

---

# **noiseprotocol**

***Release 0.3.0***

**Feb 24, 2019**



---

## Contents:

---

<b>1</b>	<b>Documentation for the Code</b>	<b>1</b>
<b>2</b>	<b>Indices and tables</b>	<b>5</b>
	<b>Python Module Index</b>	<b>7</b>



---

Documentation for the Code

---

**class** `noise.state.CipherState` (*noise\_protocol*)

Implemented as per Noise Protocol specification - paragraph 5.1.

The `initialize_key()` function takes additional required argument - `noise_protocol`.

This class holds an instance of Cipher wrapper. It manages initialisation of underlying cipher function with appropriate key in `initialize_key()` and `rekey()` methods.

**decrypt\_with\_ad** (*ad: bytes, ciphertext: bytes*) → bytes

If `k` is non-empty returns `DECRYPT(k, n++, ad, ciphertext)`. Otherwise returns ciphertext. If an authentication failure occurs in `DECRYPT()` then `n` is not incremented and an error is signaled to the caller.

**Parameters**

- **ad** – bytes sequence
- **ciphertext** – bytes sequence

**Returns** plaintext bytes sequence

**encrypt\_with\_ad** (*ad: bytes, plaintext: bytes*) → bytes

If `k` is non-empty returns `ENCRYPT(k, n++, ad, plaintext)`. Otherwise returns plaintext.

**Parameters**

- **ad** – bytes sequence
- **plaintext** – bytes sequence

**Returns** ciphertext bytes sequence

**has\_key** ()

**Returns** True if `self.k` is not an instance of Empty

**initialize\_key** (*key*)

**Parameters** **key** – Key to set within CipherState

**class** noise.state.HandshakeState

Implemented as per Noise Protocol specification - paragraph 5.3.

The initialize() function takes different required argument - noise\_protocol, which contains handshake\_pattern.

**classmethod initialize** (noise\_protocol: NoiseProtocol, initiator: bool, prologue: bytes = b'',  
s: \_KeyPair = None, e: \_KeyPair = None, rs: \_KeyPair = None, re:  
\_KeyPair = None) → HandshakeState

Constructor method. Comments below are mostly copied from specification. Instead of taking handshake\_pattern as an argument, we take full NoiseProtocol object, that way we have access to protocol name and crypto functions

**Parameters**

- **noise\_protocol** – a valid NoiseProtocol instance
- **initiator** – boolean indicating the initiator or responder role
- **prologue** – byte sequence which may be zero-length, or which may contain context information that both parties want to confirm is identical
- **s** – local static key pair
- **e** – local ephemeral key pair
- **rs** – remote party's static public key
- **re** – remote party's ephemeral public key

**Returns** initialized HandshakeState instance

**read\_message** (message: Union[bytes, bytearray], payload\_buffer: bytearray)

Comments below are mostly copied from specification.

**Parameters**

- **message** – byte sequence containing a Noise handshake message
- **payload\_buffer** – buffer-like object

**Returns** None or result of SymmetricState.split() - tuple (CipherState, CipherState)

**write\_message** (payload: Union[bytes, bytearray], message\_buffer: bytearray)

Comments below are mostly copied from specification.

**Parameters**

- **payload** – byte sequence which may be zero-length
- **message\_buffer** – buffer-like object

**Returns** None or result of SymmetricState.split() - tuple (CipherState, CipherState)

**class** noise.state.SymmetricState

Implemented as per Noise Protocol specification - paragraph 5.2.

The initialize\_symmetric function takes different required argument - noise\_protocol, which contains protocol\_name.

**decrypt\_and\_hash** (ciphertext: bytes) → bytes

Sets plaintext = DecryptWithAd(h, ciphertext), calls MixHash(ciphertext), and returns plaintext. Note that if k is empty, the DecryptWithAd() call will set plaintext equal to ciphertext.

**Parameters** **ciphertext** – bytes sequence

**Returns** plaintext bytes sequence

**encrypt\_and\_hash** (*plaintext: bytes*) → bytes

Sets ciphertext = EncryptWithAd(h, plaintext), calls MixHash(ciphertext), and returns ciphertext. Note that if k is empty, the EncryptWithAd() call will set ciphertext equal to plaintext.

**Parameters** **plaintext** – bytes sequence

**Returns** ciphertext bytes sequence

**classmethod initialize\_symmetric** (*noise\_protocol: NoiseProtocol*) → SymmetricState

Instead of taking protocol\_name as an argument, we take full NoiseProtocol object, that way we have access to protocol name and crypto functions

Comments below are mostly copied from specification.

**Parameters** **noise\_protocol** – a valid NoiseProtocol instance

**Returns** initialised SymmetricState instance

**mix\_hash** (*data: bytes*)

Sets h = HASH(h + data).

**Parameters** **data** – bytes sequence

**mix\_key** (*input\_key\_material: bytes*)

**Parameters** **input\_key\_material** –

**Returns**

**split** ()

Returns a pair of CipherState objects for encrypting/decrypting transport messages.

**Returns** tuple (CipherState, CipherState)





## CHAPTER 2

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



**n**

`noise.state, 1`



## C

CipherState (class in noise.state), 1

## D

decrypt\_and\_hash() (noise.state.SymmetricState method), 2

decrypt\_with\_ad() (noise.state.CipherState method), 1

## E

encrypt\_and\_hash() (noise.state.SymmetricState method), 2

encrypt\_with\_ad() (noise.state.CipherState method), 1

## H

HandshakeState (class in noise.state), 1

has\_key() (noise.state.CipherState method), 1

## I

initialize() (noise.state.HandshakeState class method), 2

initialize\_key() (noise.state.CipherState method), 1

initialize\_symmetric() (noise.state.SymmetricState class method), 3

## M

mix\_hash() (noise.state.SymmetricState method), 3

mix\_key() (noise.state.SymmetricState method), 3

## N

noise.state (module), 1

## R

read\_message() (noise.state.HandshakeState method), 2

## S

split() (noise.state.SymmetricState method), 3

SymmetricState (class in noise.state), 2

## W

write\_message() (noise.state.HandshakeState method), 2