

Implementation relies on the Python package cryptography.

static _generate_priv()

Return type
bytes

Returns
A freshly generated private key, capable of performing Diffie-Hellman key exchanges with the public key of another party.

Note: This function is recommended to generate a key pair based on the Curve25519 or Curve448 elliptic curves (https://signal.org/docs/specifications/doubleratchet/#recommended-cryptographic-algorithms).

---

static _derive_pub(*priv*)

Derive the public key from a private key as generated by *_generate_priv()*.

Parameters
priv (bytes) – The private key as returned by *_generate_priv()*.

Return type
bytes

Returns
The public key corresponding to the private key.

static _perform_diffie_hellman(*own_priv*, *other_pub*)

Parameters

- **own_priv** (bytes) – The own ratchet private key.
- **other_pub** (bytes) – The ratchet public key of the other party.

Return type
bytes

Returns
A shared secret agreed on via Diffie-Hellman. This method is recommended to perform X25519 or X448. There is no need to check for invalid public keys (https://signal.org/docs/specifications/doubleratchet/#recommended-cryptographic-algorithms).

4.10.3 Module: diffie_hellman_ratchet_curve448

class doubleratchet.recommended.diffie_hellman_ratchet_curve448.**DiffieHellmanRatchet**

Bases: *DiffieHellmanRatchet*

An implementation of *DiffieHellmanRatchet* using Curve448 keys and performing X448 key exchanges.

Implementation relies on the Python package cryptography.

static _generate_priv()

Return type
bytes

Returns
A freshly generated private key, capable of performing Diffie-Hellman key exchanges with the public key of another party.

Note: This function is recommended to generate a key pair based on the Curve25519 or Curve448 elliptic curves (https://signal.org/docs/specifications/doubleratchet/#recommended-cryptographic-algorithms).

---

static _derive_pub(*priv*)

Derive the public key from a private key as generated by *_generate_priv()*.

Parameters
priv (bytes) – The private key as returned by *_generate_priv()*.

Return type
bytes

Returns
The public key corresponding to the private key.

static _perform_diffie_hellman(*own_priv*, *other_pub*)

Parameters

- **own_priv** (bytes) – The own ratchet private key.
- **other_pub** (bytes) – The ratchet public key of the other party.

Return type
bytes

Returns
A shared secret agreed on via Diffie-Hellman. This method is recommended to perform X25519 or X448. There is no need to check for invalid public keys (https://signal.org/docs/specifications/doubleratchet/#recommended-cryptographic-algorithms).

4.10.4 Module: hash_function

4.10.5 Module: kdf_hkdf

class doubleratchet.recommended.kdf_hkdf.**KDF**

Bases: *KDF*

This KDF implemention uses HKDF with SHA-256 or SHA-512, using the KDF key as HKDF salt, the KDF data as HKDF input key material, and an application-specific byte sequence as HKDF info. The info value should be chosen to be distinct from other uses of HKDF in the application.

https://signal.org/docs/specifications/doubleratchet/#recommended-cryptographic-algorithms

abstract static _get_hash_function()

Return type
HashFunction

abstract static _get_info()

Return type
bytes

async classmethod derive(*key*, *data*, *length*)

Parameters

- **key** (bytes) – The KDF key.
- **data** (bytes) – The input data.
- **length** (int) – The desired size of the output data, in bytes.

Return type
bytes

Returns
length bytes of output data, derived from the KDF key and the input data.

4.10.6 Module: kdf_separate_hmacs

class doubleratchet.recommended.kdf_separate_hmacs.**KDF**

Bases: *KDF*

This implementation uses HMAC with SHA-256 or SHA-512 to derive multiple outputs from a single KDF key. These outputs are concatenated and returned as a whole. The KDF key is used as the HMAC key, the KDF data is split into single bytes and one HMAC is calculated for each byte. For example, passing `b"\x01\x02"` as the KDF data results in two HMACs being calculated, one using `b"\x01"` as the HMAC input and the other using `b"\x02"`. The two HMAC outputs are concatenated and returned. Note that the length of the output is fixed to a multiple of the HMAC digest size, based on the length of the KDF data.

https://signal.org/docs/specifications/doubleratchet/#recommended-cryptographic-algorithms

abstract static _get_hash_function()

Return type
HashFunction

async classmethod derive(*key*, *data*, *length*)

Parameters

- **key** (bytes) – The KDF key.
- **data** (bytes) – The input data.
- **length** (int) – The desired size of the output data, in bytes.

Return type
bytes

Returns
length bytes of output data, derived from the KDF key and the input data.

PYTHON MODULE INDEX

d

INDEX

Symbols

A

C

D

E

F

H

I

J

K

L

M

N

O

P

R

S

V