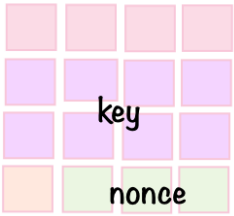
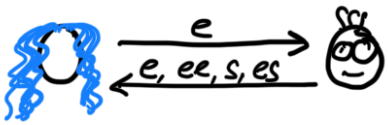
Noise protocol used in stratum v2

Alice and Bob can have:



e = ephemeral key
s = static key
h = handshake hash (useful for auth)
ck = chaining key (for making transport msg ciphers)

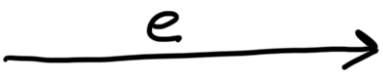
NX handshake



NX handshake

ALICE

initiator

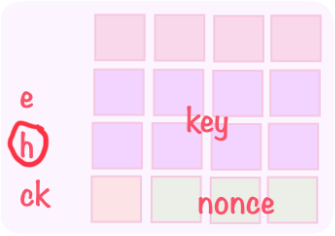


BOB

responder

Alice sends her e (unencrypted but encoded using ellswift as pseudorandom bytes)

operations:
- h is updated to include e by hashing



When Bob receives her e

operations:
- h is updated to include e by hashing

ALICE

INITIATOR

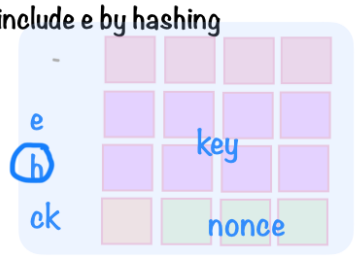
BOB

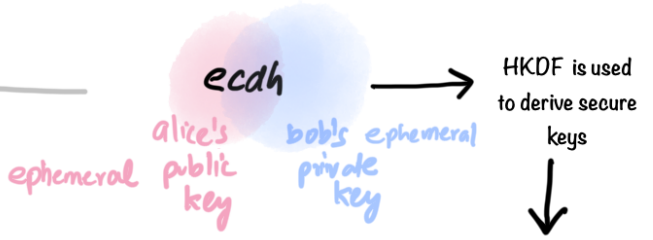
RESPONDER

Bob sends his e (unencrypted but encoded using ellswift as pseudorandom bytes)

operations:

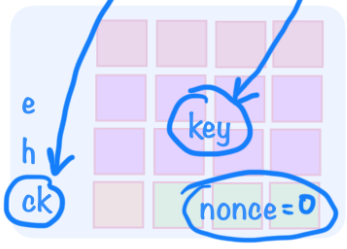
- h is updated to include e by hashing





operations:

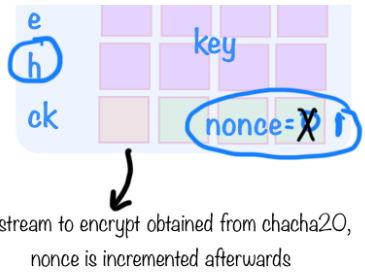
1. k is updated
2. ck is updated



Bob sends his s (encrypted and encoded using ellswift as pseudorandom bytes)

operations:

- h up till now is used as aad in the encrypted static key message
- h is updated to include s by hashing



keystream to encrypt obtained from chacha20, nonce is incremented afterwards

es

ecdh

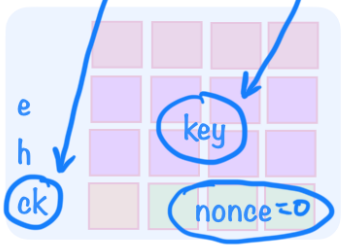
HKDF is used to derive secure keys

alice's ephemeral public key

bobs static private key

secure key

operations:
1. k is updated
2. ck is updated



nonce is set to 0

CERTIFICATE

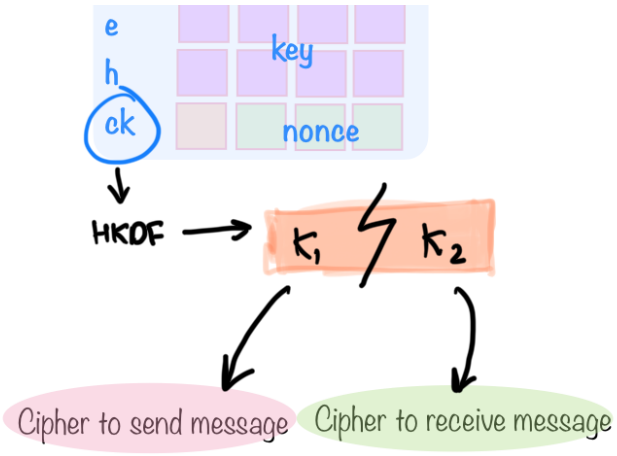
CERTIFICATE consists of:

version	2 bytes
valid_from	4 bytes
valid_to	4 bytes
signature signed using pool authority key	64 bytes

Bob sends an encrypted certificate

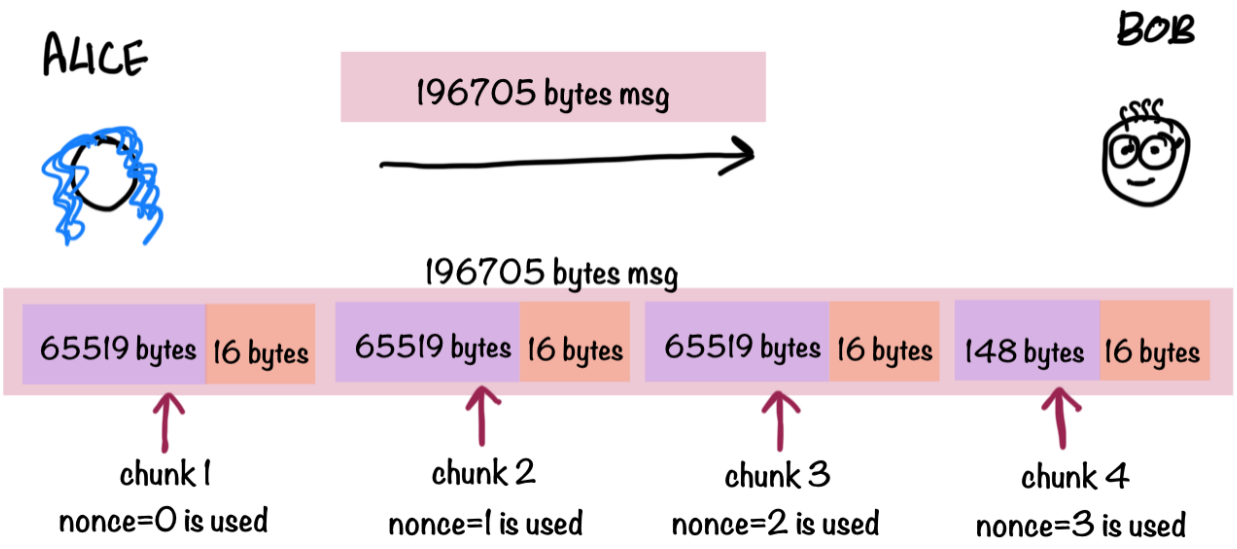
operations:
- h up till now is used as aad in the encrypted certificate message
- h is updated to include the encrypted message by hashing

Finally, Bob uses the chaining key ck to derive keys to instantiate ciphers for sending and receiving transport messages



Transport phase

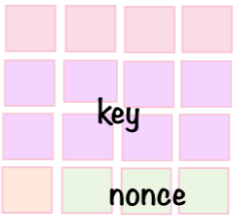
Alice wants to send a 196705 bytes packet to Bob.
However, the noise protocol framework supports message of max length 65535 bytes.
So it's chunked into pieces so that each chunk after encryption is 65535 bytes (including 16 bytes MAC tag for each chunk)



ALICE

Alice has 2 ciphers - k1 for encrypting+sending messages and k2 for receiving+decrypting messages.

The cipher uses:
- chacha20 for packet encryption
- poly1305 for packet authentication



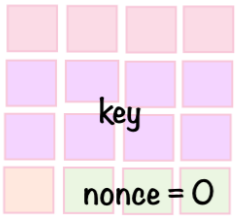
chacha20 algo scrambles the matrix to produce 64 bytes keystream at a time

65519 bytes

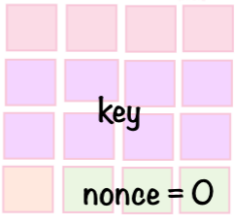
64 bytes	64 bytes	64 bytes	...	64 bytes	47 bytes	16 bytes MAC
1	2	3		1023	1024	

64 bytes keystream to encipher the plaintext is derived from chacha20 with counter = 1, 2, ..

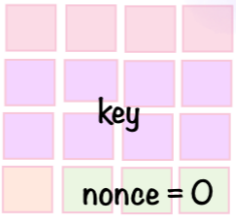
key to generate Poly1305 MAC tag is derived from the same chacha20 with counter=0



counter=1



counter=2



counter=0