Fed Up Getting Shattered and Log Jammed?

A New Generation of Crypto is Coming

David Wong

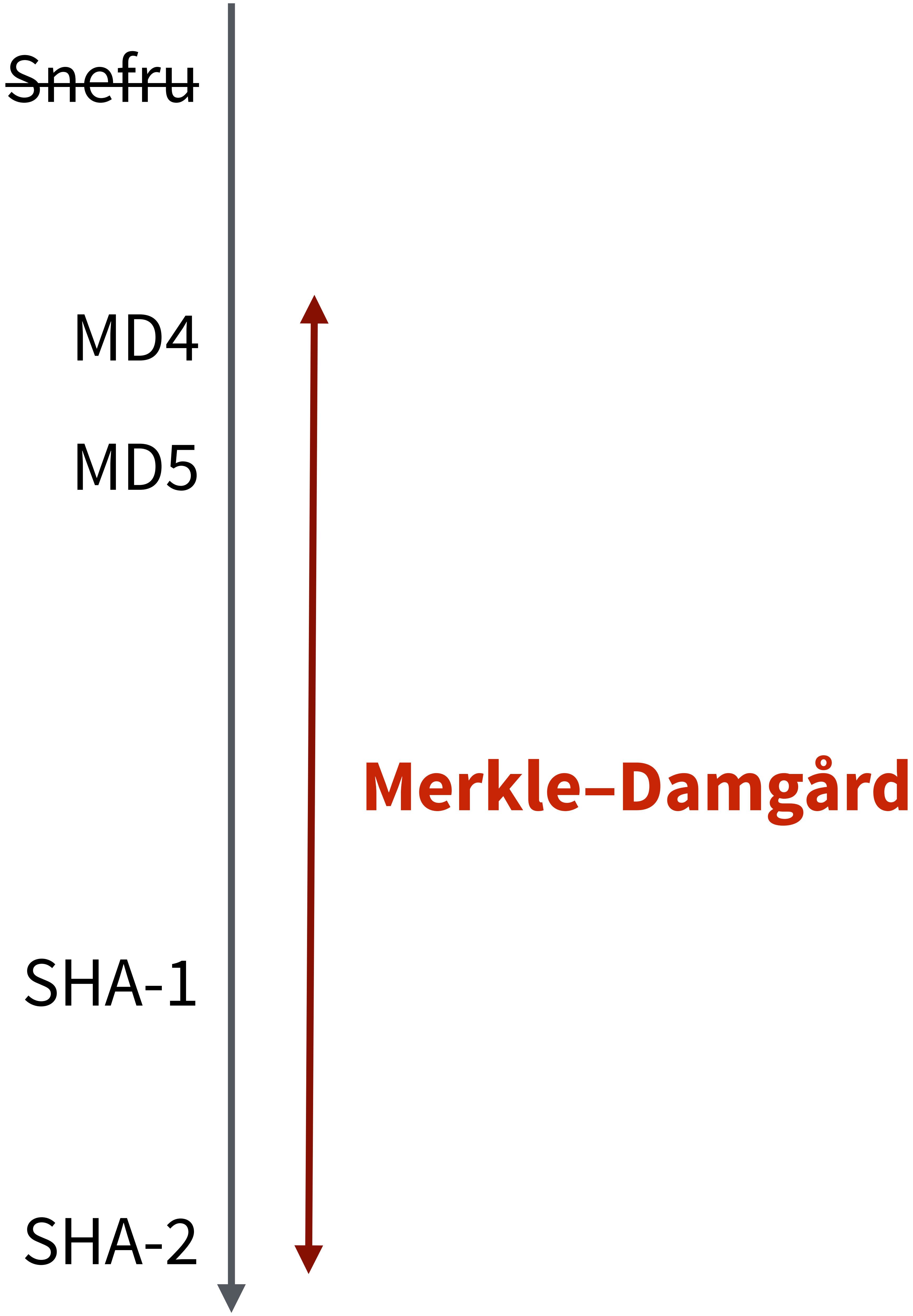
ncc group

Snefru

MD4

~~Snefru~~

MD4



~~Snefru~~

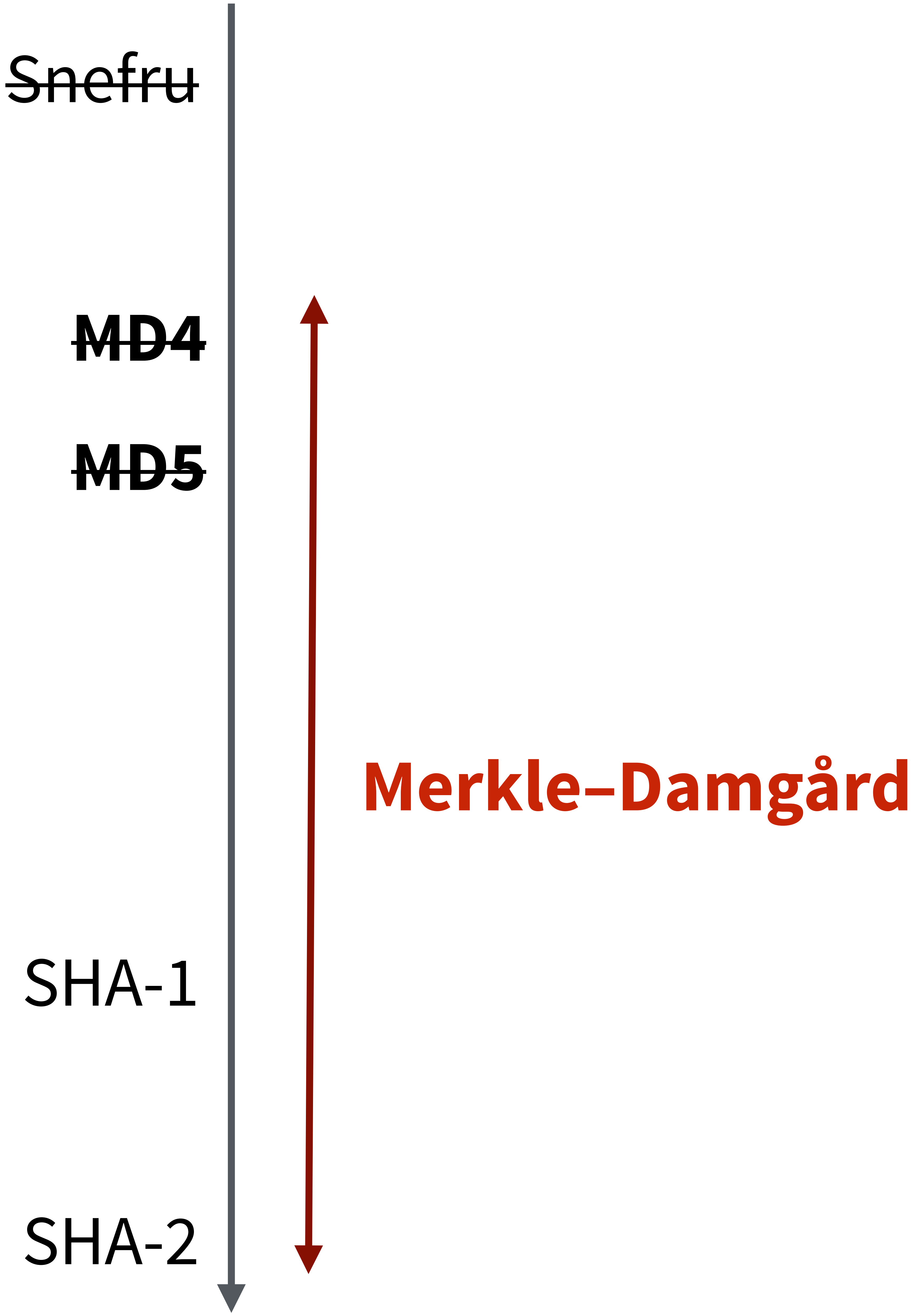
MD4

MD5

Merkle–Damgård

SHA-1

SHA-2



~~Snefru~~

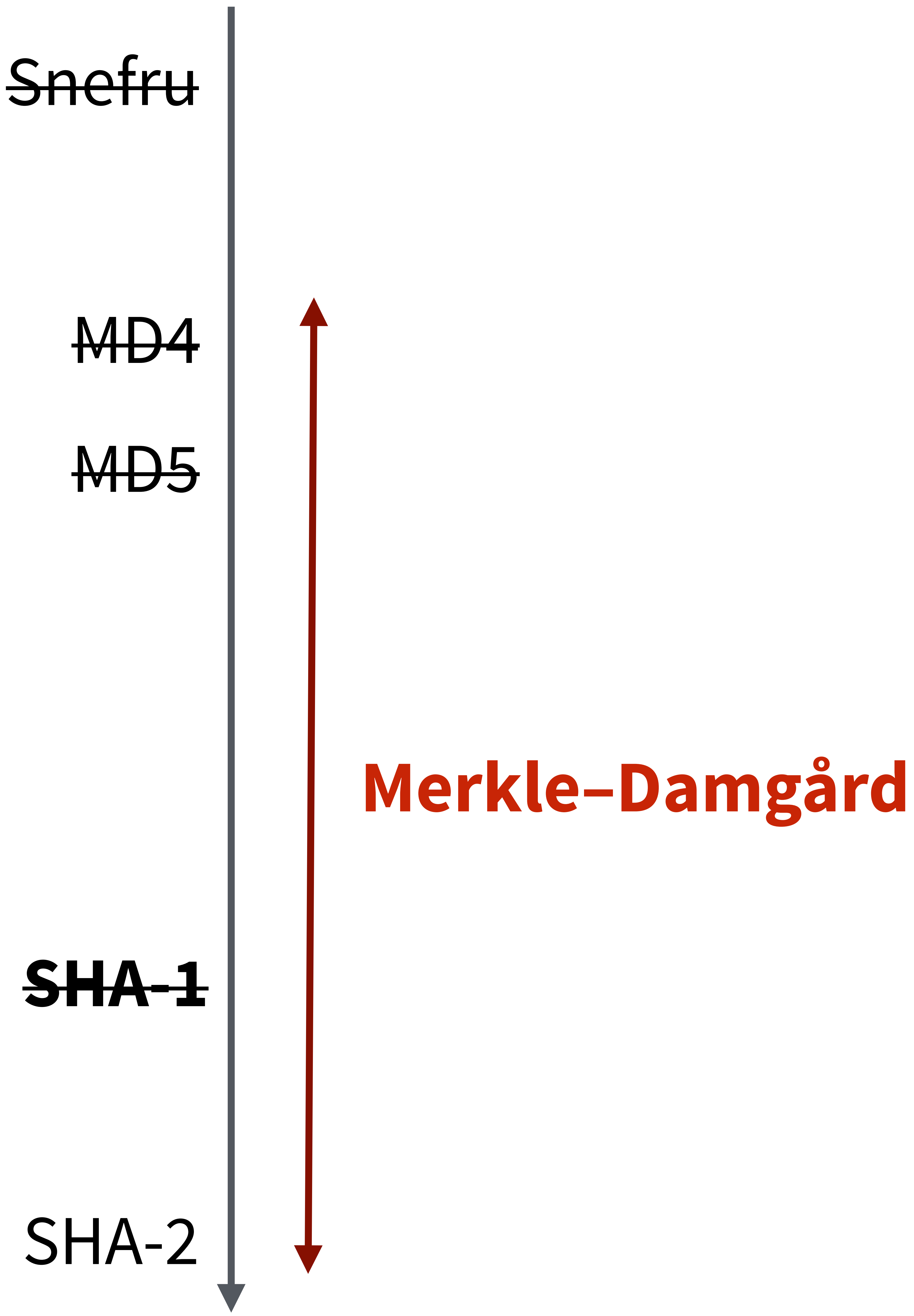
~~**MD4**~~

~~**MD5**~~

Merkle–Damgård

SHA-1

SHA-2



~~Snefru~~

~~MD4~~

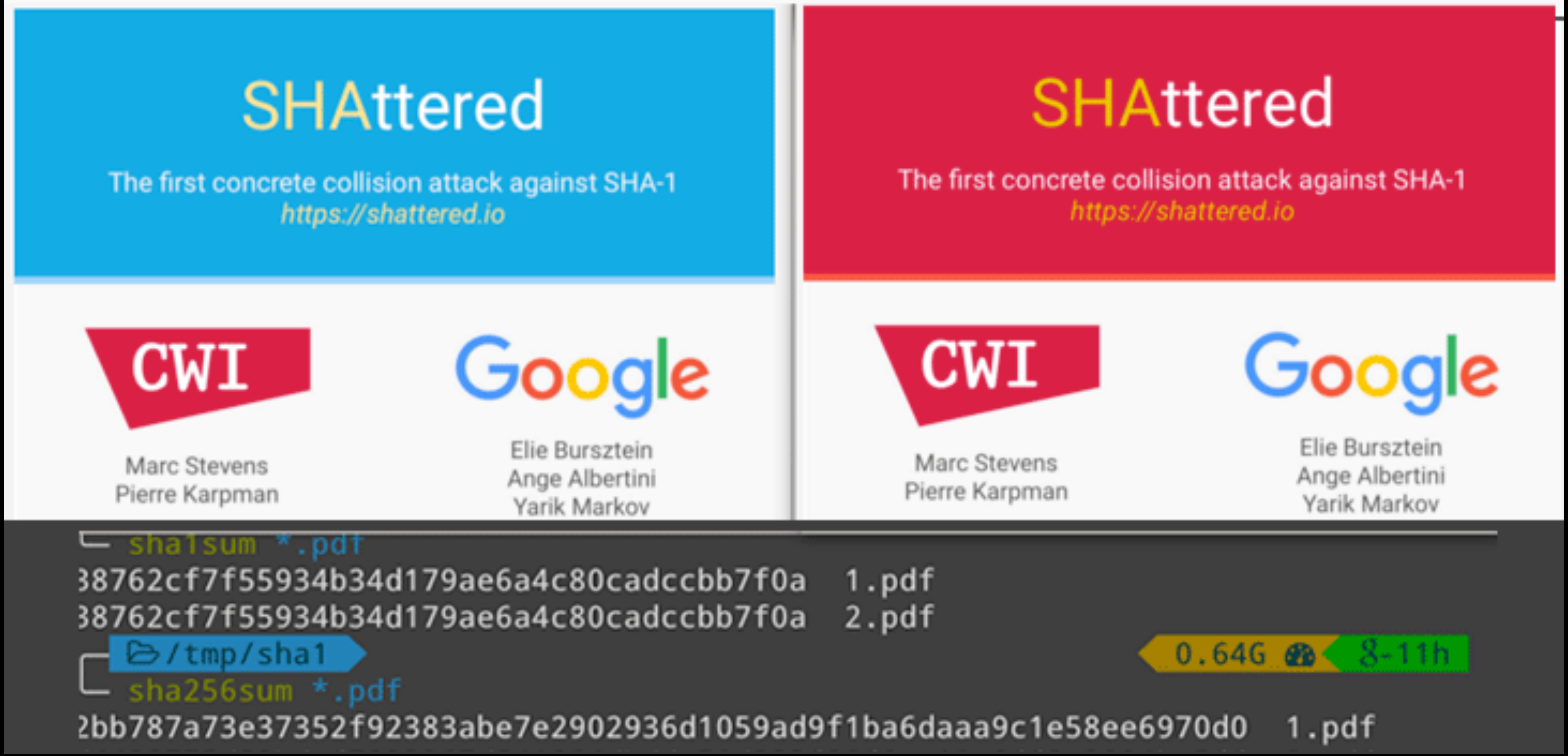
~~MD5~~

~~**SHA-1**~~

SHA-2

Merkle–Damgård

Collision Attack: Two Different Documents, But Same SHA-1 Hash Fingerprint



SHAttered

The first concrete collision attack against SHA-1
https://shattered.io

CWI

Marc Stevens
Pierre Karpman

Google

Elie Bursztein
Ange Albertini
Yarik Markov

SHAttered

The first concrete collision attack against SHA-1
https://shattered.io

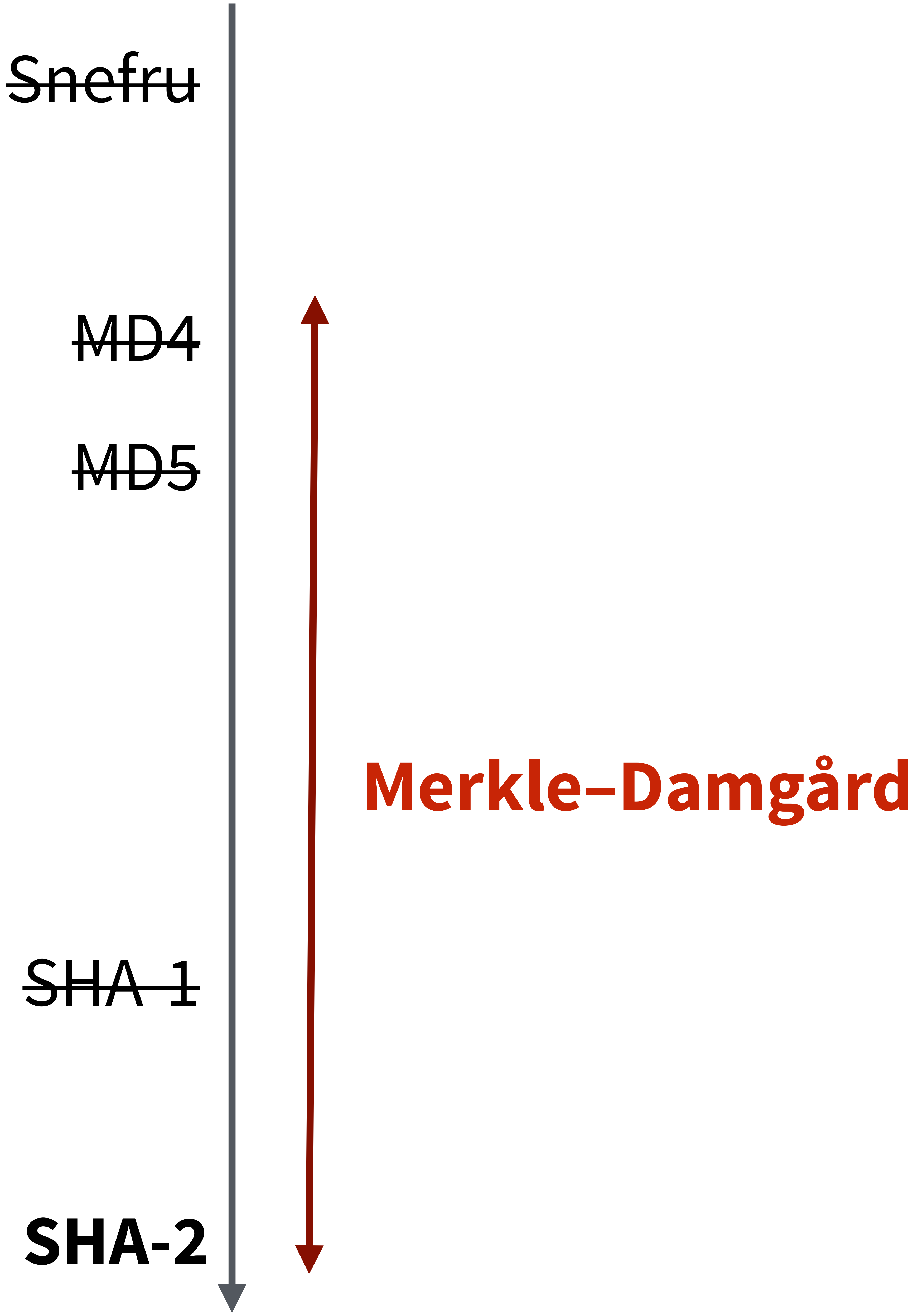
CWI

Marc Stevens
Pierre Karpman

Google

Elie Bursztein
Ange Albertini
Yarik Markov

```
└ sha1sum *.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a  1.pdf
38762cf7f55934b34d179ae6a4c80cadccbb7f0a  2.pdf
┌ /tmp/sha1                                        0.64G  8-11h
└ sha256sum *.pdf
2bb787a73e37352f92383abe7e2902936d1059ad9f1ba6daaa9c1e58ee6970d0  1.pdf
```



~~Snefru~~

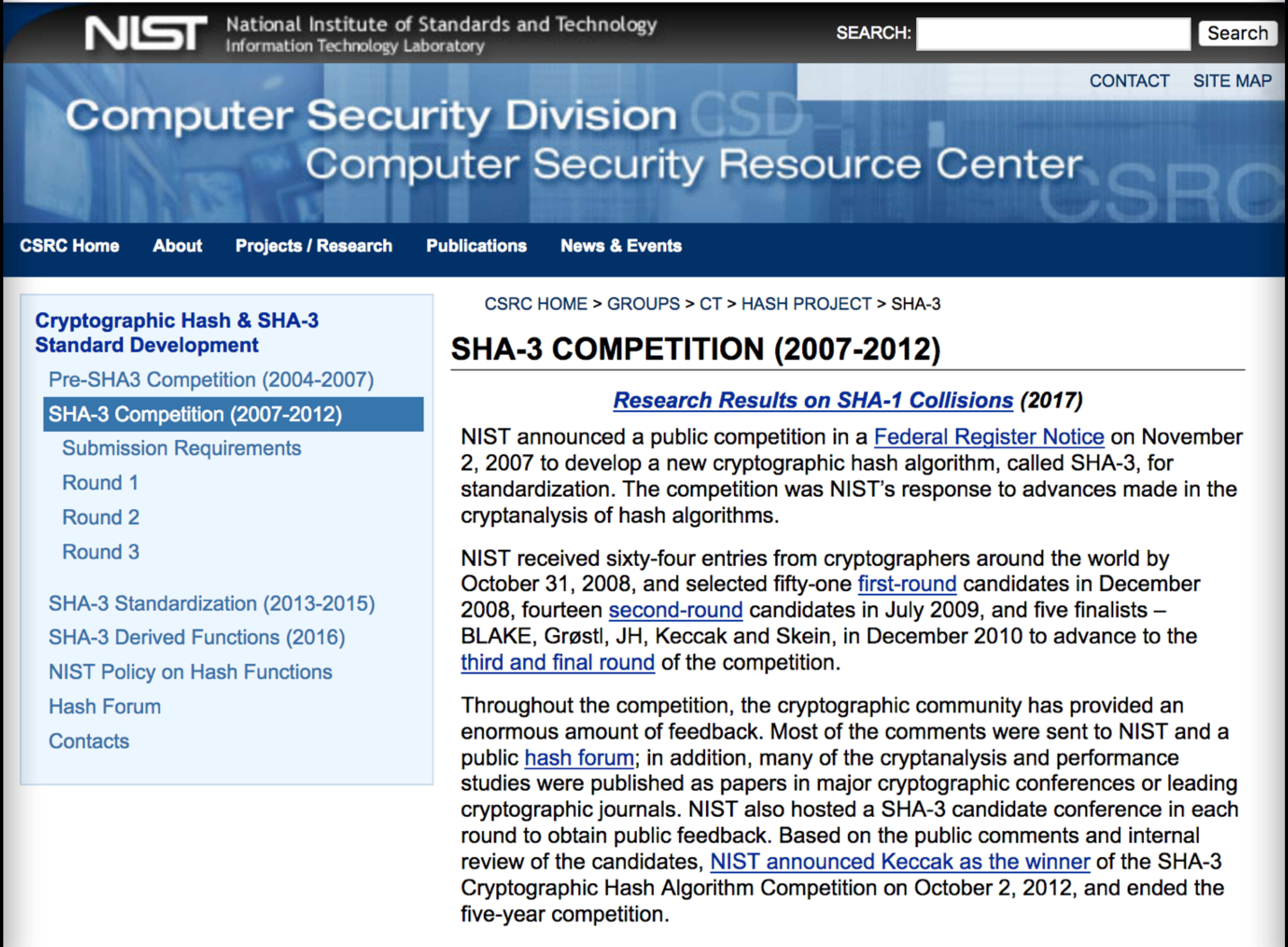
~~MD4~~

~~MD5~~

~~SHA-1~~

SHA-2

Merkle–Damgård


NIST National Institute of Standards and Technology
Information Technology Laboratory
SEARCH:
Search
CONTACT
SITE MAP
Computer Security Division CSD
Computer Security Resource Center CSRC
CSRC Home
About
Projects / Research
Publications
News & Events
Cryptographic Hash & SHA-3 Standard Development
Pre-SHA3 Competition (2004-2007)
SHA-3 Competition (2007-2012)
Submission Requirements
Round 1
Round 2
Round 3
SHA-3 Standardization (2013-2015)
SHA-3 Derived Functions (2016)
NIST Policy on Hash Functions
Hash Forum
Contacts

CSRC HOME > GROUPS > CT > HASH PROJECT > SHA-3

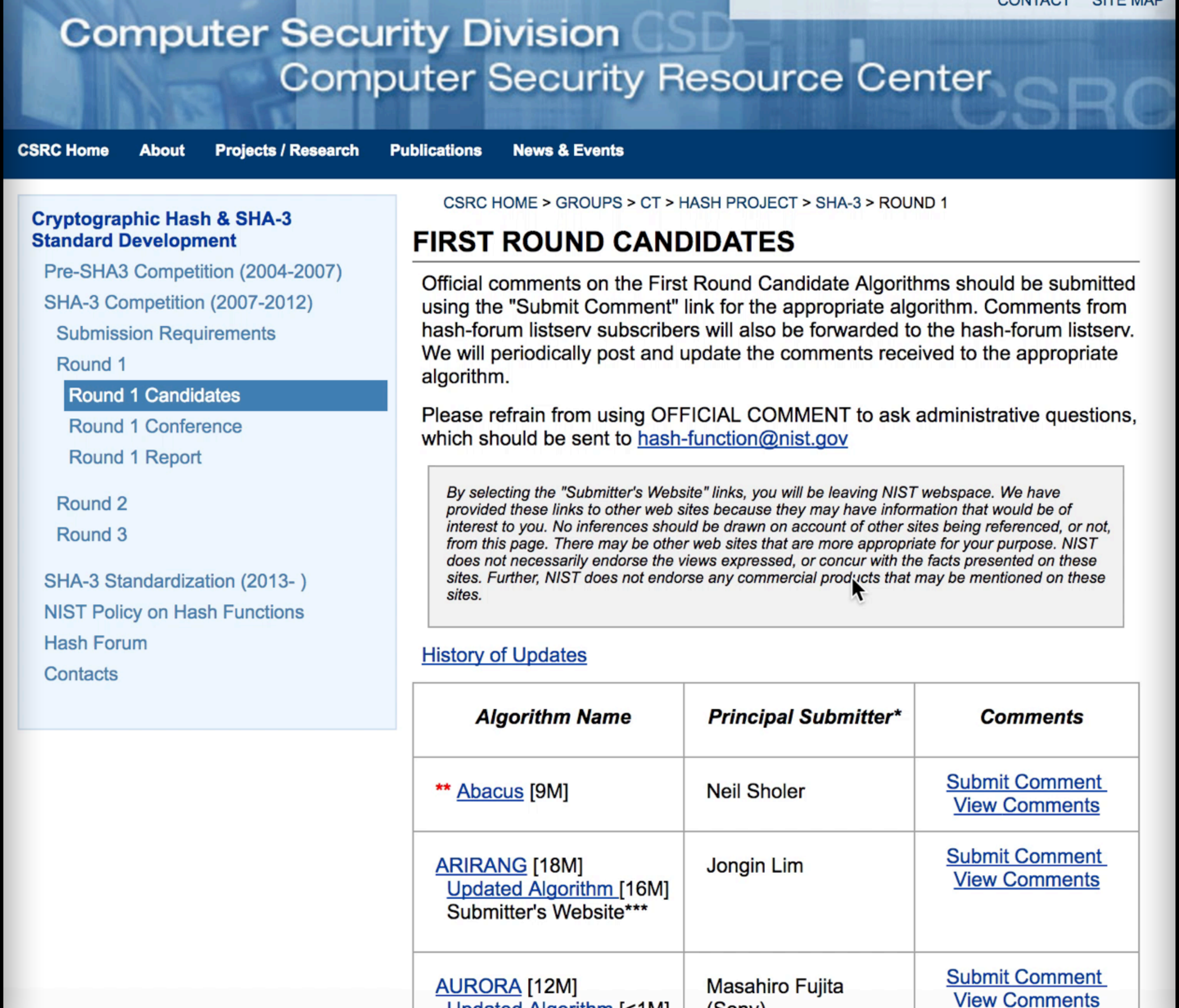
SHA-3 COMPETITION (2007-2012)

***Research Results on SHA-1 Collisions** (2017)*

NIST announced a public competition in a Federal Register Notice on November 2, 2007 to develop a new cryptographic hash algorithm, called SHA-3, for standardization. The competition was NIST's response to advances made in the cryptanalysis of hash algorithms.

NIST received sixty-four entries from cryptographers around the world by October 31, 2008, and selected fifty-one first-round candidates in December 2008, fourteen second-round candidates in July 2009, and five finalists – BLAKE, Grøstl, JH, Keccak and Skein, in December 2010 to advance to the third and final round of the competition.

Throughout the competition, the cryptographic community has provided an enormous amount of feedback. Most of the comments were sent to NIST and a public hash forum; in addition, many of the cryptanalysis and performance studies were published as papers in major cryptographic conferences or leading cryptographic journals. NIST also hosted a SHA-3 candidate conference in each round to obtain public feedback. Based on the public comments and internal review of the candidates, NIST announced Keccak as the winner of the SHA-3 Cryptographic Hash Algorithm Competition on October 2, 2012, and ended the five-year competition.



CONTACT SITE MAP
Computer Security Division CSD
Computer Security Resource Center CSRC
CSRC Home
About
Projects / Research
Publications
News & Events

Cryptographic Hash & SHA-3 Standard Development
Pre-SHA3 Competition (2004-2007)
SHA-3 Competition (2007-2012)
Submission Requirements
Round 1
Round 1 Candidates
Round 1 Conference
Round 1 Report
Round 2
Round 3
SHA-3 Standardization (2013-)
NIST Policy on Hash Functions
Hash Forum
Contacts

CSRC HOME > GROUPS > CT > HASH PROJECT > SHA-3 > ROUND 1

FIRST ROUND CANDIDATES

Official comments on the First Round Candidate Algorithms should be submitted using the "Submit Comment" link for the appropriate algorithm. Comments from hash-forum listserv subscribers will also be forwarded to the hash-forum listserv. We will periodically post and update the comments received to the appropriate algorithm.

Please refrain from using OFFICIAL COMMENT to ask administrative questions, which should be sent to hash-function@nist.gov

By selecting the "Submitter's Website" links, you will be leaving NIST webspace. We have provided these links to other web sites because they may have information that would be of interest to you. No inferences should be drawn on account of other sites being referenced, or not, from this page. There may be other web sites that are more appropriate for your purpose. NIST does not necessarily endorse the views expressed, or concur with the facts presented on these sites. Further, NIST does not endorse any commercial products that may be mentioned on these sites.

History of Updates

Algorithm Name	***Principal Submitter****	***Comments***
** Abacus [9M]	Neil Sholer	Submit Comment View Comments
ARIRANG [18M] Updated Algorithm [16M] Submitter's Website***	Jongin Lim	Submit Comment View Comments
AURORA [12M] Updated Algorithm [<1M]	Masahiro Fujita (Sony)	Submit Comment View Comments

Keccak

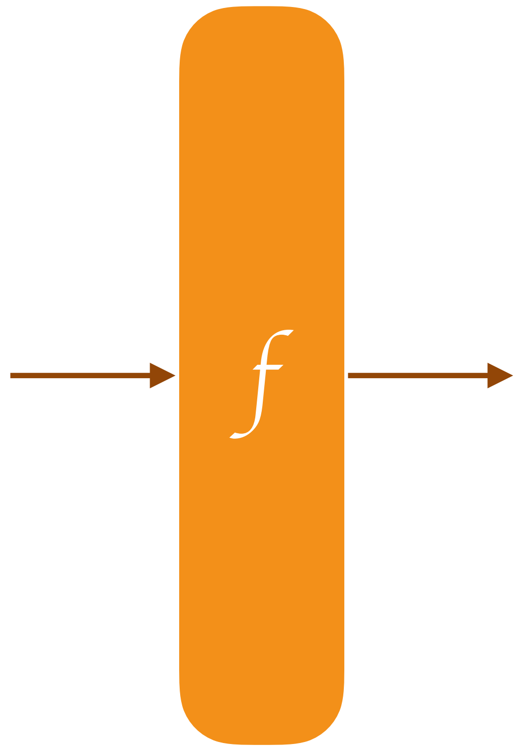
BLAKE, Grøstl, JH, Skein

outline

1. Intro
2. **SHA-3**
3. Strobe, a protocol framework derived from SHA-3
4. Noise, a full protocol framework not derived from SHA-3
5. Strobe + Noise = Disco

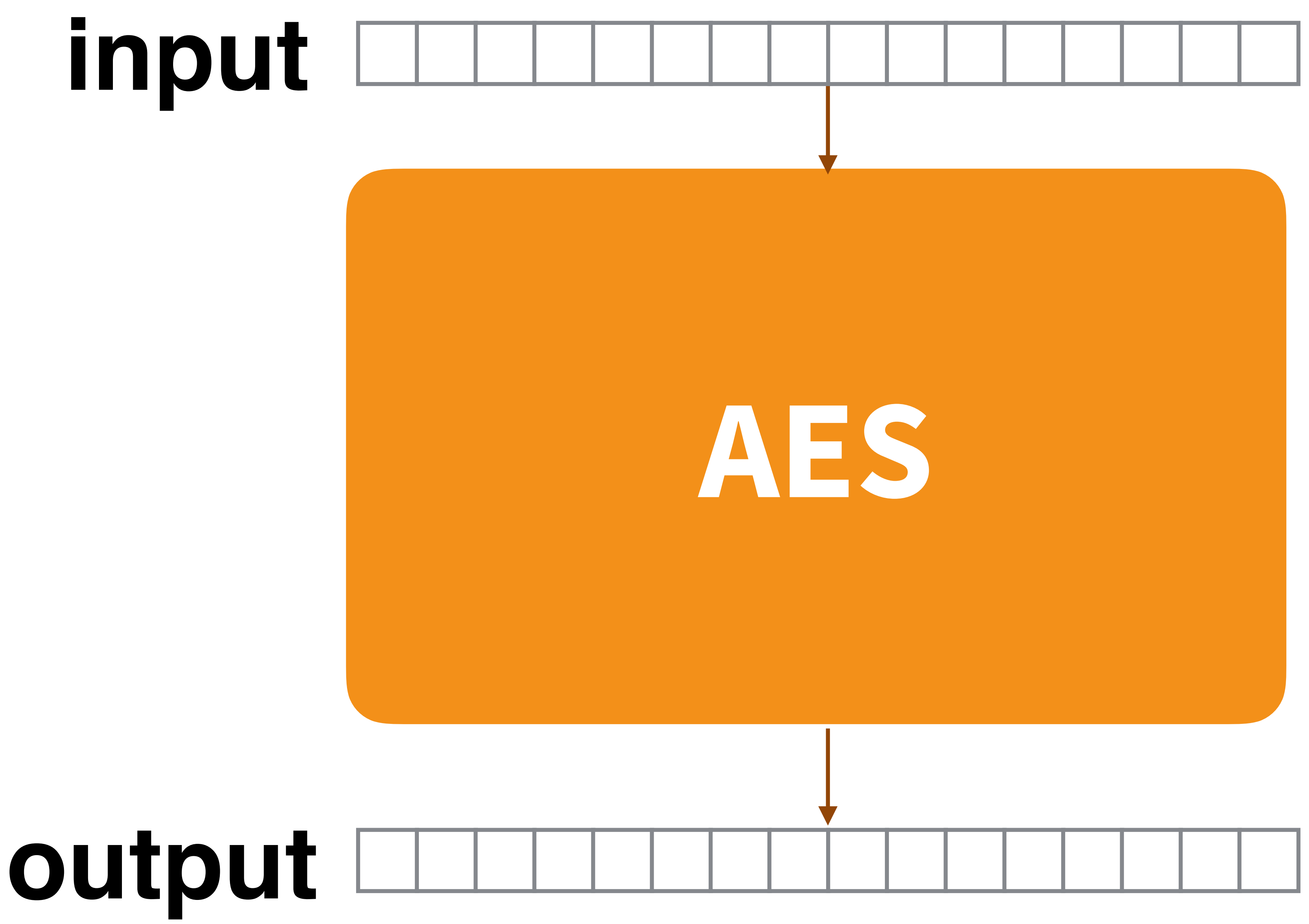
Part I: SHA-3

Big things have small beginnings

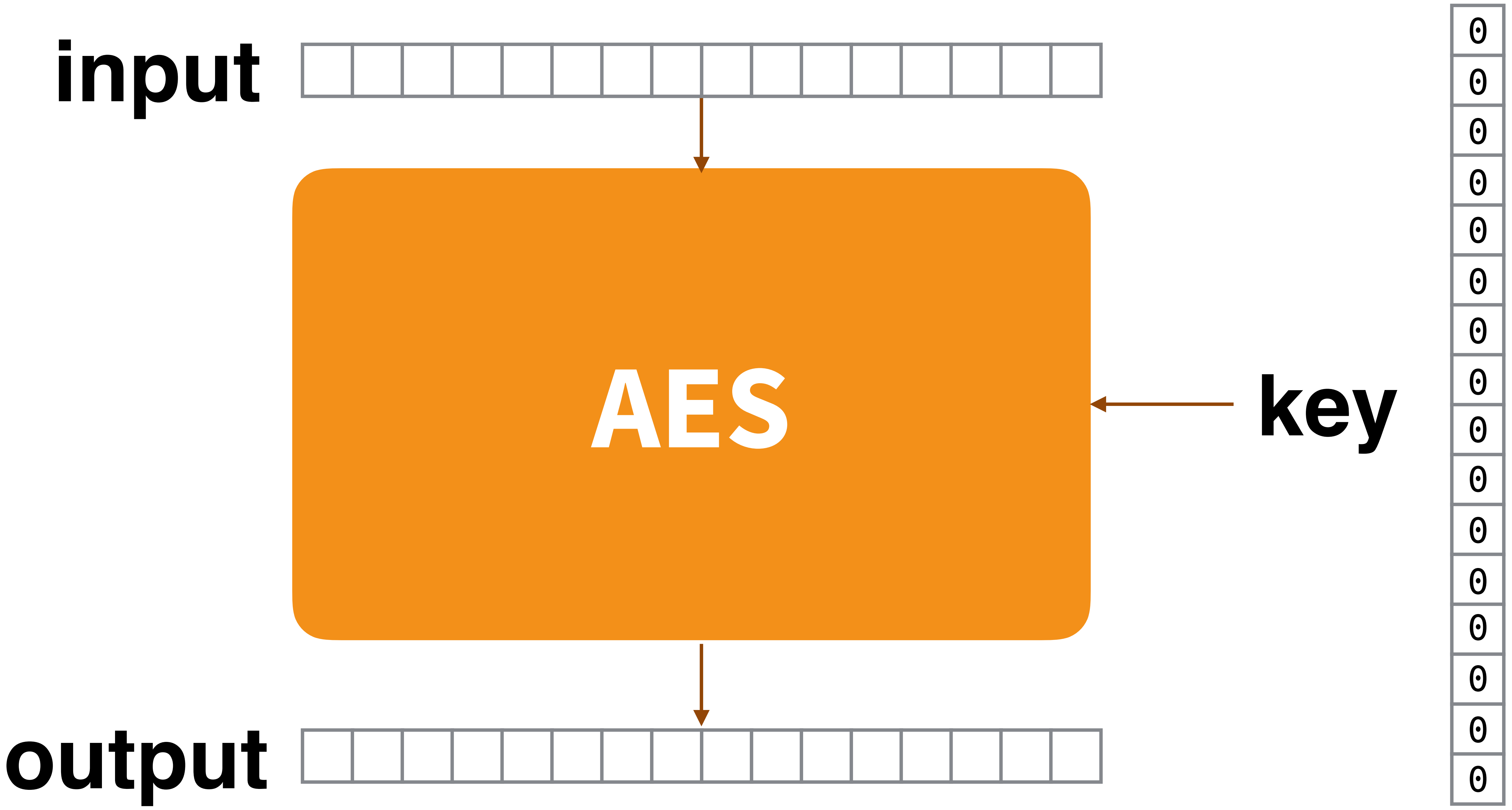


permutation-based cryptography

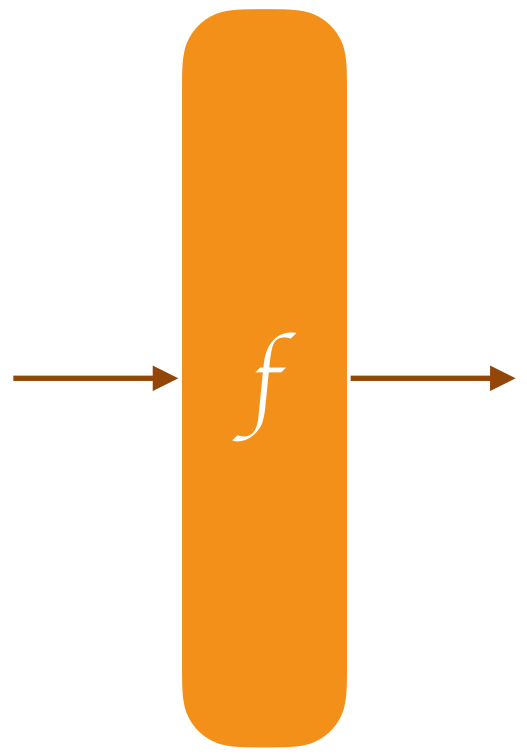
AES is a permutation



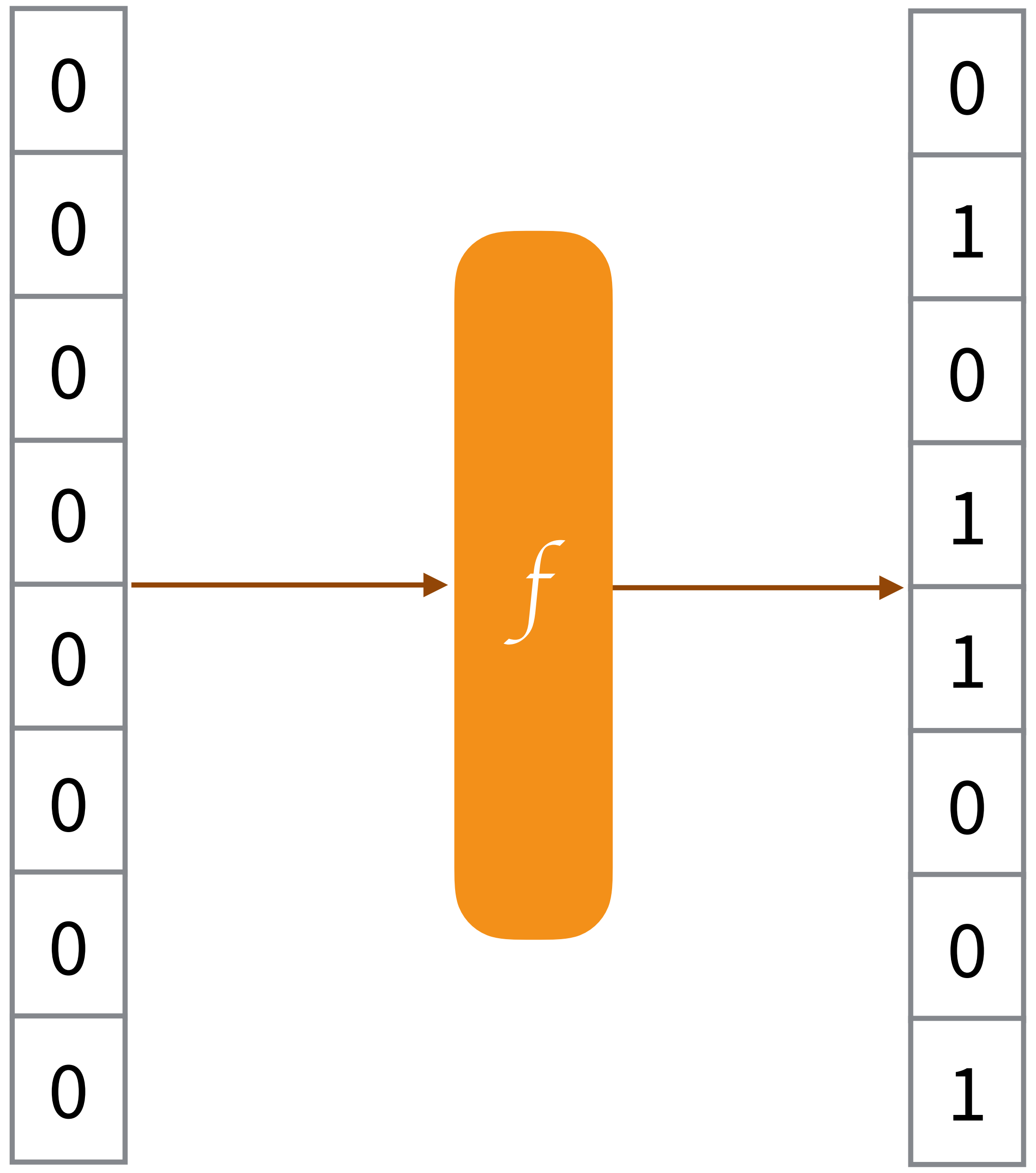
AES is a permutation



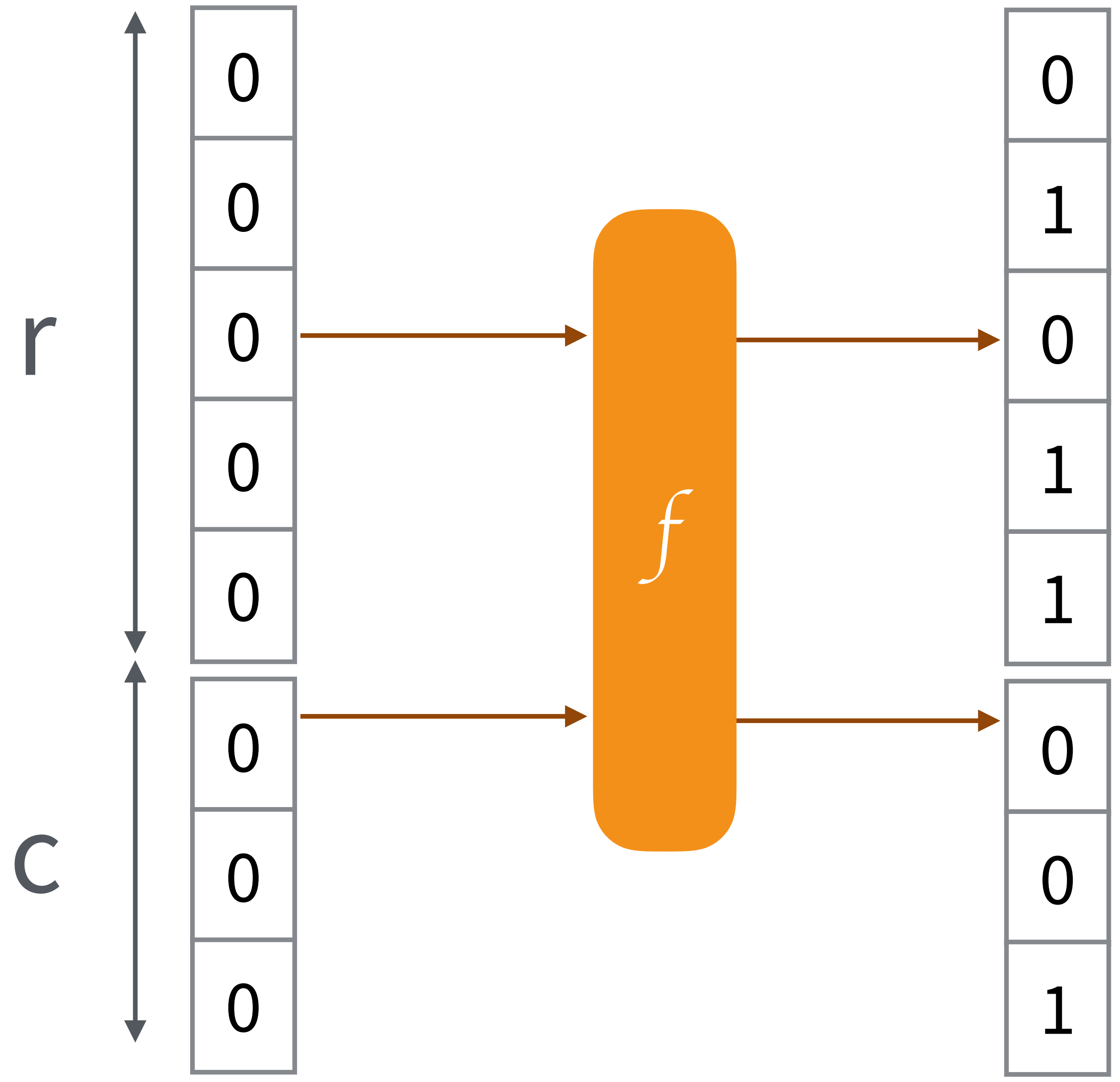
Sponge Construction



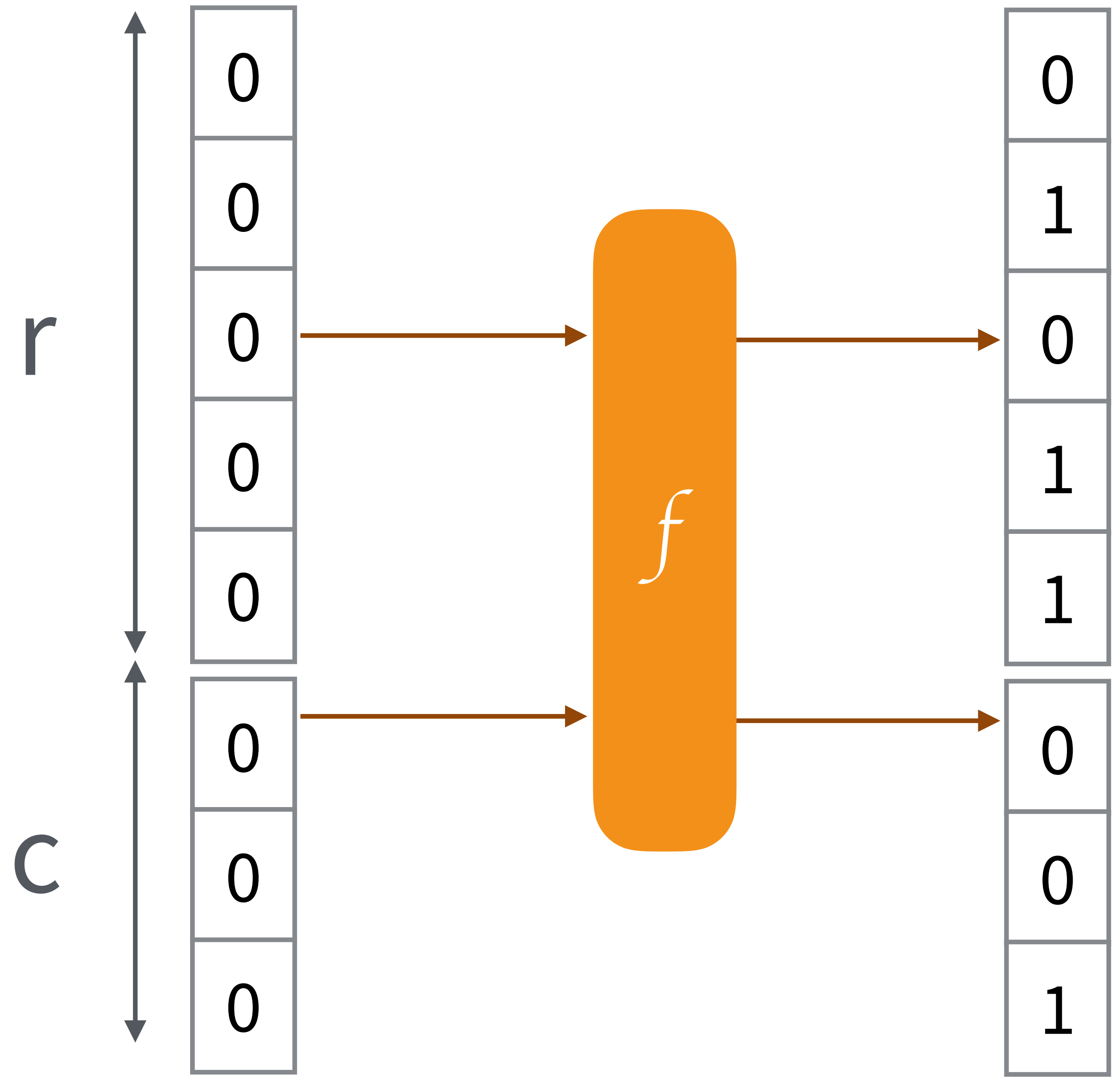
Sponge Construction

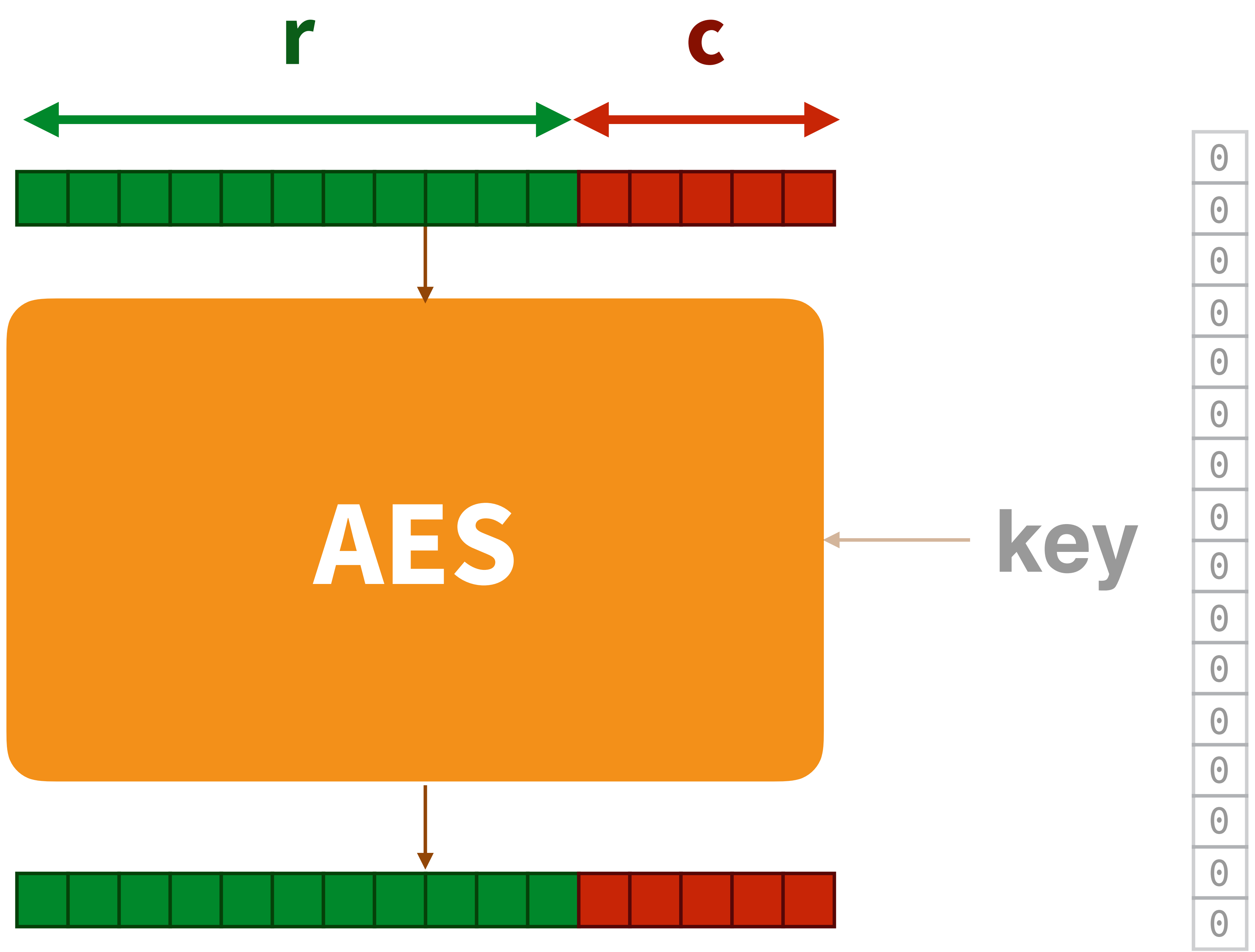


Sponge Construction

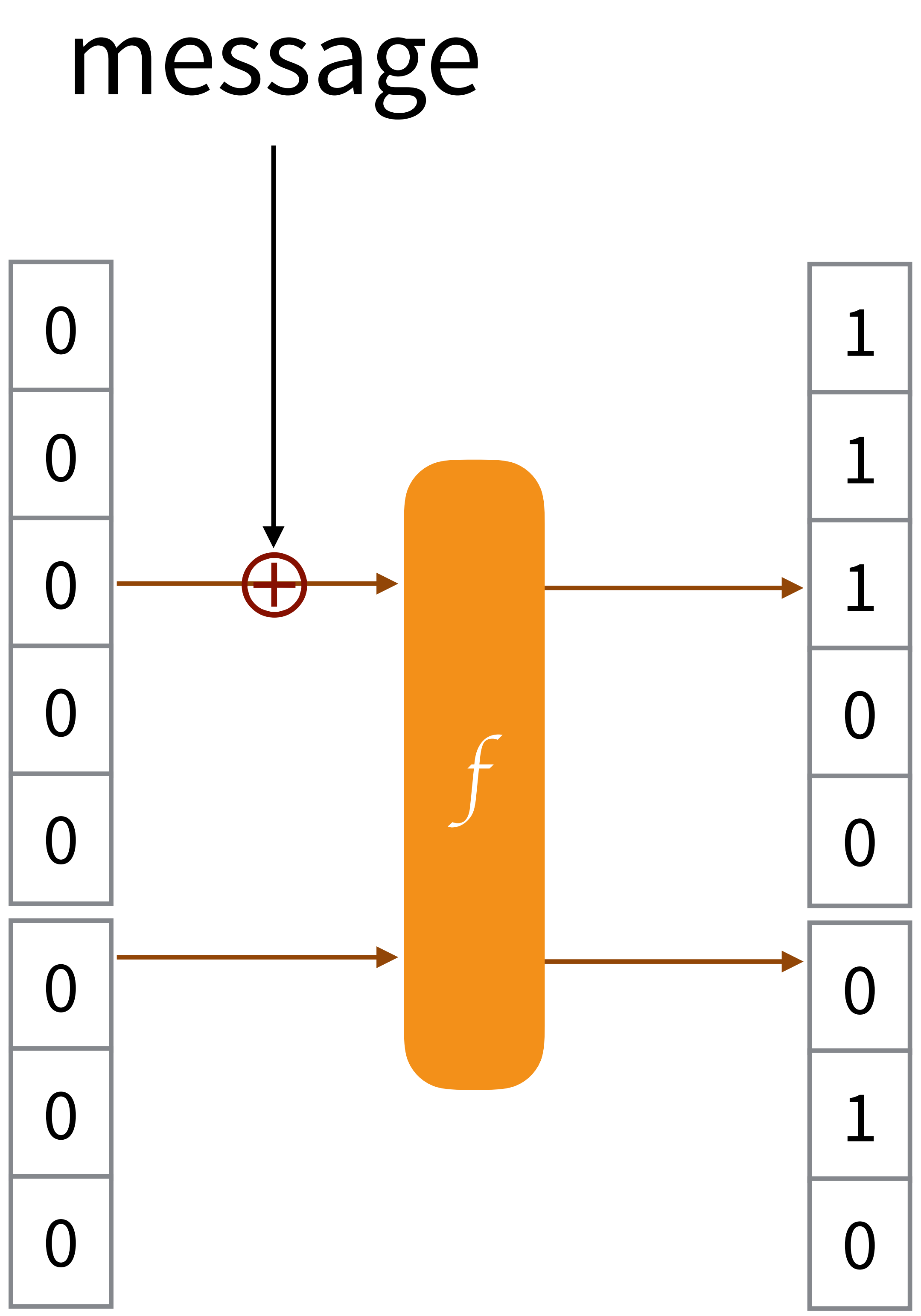


Sponge Construction

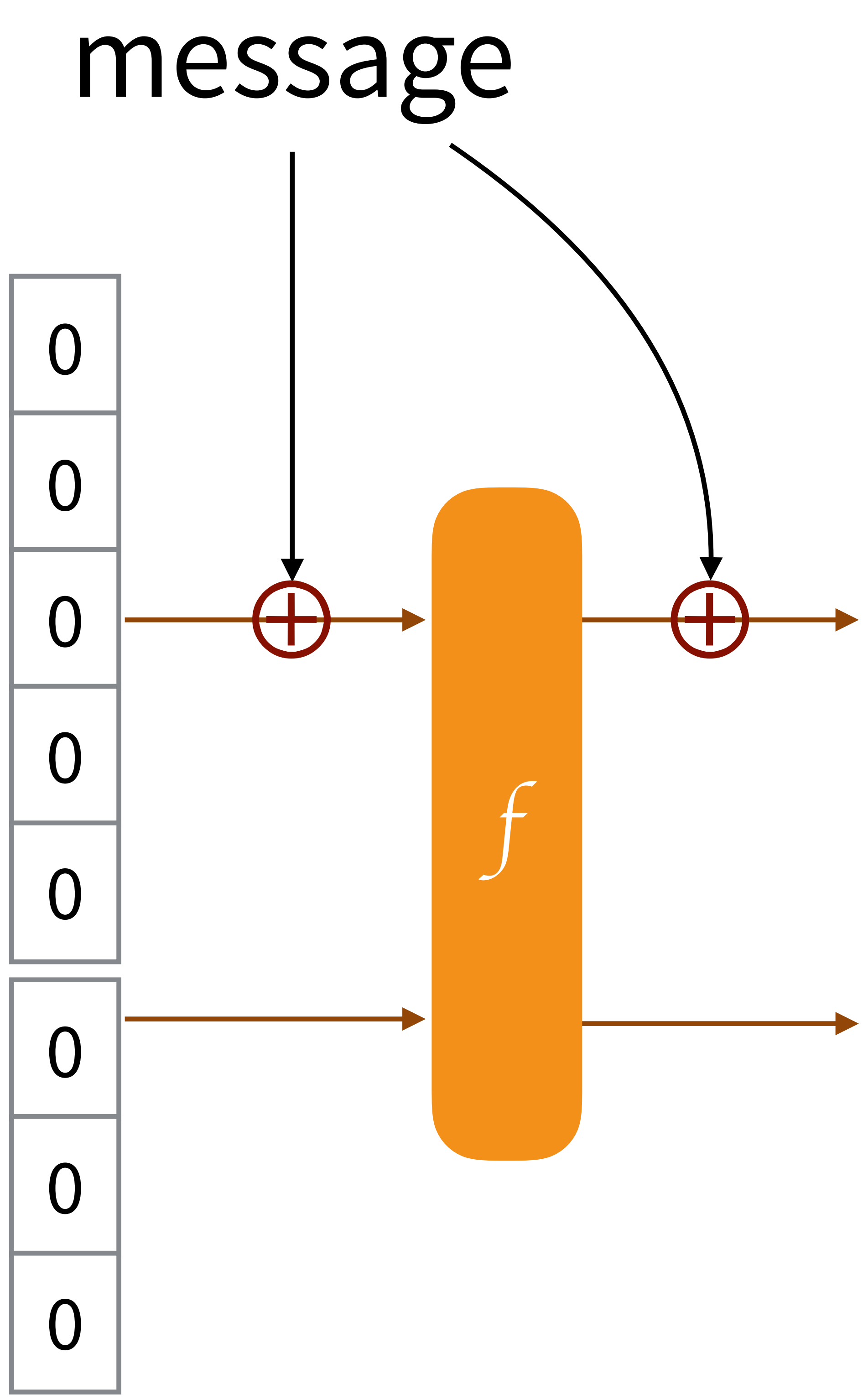




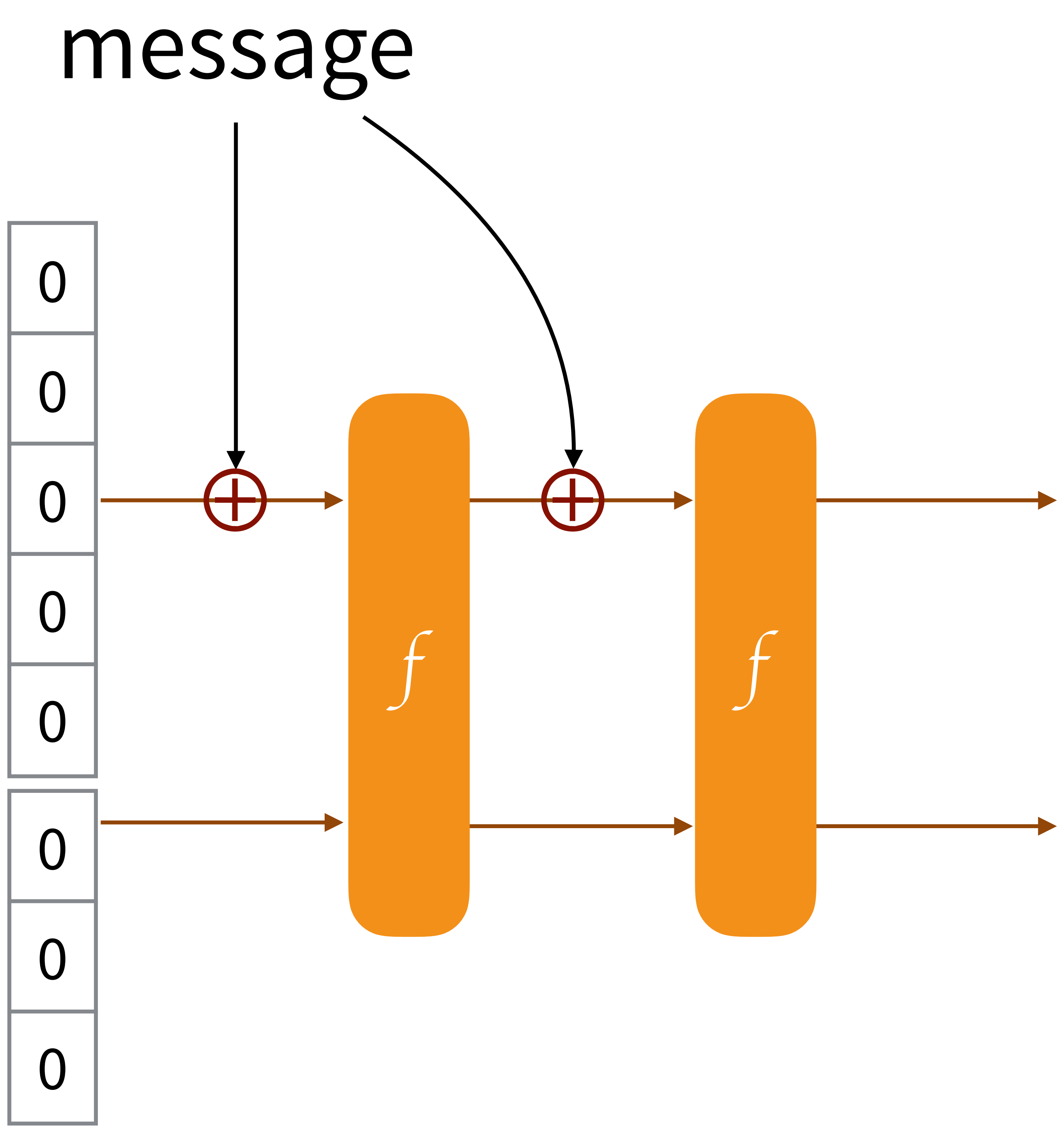
Sponge Construction



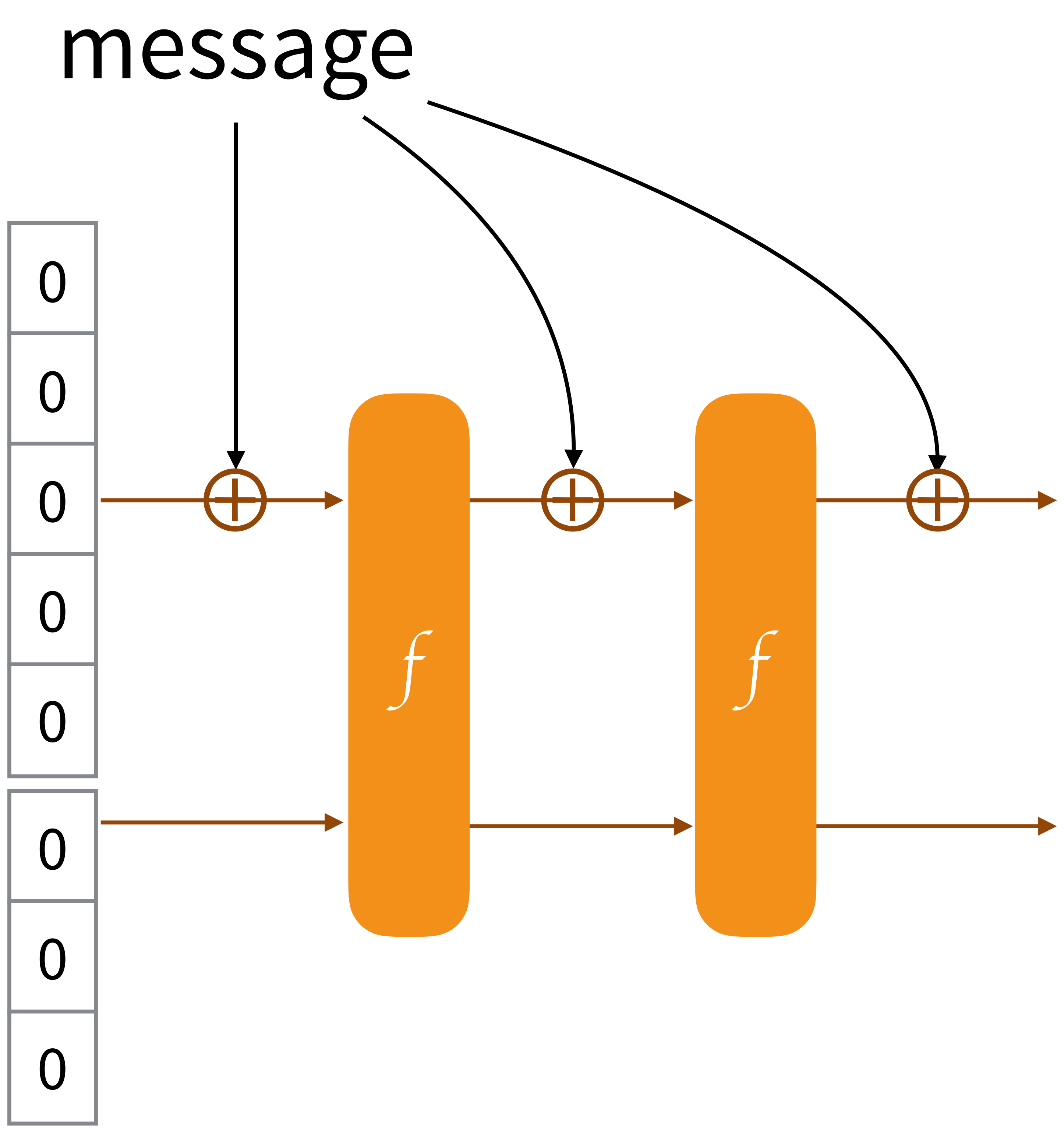
Sponge Construction



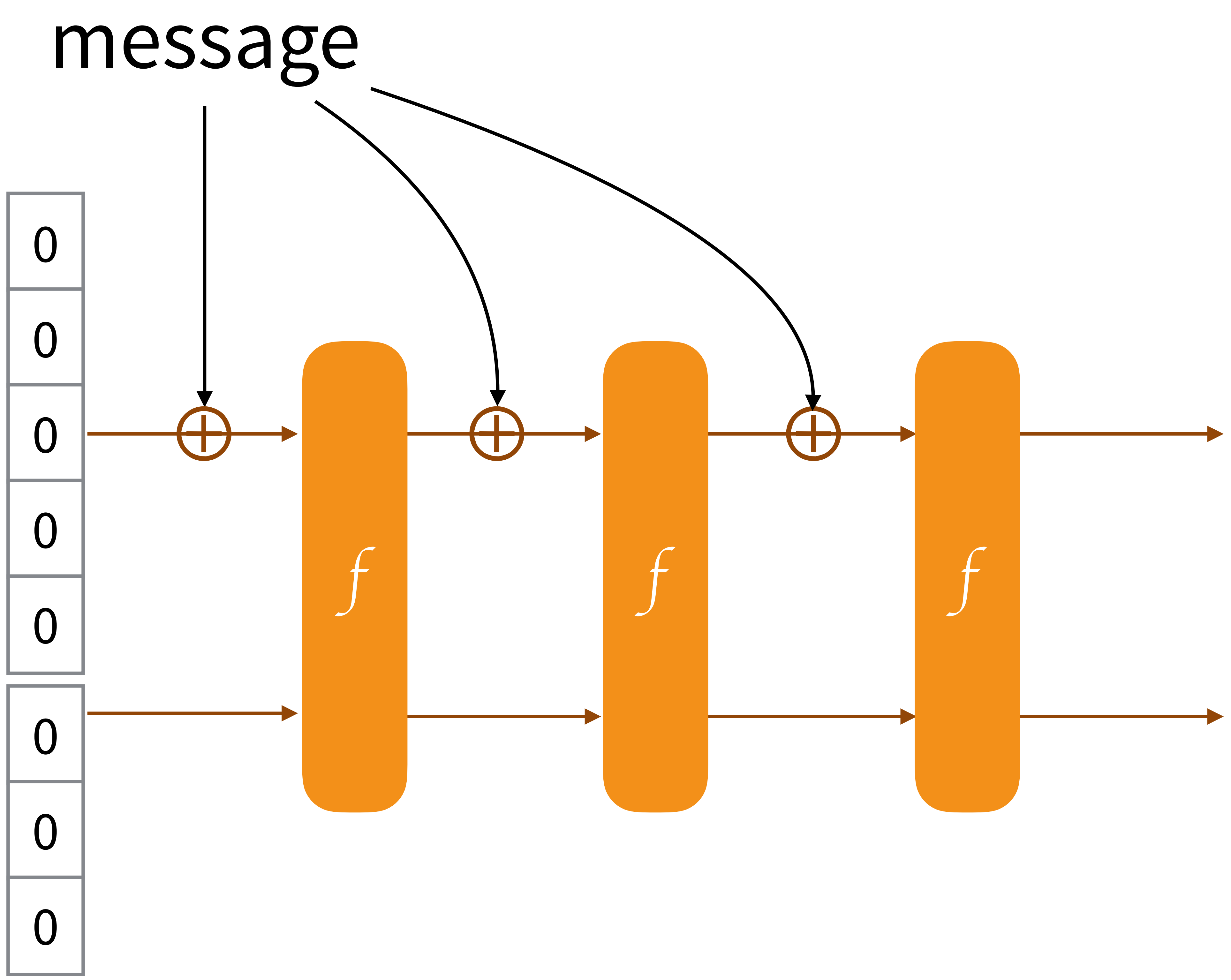
Sponge Construction



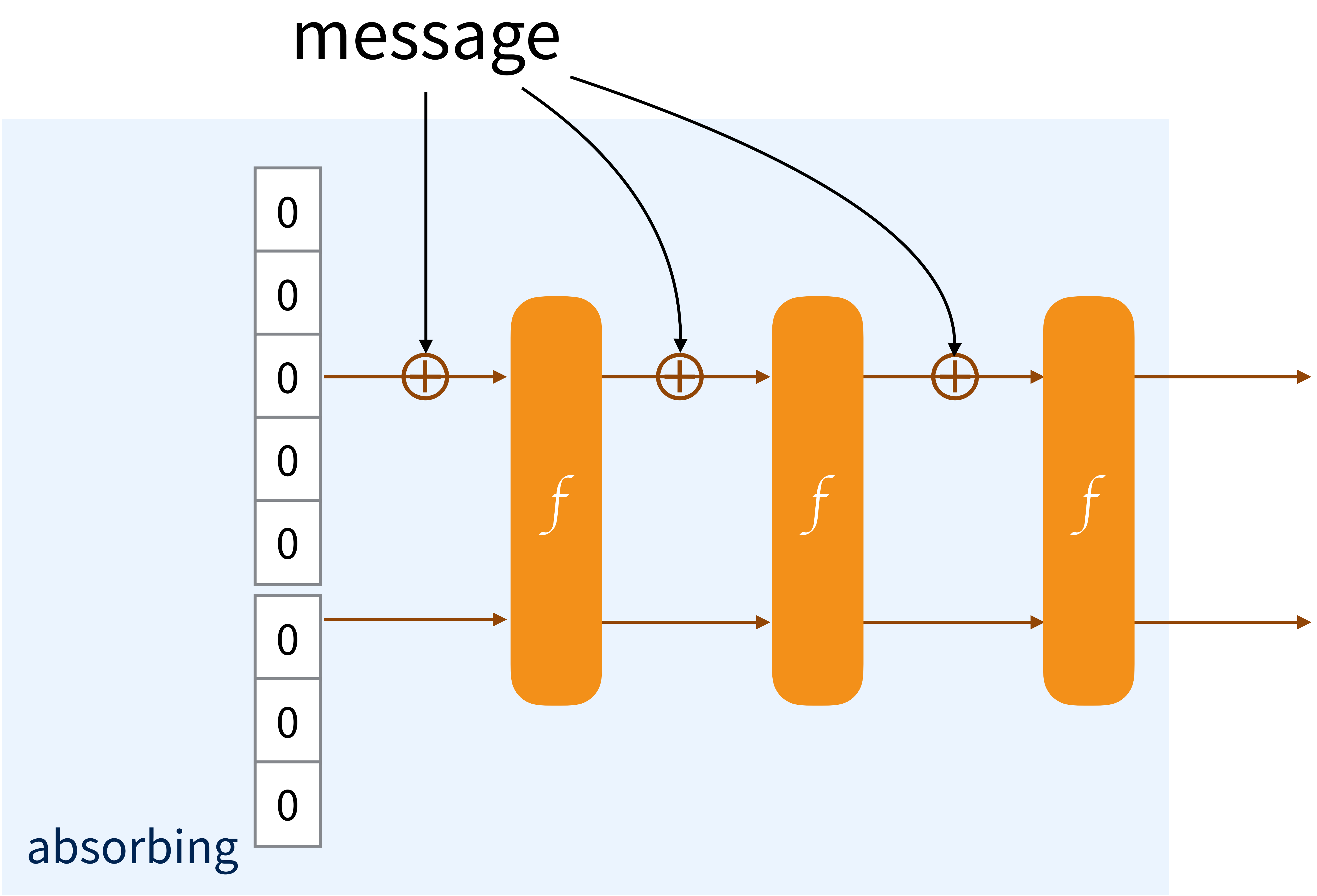
Sponge Construction



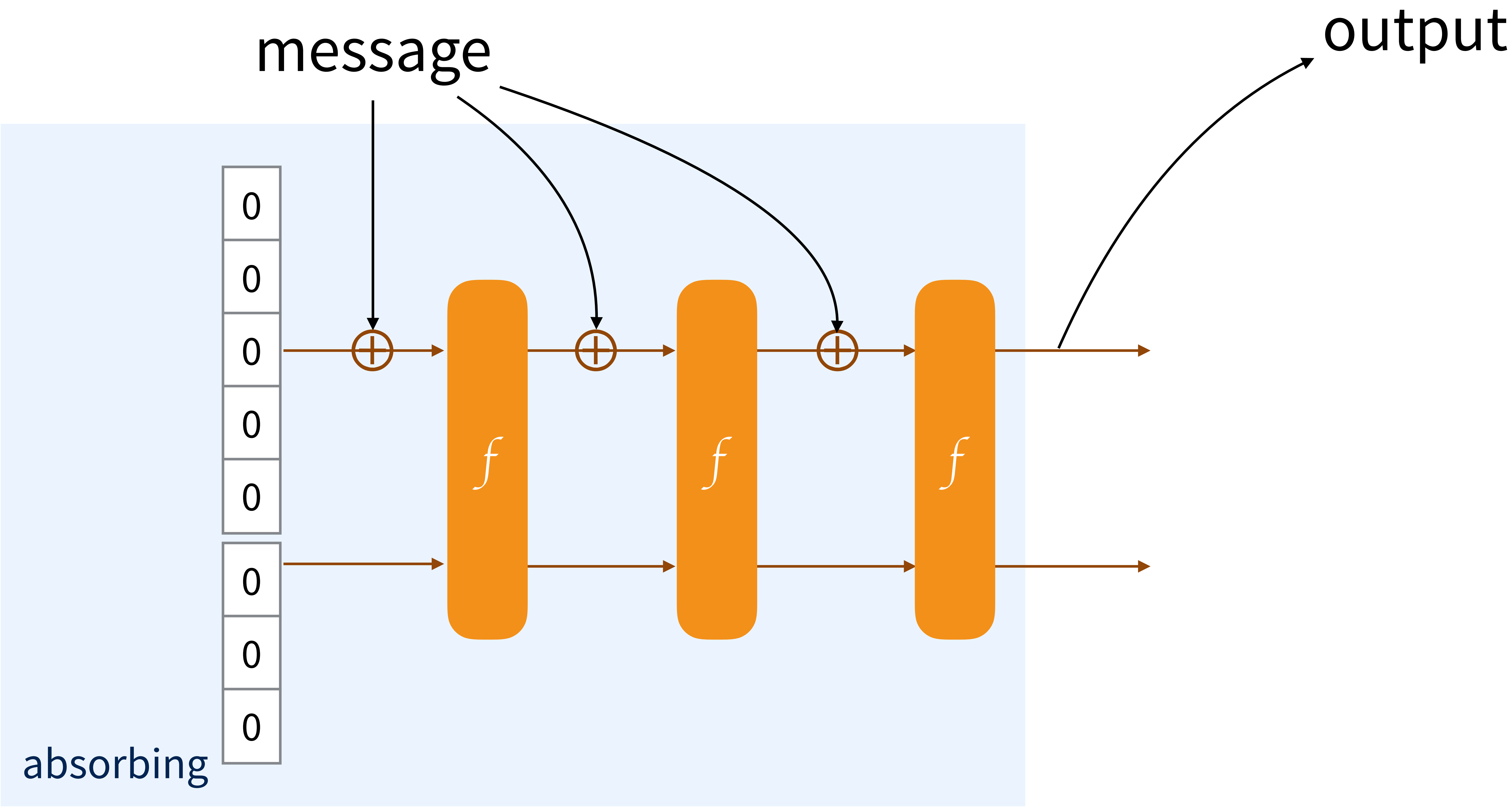
Sponge Construction



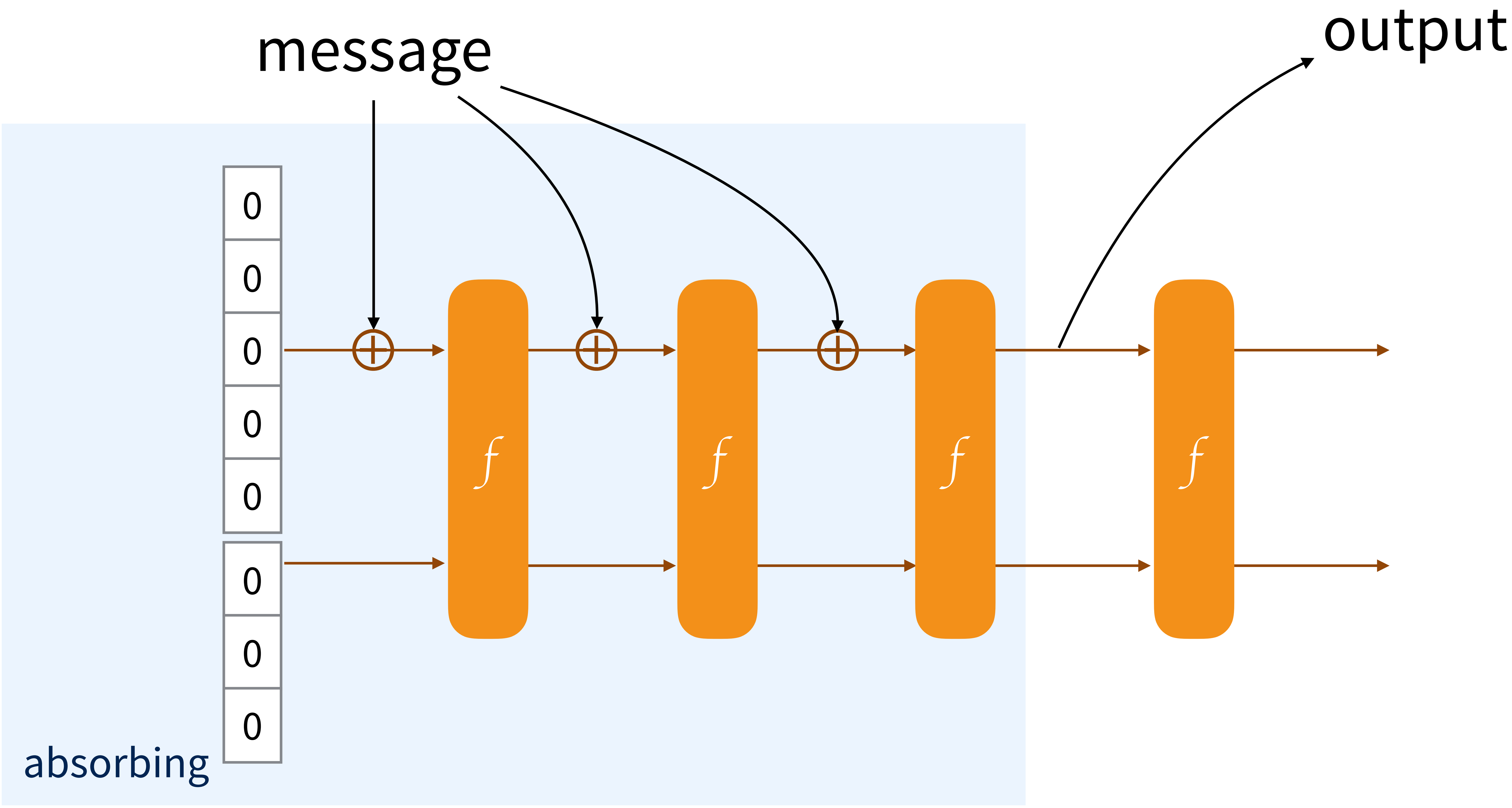
Sponge Construction



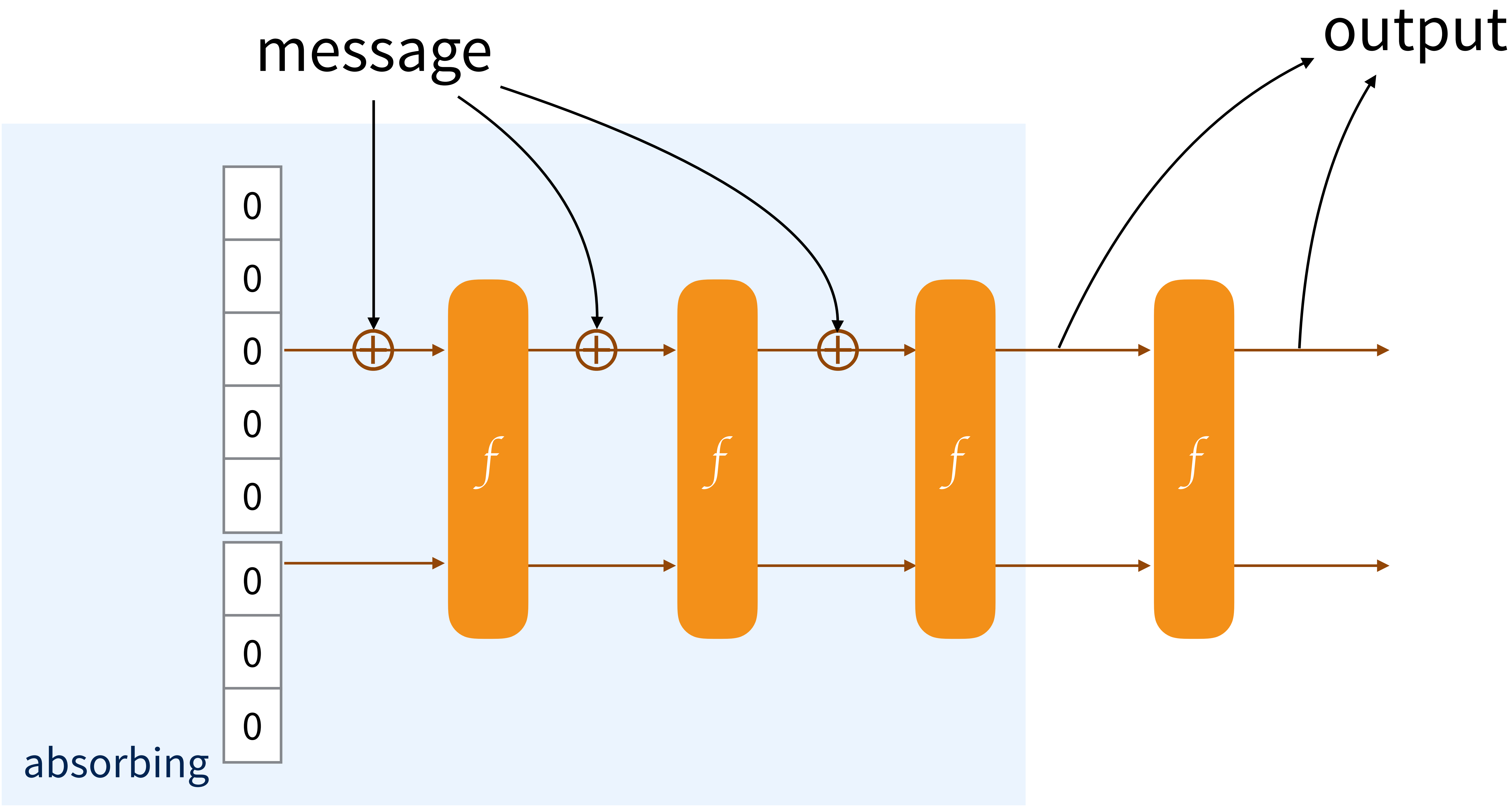
Sponge Construction



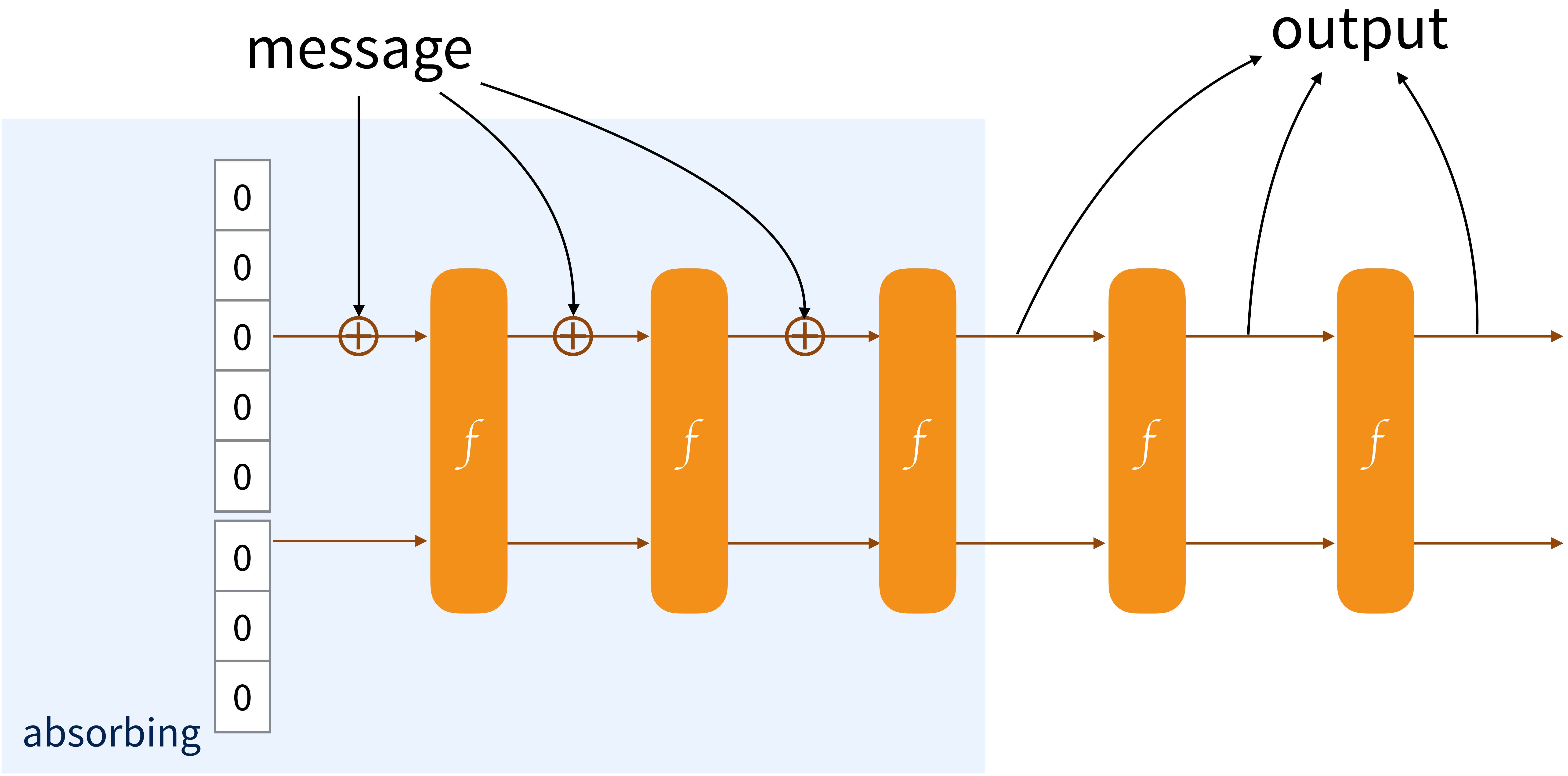
Sponge Construction



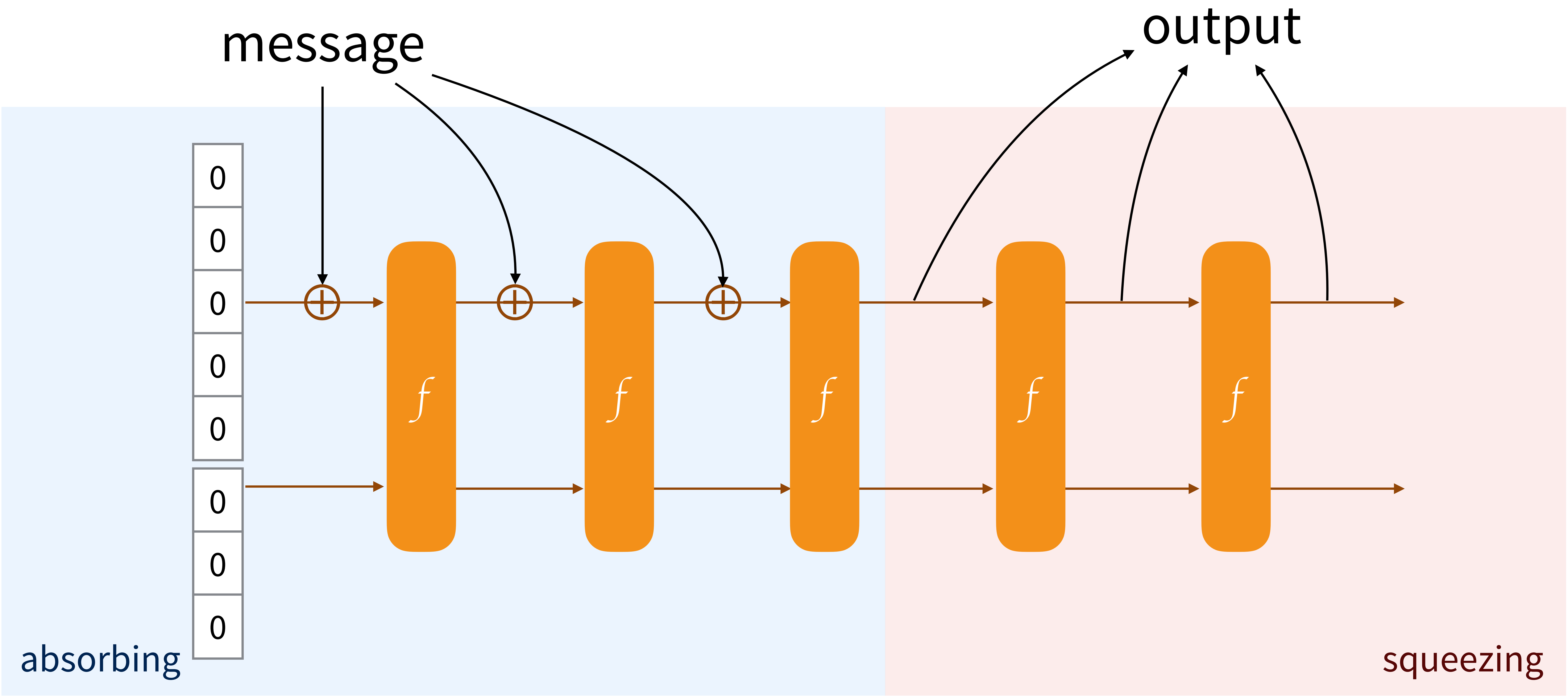
Sponge Construction

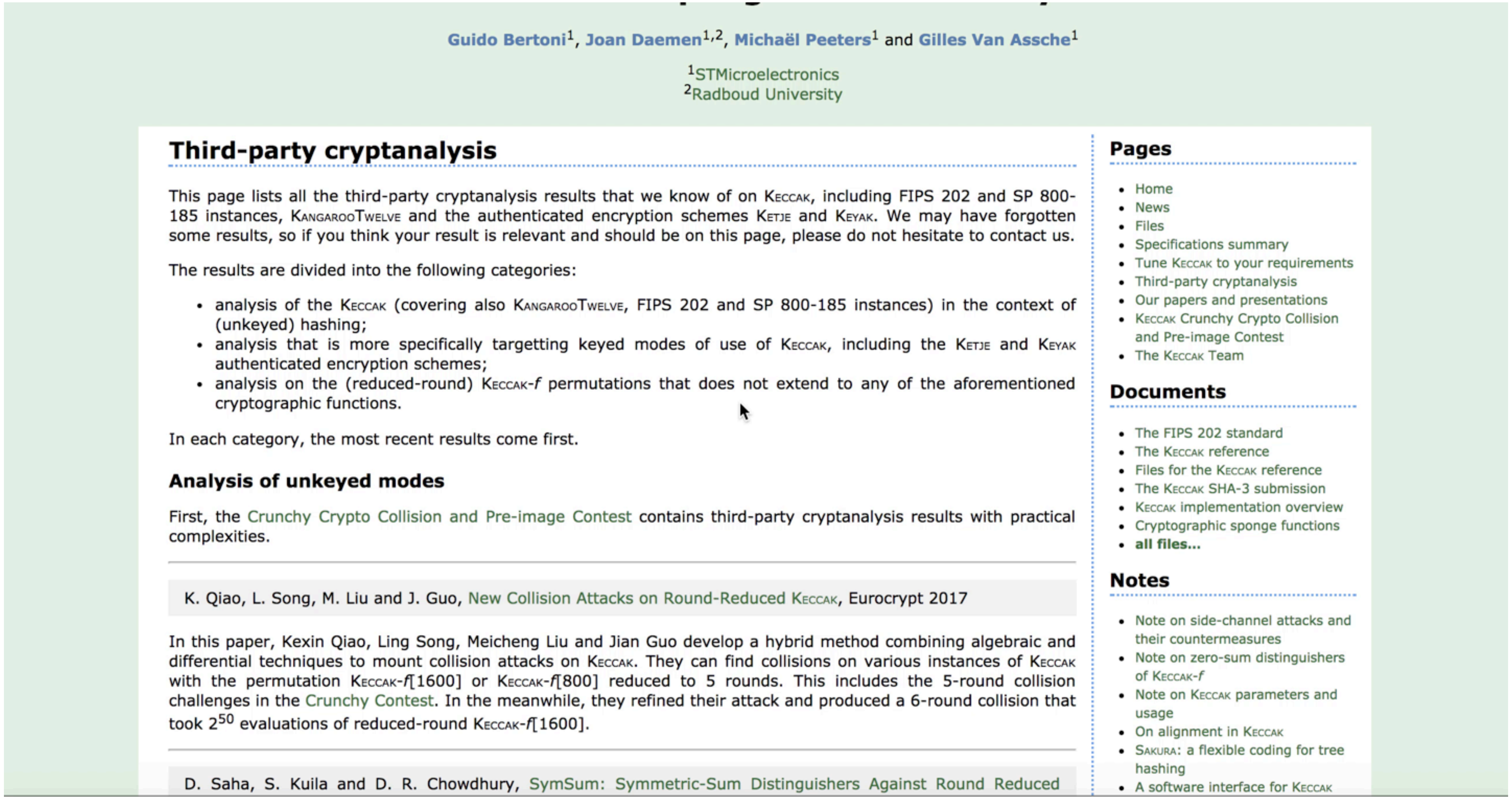


Sponge Construction



Sponge Construction





Guido Bertoni[1], Joan Daemen[1,2], Michaël Peeters[1] and Gilles Van Assche[1]

[1]STMicroelectronics
[2]Radboud University

Third-party cryptanalysis

This page lists all the third-party cryptanalysis results that we know of on Keccak, including FIPS 202 and SP 800-185 instances, KangarooTwelve and the authenticated encryption schemes Ketje and Keyak. We may have forgotten some results, so if you think your result is relevant and should be on this page, please do not hesitate to contact us.

The results are divided into the following categories:

- analysis of the Keccak (covering also KangarooTwelve, FIPS 202 and SP 800-185 instances) in the context of (unkeyed) hashing;
- analysis that is more specifically targetting keyed modes of use of Keccak, including the Ketje and Keyak authenticated encryption schemes;
- analysis on the (reduced-round) Keccak-*f* permutations that does not extend to any of the aforementioned cryptographic functions.

In each category, the most recent results come first.

Analysis of unkeyed modes

First, the Crunchy Crypto Collision and Pre-image Contest contains third-party cryptanalysis results with practical complexities.

K. Qiao, L. Song, M. Liu and J. Guo, New Collision Attacks on Round-Reduced Keccak, Eurocrypt 2017

In this paper, Kexin Qiao, Ling Song, Meicheng Liu and Jian Guo develop a hybrid method combining algebraic and differential techniques to mount collision attacks on Keccak. They can find collisions on various instances of Keccak with the permutation Keccak-*f*[1600] or Keccak-*f*[800] reduced to 5 rounds. This includes the 5-round collision challenges in the Crunchy Contest. In the meanwhile, they refined their attack and produced a 6-round collision that took 2^{50} evaluations of reduced-round Keccak-*f*[1600].

D. Saha, S. Kuila and D. R. Chowdhury, SymSum: Symmetric-Sum Distinguishers Against Round Reduced

Pages

- Home
- News
- Files
- Specifications summary
- Tune Keccak to your requirements
- Third-party cryptanalysis
- Our papers and presentations
- Keccak Crunchy Crypto Collision and Pre-image Contest
- The Keccak Team

Documents

- The FIPS 202 standard
- The Keccak reference
- Files for the Keccak reference
- The Keccak SHA-3 submission
- Keccak implementation overview
- Cryptographic sponge functions
- **all files...**

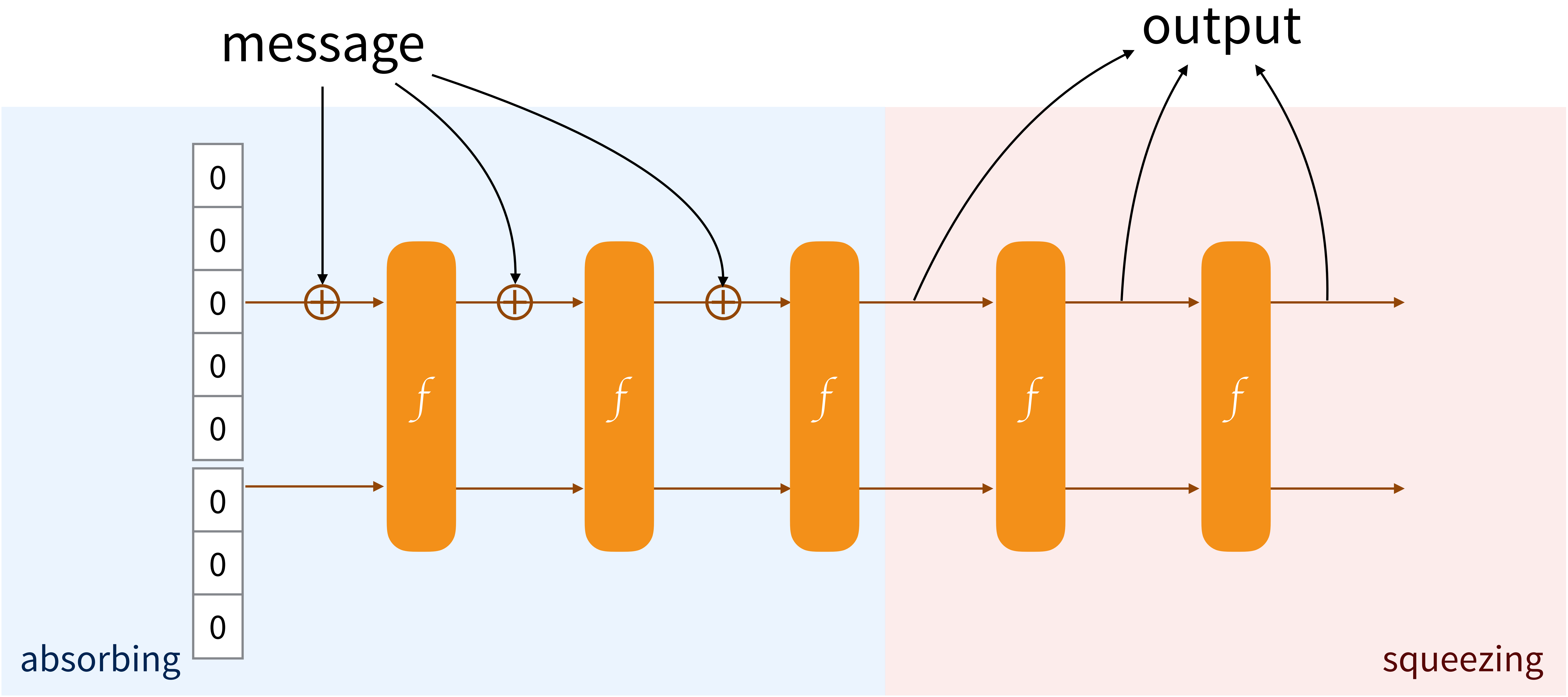
Notes

- Note on side-channel attacks and their countermeasures
- Note on zero-sum distinguishers of Keccak-*f*
- Note on Keccak parameters and usage
- On alignment in Keccak
- Sakura: a flexible coding for tree hashing
- A software interface for Keccak

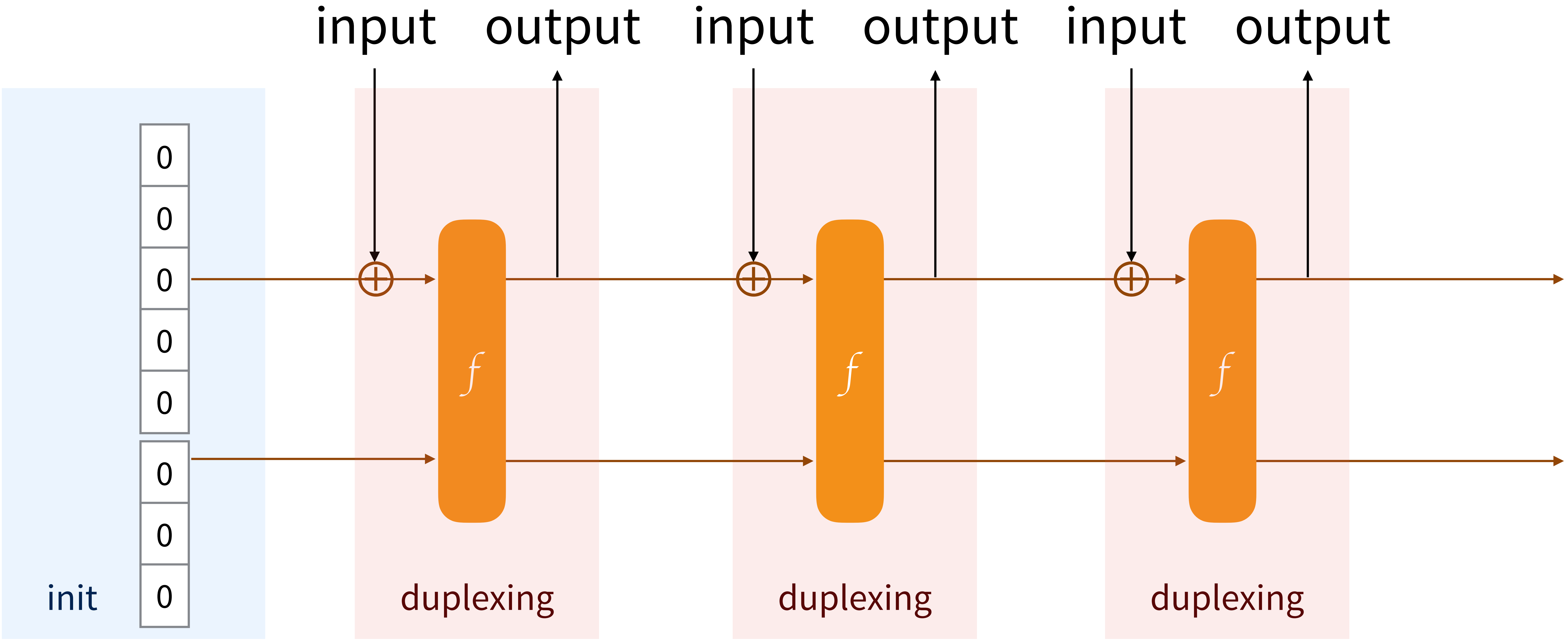
Part II: Strobe

From SHA-3 to protocols

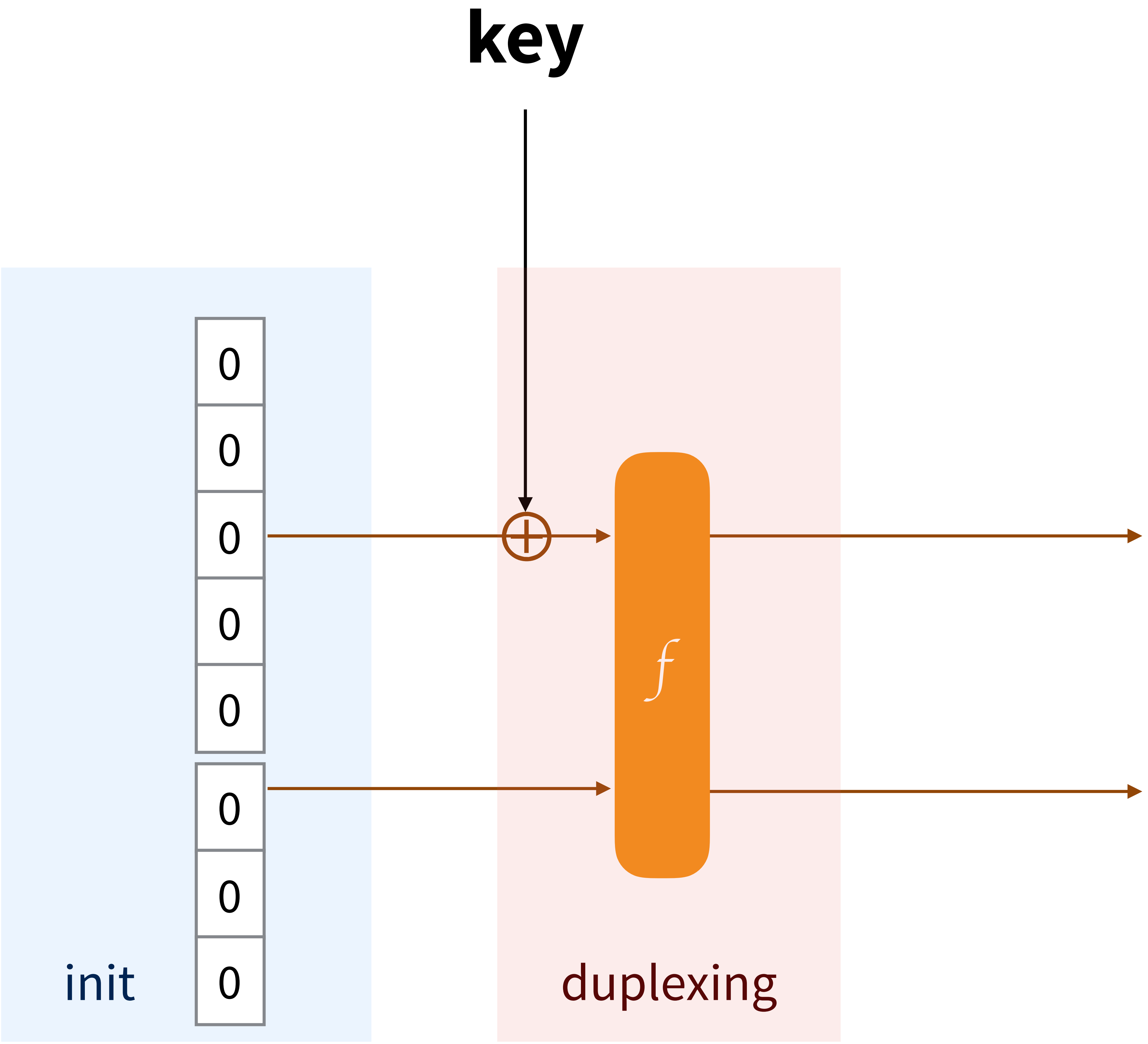
Sponge Construction



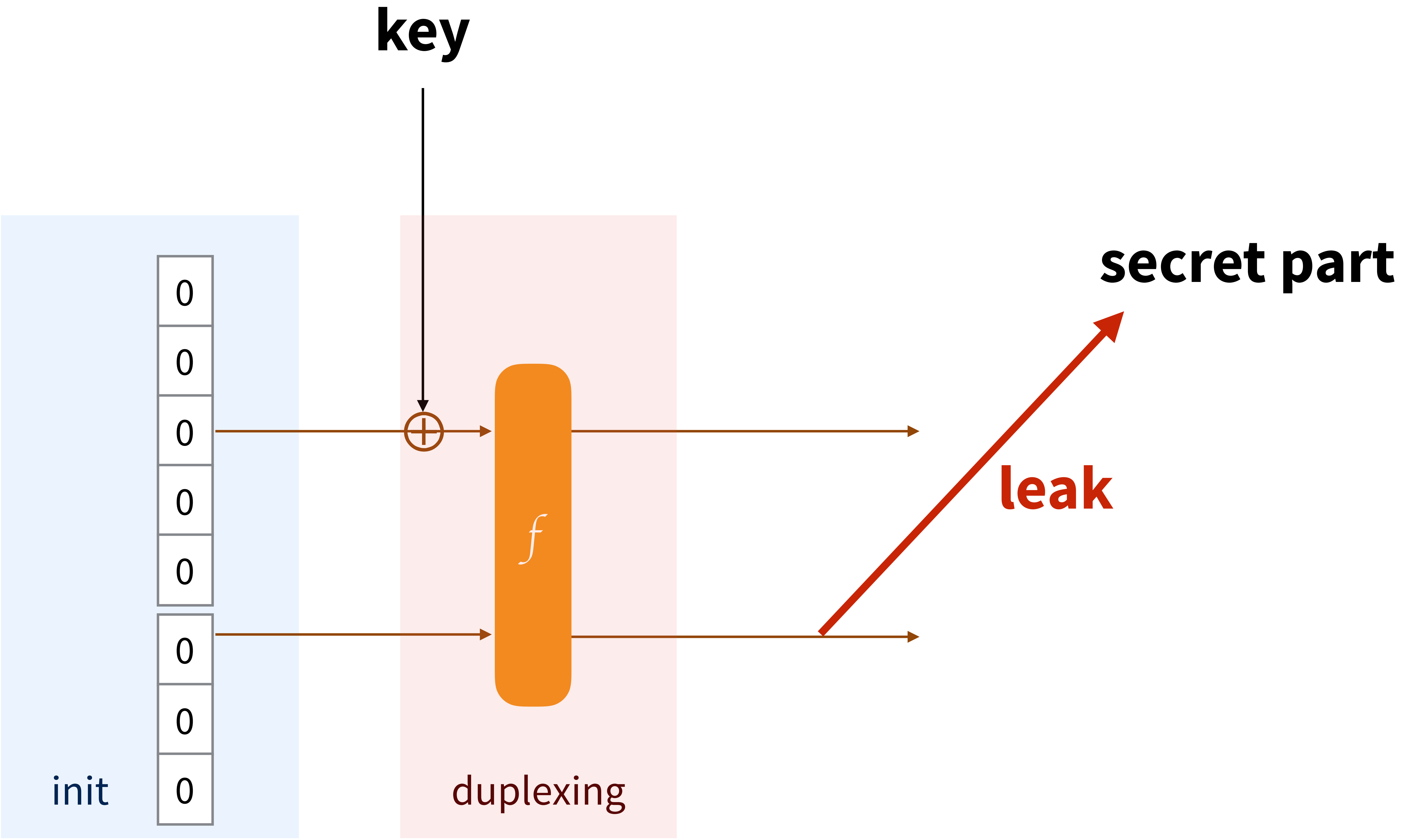
Duplex Construction



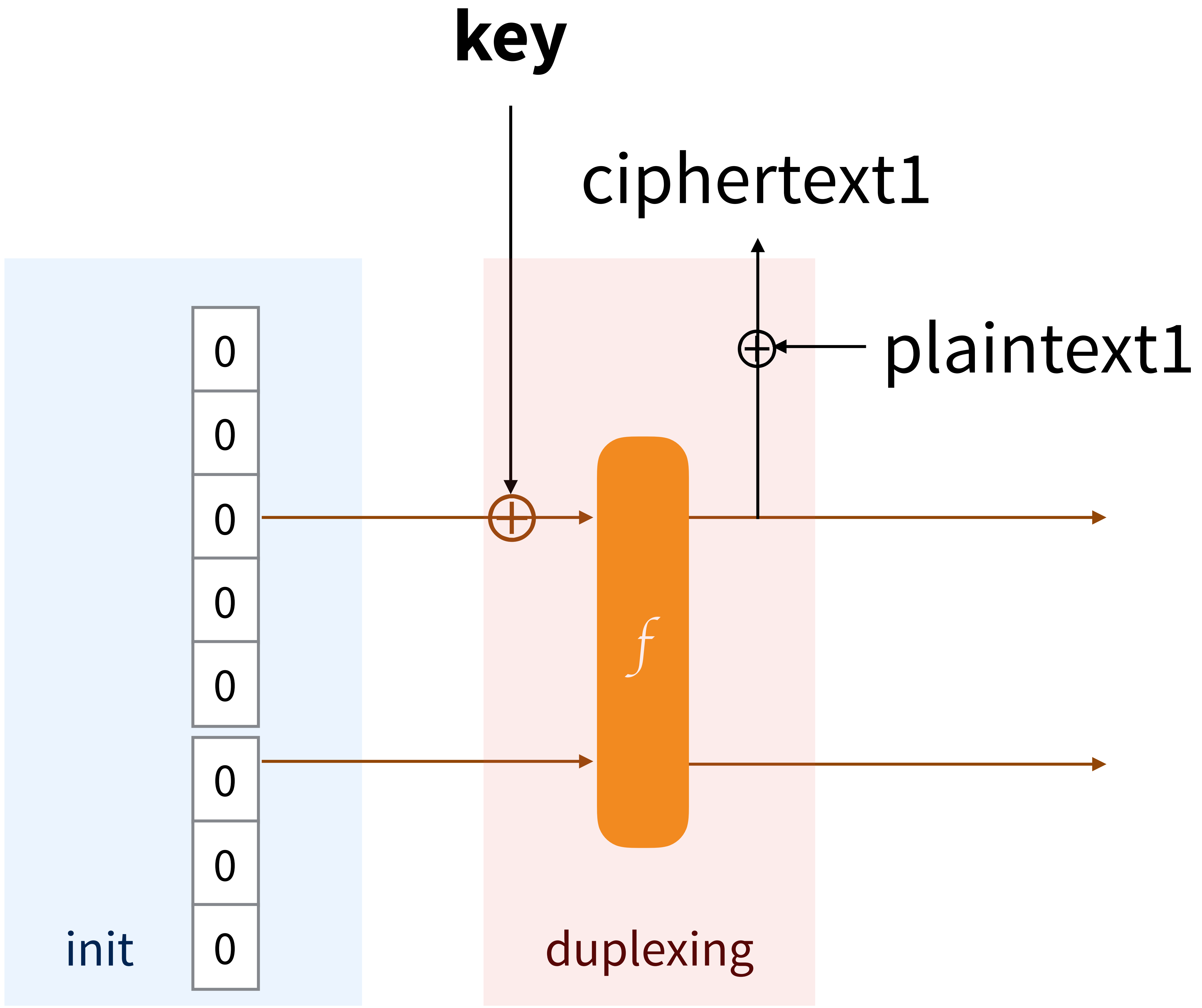
Keyed-mode



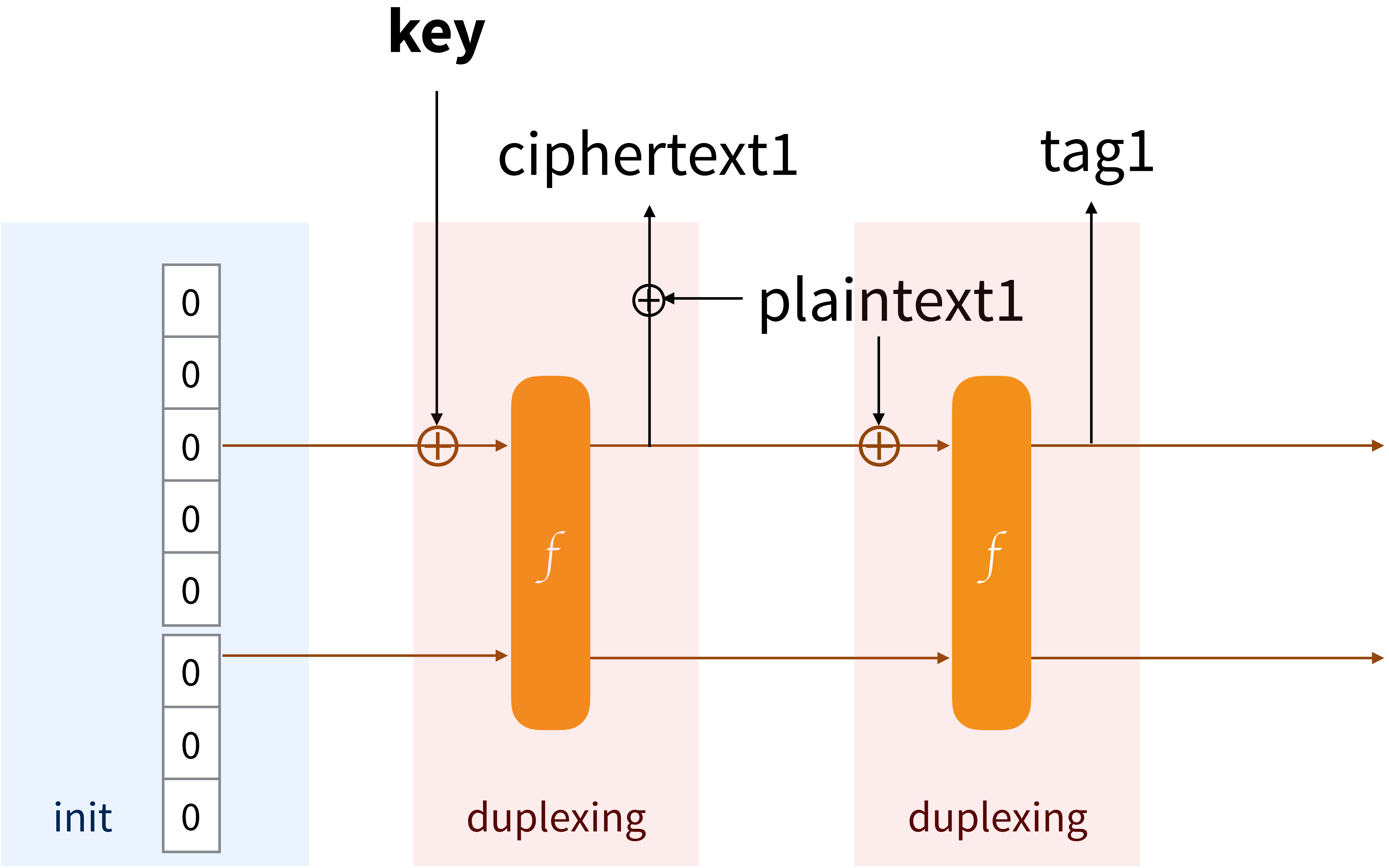
Keyed-mode



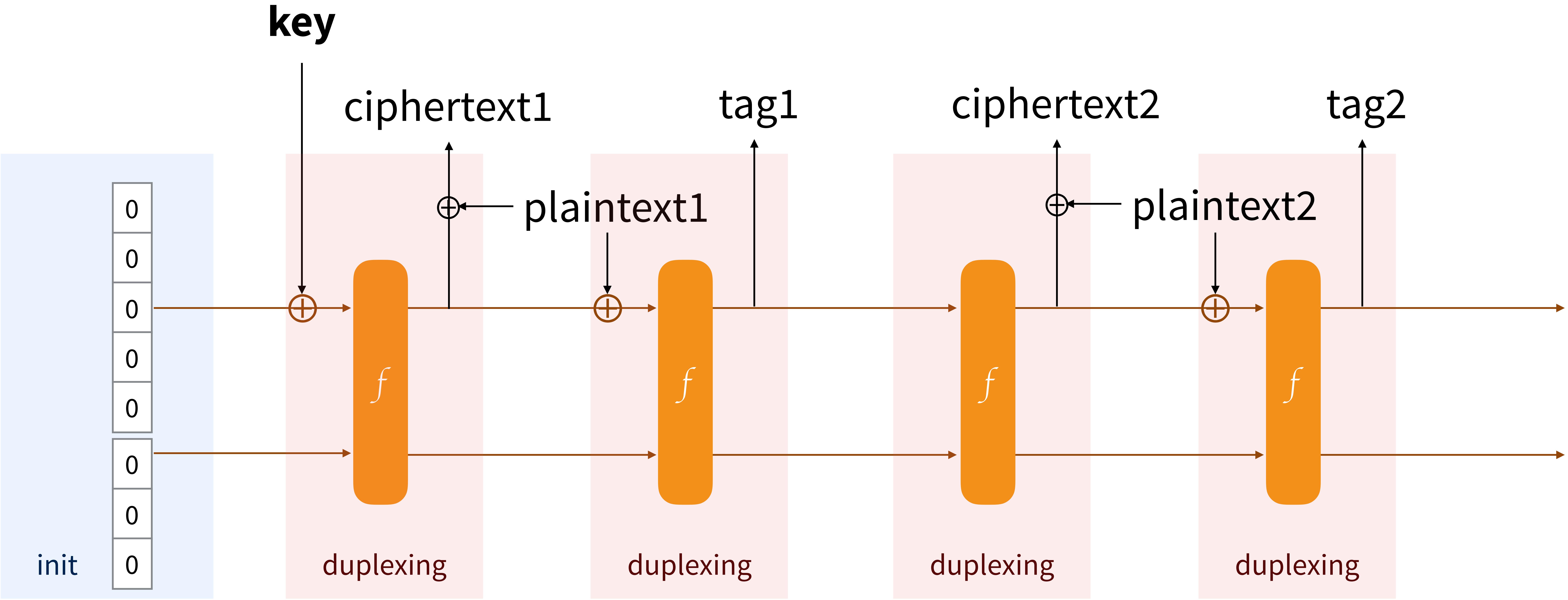
Encryption



Authenticated Encryption



Sessions

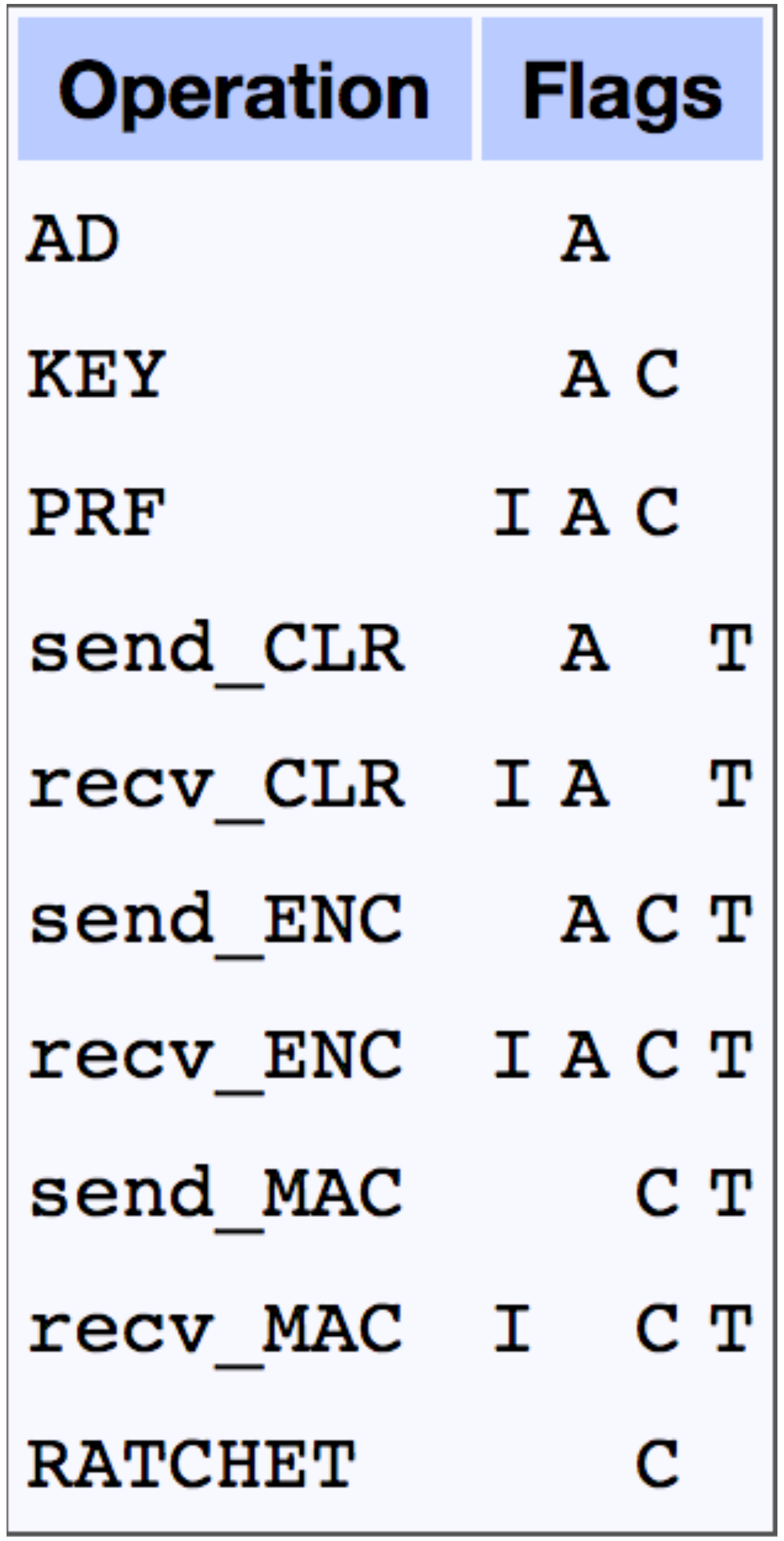

key
ciphertext1
tag1
ciphertext2
tag2
plaintext1
plaintext2
0
0
0
0
0
0
0
0
init
f
f
f
f
duplexing
duplexing
duplexing
duplexing

Strobe

```
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
}
```

Strobe

```
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
```



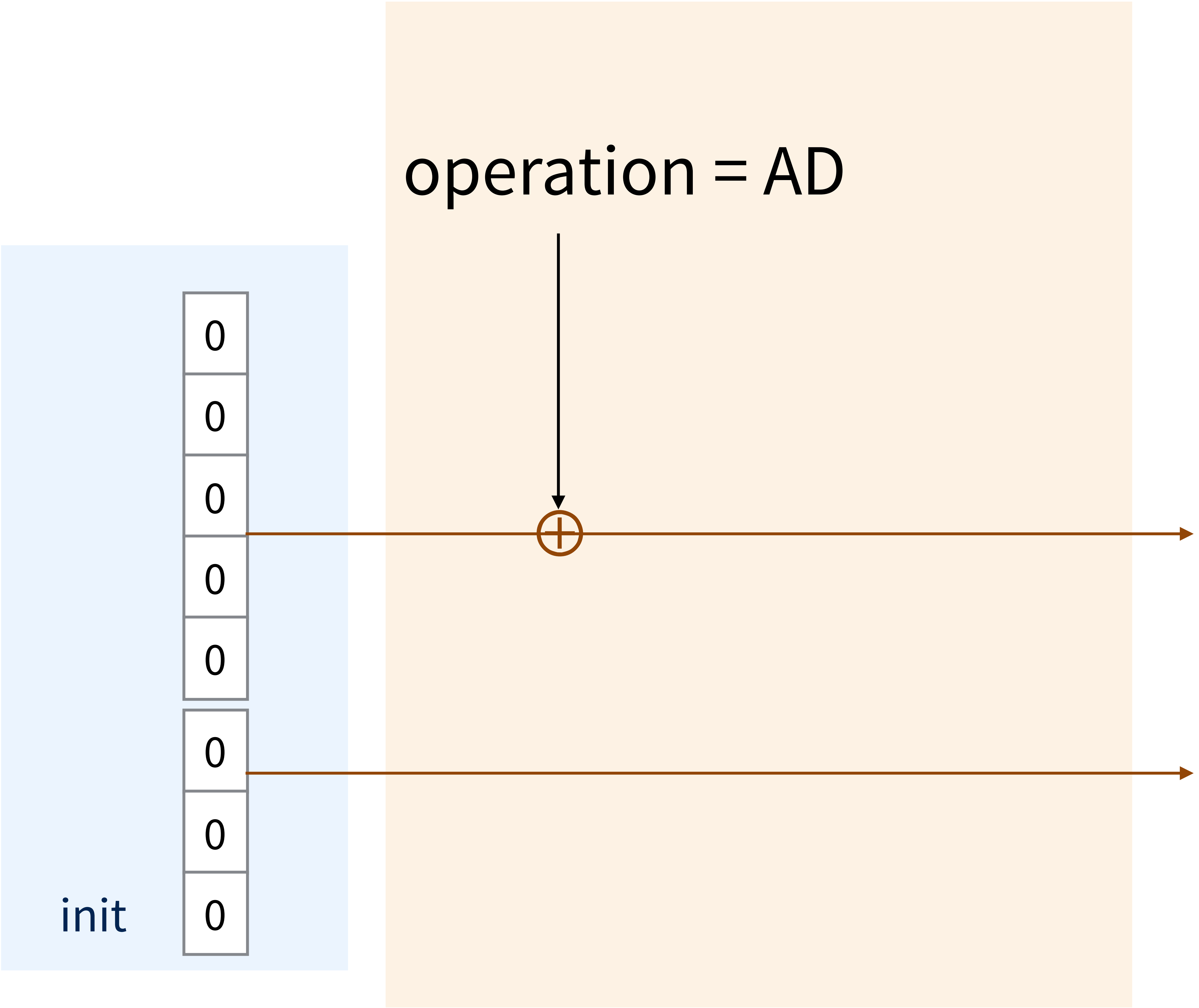
Operation	Flags			
AD		A		
KEY		A	C	
PRF	I	A	C	
send_CLR		A		T
recv_CLR	I	A		T
send_ENC		A	C	T
recv_ENC	I	A	C	T
send_MAC			C	T
recv_MAC	I		C	T
RATCHET			C	

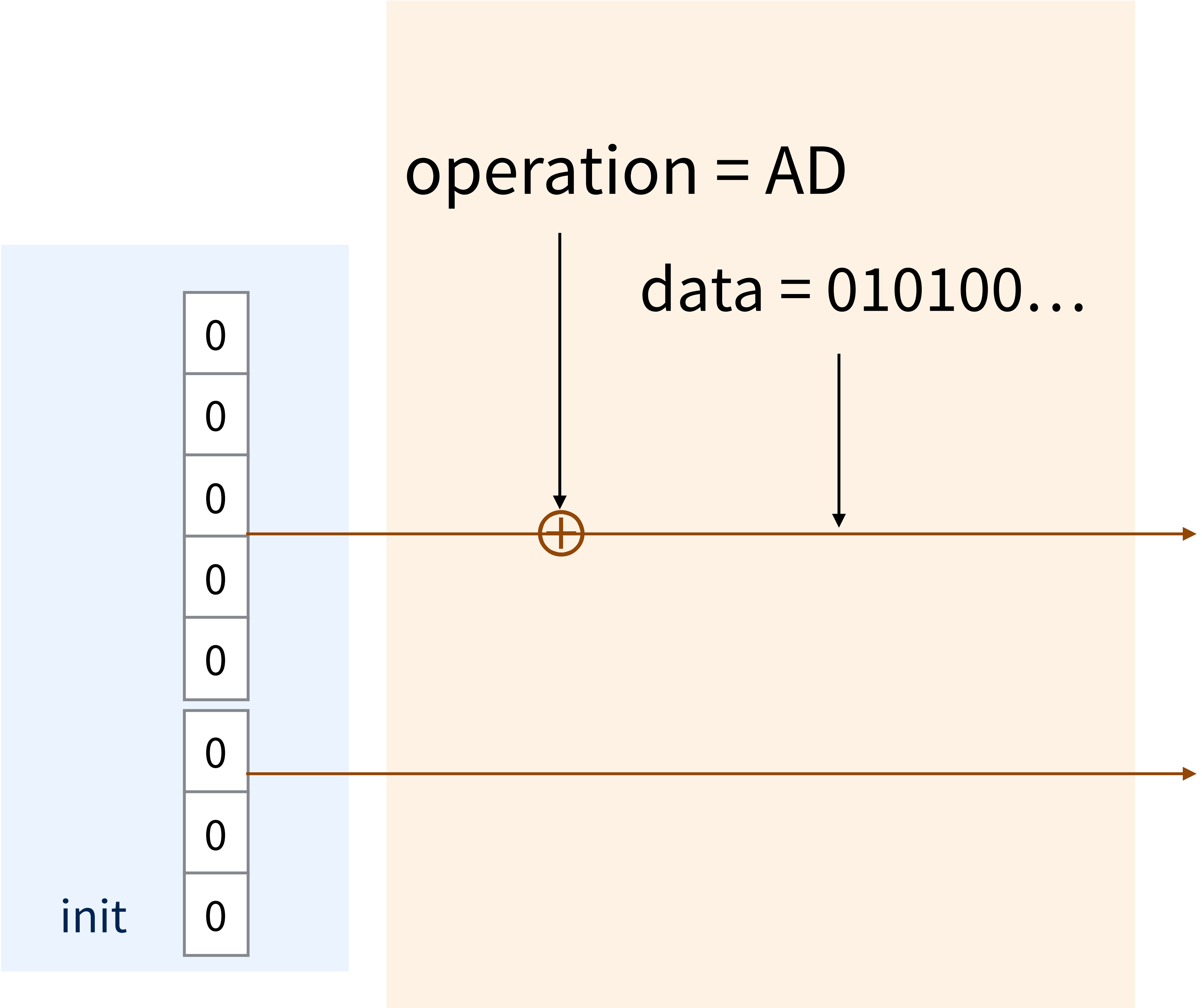
Hash Function

```
myHash = Strobe_init(“hash”)
myHash.AD(“something to be hashed”)
hash = myHash.PRF(outputLen=16)
```

Key Derivation Function

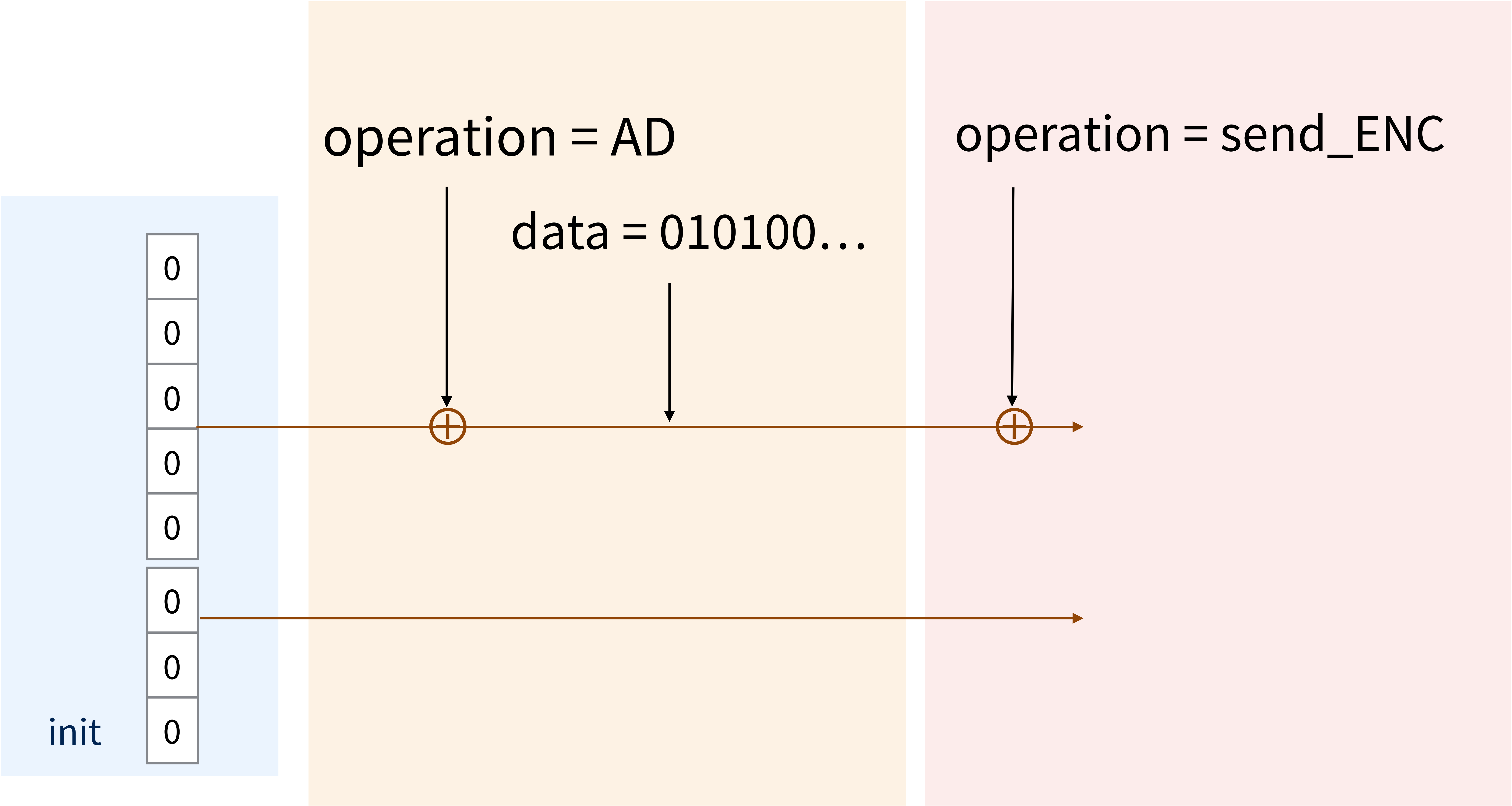
```
KDF = Strobe_init(“deriving keys”)
KDF.AD(keyExchangeOutput)
keys = KDF.PRF(outputLen=32)
key1 = keys[:16]
key2 = keys[16:]
```


operation = AD
0
0
0
0
0
0
0
0
init



operation = AD

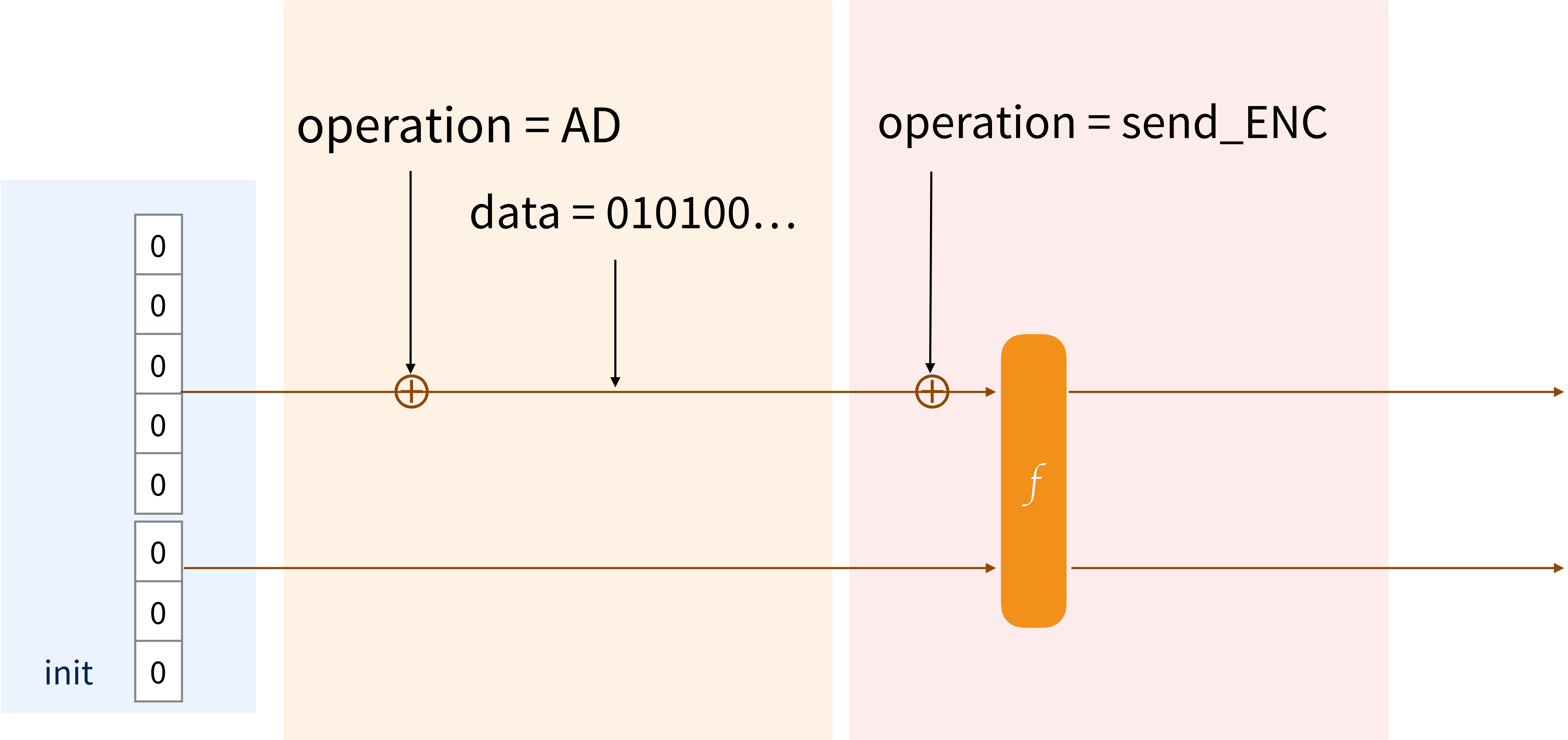
data = 010100…

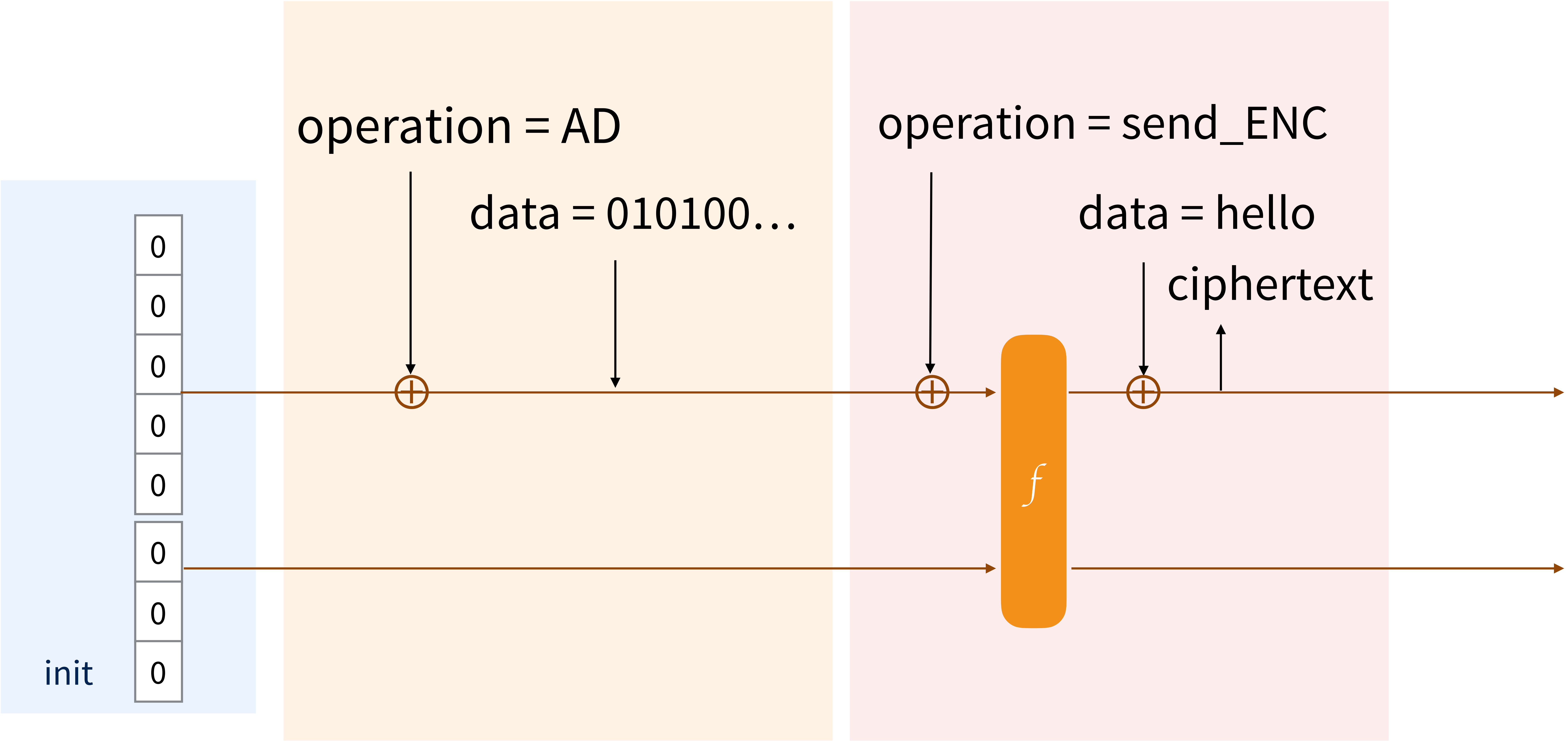


operation = AD

data = 010100...

operation = send_ENC


operation = AD
operation = send_ENC
data = 010100…
0
0
0
0
0
0
0
0
init
f


operation = AD
data = 010100…
operation = send_ENC
data = hello
ciphertext
0
0
0
0
0
0
0
0
init
f

operation = AD
data = 010100…
operation = send_ENC
data = hello
ciphertext
operation = send_MAC
tag
len = 16
0
0
0
0
0
0
0
0
init
f
f

send_AEAD

operation = AD

data = 010100…

operation = send_ENC

data = hello

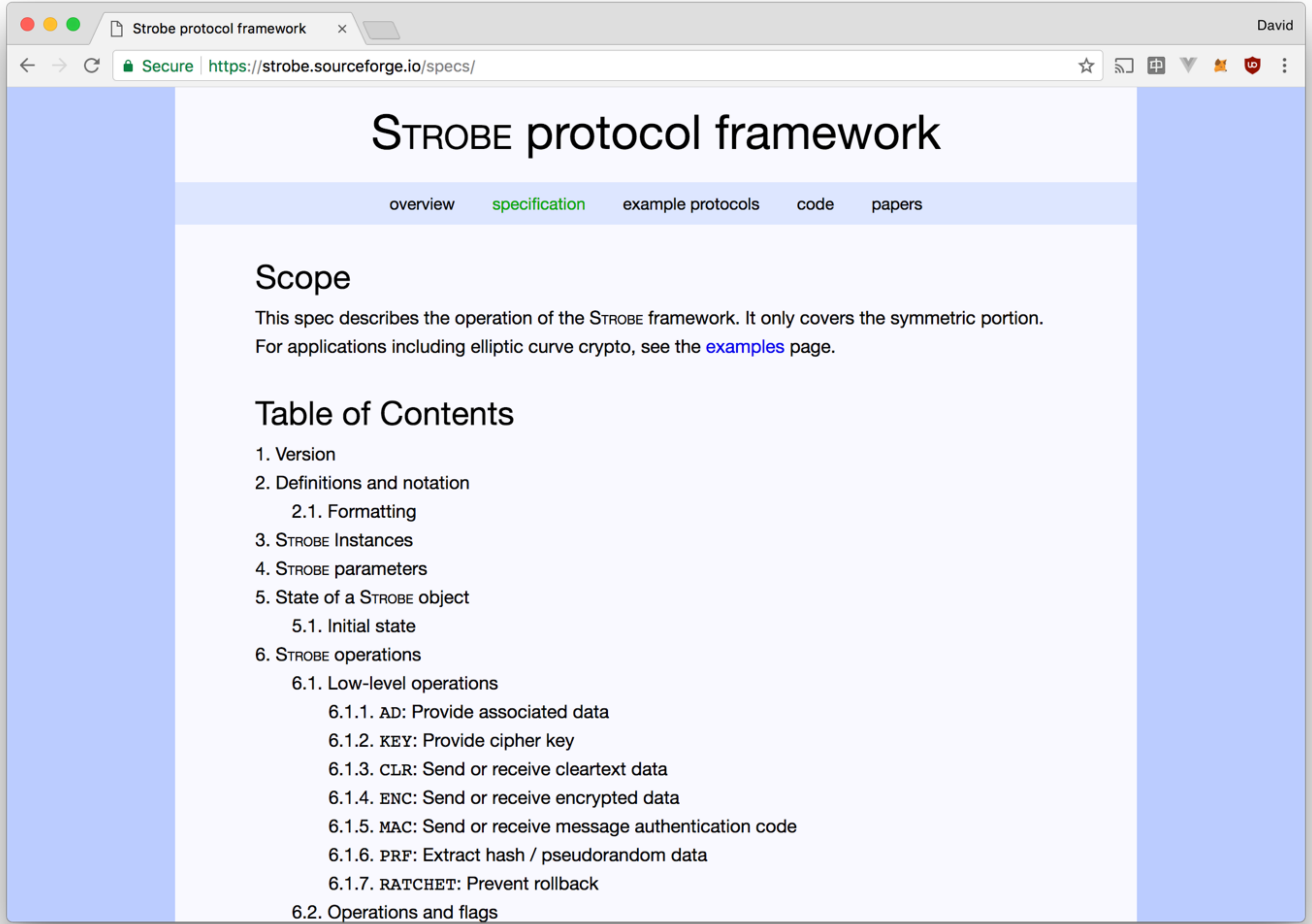
ciphertext

operation = send_MAC

tag

len = 16

0
0
0
0
0
0
0
0
init
f
f



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

strobe.sourceforge.io

Strobe

- **flexible** framework to support a large number of protocols
- large **symmetric cryptography** library
- fits into **tiny** IoT devices (**less than 1000 lines of code)**
- relies on strong **SHA-3** standard

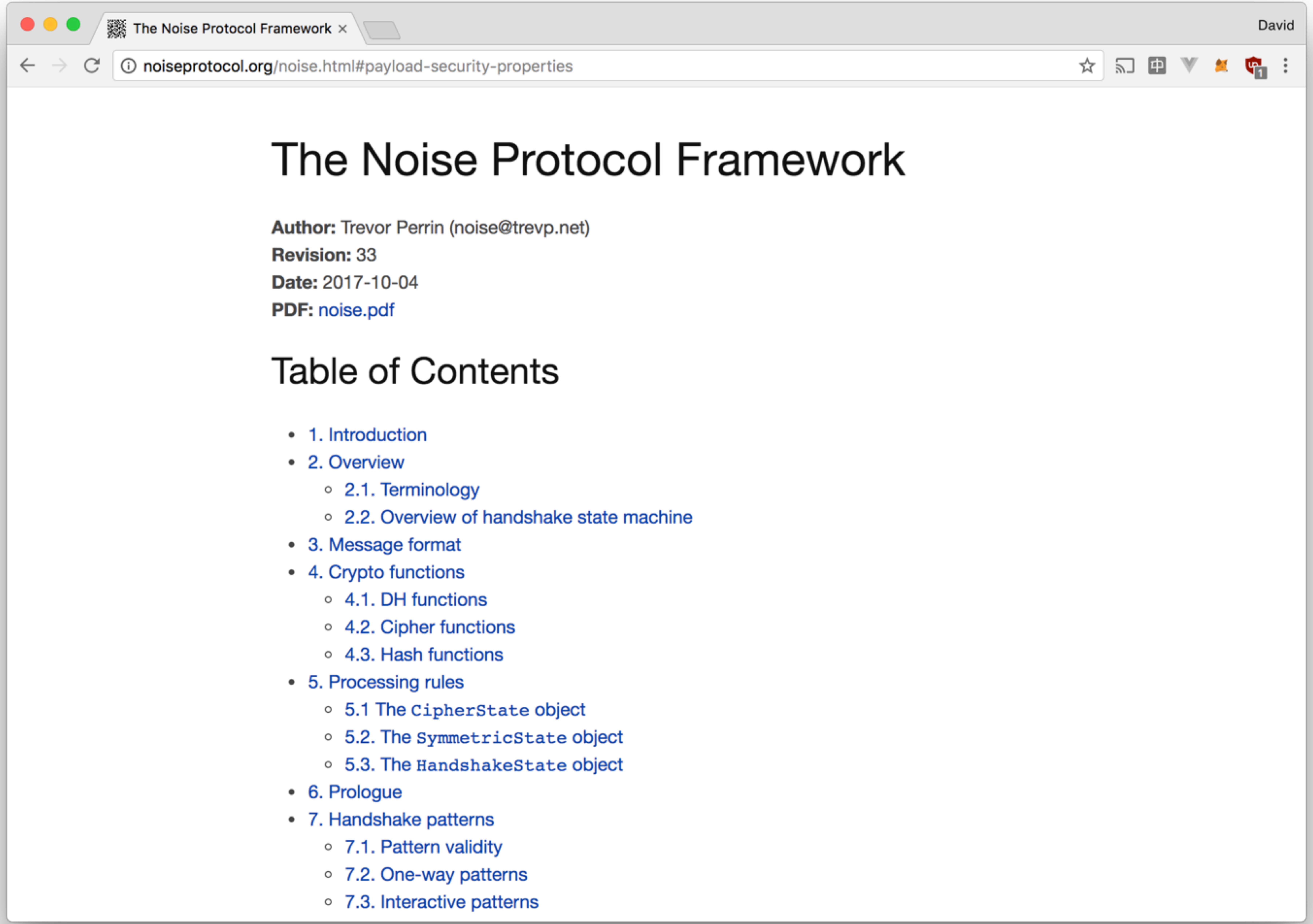
Part III: Noise

A modern protocol framework

TLS

- TLS is the **de facto standard** for securing communications
- **complex** specification
  - supported by other complex specs (asn.1, x509, extensions, …)
- design carrying a lot of **legacy** decisions
- **huge** and **scary** libraries
  - **cumbersome** configuration…
- often **dangerously** re-implemented (custom implementations)
  - or re-invented (proprietary protocols)

Complexity is the enemy of security


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

www.noiseprotocol.org

The Noise Protocol Framework

- it’s a protocol **framework** to achieve something like TLS
- “easy” to **understand**, to **analyze**, to **extend** and to **implement**
- no need for a **PKI**
- many handshakes to choose from (**flexible**)
- it’s **straight forward** to implement (<2k LOC)
  - and **small** (18kb for Arduino by Virgil Security)
- there are already **libraries** that you can leverage
- **minimal** (or zero) configuration
- if you have a good excuse not to use TLS, **Noise is the answer**

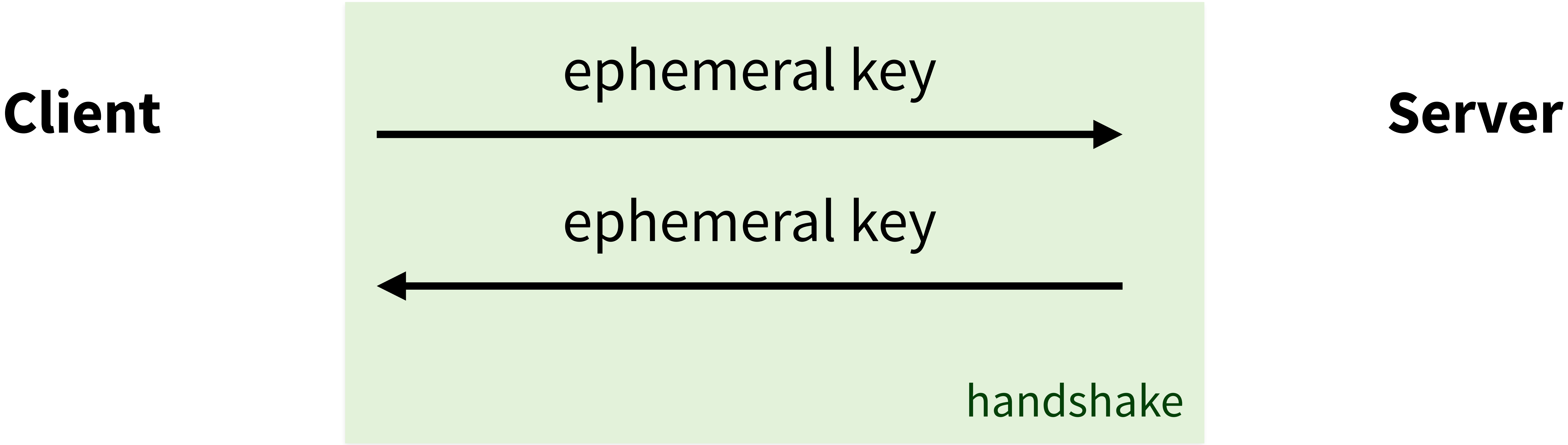
The **crypto** functions

- **DH**
  - 25519
  - 448

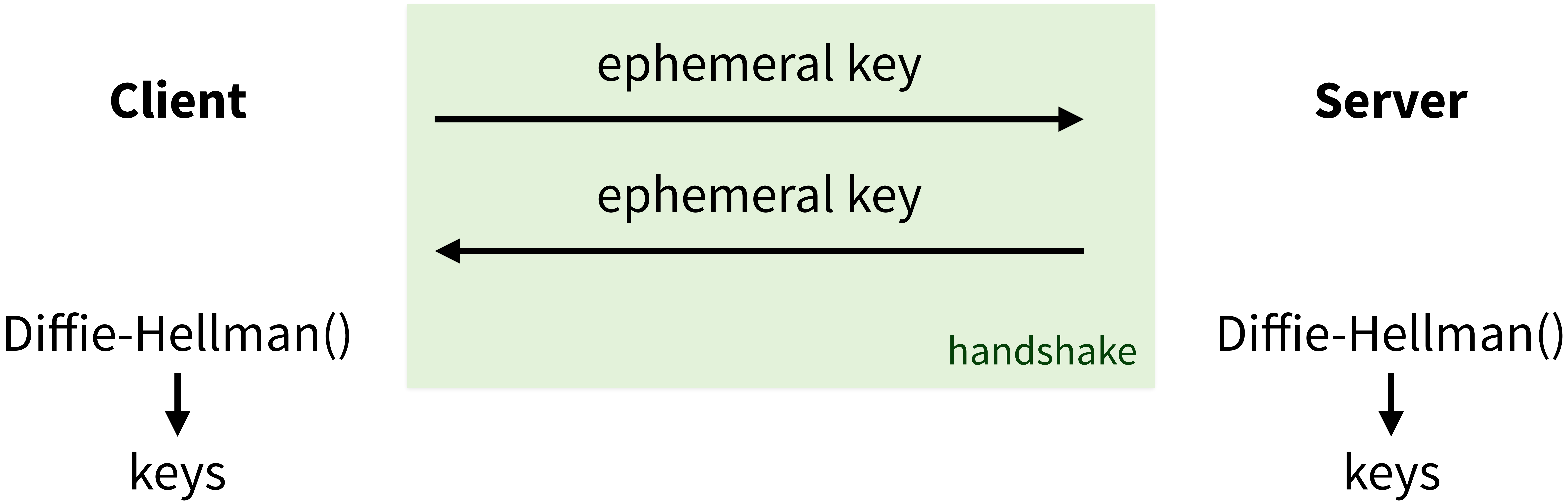
- **AEAD**
  - Chacha20-Poly1305
  - AES-GCM

- **HASH**
  - SHA-256
  - SHA-512
  - BLAKE2s
  - BLAKE2b

A **simple** state machine



A **simple** state machine



Client

Server

ephemeral key

ephemeral key

handshake

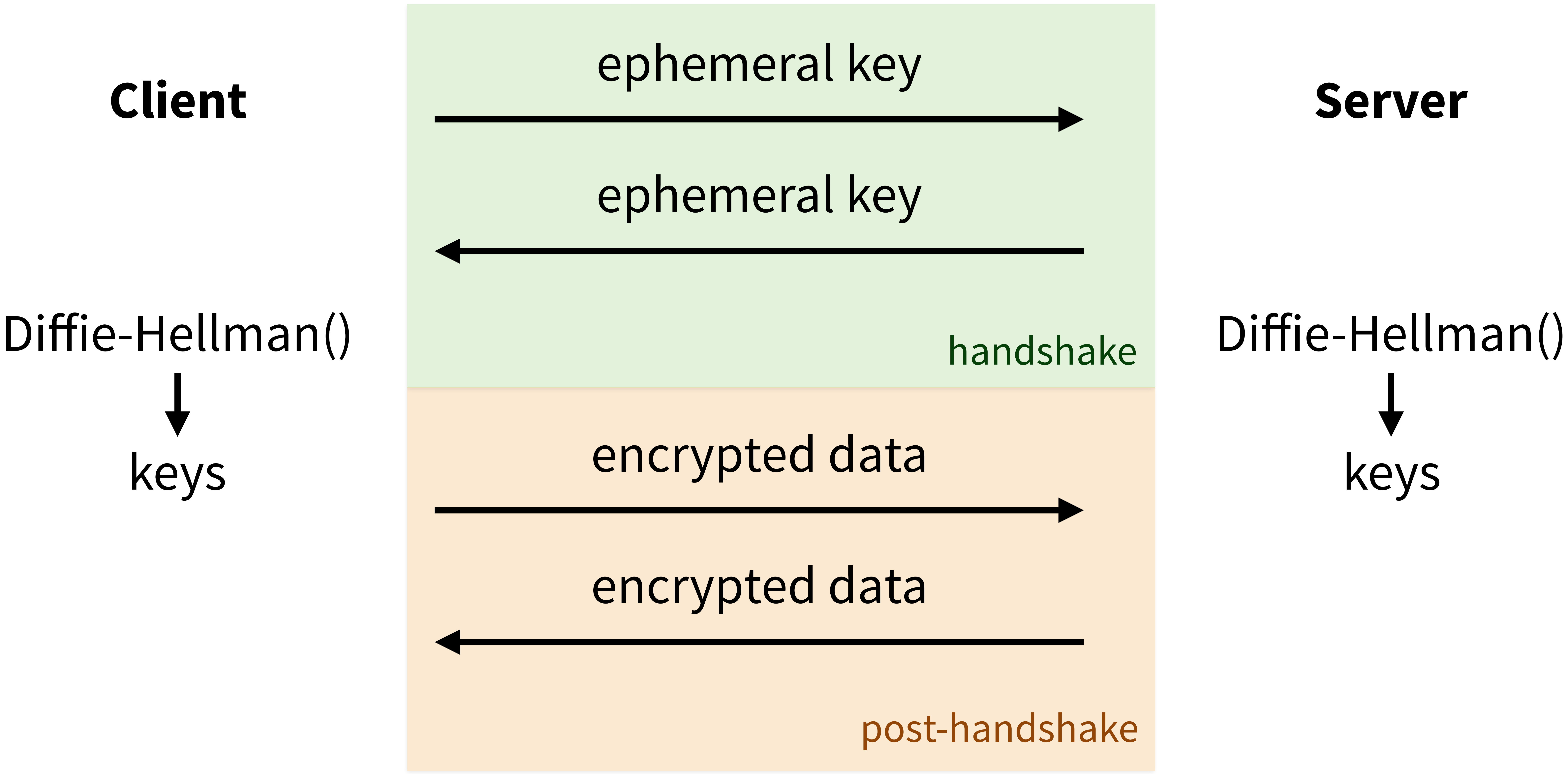
Diffie-Hellman()

keys

Diffie-Hellman()

keys

A **simple** state machine



Client

Server

ephemeral key

ephemeral key

handshake

Diffie-Hellman()

keys

Diffie-Hellman()

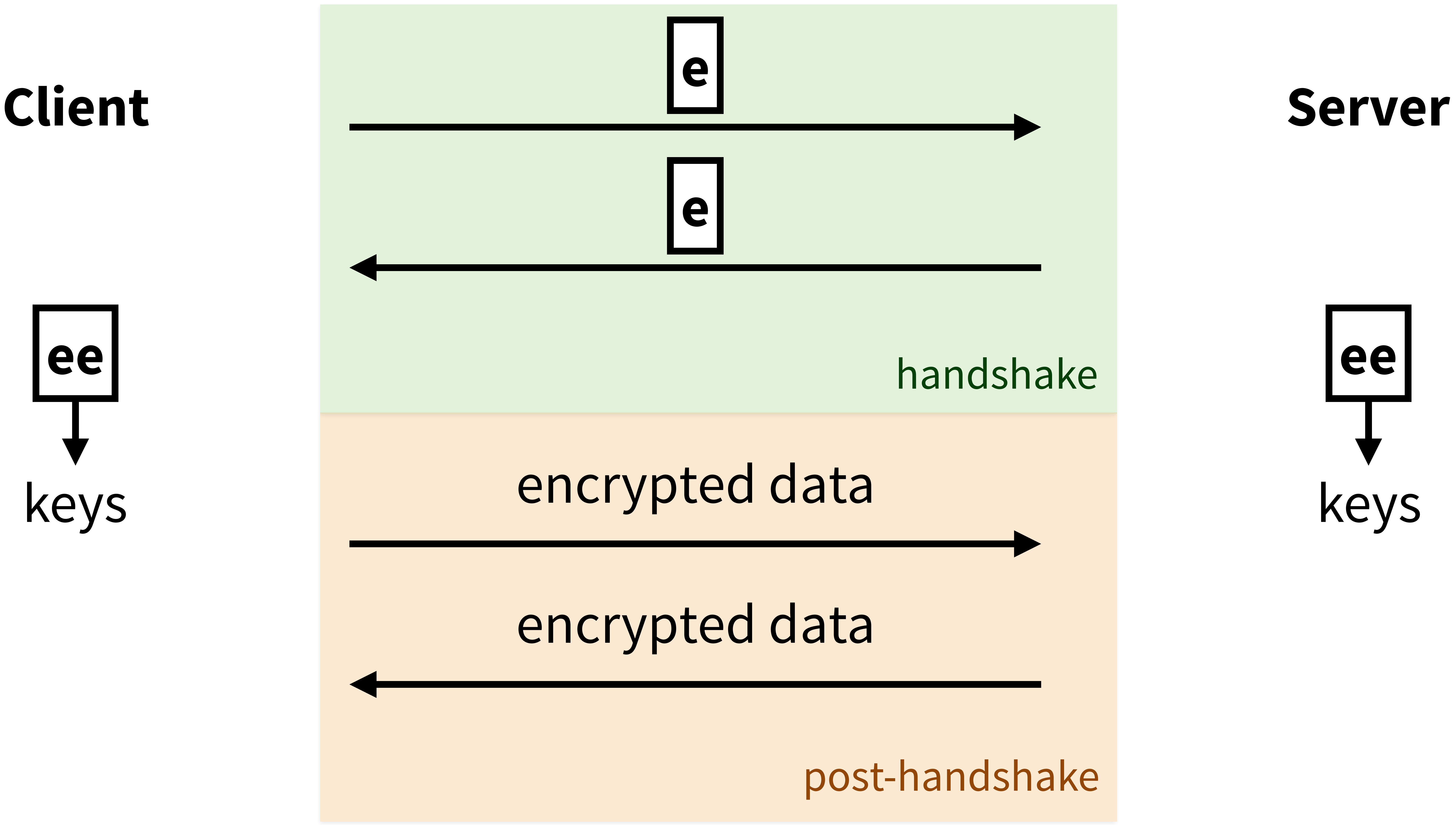
keys

encrypted data

encrypted data

post-handshake

A **simple** state machine



Handshake Patterns

→ e

← e, ee

Handshake Patterns

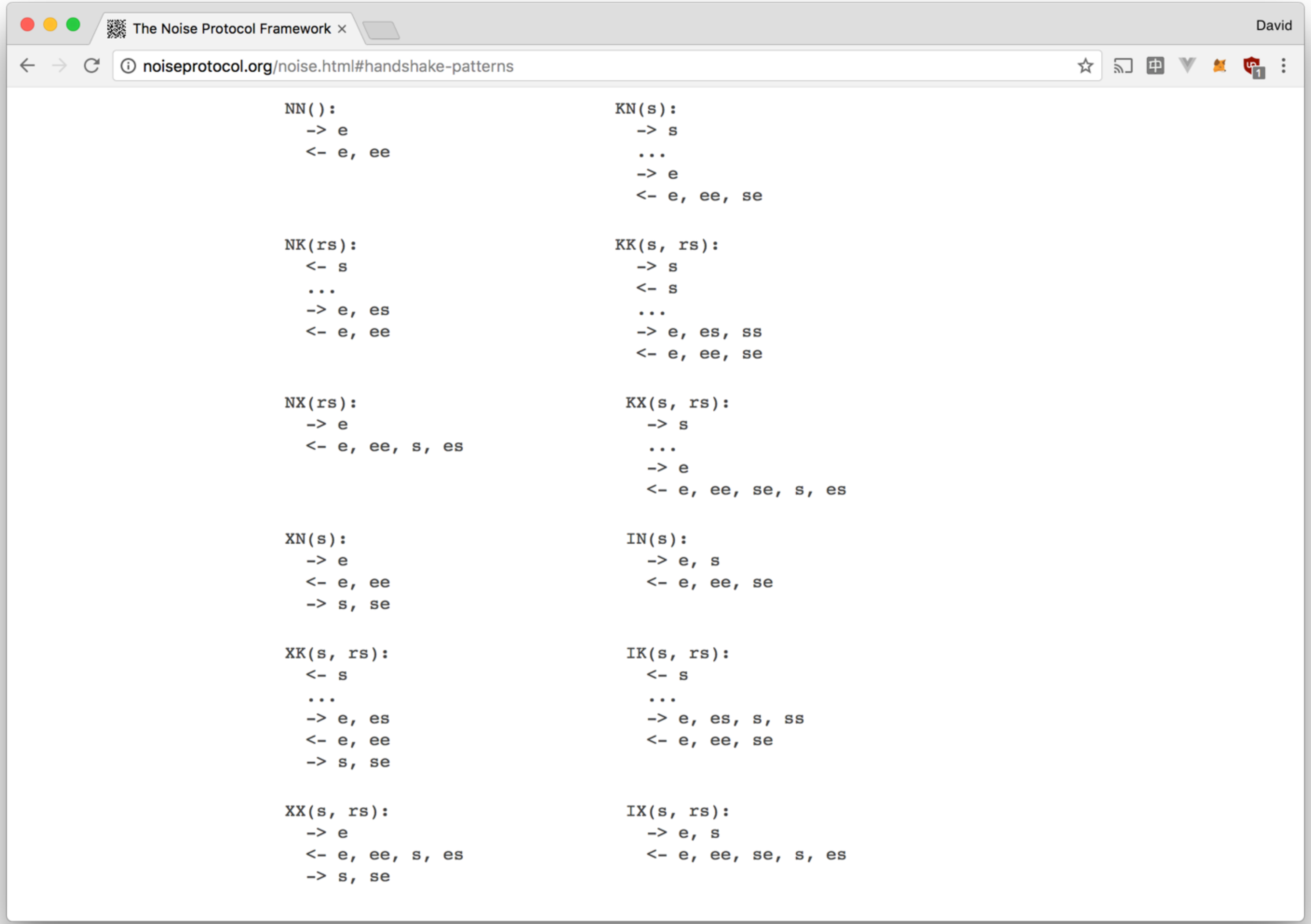
Noise_**NN**():

→ e

← e, ee

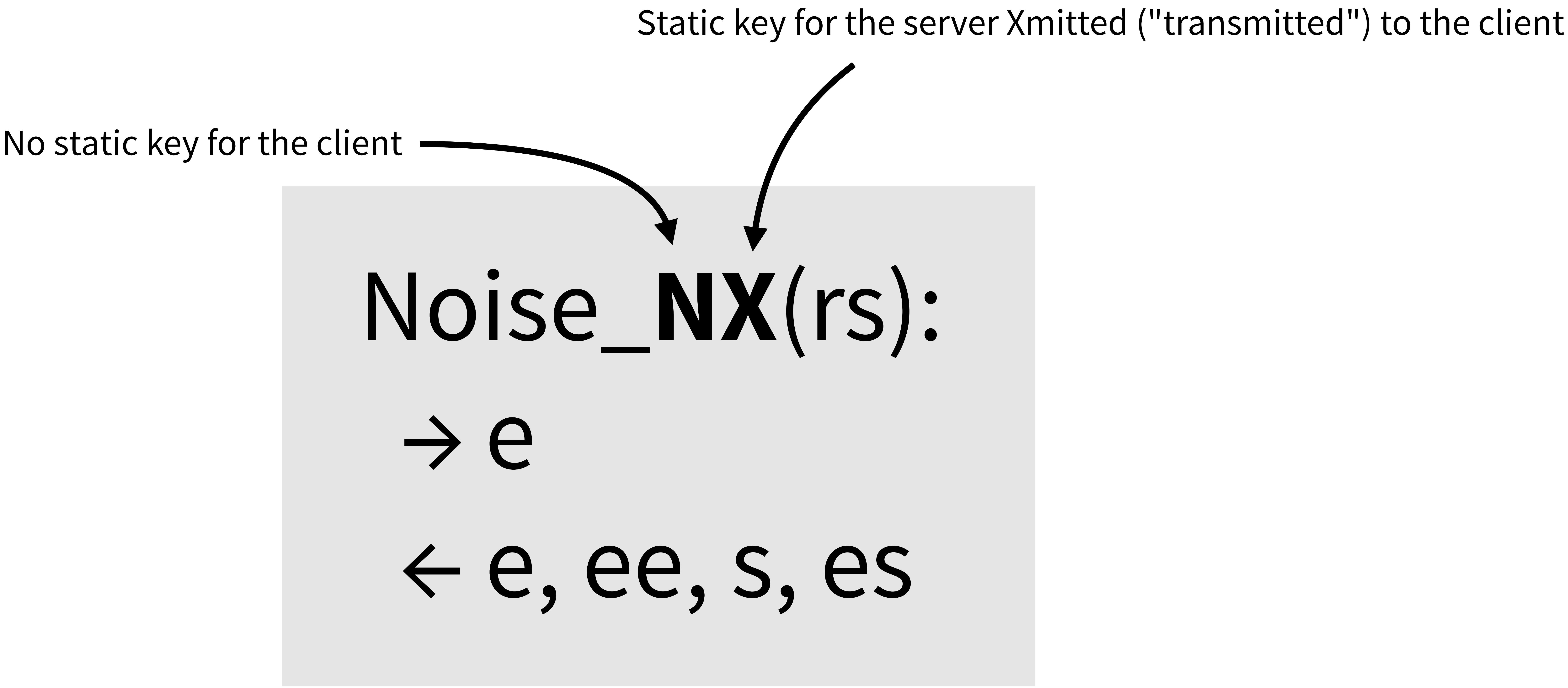
Tokens

- **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


The Noise Protocol Framework
David
noiseprotocol.org/noise.html#handshake-patterns

```
NN():                          KN(s):
  -> e                           -> s
  <- e, ee                       ...
                                 -> e
                                 <- e, ee, se

NK(rs):                        KK(s, rs):
  <- s                           -> s
  ...                            <- s
  -> e, es                       ...
  <- e, ee                       -> e, es, ss
                                 <- e, ee, se

NX(rs):                         KX(s, rs):
  -> e                            -> s
  <- e, ee, s, es                 ...
                                  -> e
                                  <- e, ee, se, s, es

XN(s):                          IN(s):
  -> e                            -> e, s
  <- e, ee                        <- e, ee, se
  -> s, se

XK(s, rs):                      IK(s, rs):
  <- s                            <- s
  ...                             ...
  -> e, es                        -> e, es, s, ss
  <- e, ee                        <- e, ee, se
  -> s, se

XX(s, rs):                      IX(s, rs):
  -> e                            -> e, s
  <- e, ee, s, es                 <- e, ee, se, s, es
  -> s, se
```

Handshake Pattern



Static key for the server Xmitted ("transmitted") to the client

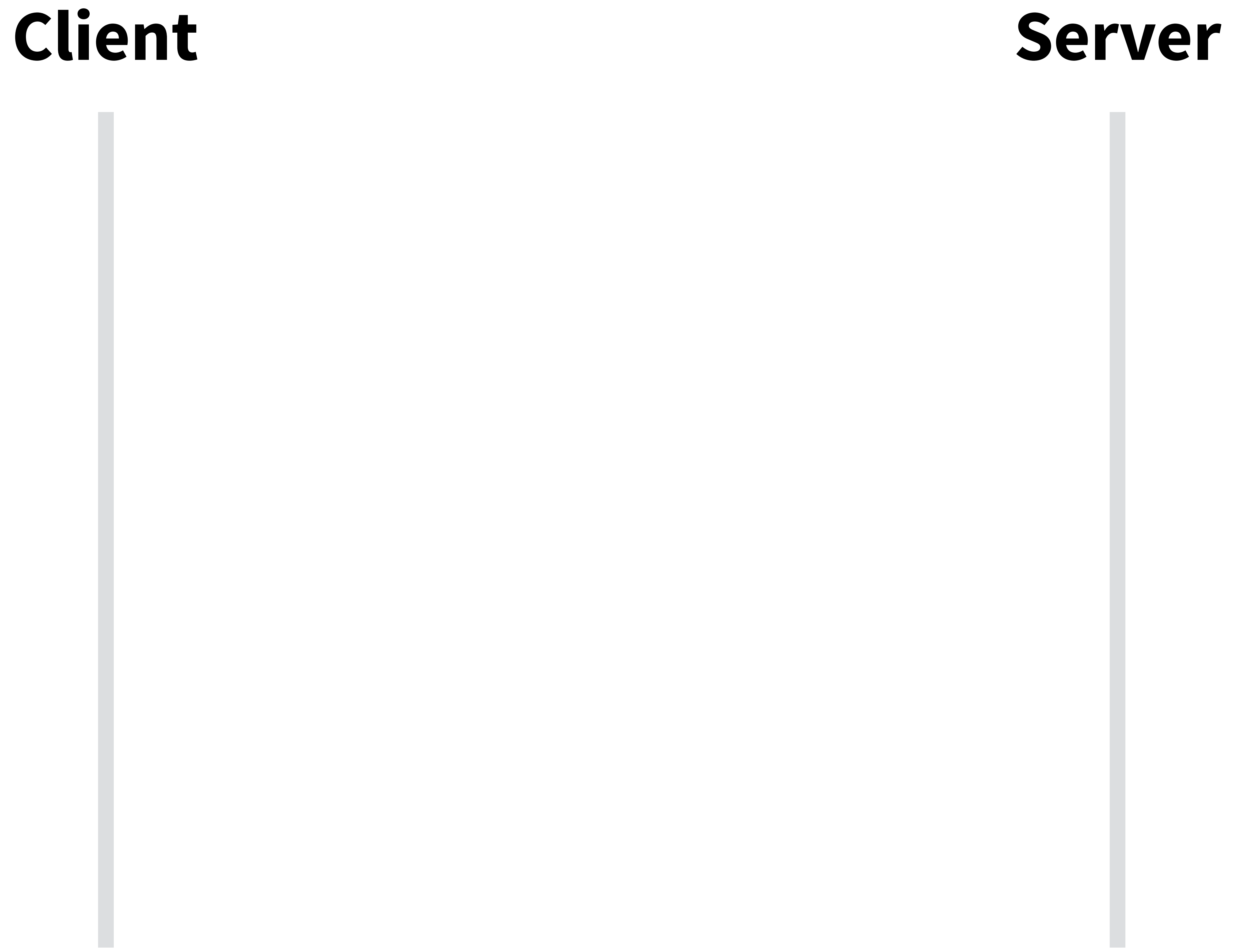
No static key for the client

```
Noise_NX(rs):
  → e
  ← e, ee, s, es
```

Noise_**NX**(rs):

→ e

← e, ee, s, es



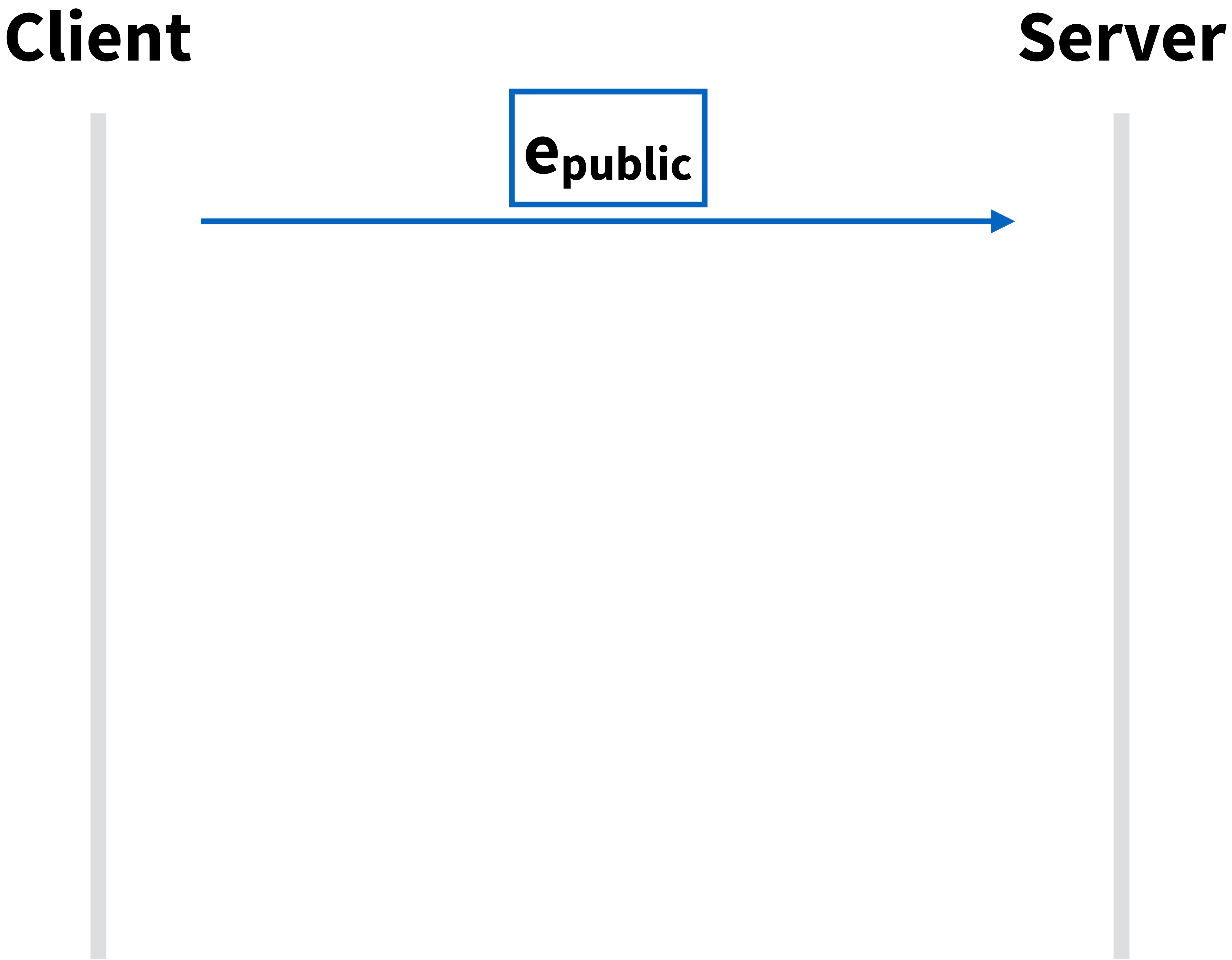
Client

Server

Noise_**NX**(rs):

→ e

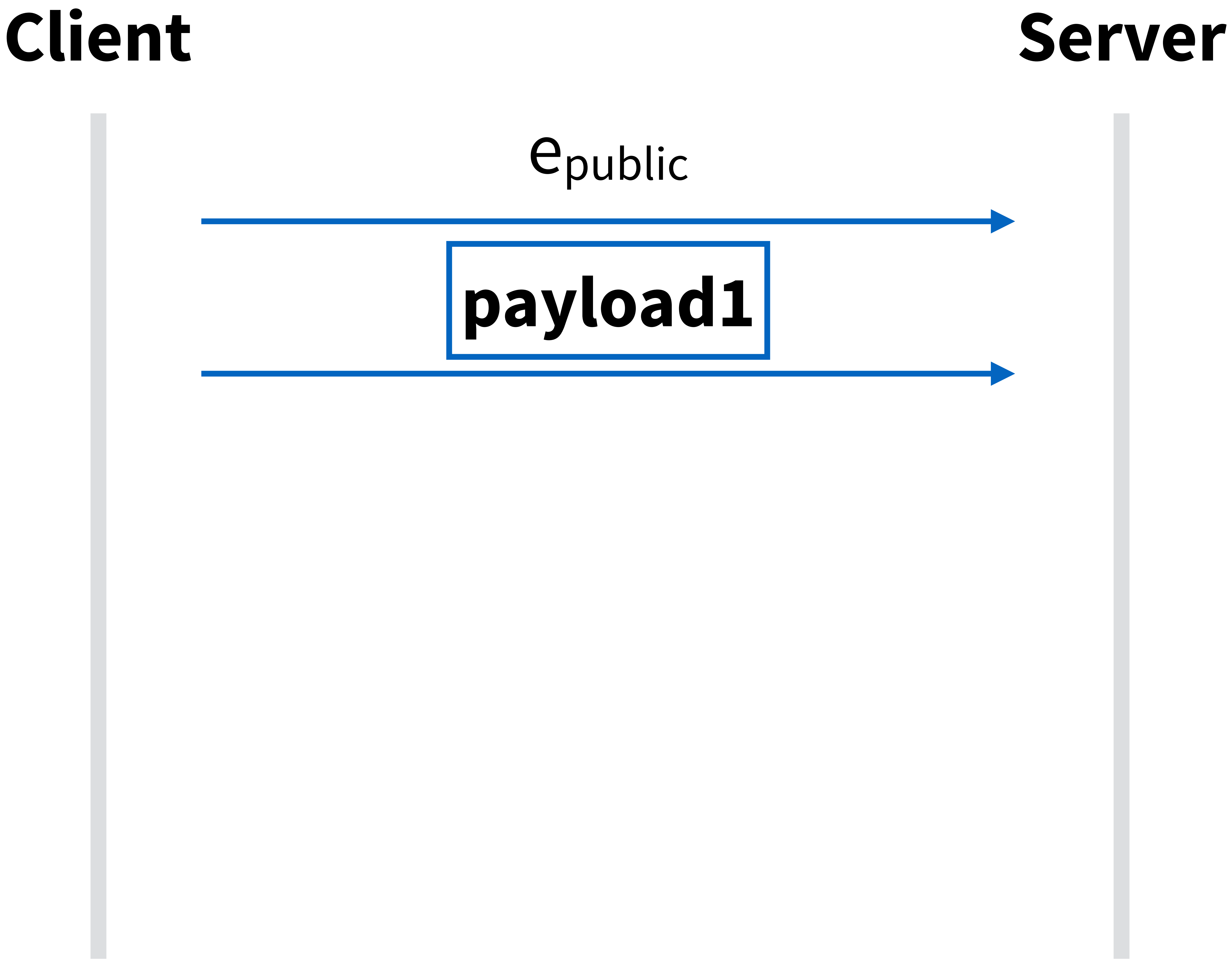
← e, ee, s, es



Noise_**NX**(rs):

→ e ●

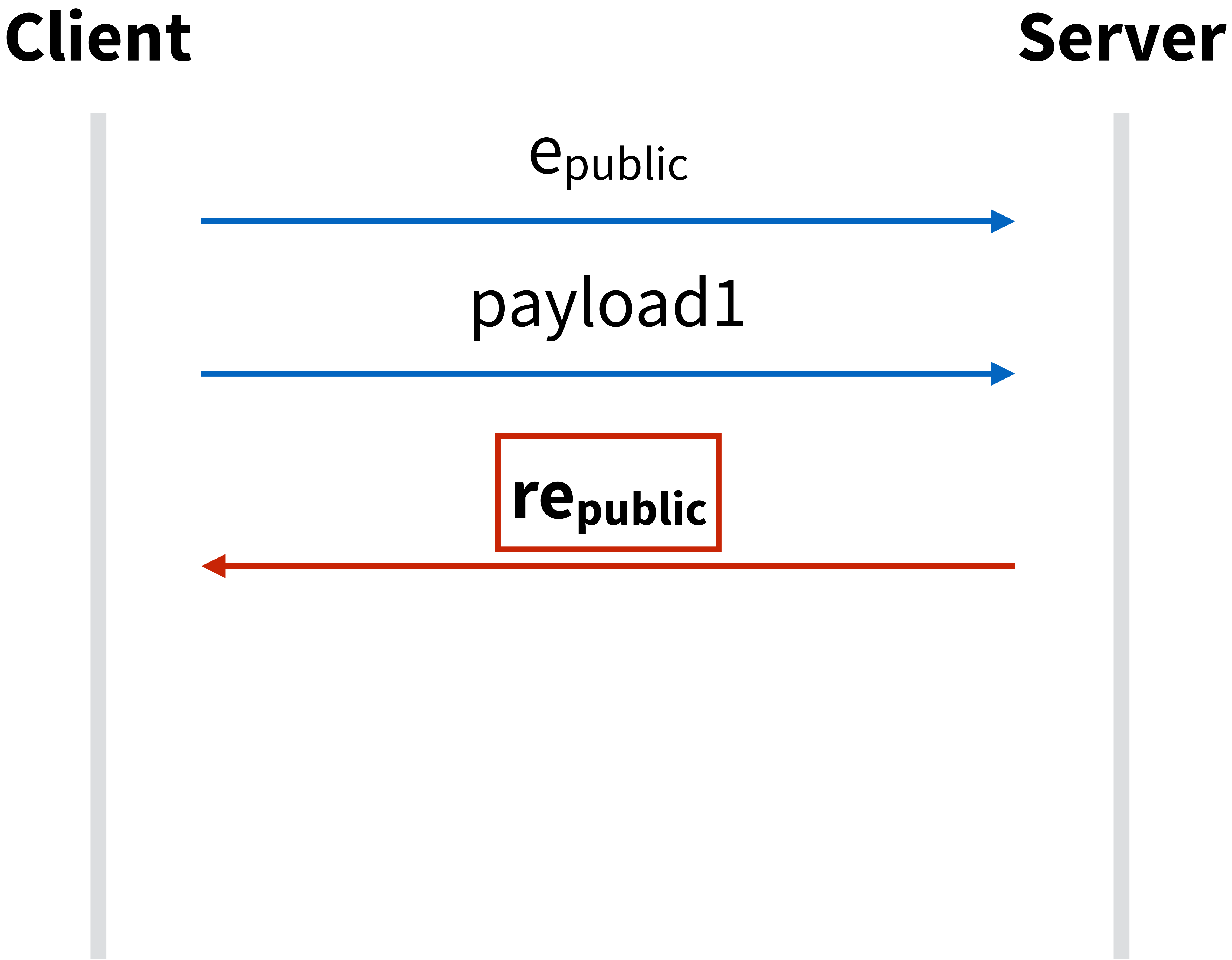
← e, ee, s, es



Noise_**NX**(rs):

→ e

← e, ee, s, es



Client

Server

e_{public}

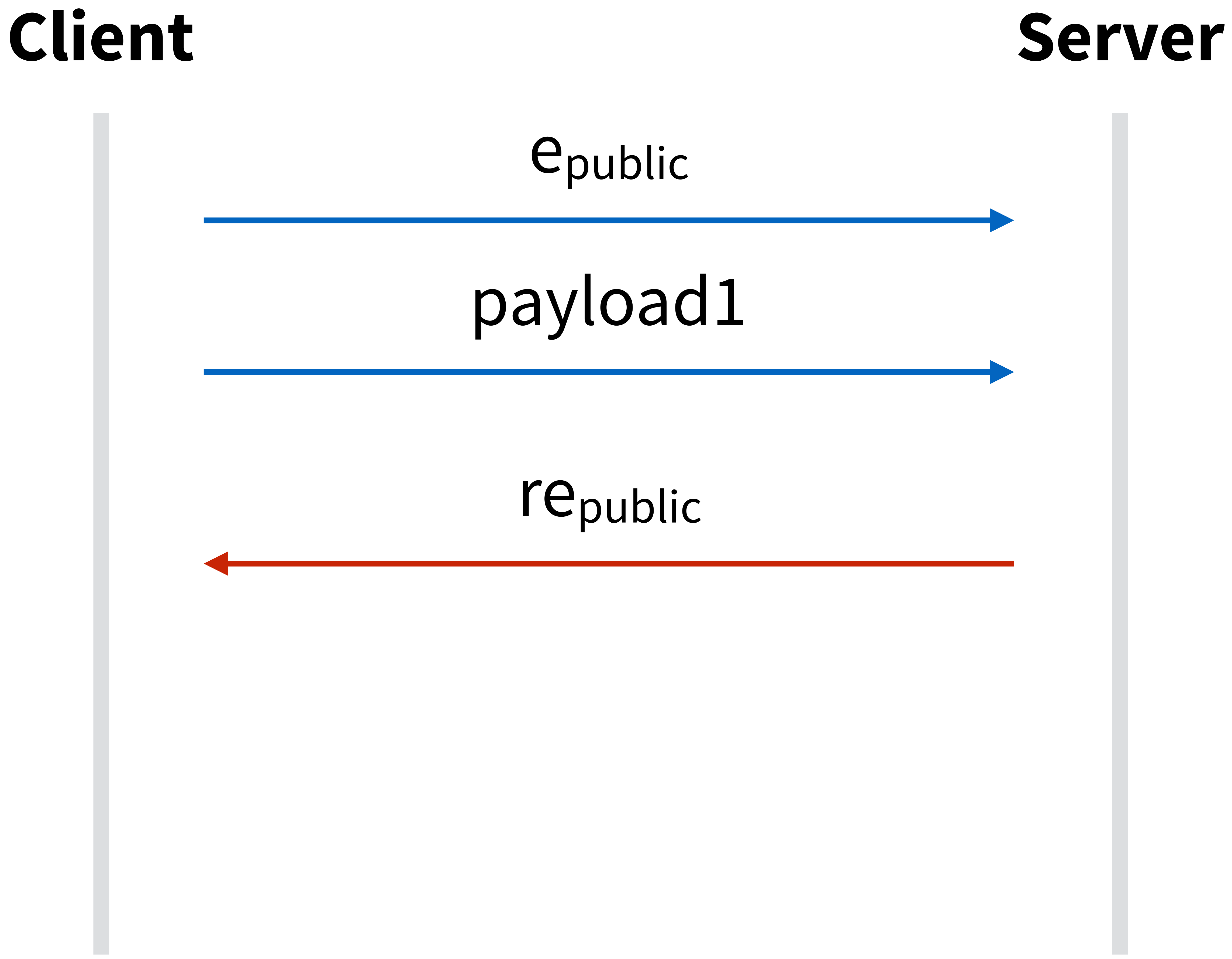
payload1

$\mathbf{re}_{\mathbf{public}}$

Noise_**NX**(rs):

→ e

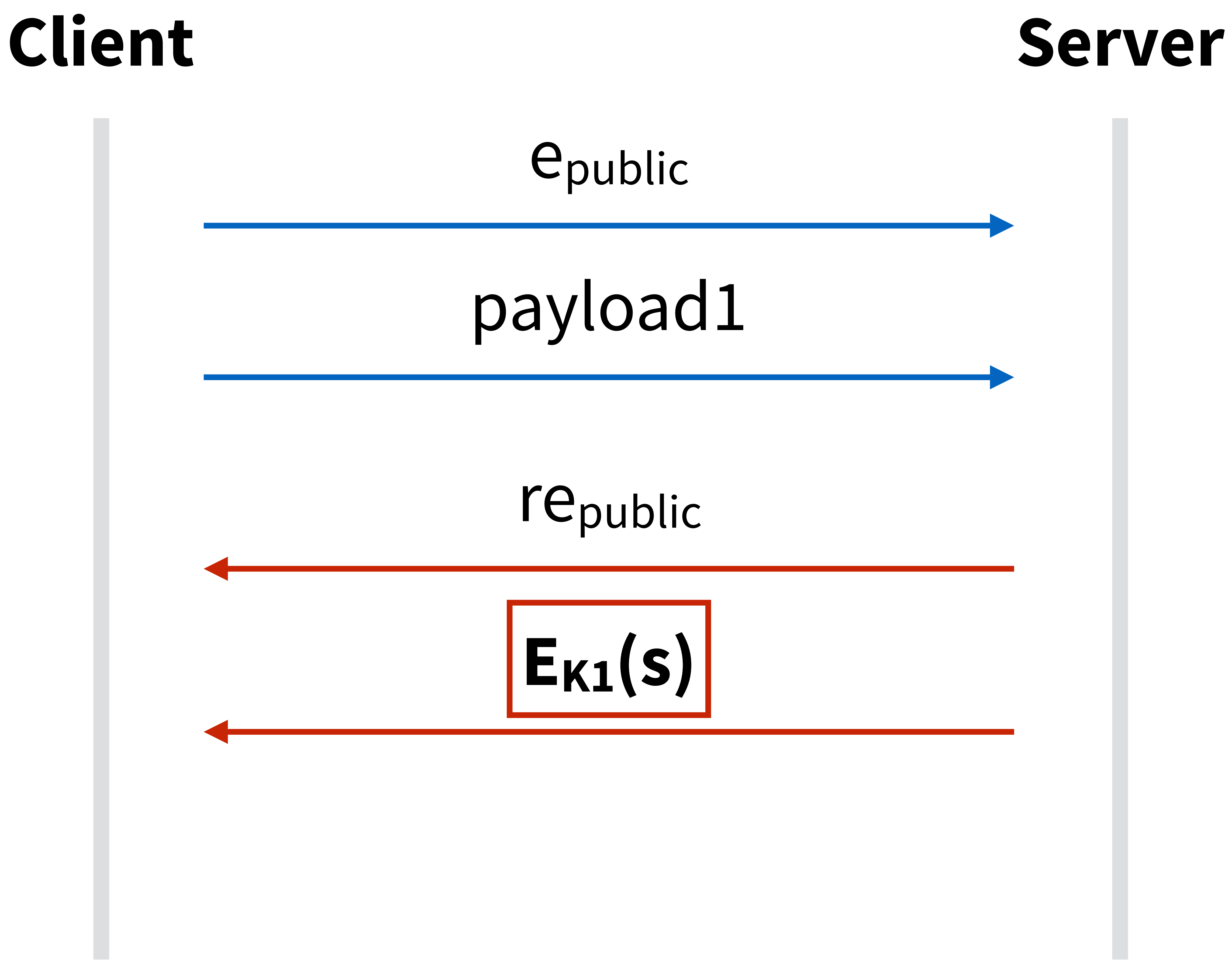
← e, ee, s, es



Noise_**NX**(rs):

→ e

← e, ee, s, es



Client

Server

e_{public}

payload1

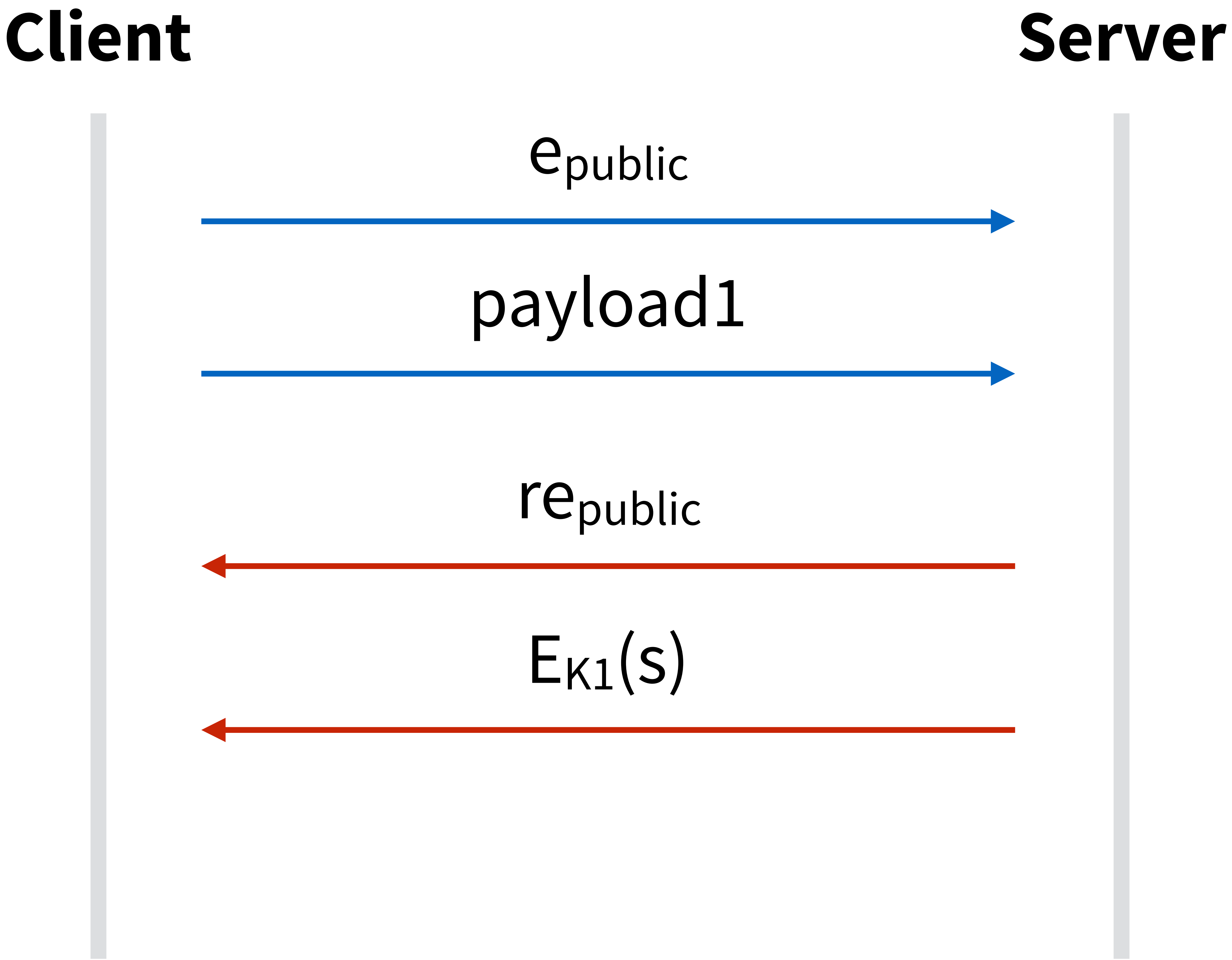
re_{public}

$\boxed{\mathbf{E_{K1}(s)}}$

Noise_**NX**(rs):

→ e

← e, ee, s, es



Client

Server

e_{public}

payload1

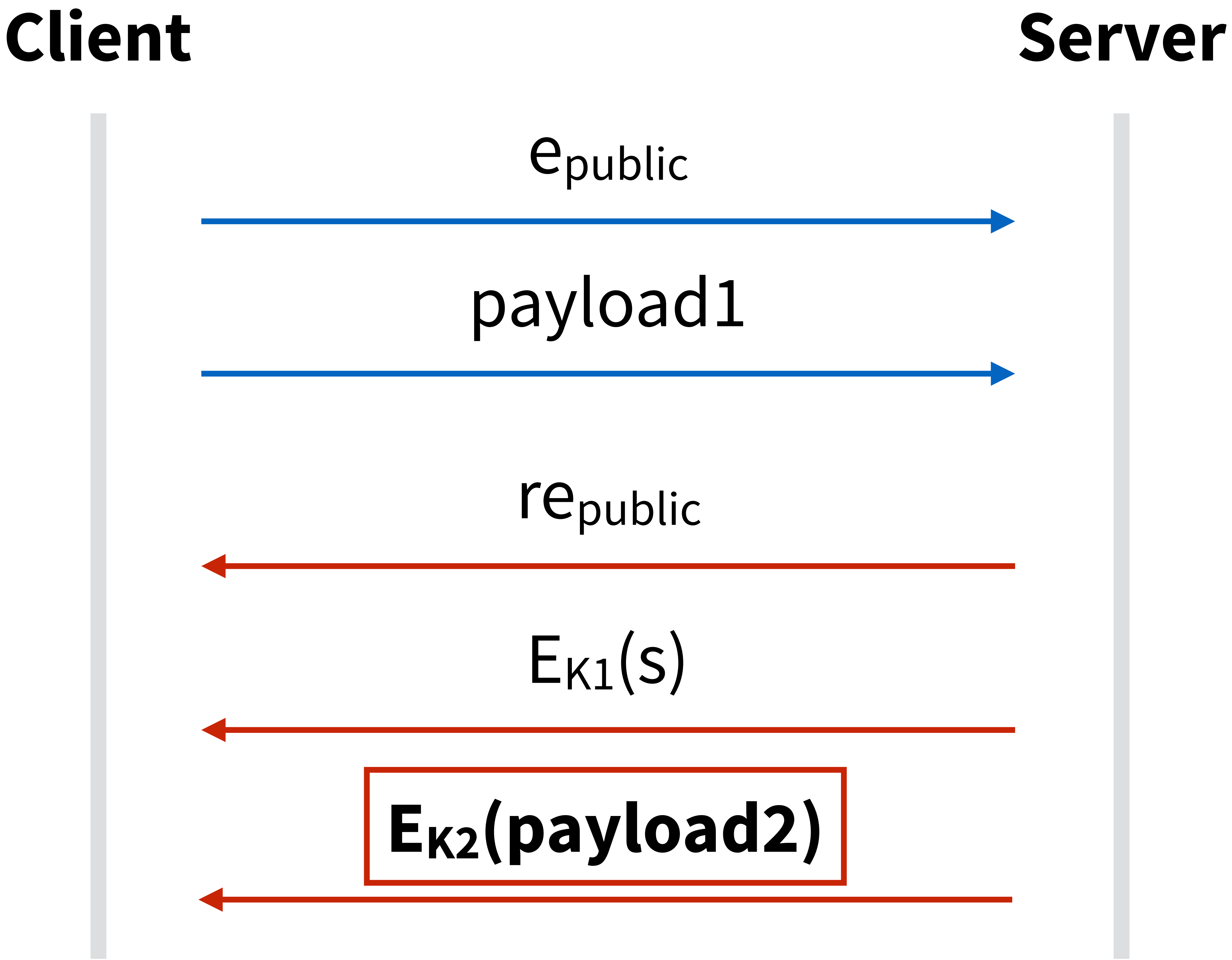
re_{public}

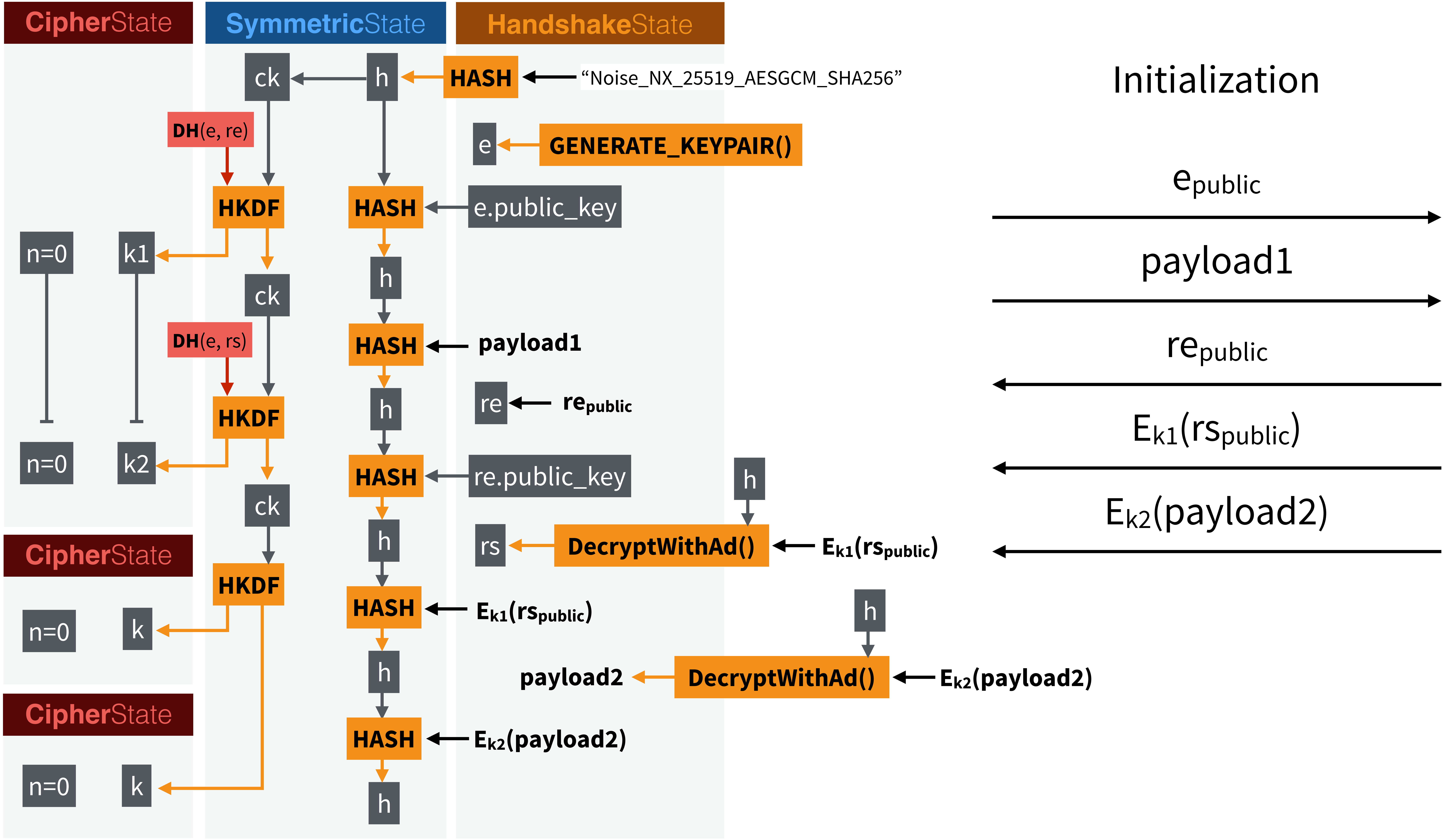
$E_{K1}(s)$

Noise_**NX**(rs):

→ e

← e, ee, s, es

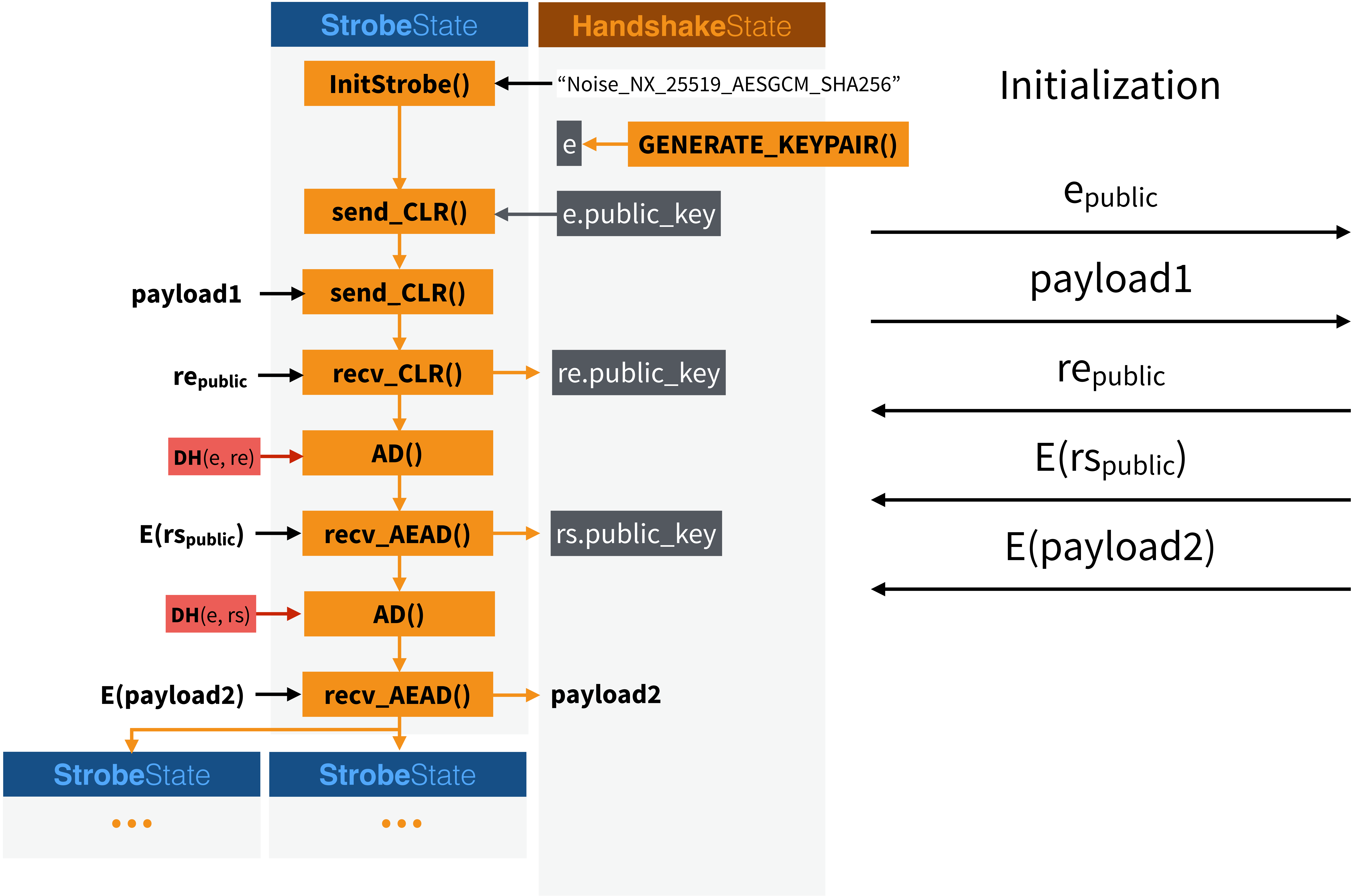



CipherState
SymmetricState
HandshakeState
Initialization
ck
h
HASH
“Noise_NX_25519_AESGCM_SHA256”
DH(e, re)
e
GENERATE_KEYPAIR()
HKDF
HASH
e.public_key
n=0
k1
ck
h
DH(e, rs)
HASH
payload1
re
re_{public}
HKDF
h
n=0
k2
HASH
re.public_key
ck
h
rs
DecryptWithAd()
$E_{k1}(rs_{public})$
CipherState
HKDF
HASH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

Part IV: Noise + Strobe = Disco

A modern cryptographic {protocol, library} based on
SHA-3 and Curve25519

Initialization



StrobeState

HandshakeState

InitStrobe() ← “Noise_NX_25519_AESGCM_SHA256”

e ← GENERATE_KEYPAIR()

send_CLR() ← e.public_key

payload1 → send_CLR()

re_{public} → recv_CLR() → re.public_key

DH(e, re) → AD()

E(rs_{public}) → recv_AEAD() → rs.public_key

DH(e, rs) → AD()

E(payload2) → recv_AEAD() → payload2

StrobeState

• • •

StrobeState

• • •

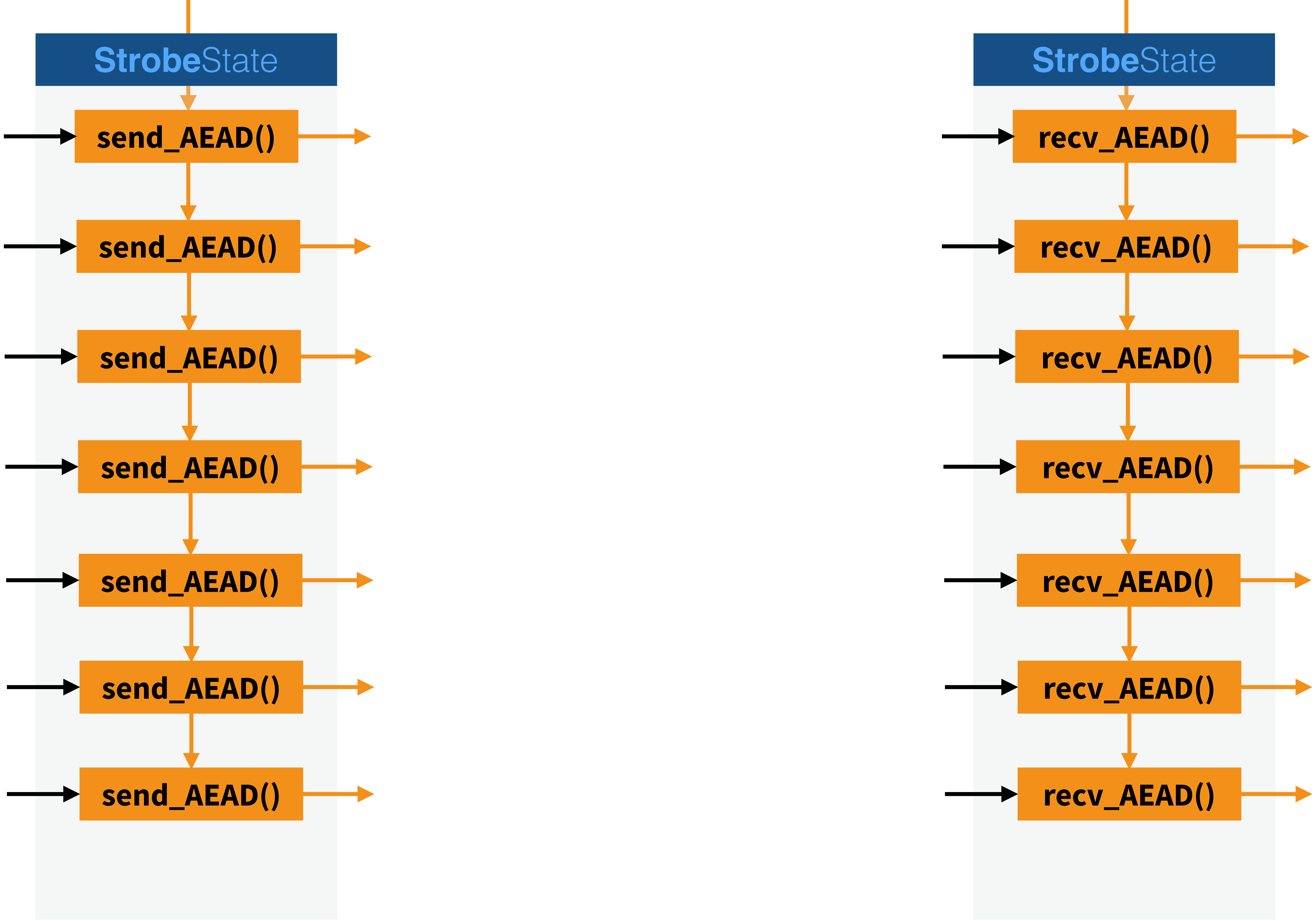
e_{public} →

payload1 →

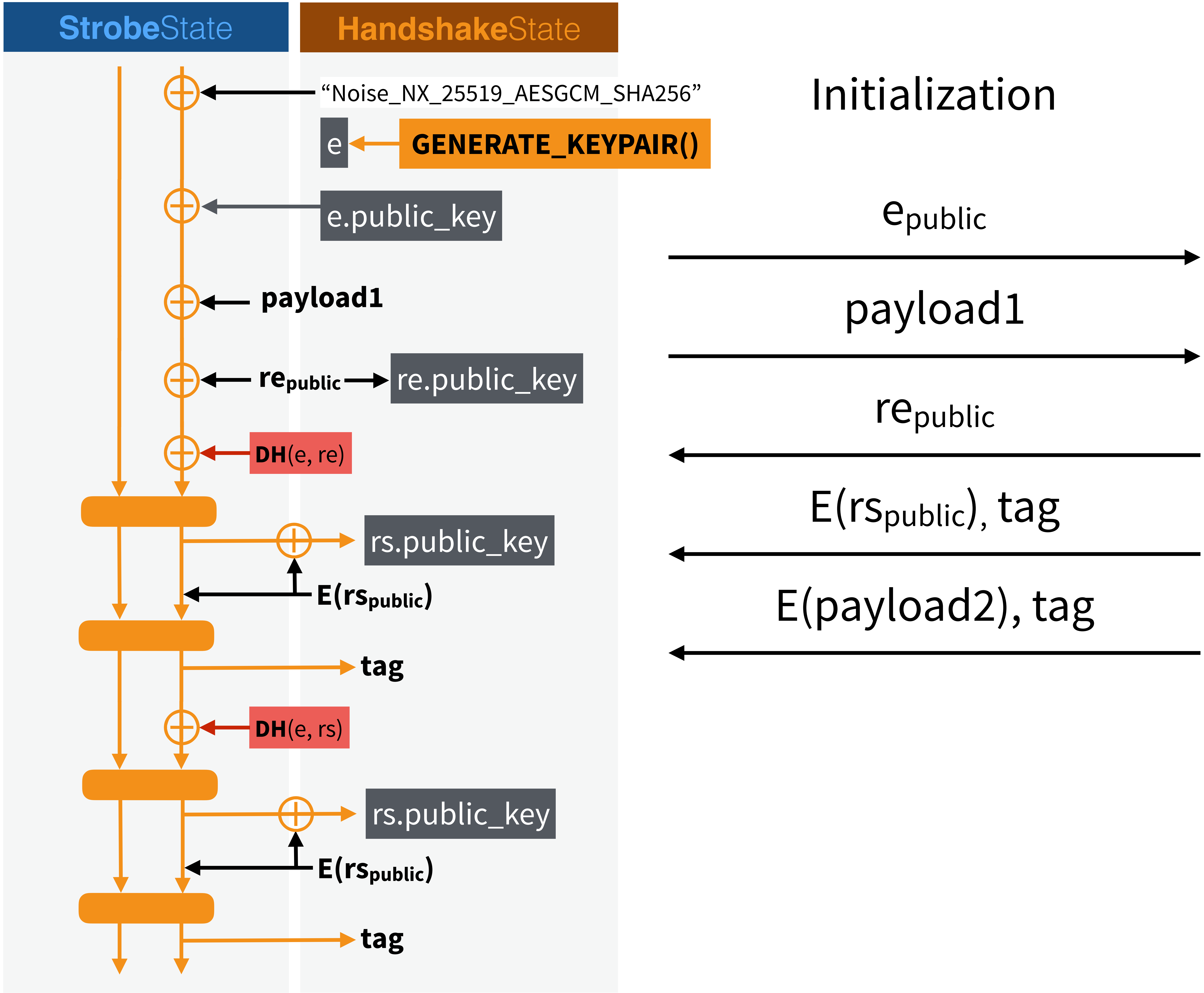
re_{public} ←

E(rs_{public}) ←

E(payload2) ←



no need for **IVs** or **nonces**



StrobeState

HandshakeState

“Noise_NX_25519_AESGCM_SHA256”

e

GENERATE_KEYPAIR()

e.public_key

payload1

re$_{\textbf{public}}$

re.public_key

DH(e, re)

rs.public_key

E(rs$_{\textbf{public}}$**)**

tag

DH(e, rs)

rs.public_key

E(rs$_{\textbf{public}}$**)**

tag

Initialization

e$_{\text{public}}$

payload1

re$_{\text{public}}$

E(rs$_{\text{public}}$), tag

E(payload2), tag

The **state** of **Disco**

- **Noise** is still a draft
- **Strobe** is alpha (v1.0.2)
- **Disco** is a draft specification extending Noise (**experimental**)
- **libDisco** is a **plug-and-play protocol+library**
  - the Golang library is here: www.discrocrypto.com
  - it's **~1000 lines of code**
    - ~2000 lines of code with Strobe
    - +2000 lines of code with X25519
- ⚠️ Disco and libdisco are still **experimental**
  - we need more eyes, more interoperability testing, …
- ⚠️ **THIS IS NOT REPLACING TLS**

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

I **tweet** my mind on
twitter.com/lyon01_david

and I work here

nccgroup