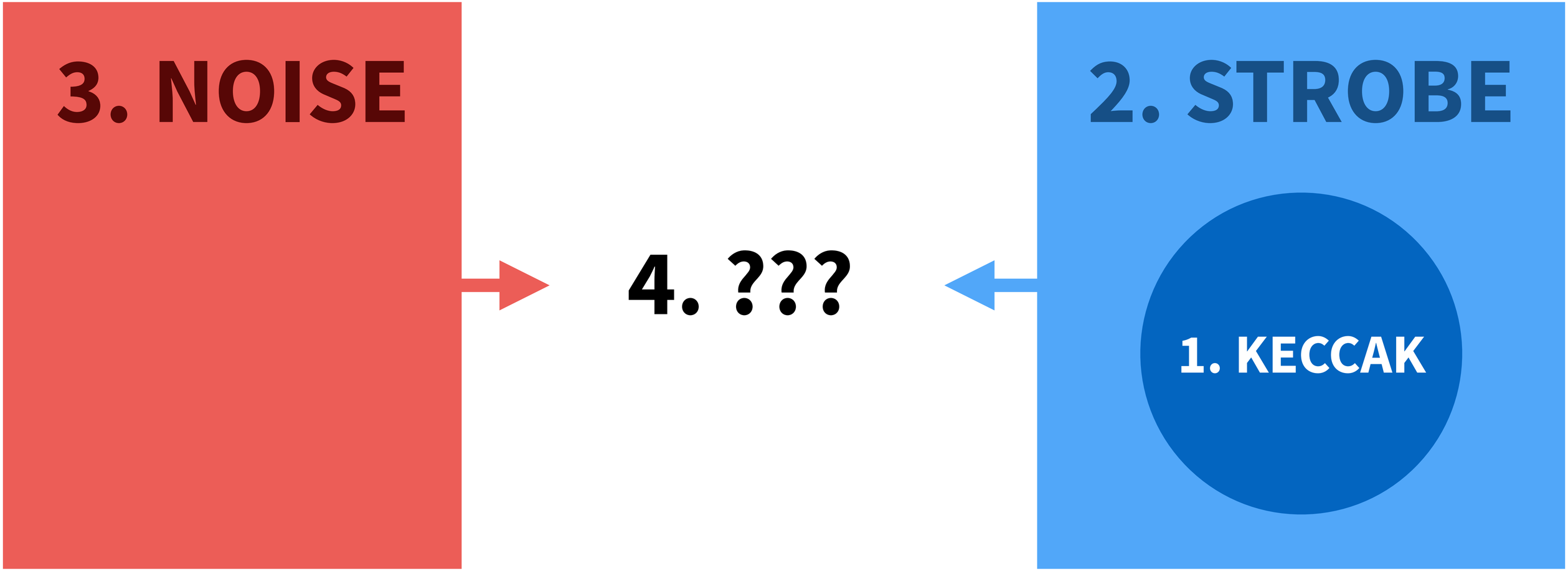
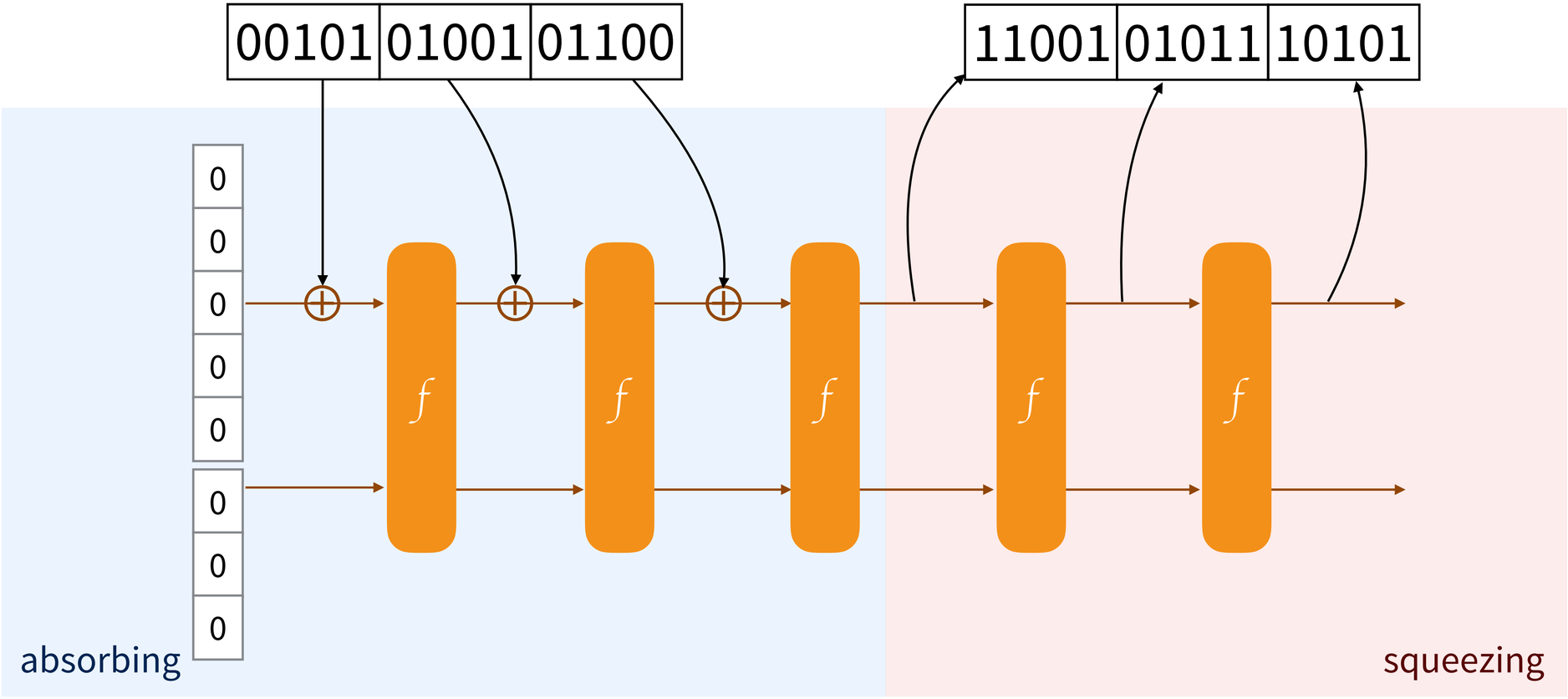
Modern Session Encryption

David Wong

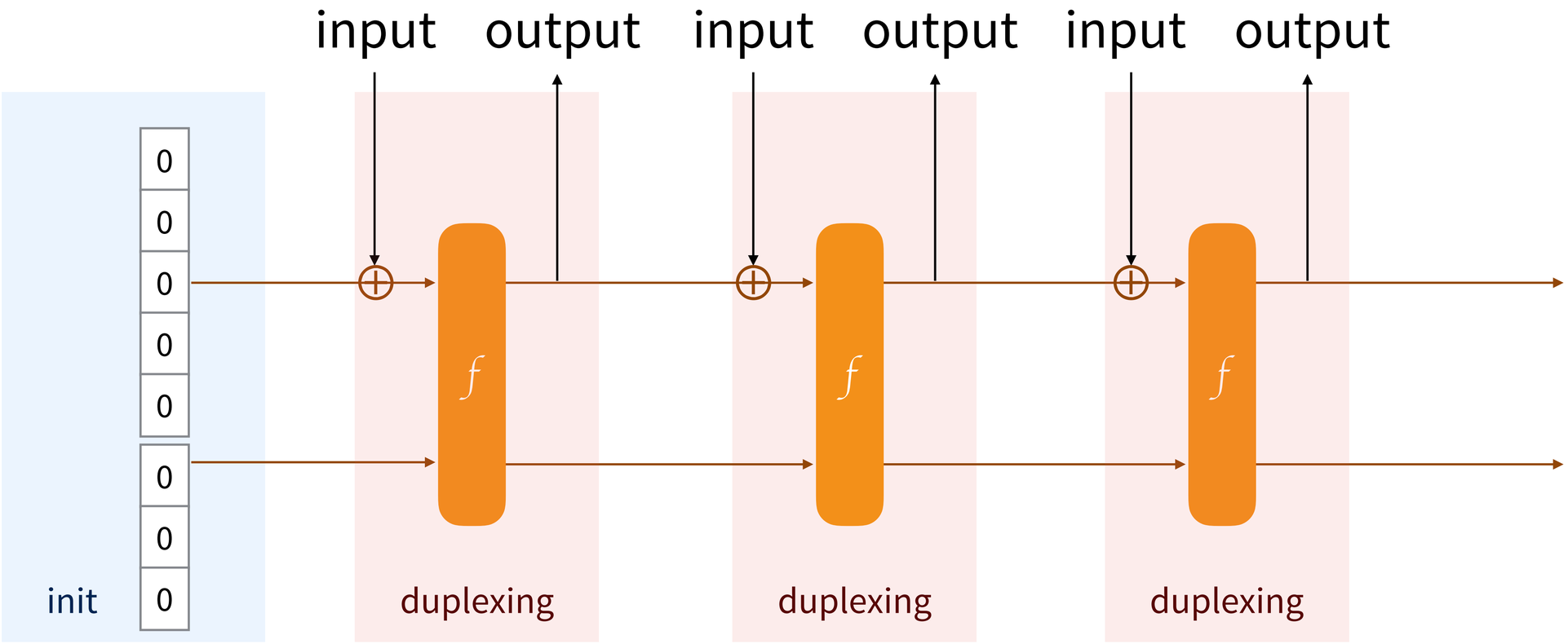
outline



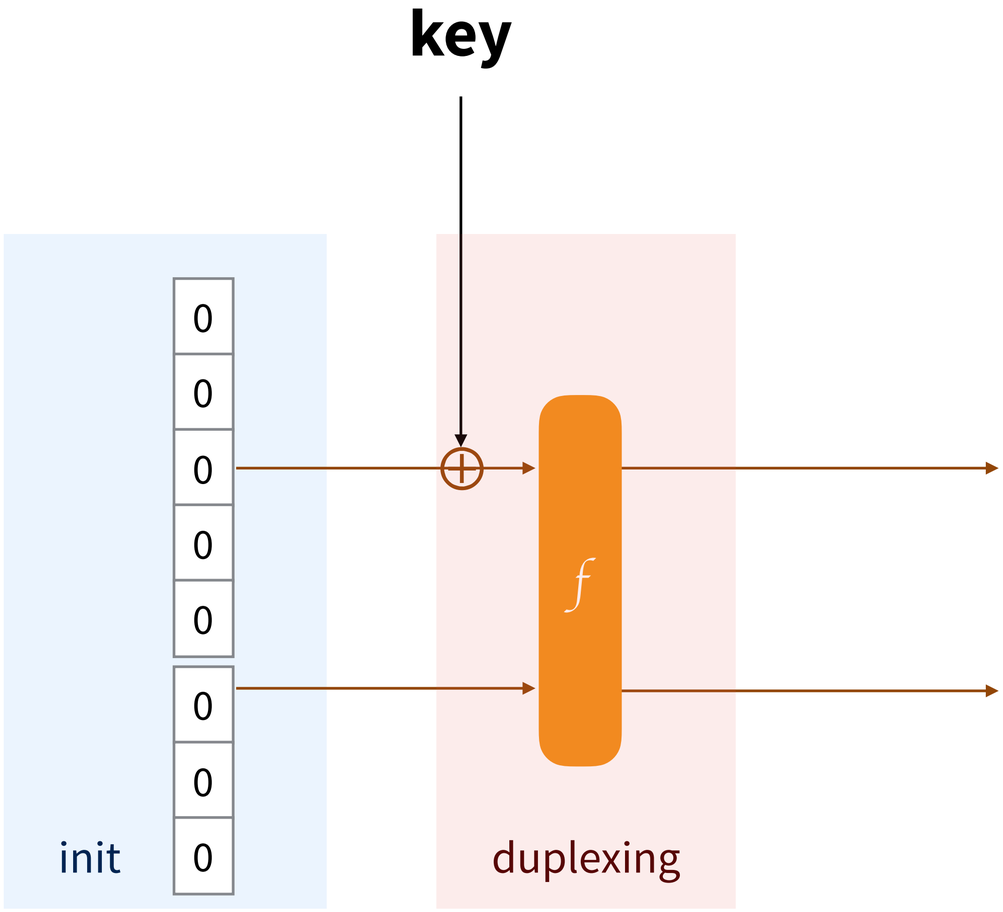
Sponge Construction



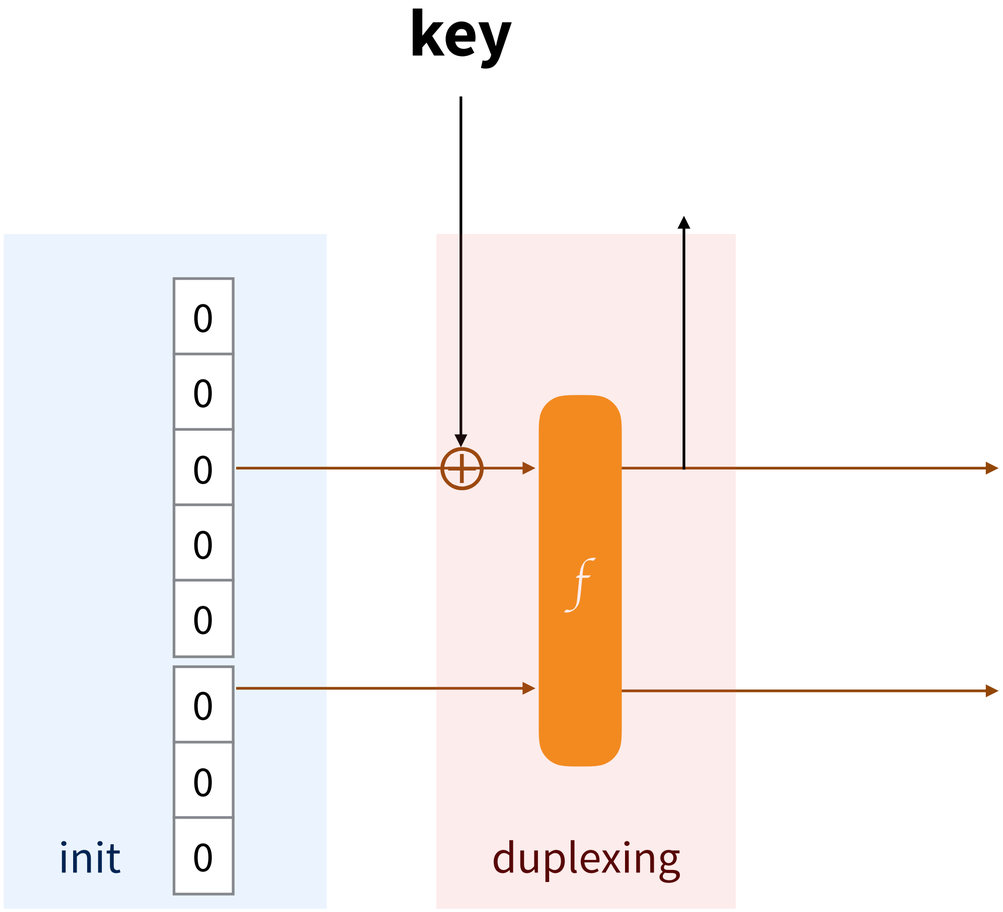
Duplex Construction



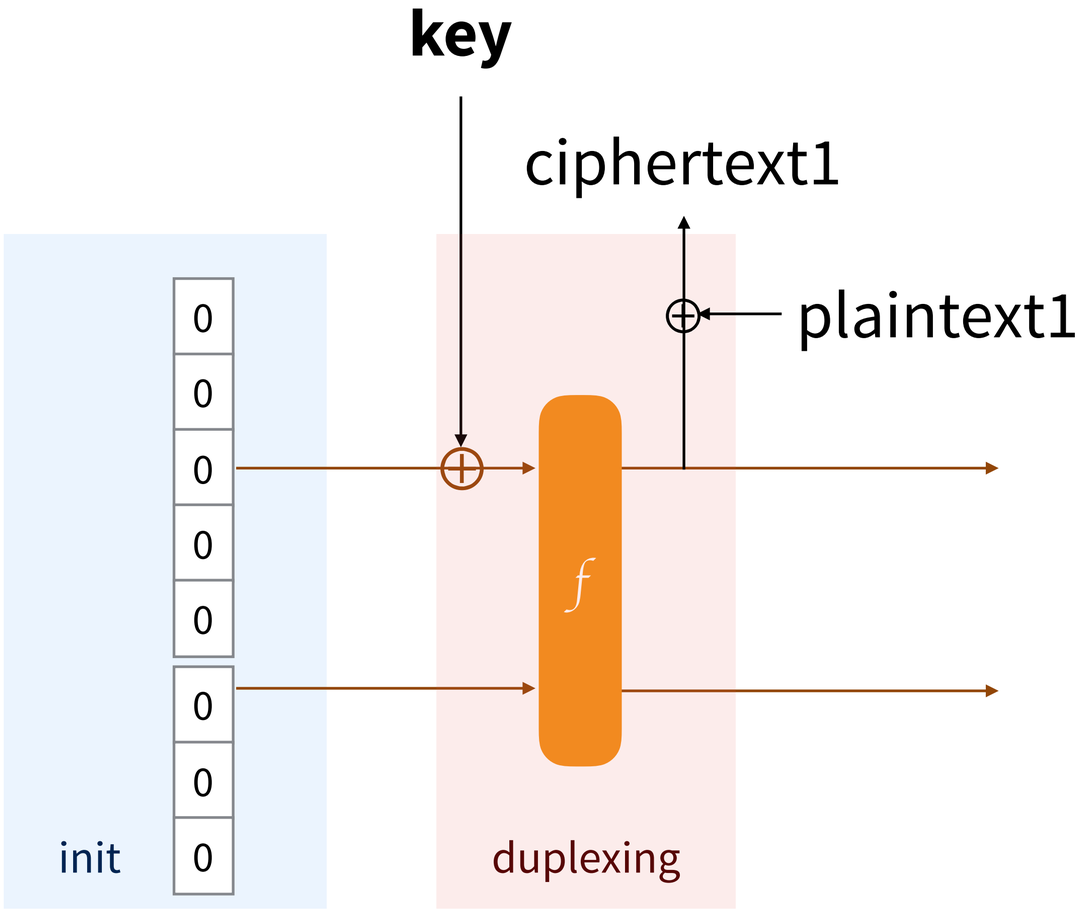
Keyed-mode



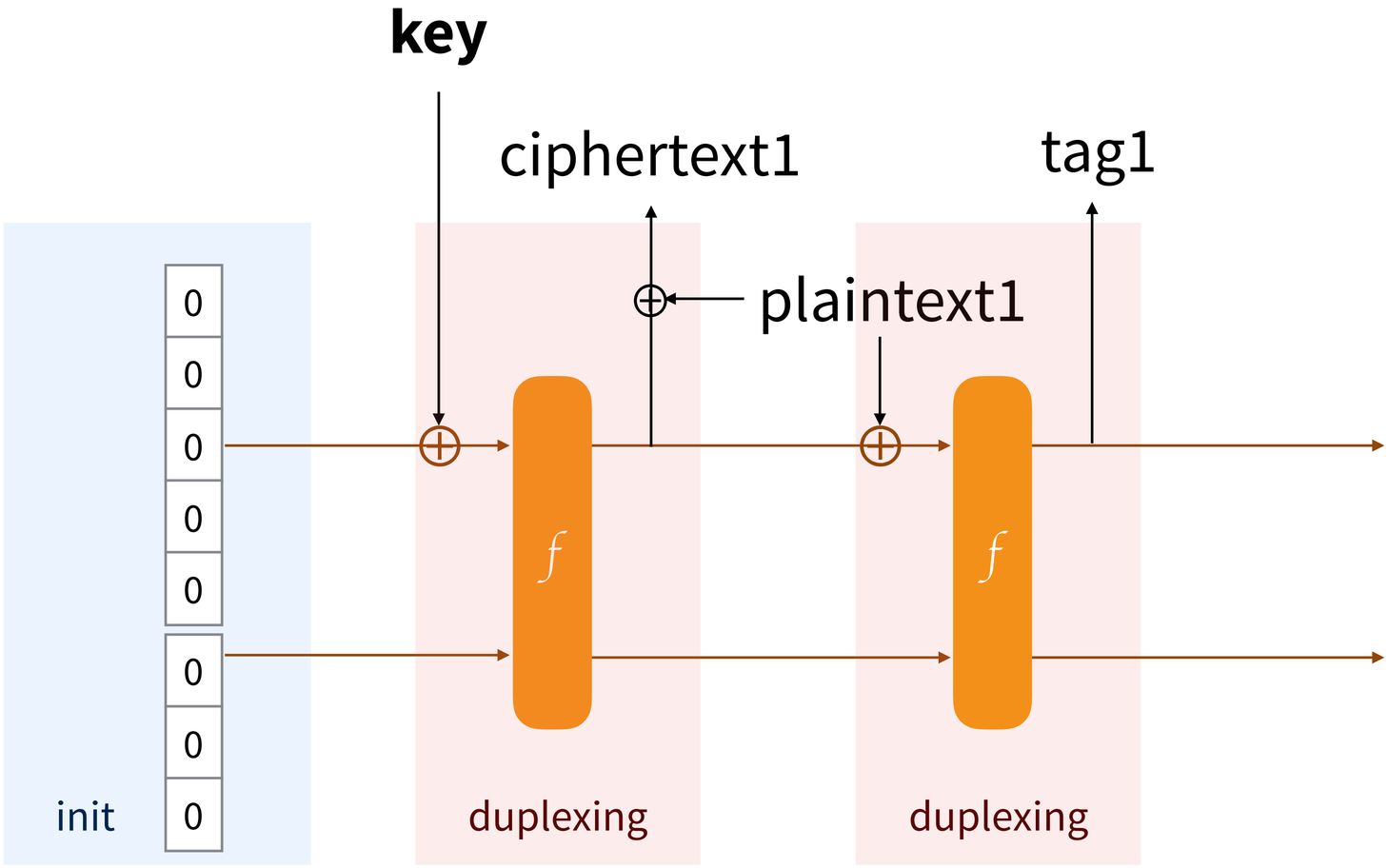
Encryption?



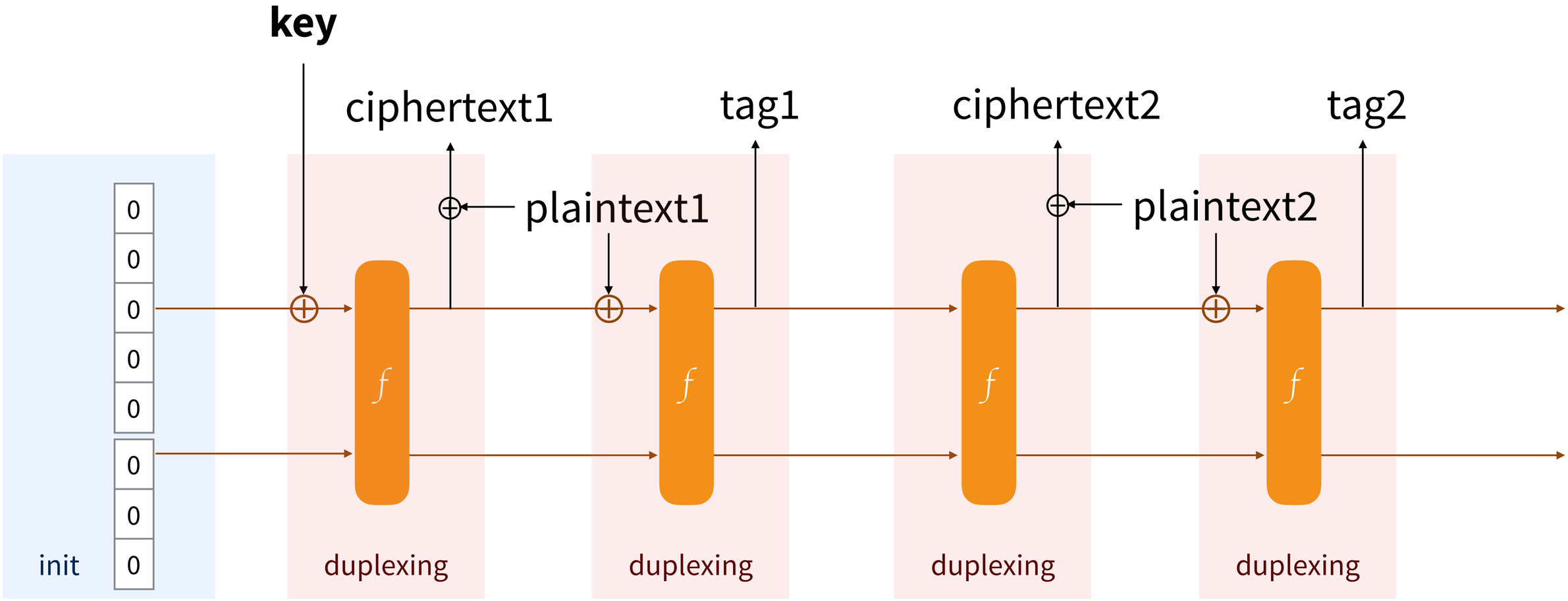
Encryption

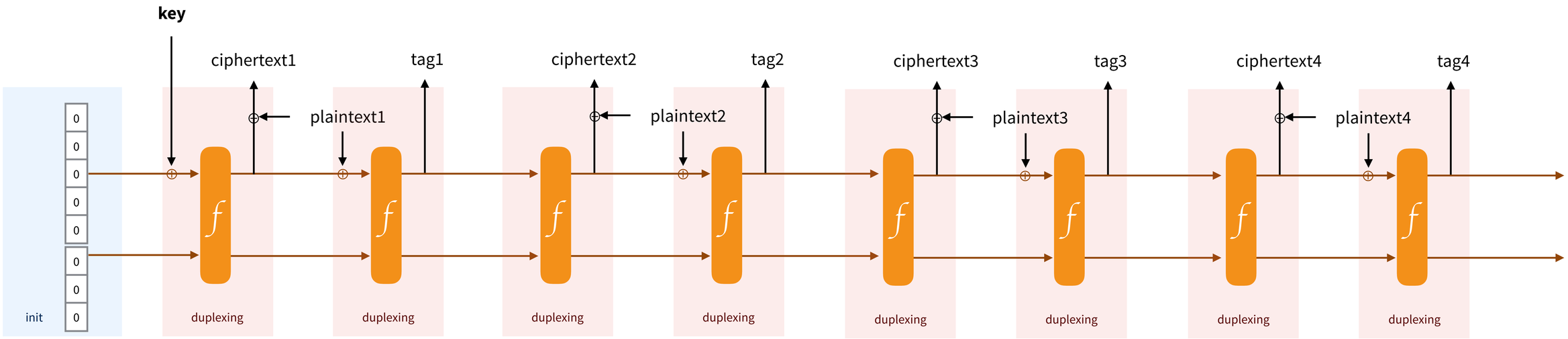


Authenticated Encryption

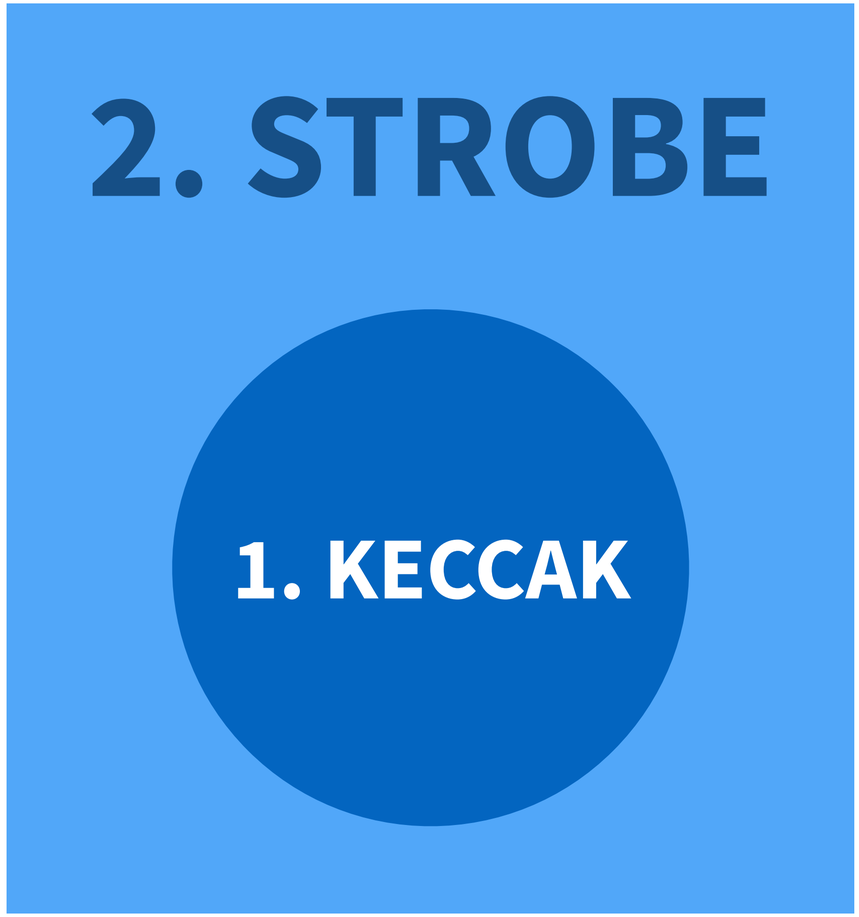


Sessions


key
ciphertext1
tag1
ciphertext2
tag2
plaintext1
plaintext2
0
0
0
0
0
0
0
0
init
f
f
f
f
duplexing
duplexing
duplexing
duplexing

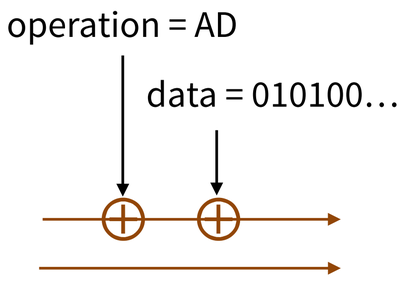

key
ciphertext1
tag1
ciphertext2
tag2
ciphertext3
tag3
ciphertext4
tag4
plaintext1
plaintext2
plaintext3
plaintext4
0
0
0
0
0
0
0
0
init
f
duplexing

outline

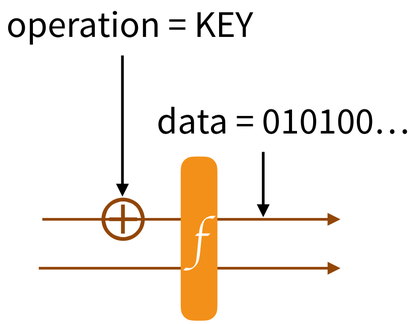


Strobe functions

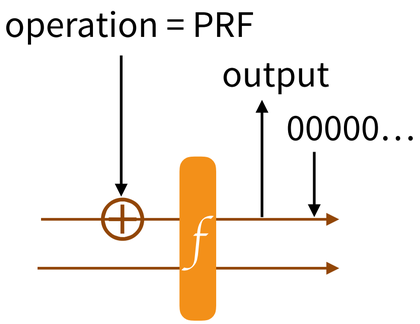
AD



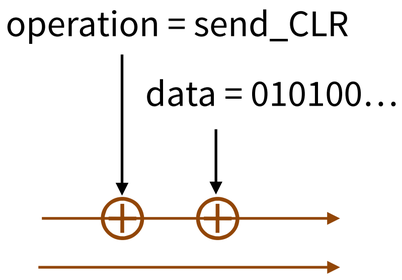
KEY



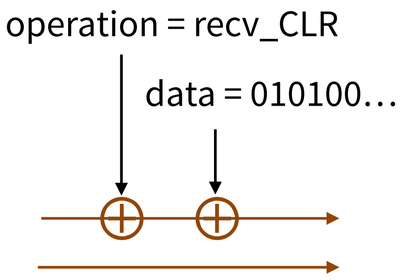
PRF



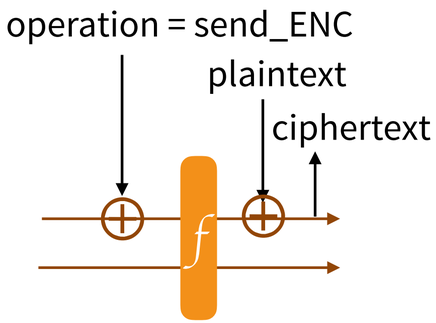
send_CLR



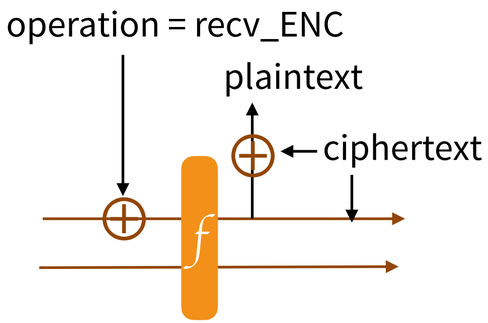
recv_CLR



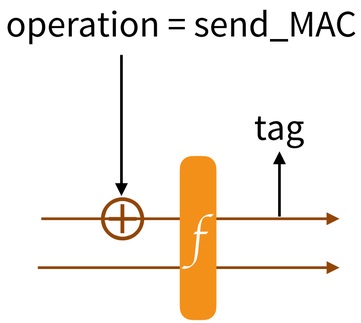
send_ENC



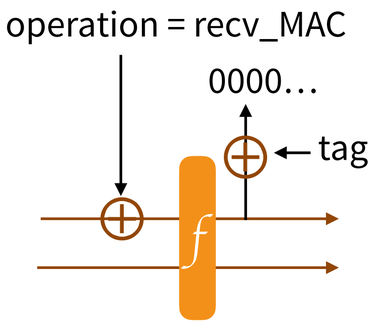
recv_ENC



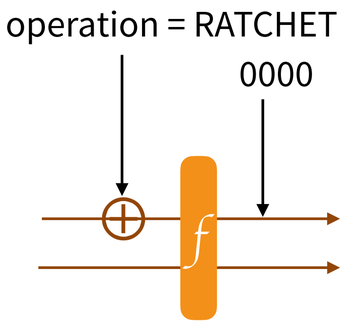
send_MAC



recv_MAC



RATCHET



Strobe protocol example

```
myProtocol = Strobe_init("myWebsite.com")
myProtocol.AD(sharedSecret)
buffer = myProtocol.send_ENC("GET /")
buffer += myProtocol.send_MAC(len=16)
// send the buffer
// receive a ciphertext
message = myProtocol.recv_ENC(ciphertext[:-16])
ok = myProtocol.recv_MAC(ciphertext[-16:])
if !ok {
 // reset the connection
}
```

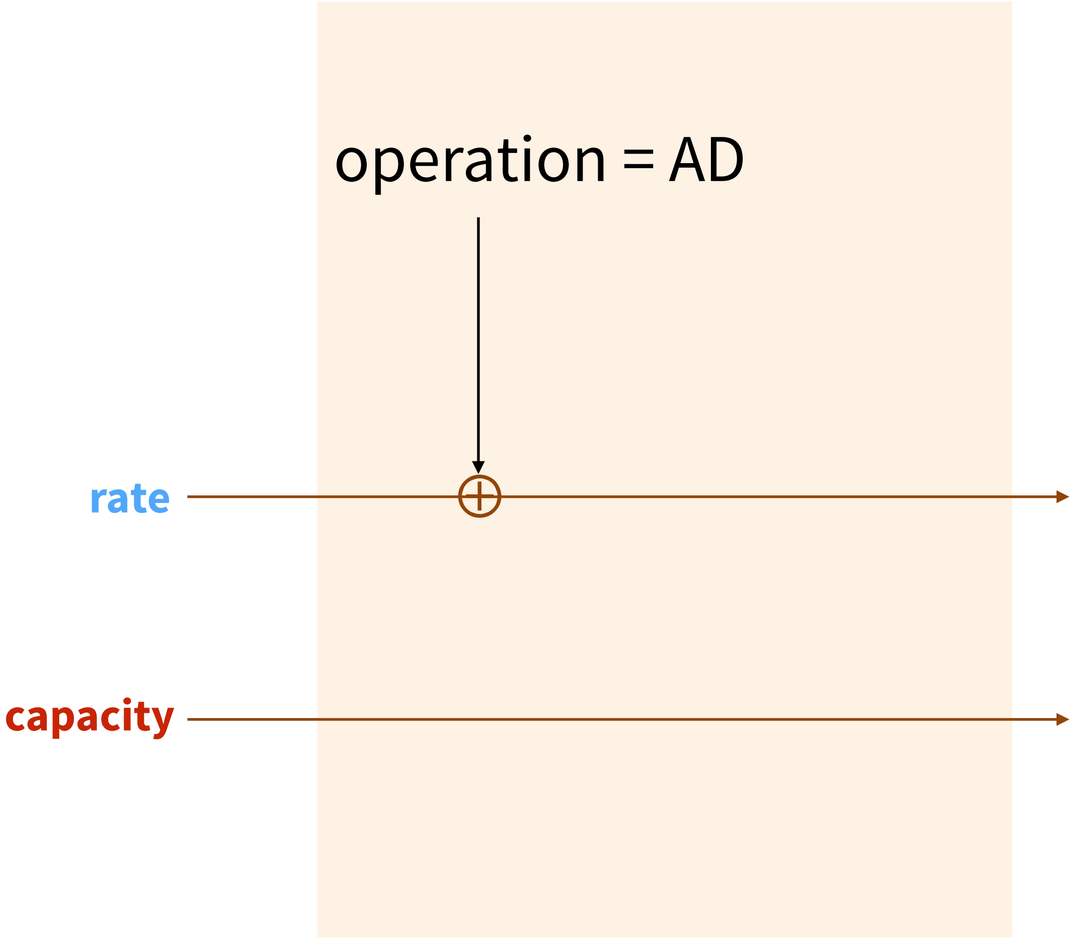
```
buffer = myProtocol.send_ENC(plaintext1)
buffer += myProtocol.send_MAC(len=16)
// send the buffer
buffer = myProtocol.send_ENC(plaintext2)
buffer += myProtocol.send_MAC(len=16)
// send the buffer
buffer = myProtocol.send_ENC(plaintext3)
buffer += myProtocol.send_MAC(len=16)
// send the buffer
buffer = myProtocol.send_ENC(plaintext4)
buffer += myProtocol.send_MAC(len=16)
// send the buffer
```

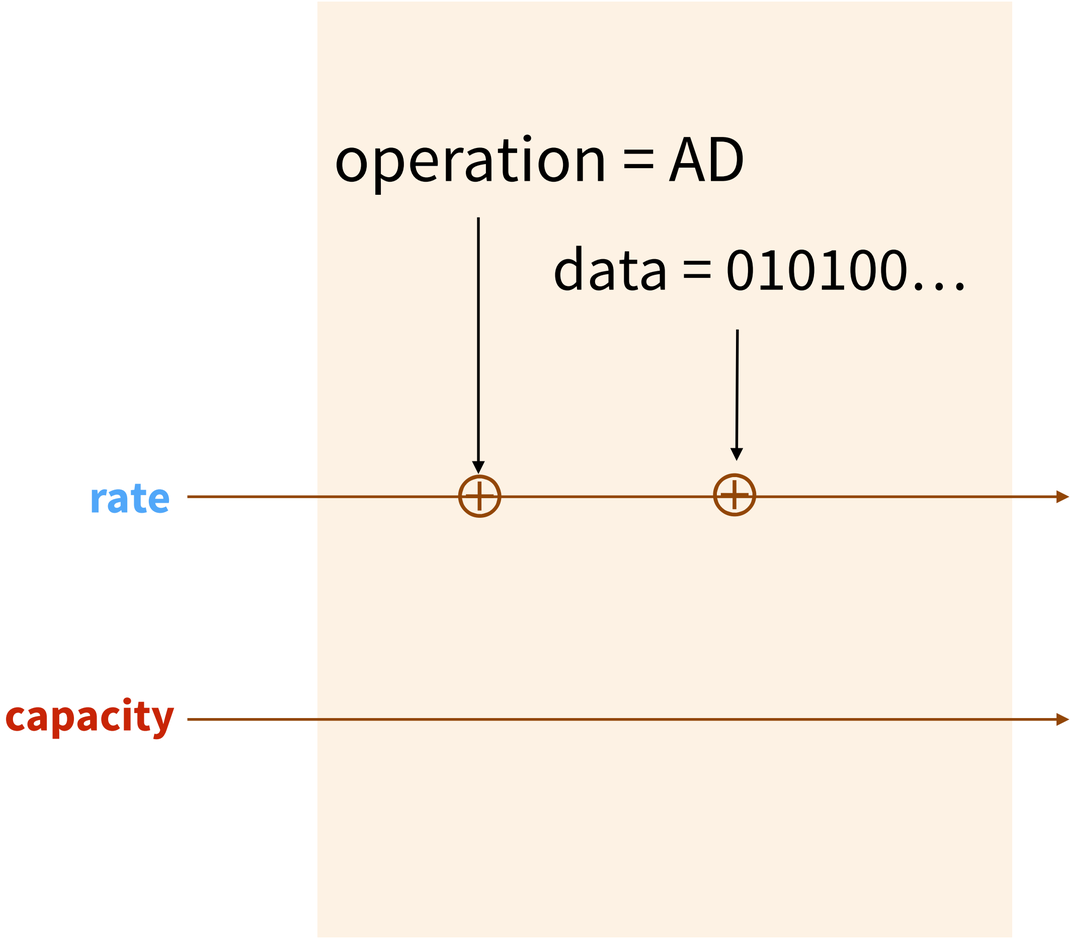
Strobe

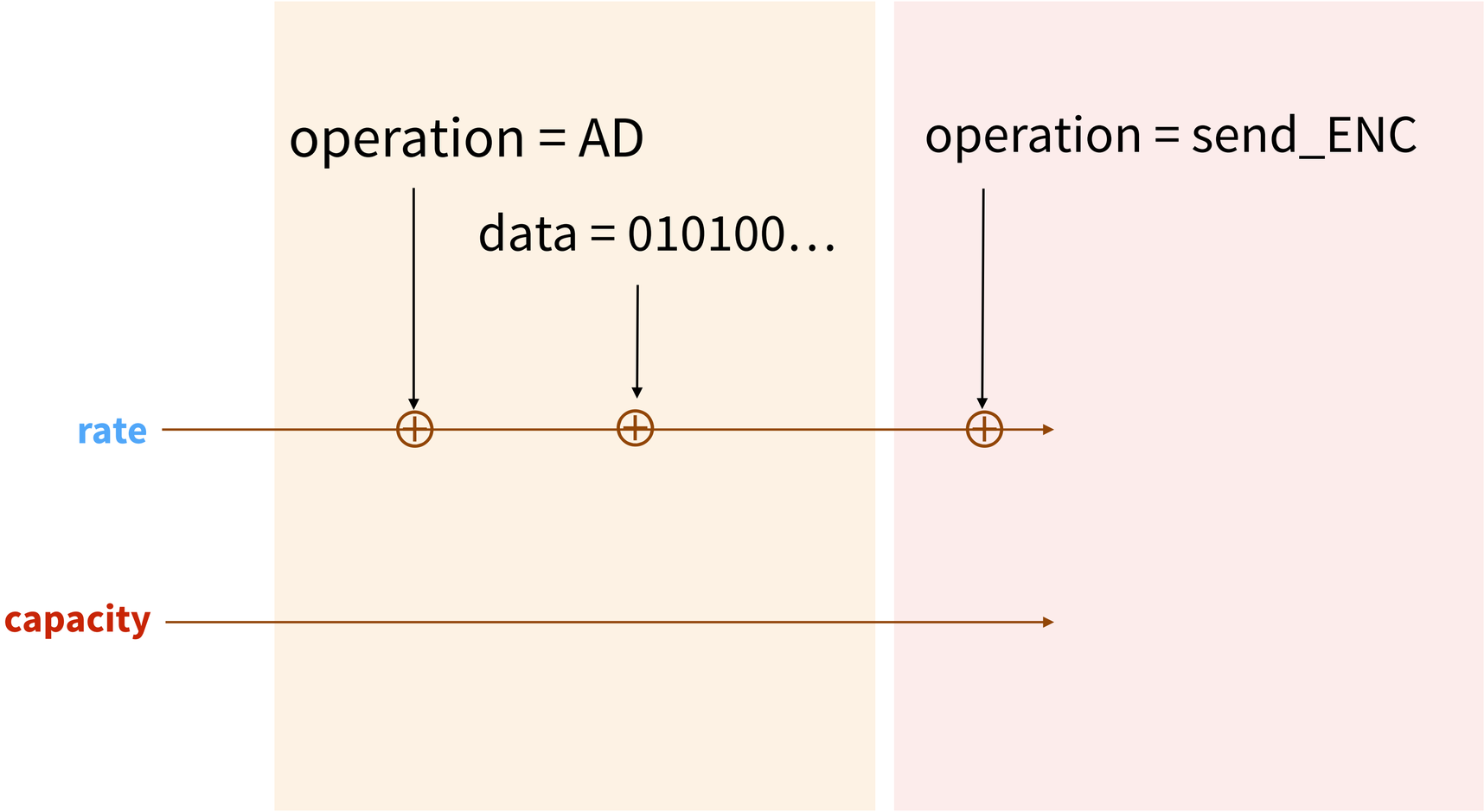
- **flexible** framework to support a large number of protocols
- large **symmetric cryptography** library

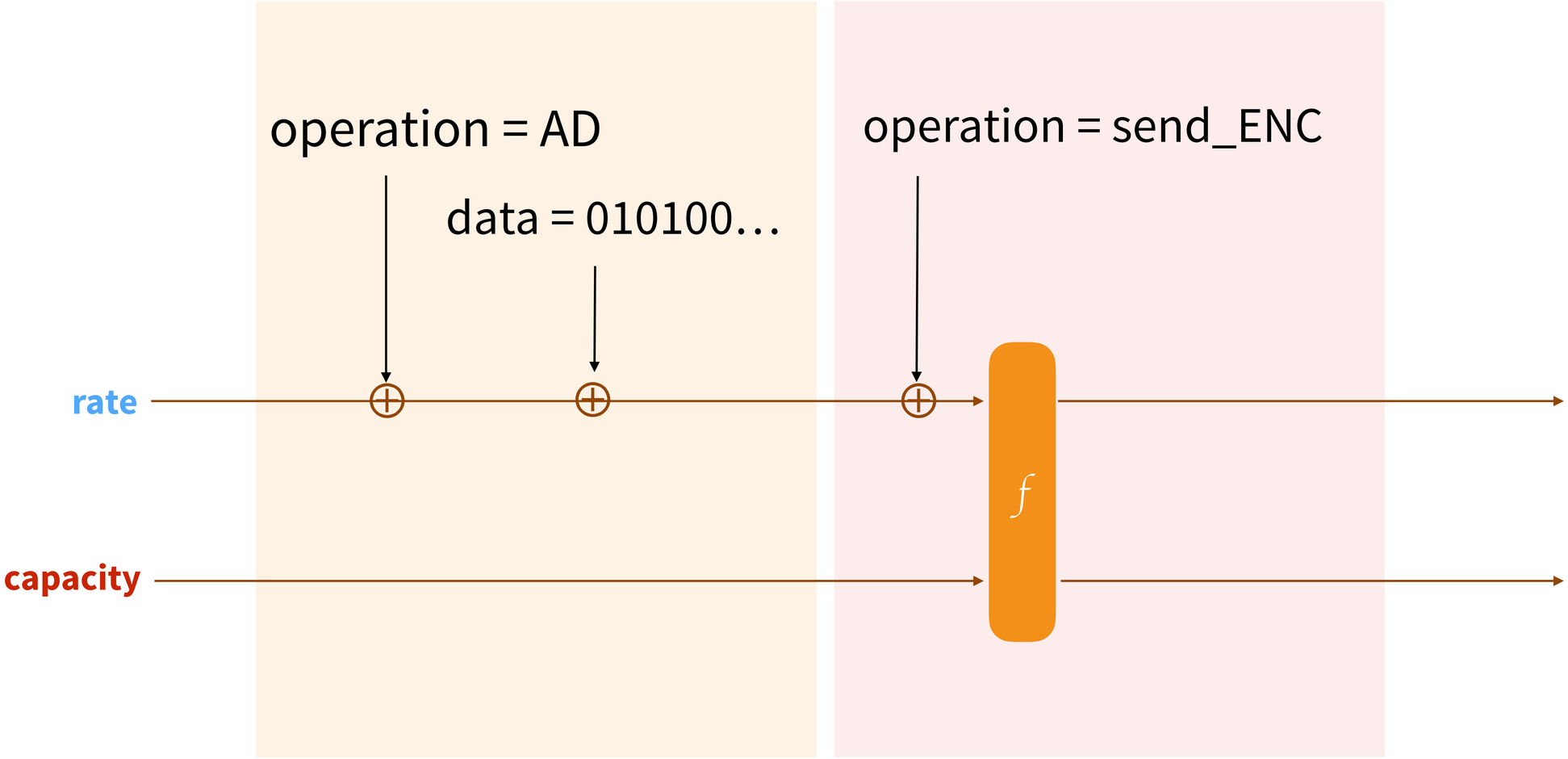
Strobe as a Hash Function

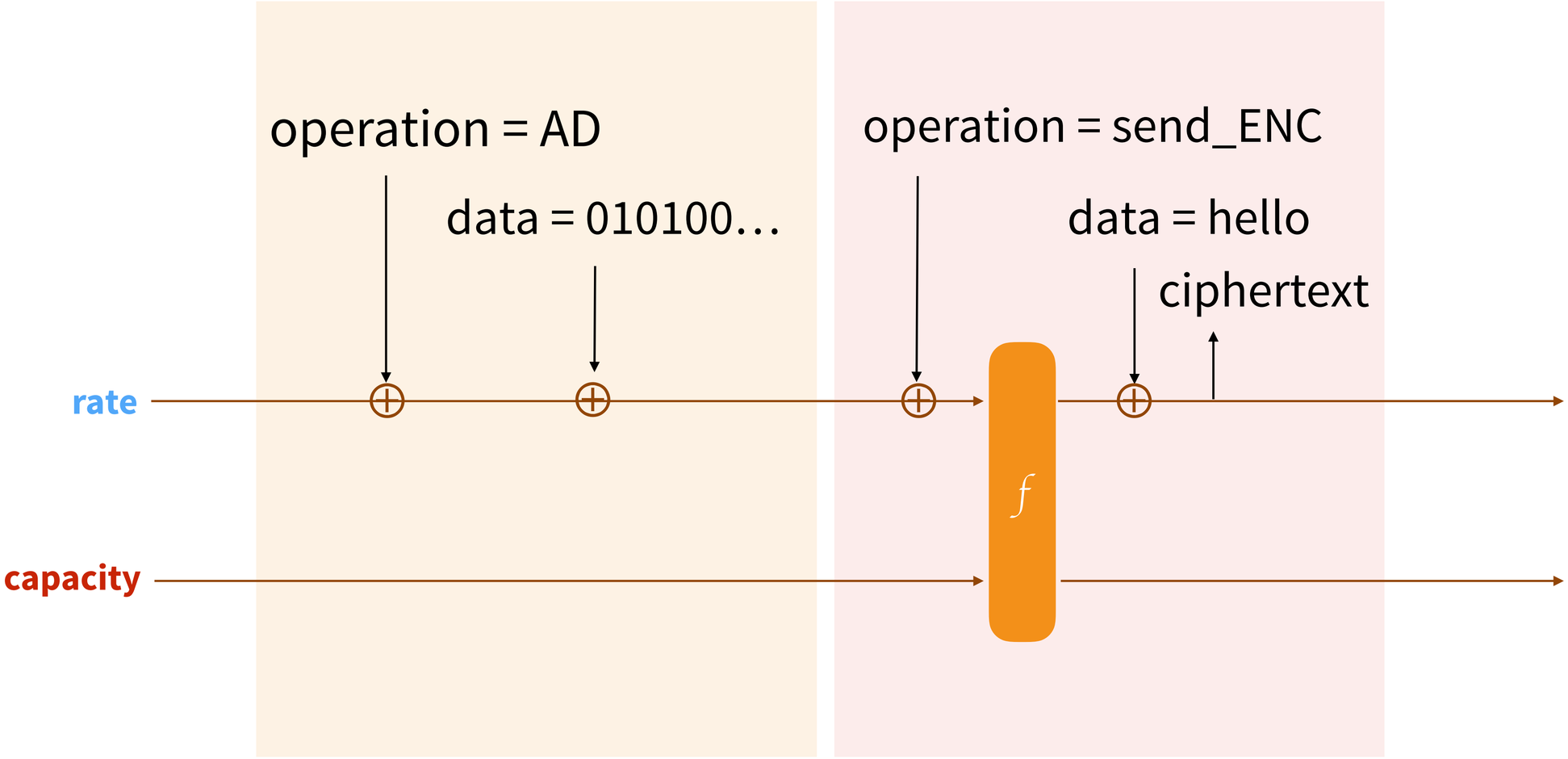
```
myHash = Strobe_init("david_wong_hash")
myHash.AD("something to be hashed")
hash = myHash.PRF(outputLen=32)
```

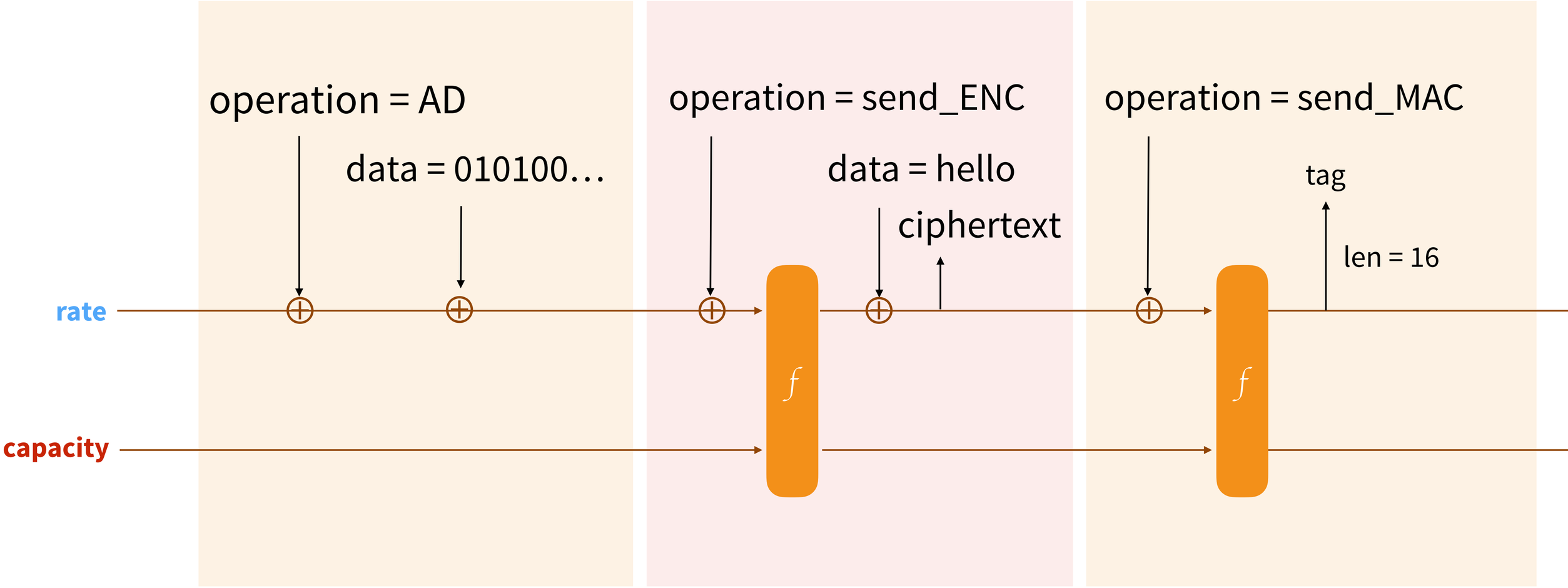

operation = AD
rate
capacity

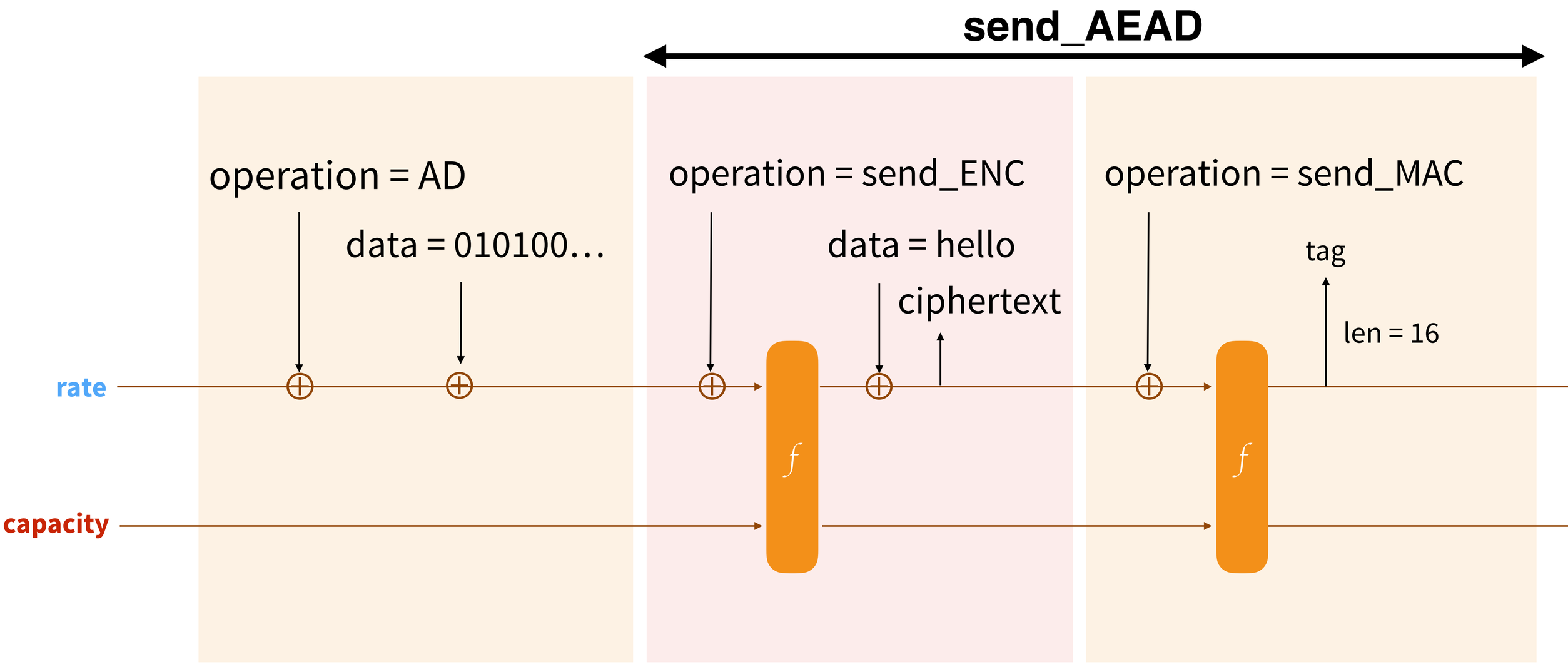

operation = AD
data = 010100…
rate
capacity


operation = AD
data = 010100…
operation = send_ENC
rate
capacity


operation = AD
data = 010100…
operation = send_ENC
rate
capacity
f


operation = AD
data = 010100…
operation = send_ENC
data = hello
ciphertext
rate
capacity
f


operation = AD
data = 010100…
operation = send_ENC
data = hello
ciphertext
operation = send_MAC
tag
len = 16
rate
capacity
f
f


send_AEAD
operation = AD
data = 010100…
operation = send_ENC
data = hello
ciphertext
operation = send_MAC
tag
len = 16
rate
capacity
f
f

Strobe

- **flexible** framework to support a large number of protocols
- large **symmetric cryptography** library
- fits into **tiny** IoT devices (**~300 lines of code)**
- relies on strong **SHA-3** standard (**SHAKE**-compliant)

strobe.sourceforge.io

Strobe protocol framework

David

Secure https://strobe.sourceforge.io/specs/

STROBE protocol framework

overview specification example protocols code papers

Scope

This spec describes the operation of the STROBE framework. It only covers the symmetric portion. For applications including elliptic curve crypto, see the examples page.

Table of Contents

outline

3. NOISE

2. STROBE

1. KECCAK

TLS

- TLS is the **de facto standard** for securing communications
- **complex** specification (TLS 1.3 is 160-page long)
  - supported by other specifications (asn.1, x509, 44 mentioned RFCs …)
- design carrying a lot of **legacy** decisions
- **cryptographic agility** and **complicated** state machine
- **huge** and **scary** libraries (OpenSSL is 700k LOC, 165 CVEs)
  - **cumbersome** configuration…
- often **dangerously** re-implemented (custom implementations)
  - or re-invented (proprietary protocols)

Complexity is the enemy of security

www.noiseprotocol.org

The Noise Protocol Framework ×

David

noiseprotocol.org/noise.html#payload-security-properties

The Noise Protocol Framework

Author: Trevor Perrin (noise@trevp.net)
Revision: 33
Date: 2017-10-04
PDF: noise.pdf

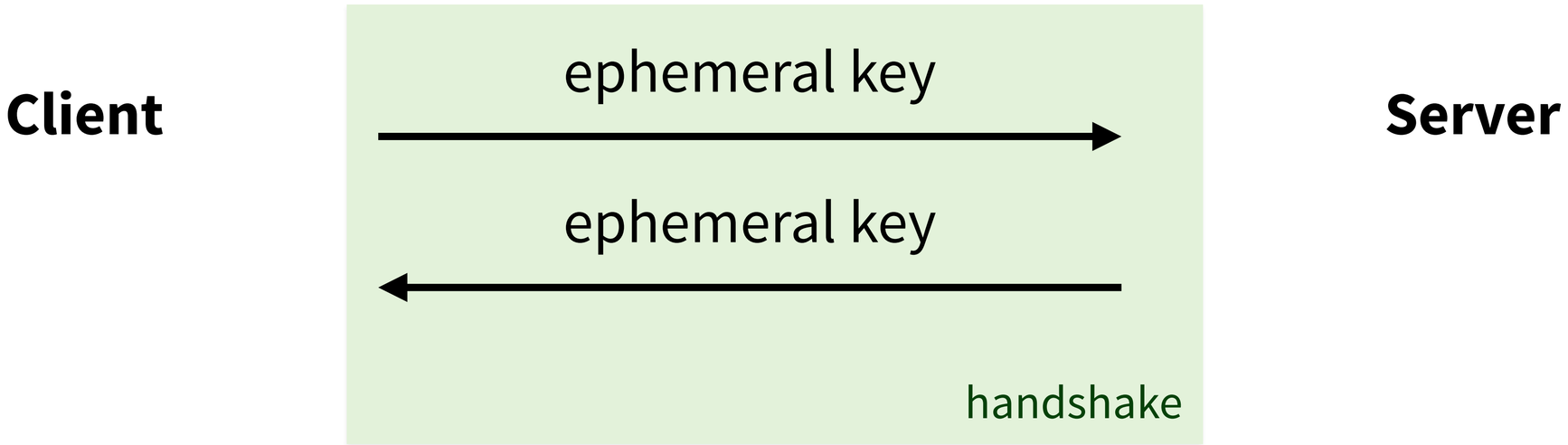
Table of Contents

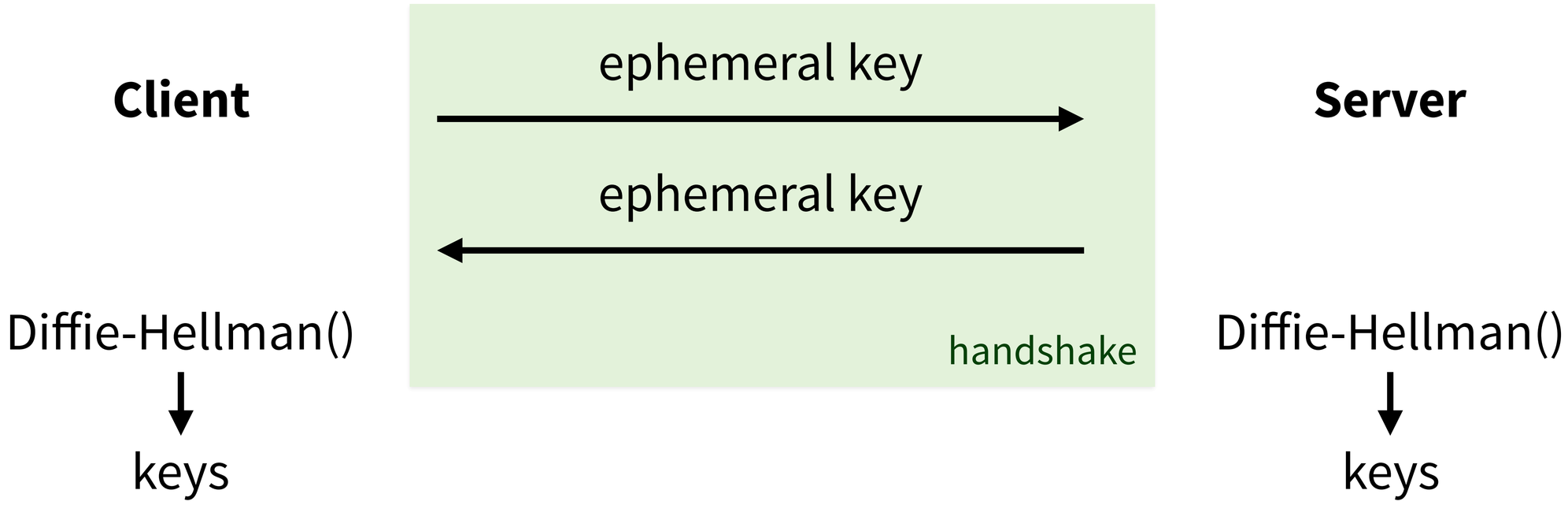
The Noise Protocol Framework

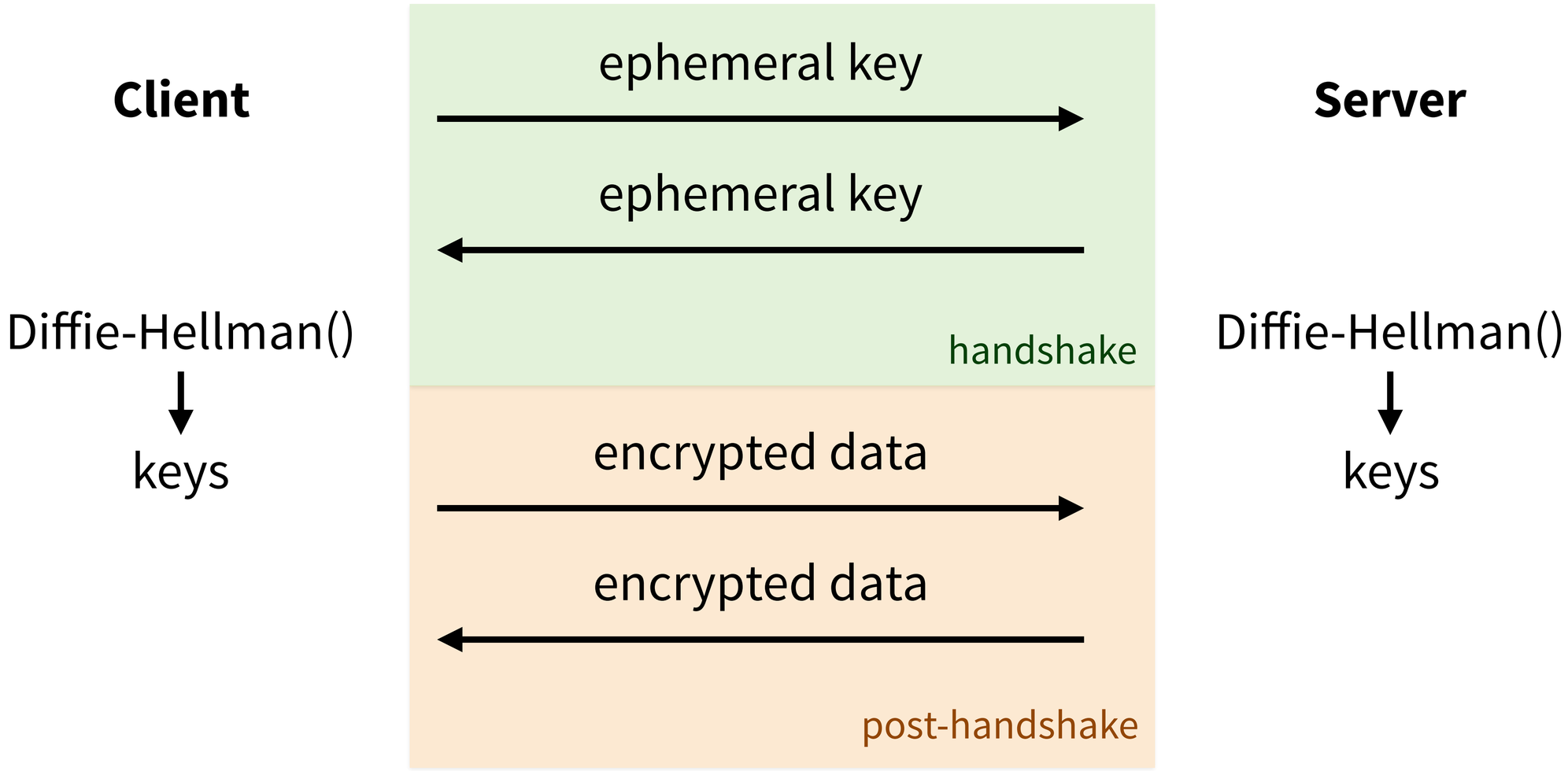
- no need for **certificates** or a **PKI**
- many handshakes to choose from (**flexible**)
- it’s **straight forward** to implement (<1k LOC, 18kb for Arduino)
- there are already **libraries** that you can leverage
- **minimal** (or zero) configuration
- used by **WhatsApp**, **Slack**, the **Bitcoin Lightning Network**, …
- if you have a good excuse not to use TLS, **Noise is the answer**

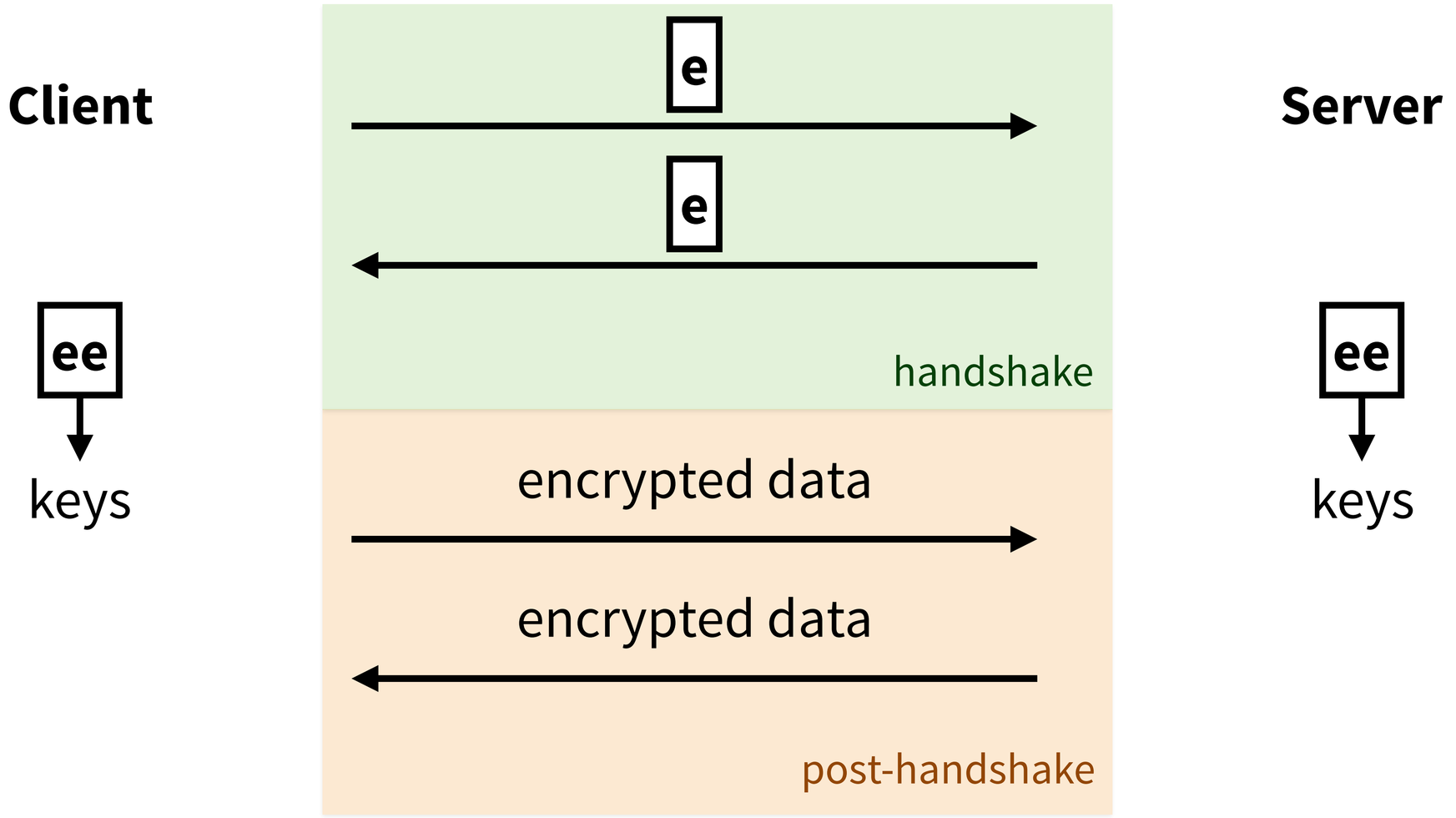
The **crypto** functions

- **DH:** X25519 or X448
- **AEAD:** Chacha20-Poly1305 or AES-GCM
- **HASH**: BLAKE2 or SHA-2


Client
ephemeral key
ephemeral key
handshake
Server


Client
ephemeral key
ephemeral key
handshake
Server
Diffie-Hellman()
keys
Diffie-Hellman()
keys


Client
Server
ephemeral key
ephemeral key
handshake
Diffie-Hellman()
Diffie-Hellman()
keys
keys
encrypted data
encrypted data
post-handshake


Client
Server
e
e
ee
ee
keys
keys
handshake
encrypted data
encrypted data
post-handshake

→ e

← e, ee

Tokens

- **e**: ephemeral key
- **s**: static key
- **ee**: **DH**(client ephemeral key, server ephemeral key)
- **es**: **DH**(client ephemeral key, server static key)
- **se**: **DH**(client static key, server ephemeral key)
- **ss**: **DH**(client static key, server static key)
- **psk**: pre-shared key

Handshake Patterns

N(rs):

← s

…

→ e, es

K(s,rs):

← s

→ s

…

→ e, es, ss

X(s,rs):

← s

…

→ e, es, s, ss

NN():

→ e

← e, ee

NK(rs):

← s

…

→ e, es

← e, ee

NX(rs):

→ e

← e, ee, s, es

XN(s):

→ e

← e, ee

→ s, se

XK(s, rs):

← s

…

→ e, es

← e, ee

→ s, se

XX(s, rs):

→ e

← e, ee, s, es

→ s, se

KN(s):

→ s

…

→ e

← e, ee, se

KK(s, rs):

← s

→ s

…

→ e, es, ss

← e, ee, se

NX(rs):

→ e

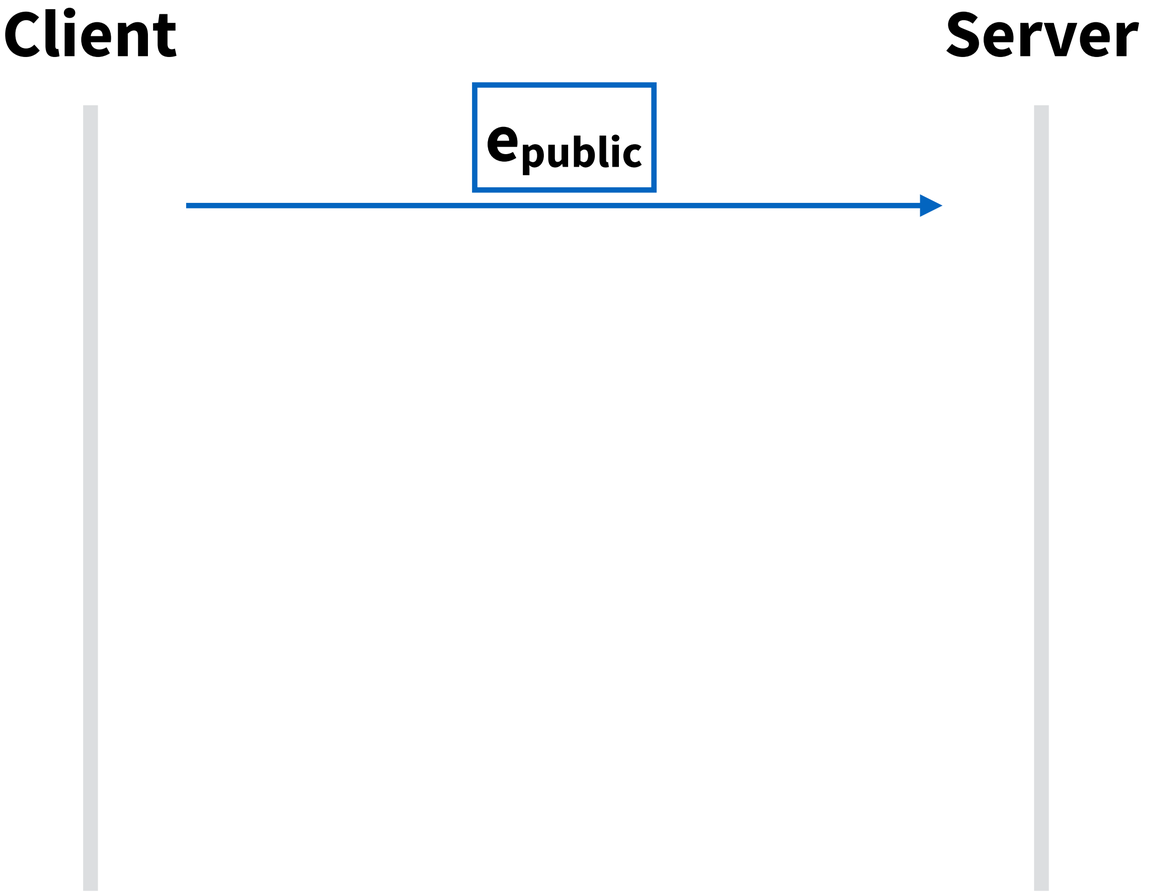
← e, ee, s, es

Client **Server**

NX(rs):

→ e

← e, ee, s, es



NX(rs):

→ e

← e, ee, s, es

Client **Server**

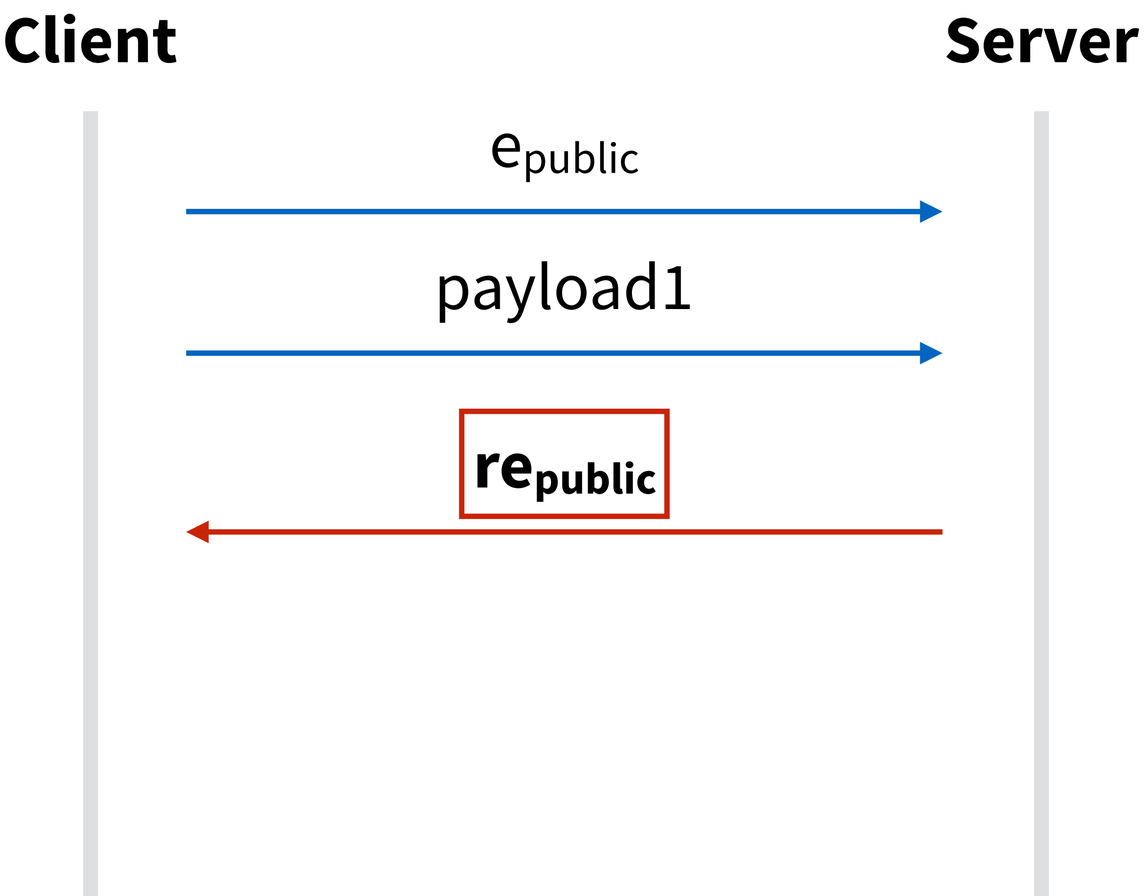
e_{public}

payload1

NX(rs):

→ e

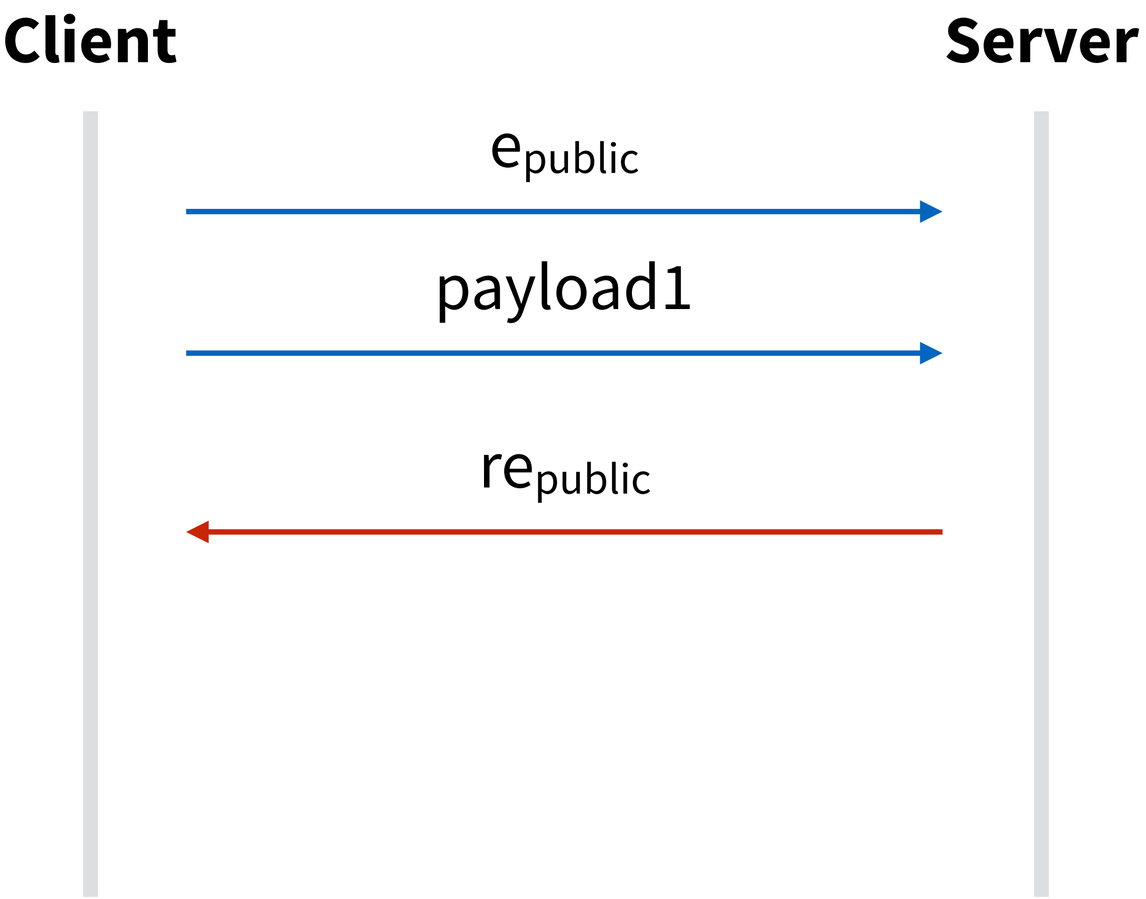
← e, ee, s, es



NX(rs):

→ e

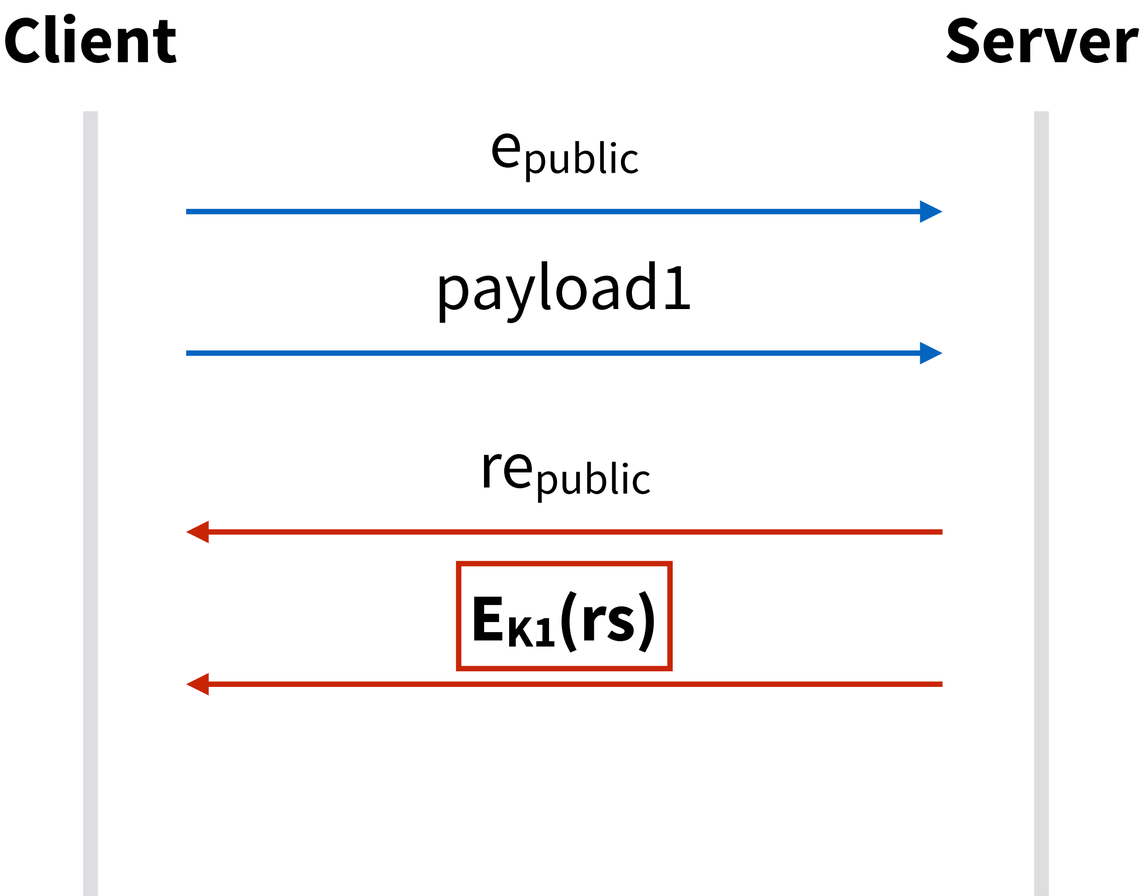
← e, ee, s, es



NX(rs):

→ e

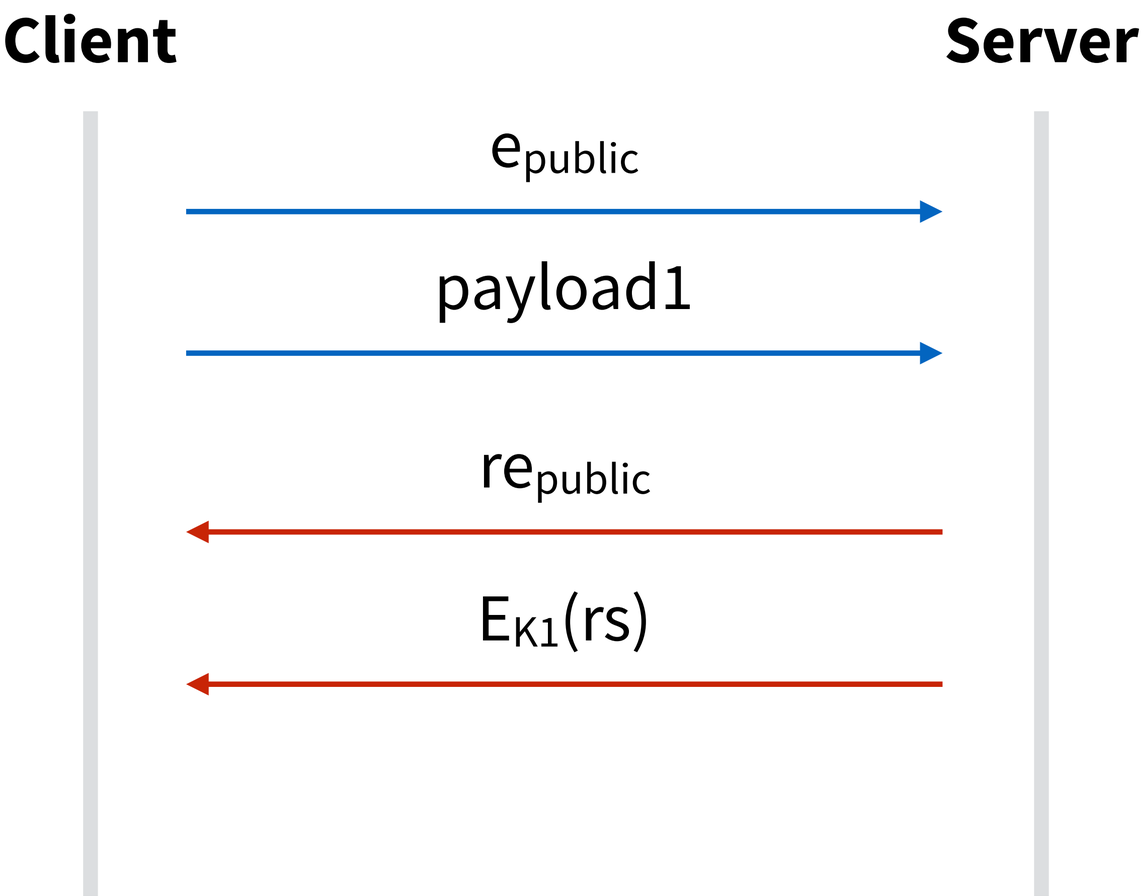
← e, ee, s, es



NX(rs):

→ e

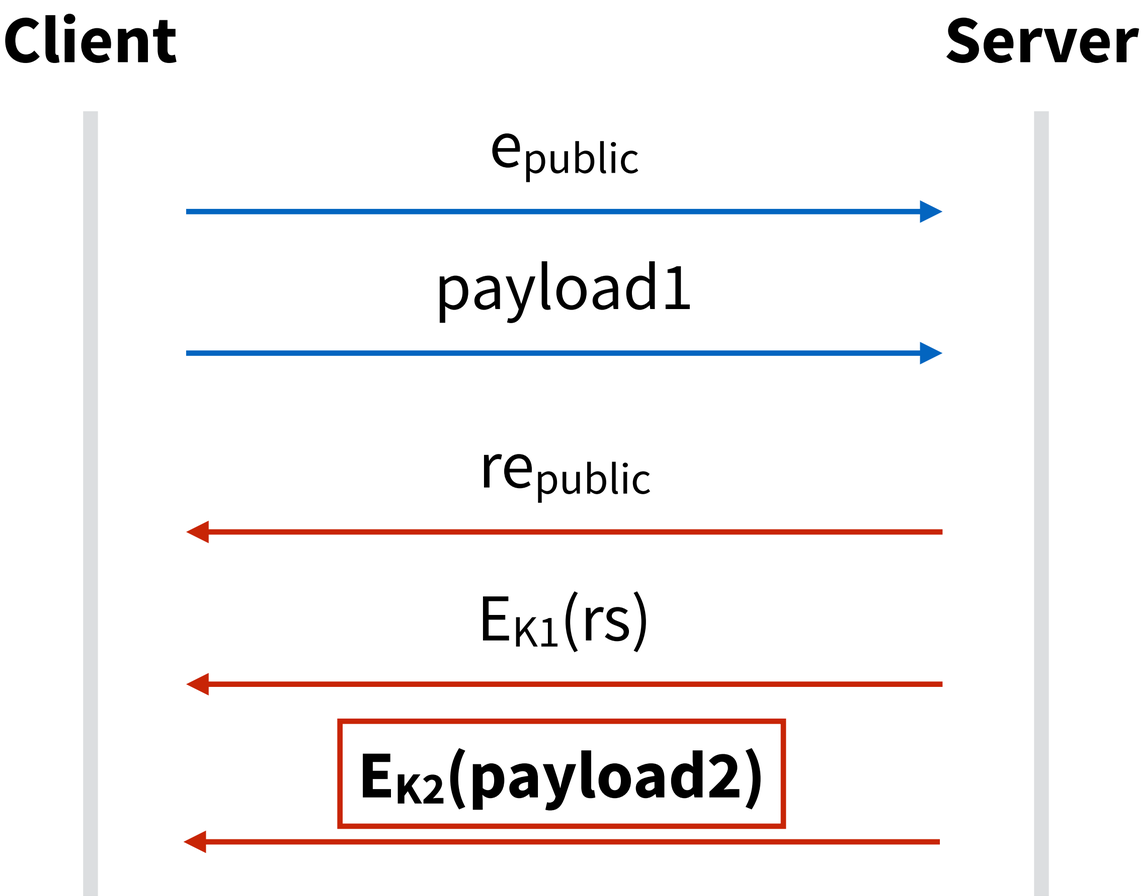
← e, ee, s, es

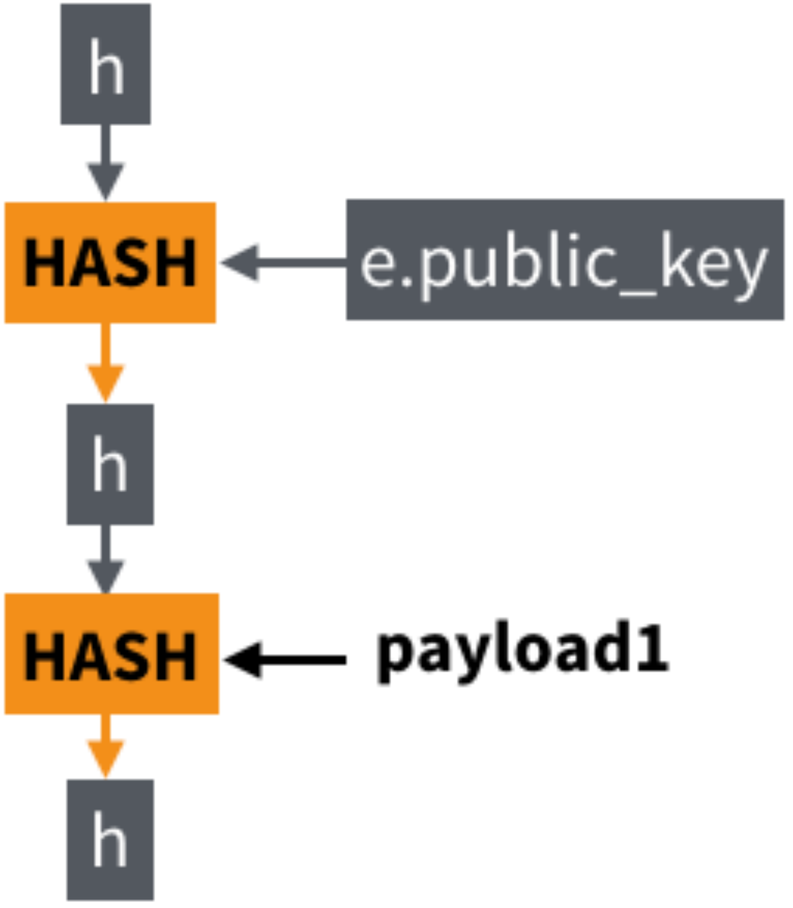


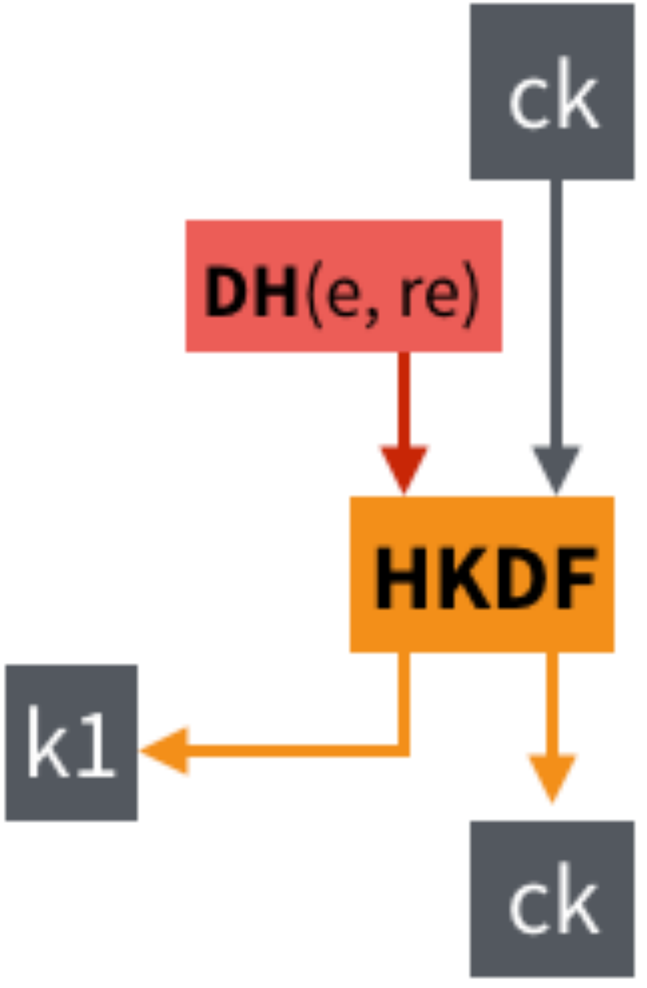
NX(rs):

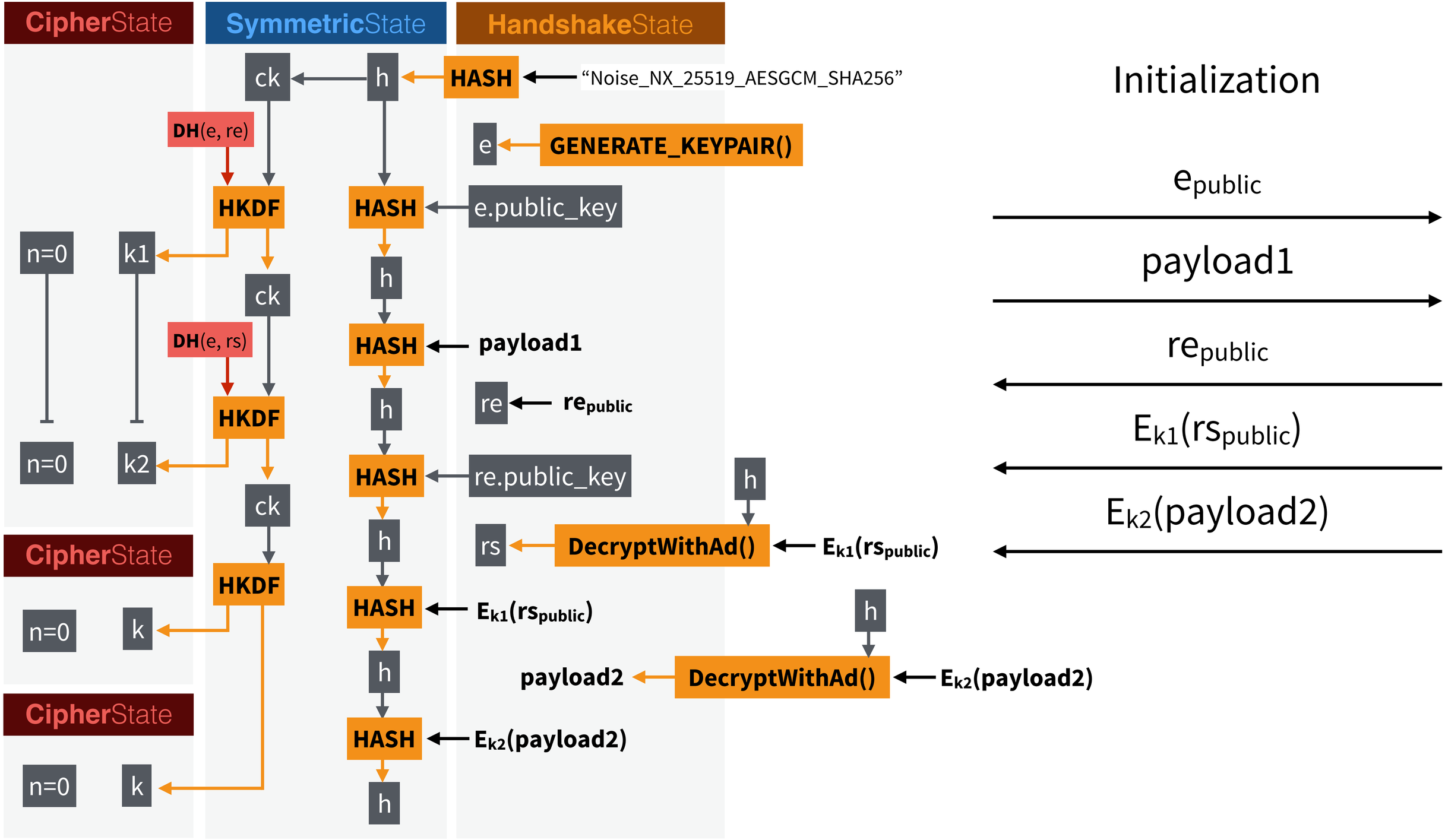
→ e

← e, ee, s, es

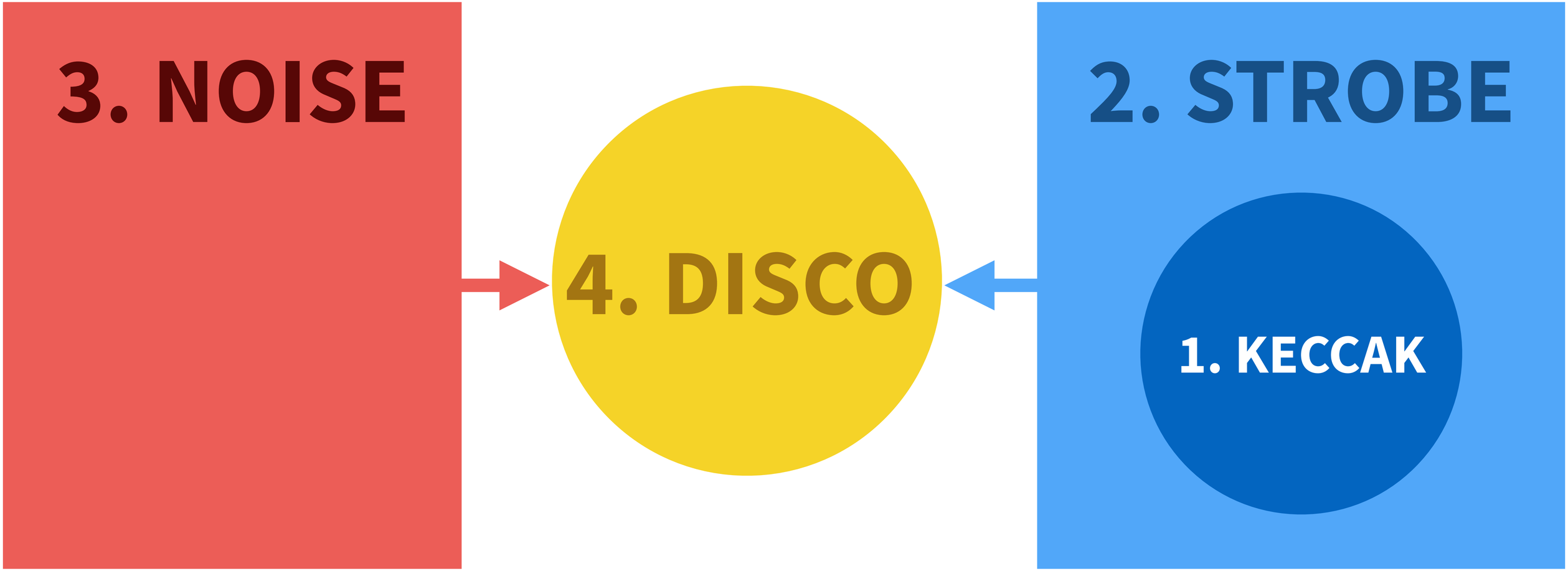


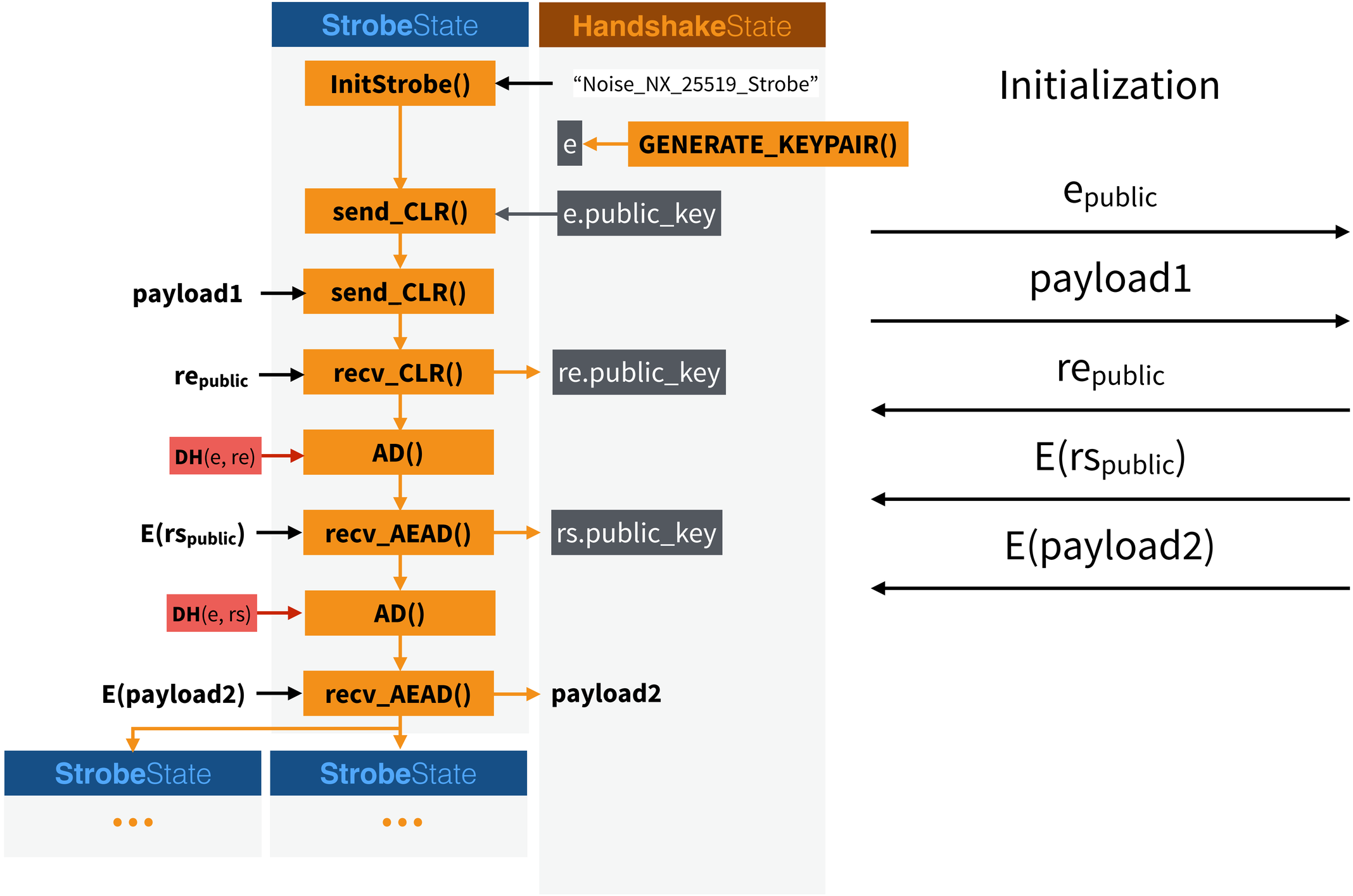

h
HASH
e.public_key
h
HASH
payload1
h

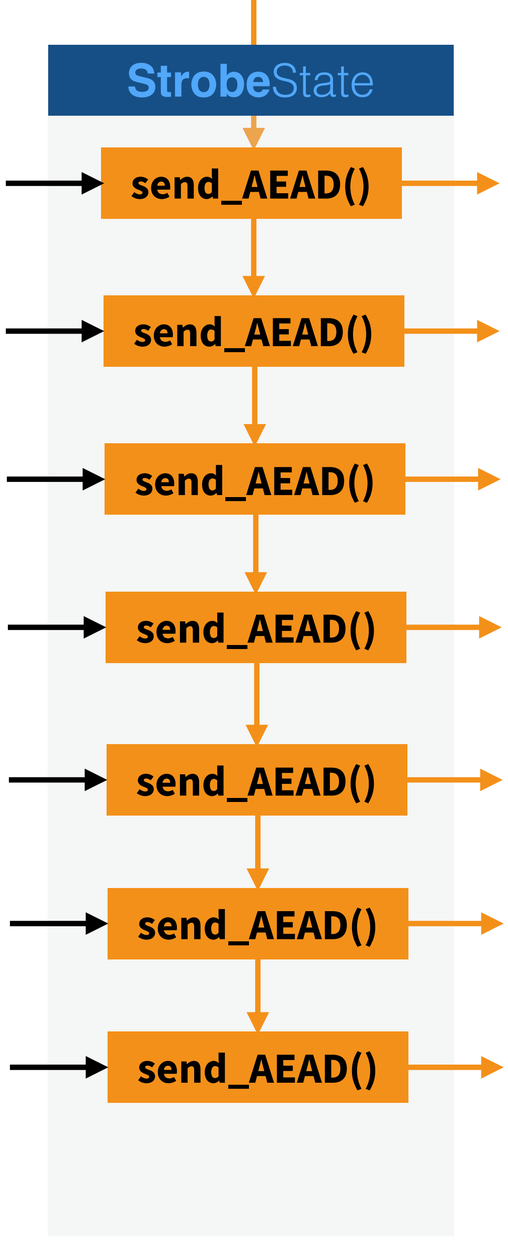

ck
DH(e, re)
HKDF
k1
ck

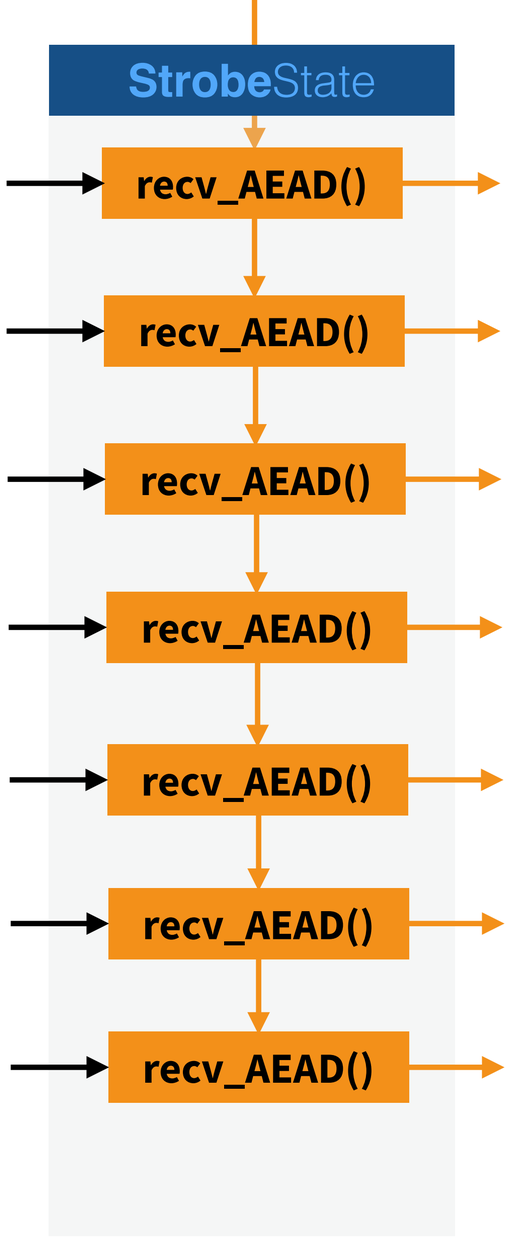

CipherState
SymmetricState
HandshakeState
ck
h
HASH
"Noise_NX_25519_AESGCM_SHA256"
Initialization
DH(e, re)
e
GENERATE_KEYPAIR()
HKDF
HASH
e.public_key
e_{public}
n=0
k1
payload1
ck
h
DH(e, rs)
HASH
payload1
re_{public}
HKDF
h
re
re_{public}
$E_{k1}(rs_{public})$
n=0
k2
HASH
re.public_key
h
$E_{k2}(payload2)$
ck
h
rs
DecryptWithAd()
$E_{k1}(rs_{public})$
CipherState
HKDF
HASH
$E_{k1}(rs_{public})$
h
n=0
k
h
payload2
DecryptWithAd()
$E_{k2}(payload2)$
CipherState
HASH
$E_{k2}(payload2)$
n=0
k
h

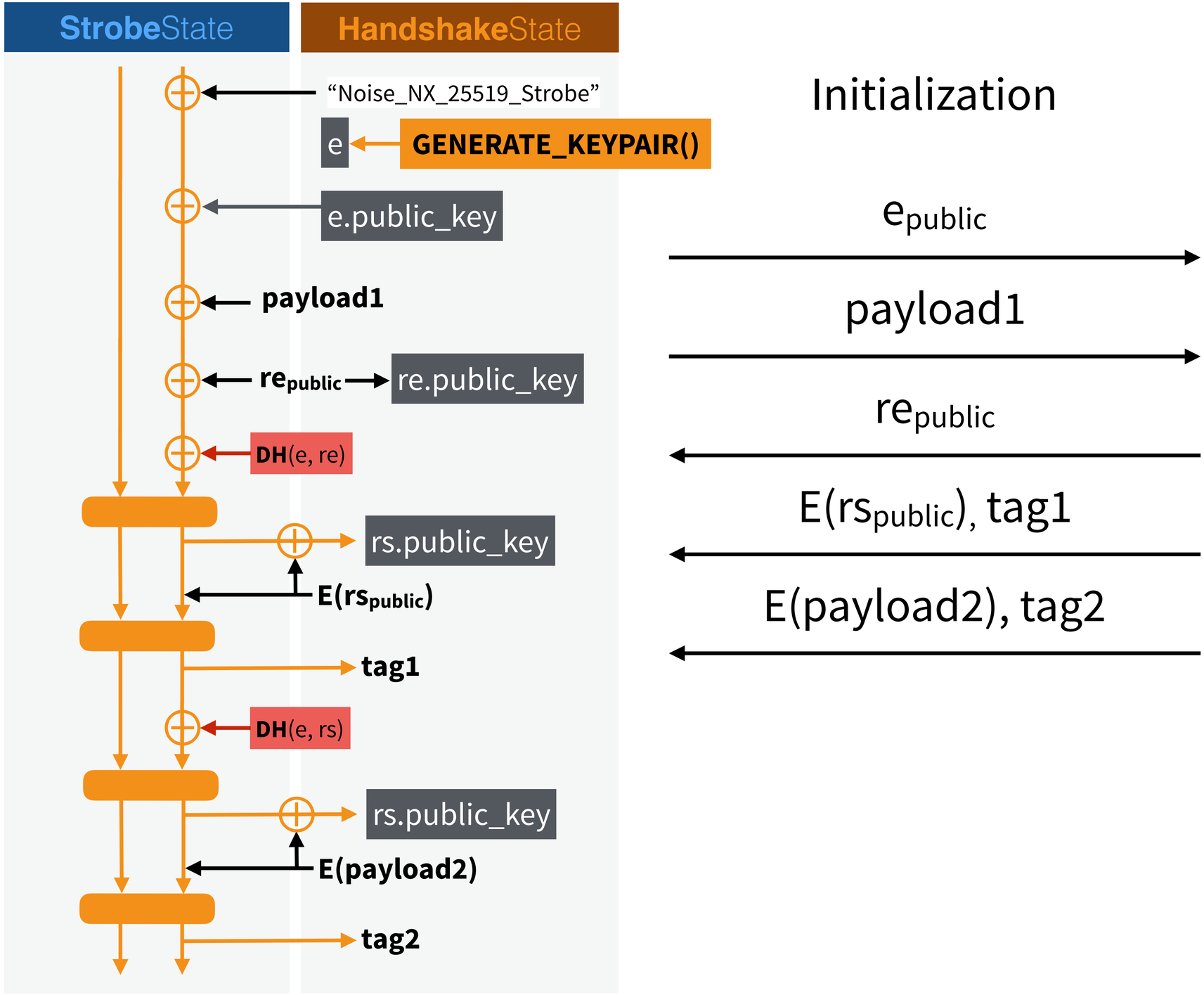
outline




StrobeState
HandshakeState
InitStrobe()
"Noise_NX_25519_Strobe"
e
GENERATE_KEYPAIR()
send_CLR()
e.public_key
payload1
send_CLR()
republic
recv_CLR()
re.public_key
DH(e, re)
AD()
E(rspublic)
recv_AEAD()
rs.public_key
DH(e, rs)
AD()
E(payload2)
recv_AEAD()
payload2
StrobeState
StrobeState
Initialization
epublic
payload1
republic
E(rspublic)
E(payload2)


StrobeState
send_AEAD()
send_AEAD()
send_AEAD()
send_AEAD()
send_AEAD()
send_AEAD()
send_AEAD()


StrobeState
recv_AEAD()
recv_AEAD()
recv_AEAD()
recv_AEAD()
recv_AEAD()
recv_AEAD()
recv_AEAD()


StrobeState
HandshakeState
"Noise_NX_25519_Strobe"
e
GENERATE_KEYPAIR()
e.public_key
payload1
re_public
re.public_key
DH(e, re)
rs.public_key
E(rs_public)
tag1
DH(e, rs)
rs.public_key
E(payload2)
tag2
Initialization
e_public
payload1
re_public
E(rs_public), tag1
E(payload2), tag2

www.discocrypto.com

Disco!

David

discocrypto.com/#/

libdisco

Home | Get Started | Protocol | Cryptographic Library | Source Code | Contact

GENERAL

- Get Started

PROTOCOL

- Overview
- Keys in Disco
  - Noise_K
  - Noise_N
  - Noise_X
  - Noise_NNpsk2
  - Noise_KK
  - Noise_NK
  - Noise_NX
  - Noise_XX
- Specification

CRYPTOGRAPHIC LIBRARY

- Overview

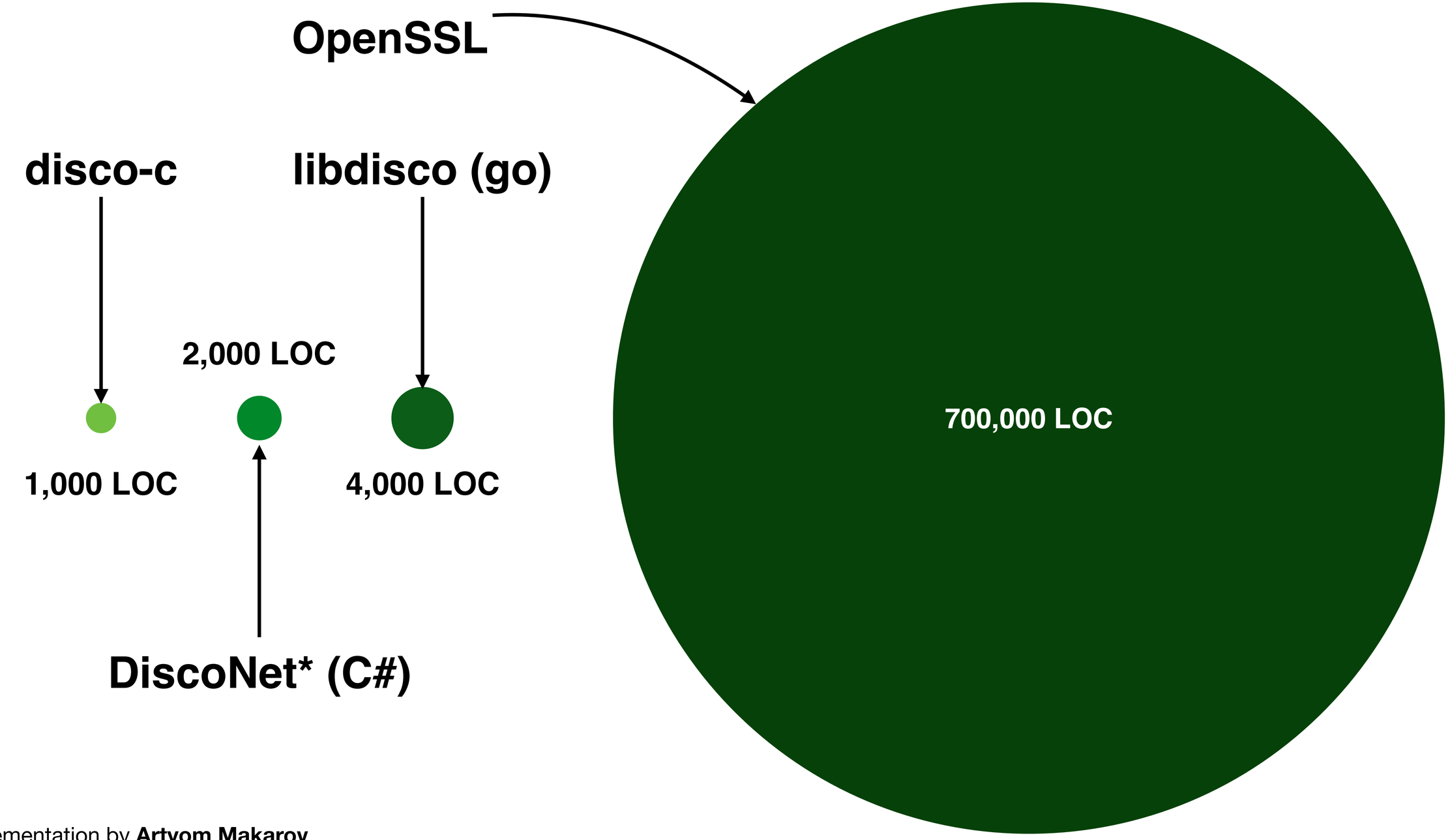
libdisco

libdisco is a **modern plug-and-play secure protocol** and a **cryptographic library** implemented in **1000 lines of code** in **Golang**. It offers different ways of securely connecting peers together, as well as different cryptographic primitives for all of an application's needs.

Warning

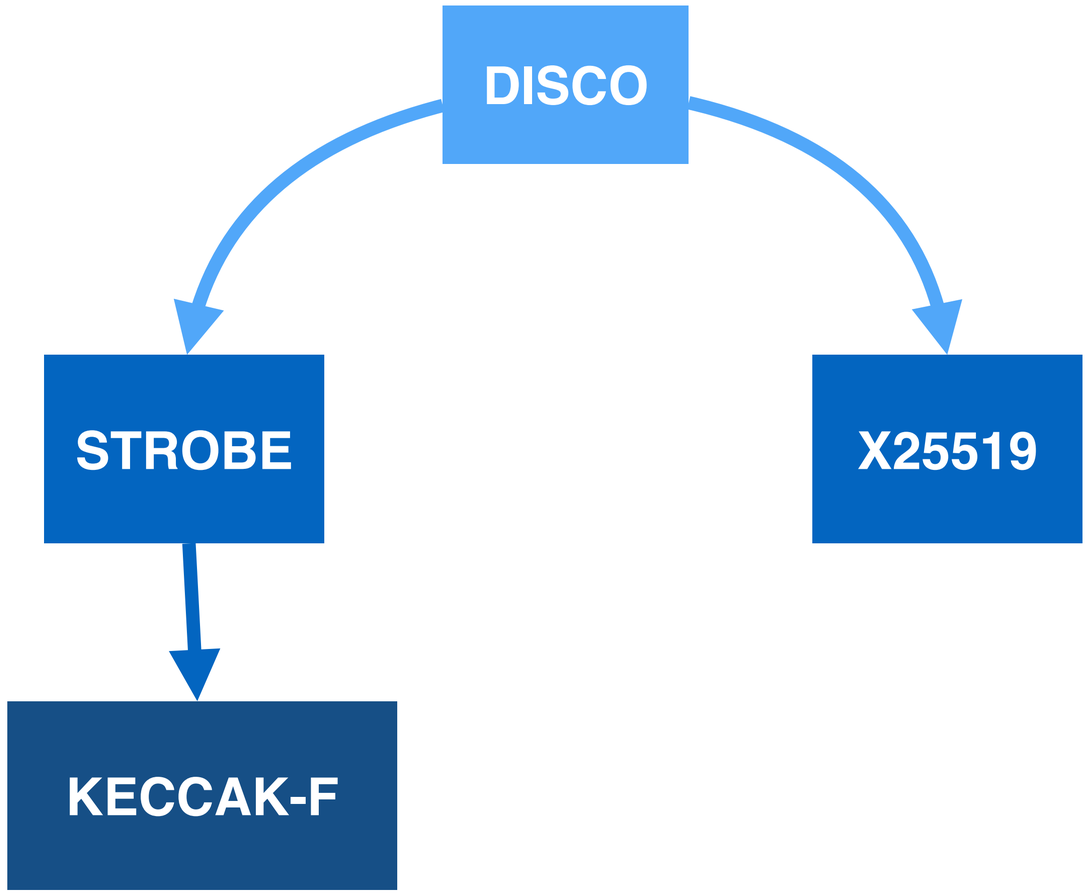
libdisco is **experimental**. It still has not been thoroughly reviewed and thus should not be used in production.

libdisco is a library built by merging the Noise protocol framework and the Strobe protocol framework. This means that it supports a subset of Noise's handshakes while offering the cryptographic primitive Strobe has to offer. In other words, you can use libdisco to securely connect peers together, or to do basic cryptographic operations like hashing or encrypting

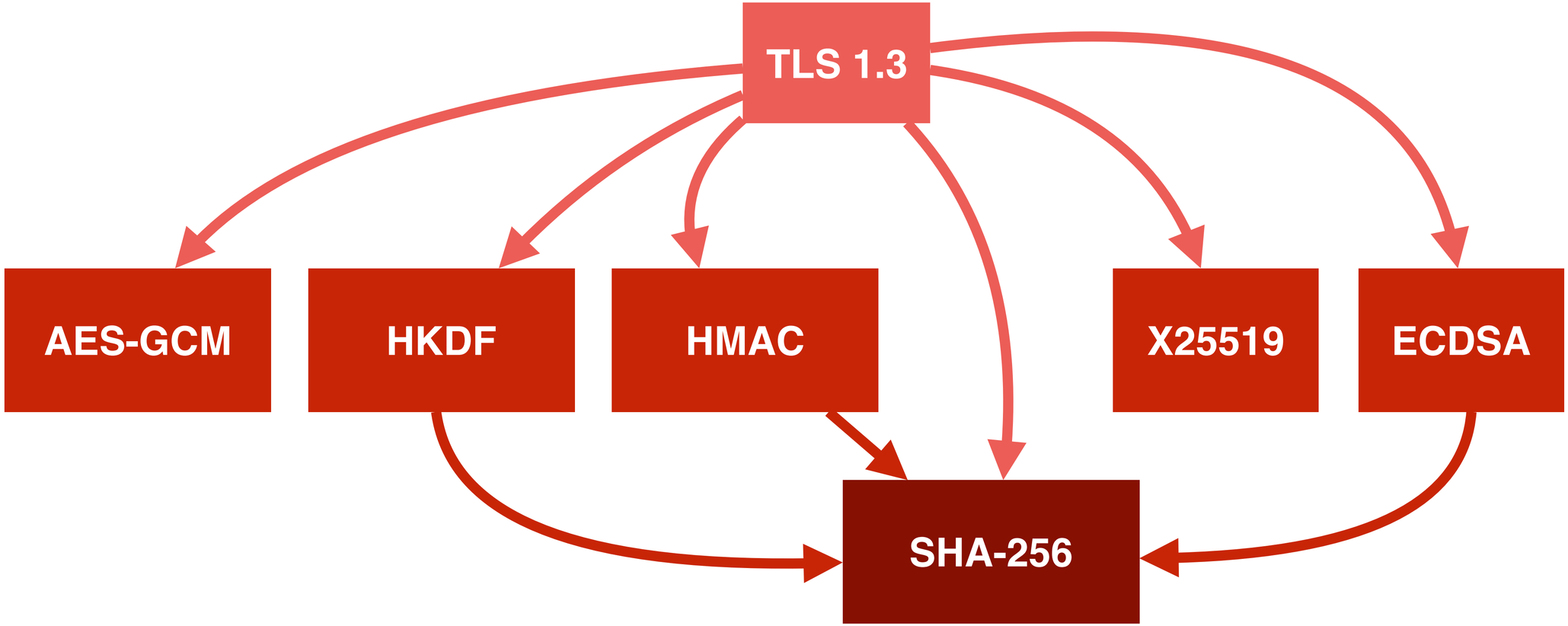


* implementation by **Artyom Makarov**

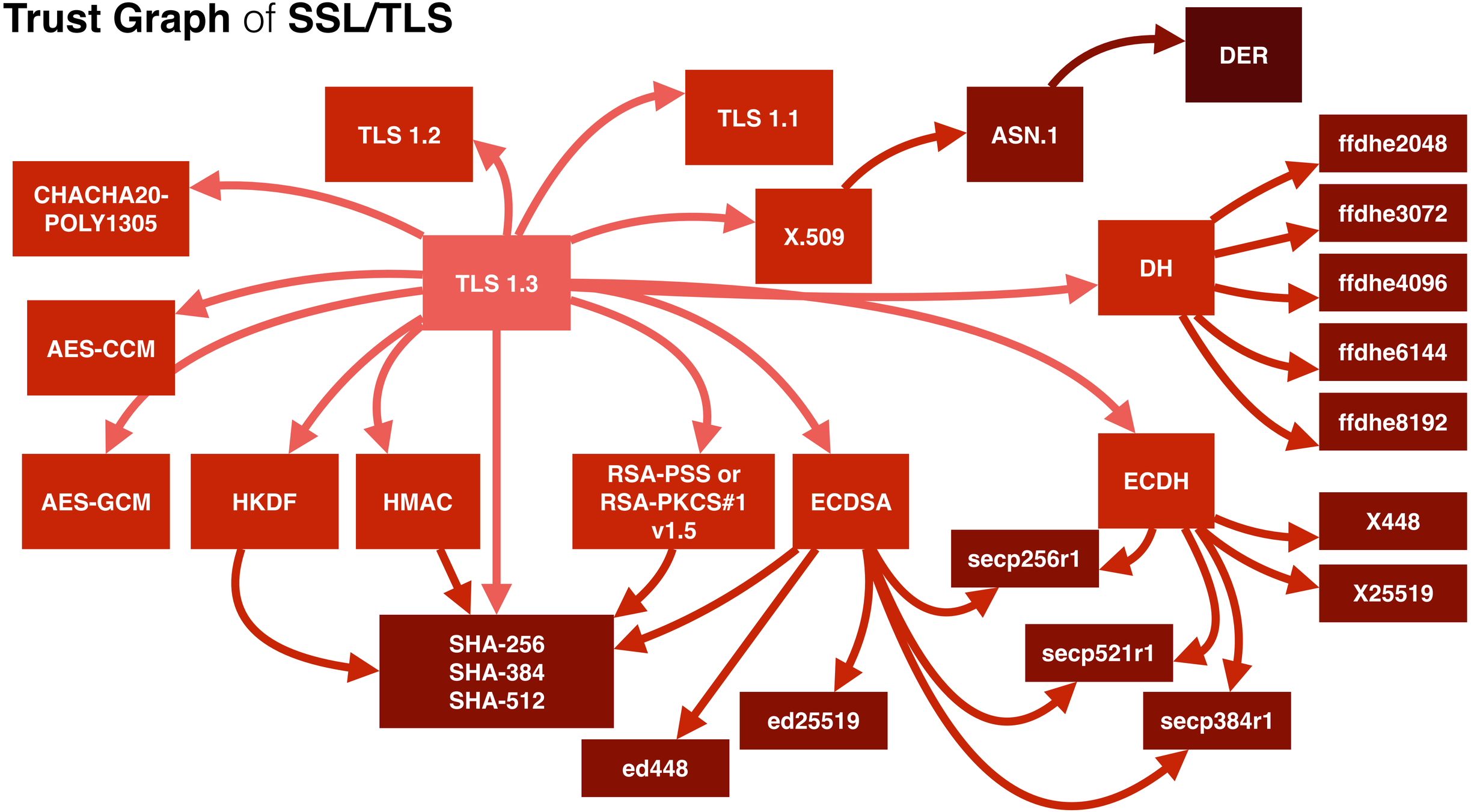
Trust Graph of **Disco**



Trust Graph of **biased SSL/TLS**



Trust Graph of SSL/TLS



The state of **Disco**

- **Disco** is a draft specification extending Noise (**experimental**)
- **Noise** is a stable draft (rev34)
- **Strobe** is alpha (v1.0.2)
- ⚠️ Disco and the implementations are still **experimental**
  - need more eyes, more interoperability testing, etc.
  - looking to formally prove handshakes with Tamarin

the **disco** is at
www.discocrypto.com

I **write** about crypto
www.cryptologie.net

follow me on
twitter.com/cryptodavidw

(and I **work** here)

nccgroup