CSE551: Advanced Computer Security

6. SSL/TLS & HTTPS

Seongil Wi

Department of Computer Science and Engineering

HW1

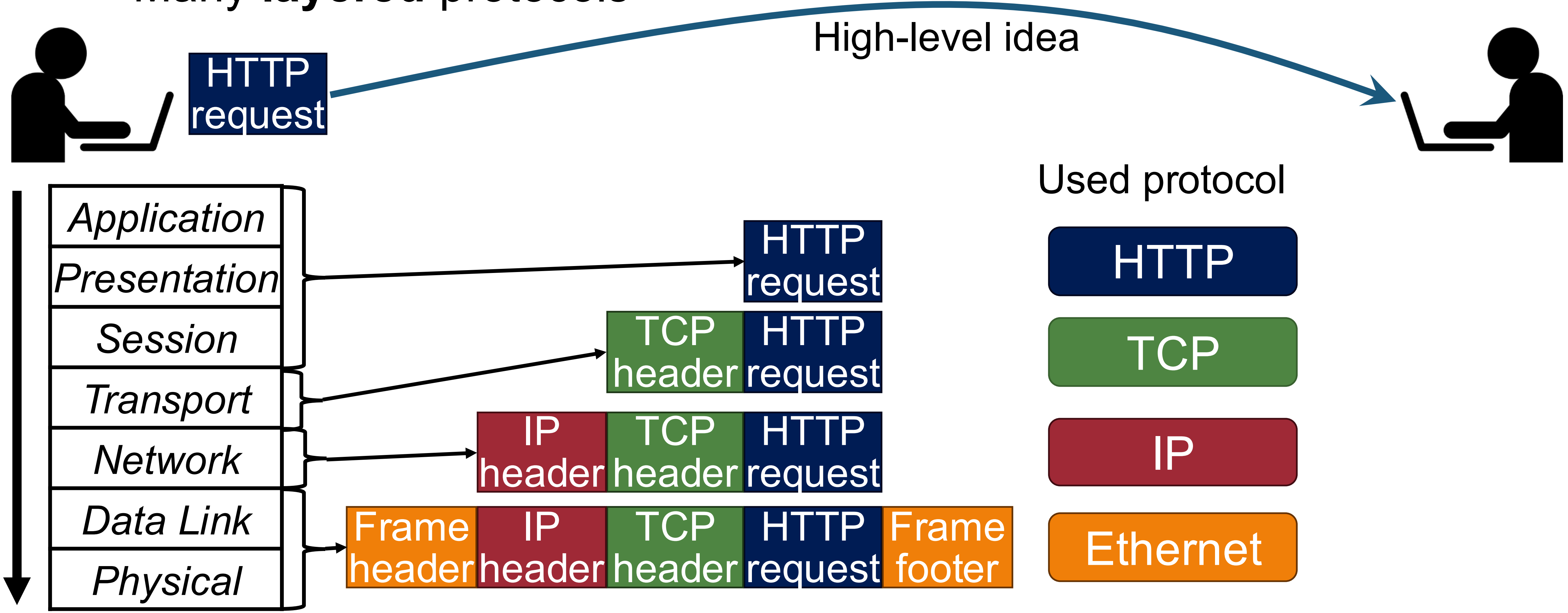
- Submit two paper critiques
- Detailed instructions will be provided later
- Due: September 9, 11:59 PM

Protocol

- A system of digital **rules** for data exchange between computers
- Many **layered** protocols

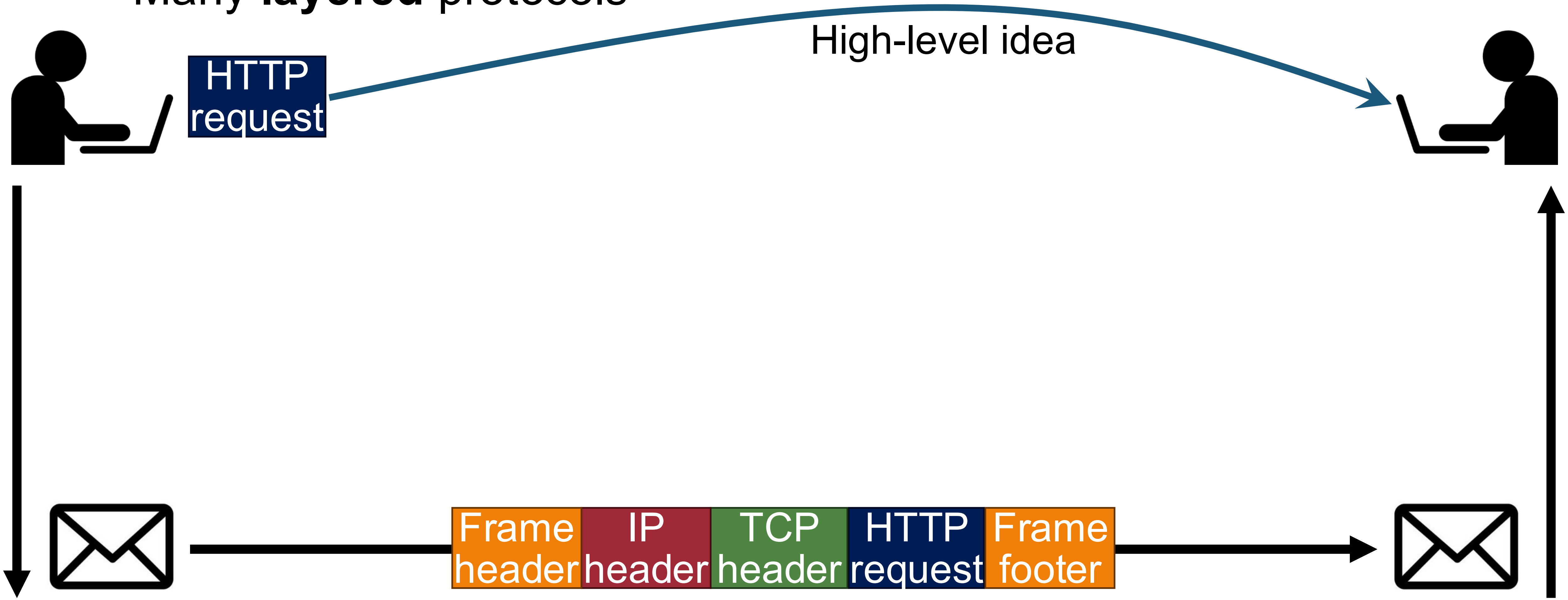
Protocol

- A system of digital **rules** for data exchange between computers
- Many **layered** protocols



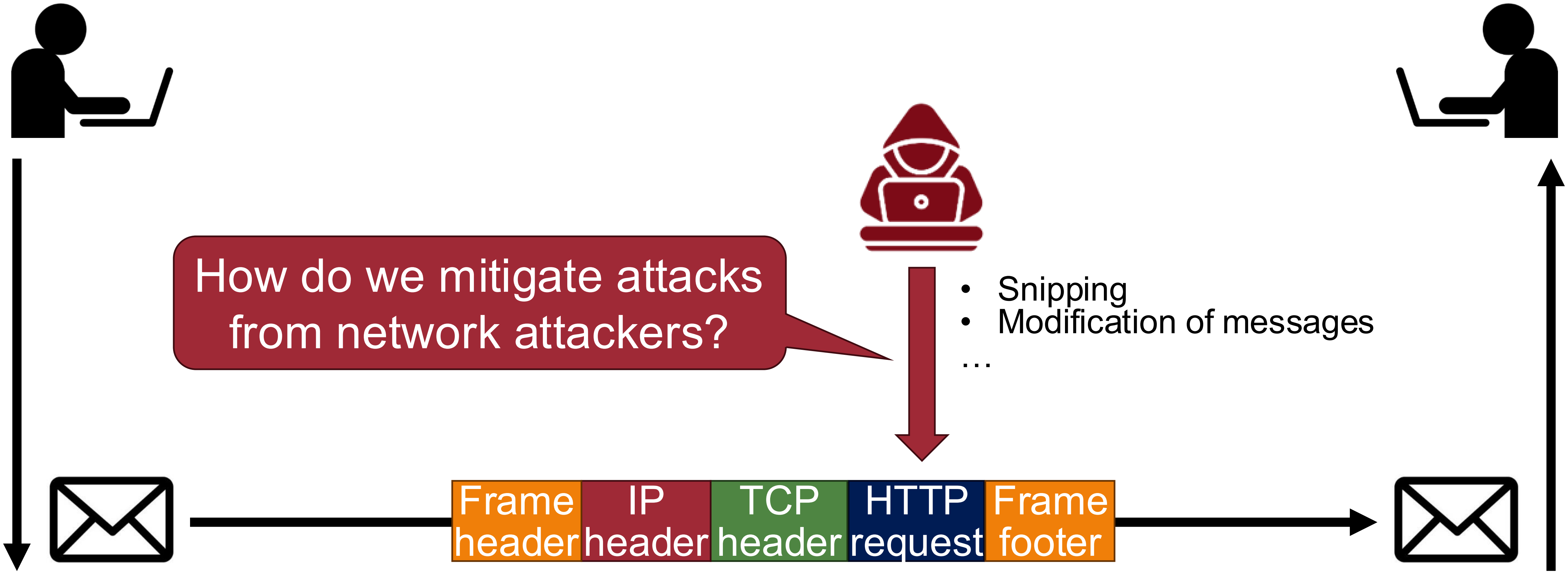
Protocol

- A system of digital **rules** for data exchange between computers
- Many **layered** protocols



Network Attackers

- A system of digital **rules** for data exchange between computers
- Many **layered** protocols



How do we mitigate attacks from network attackers?

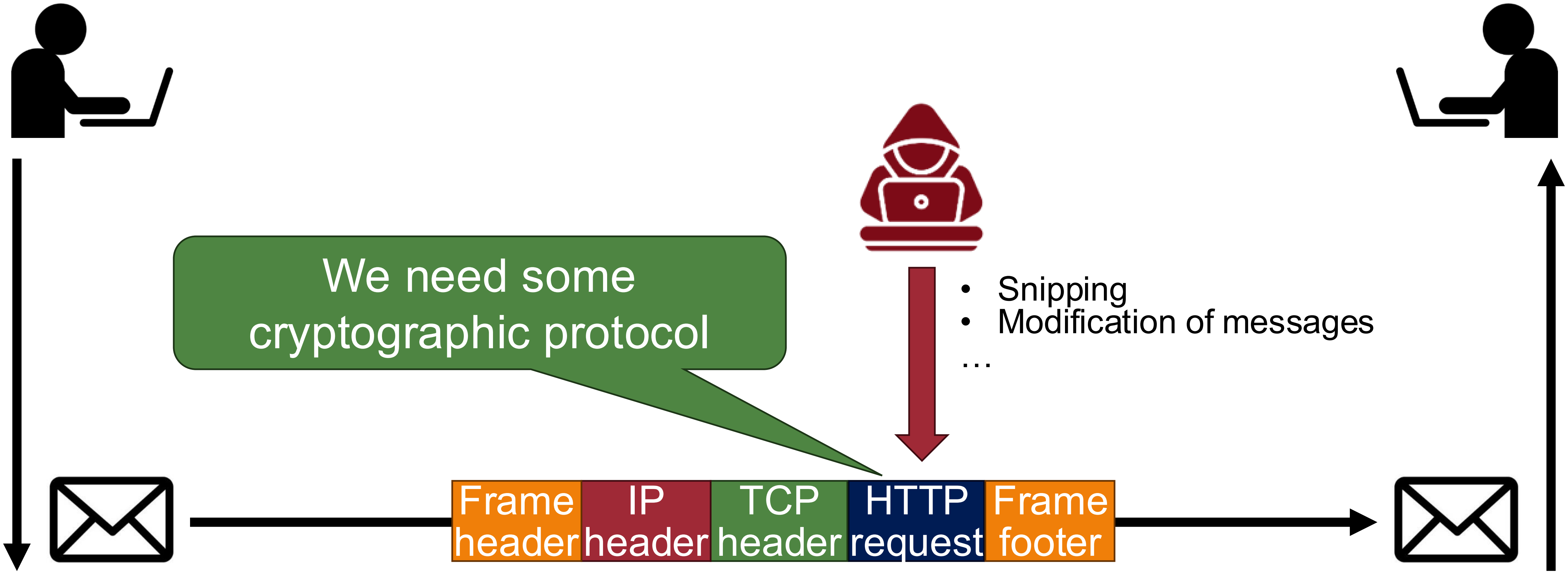
- Snipping
- Modification of messages

…

Frame header | IP header | TCP header | HTTP request | Frame footer

Motivation: Cryptographical Protocol

- A system of digital **rules** for data exchange between computers
- Many **layered** protocols



We need some cryptographic protocol

- Snipping
- Modification of messages

…

Frame header	IP header	TCP header	HTTP request	Frame footer

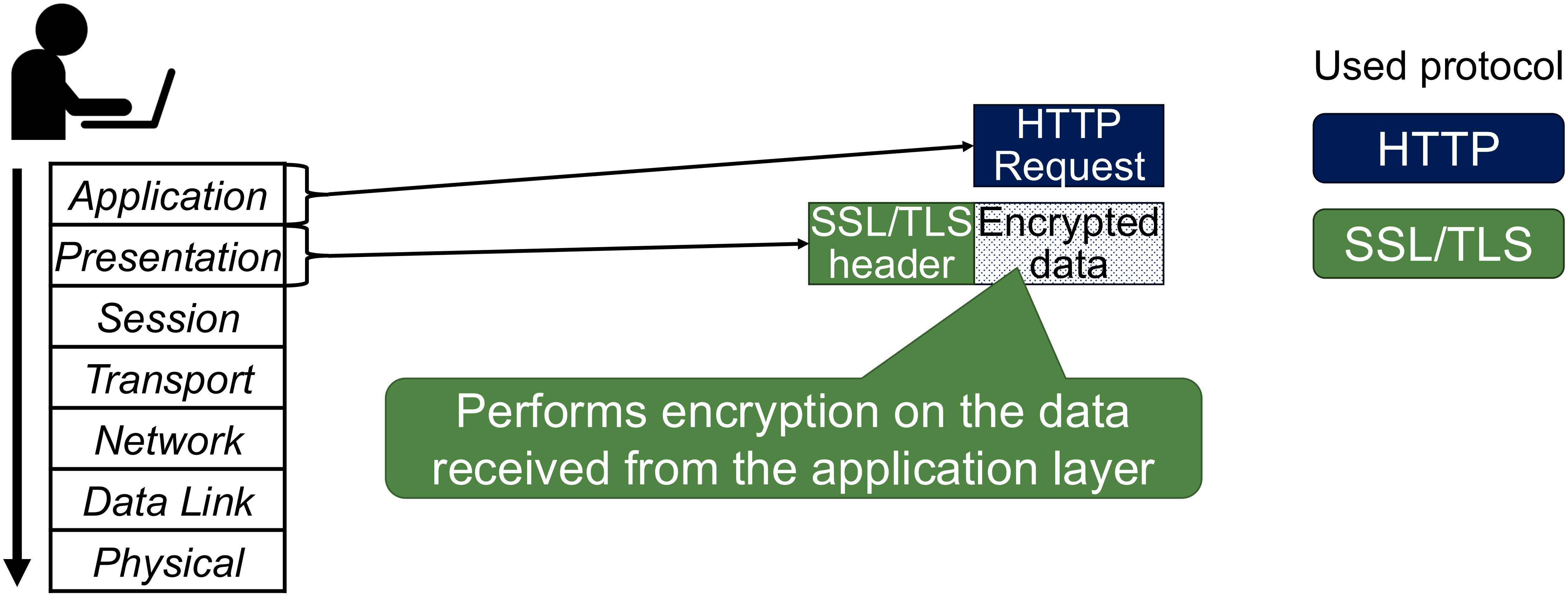
SSL/TLS

What is SSL/TLS?

- **Secure Sockets Layer (SSL)** and **Transport Layer Security (TLS)** protocols
  - Same protocol design, different crypto algorithms
  - (Reserved) port number: 443
- Security goals: achieving…
  - Confidentiality
  - Integrity
  - Authentication
- ***De facto* standard for Internet security**

SSL/TLS Basic Idea

- Adding a protocol layer for secure communication!



Application

Presentation

Session

Transport

Network

Data Link

Physical

HTTP Request

SSL/TLS header

Encrypted data

Performs encryption on the data received from the application layer

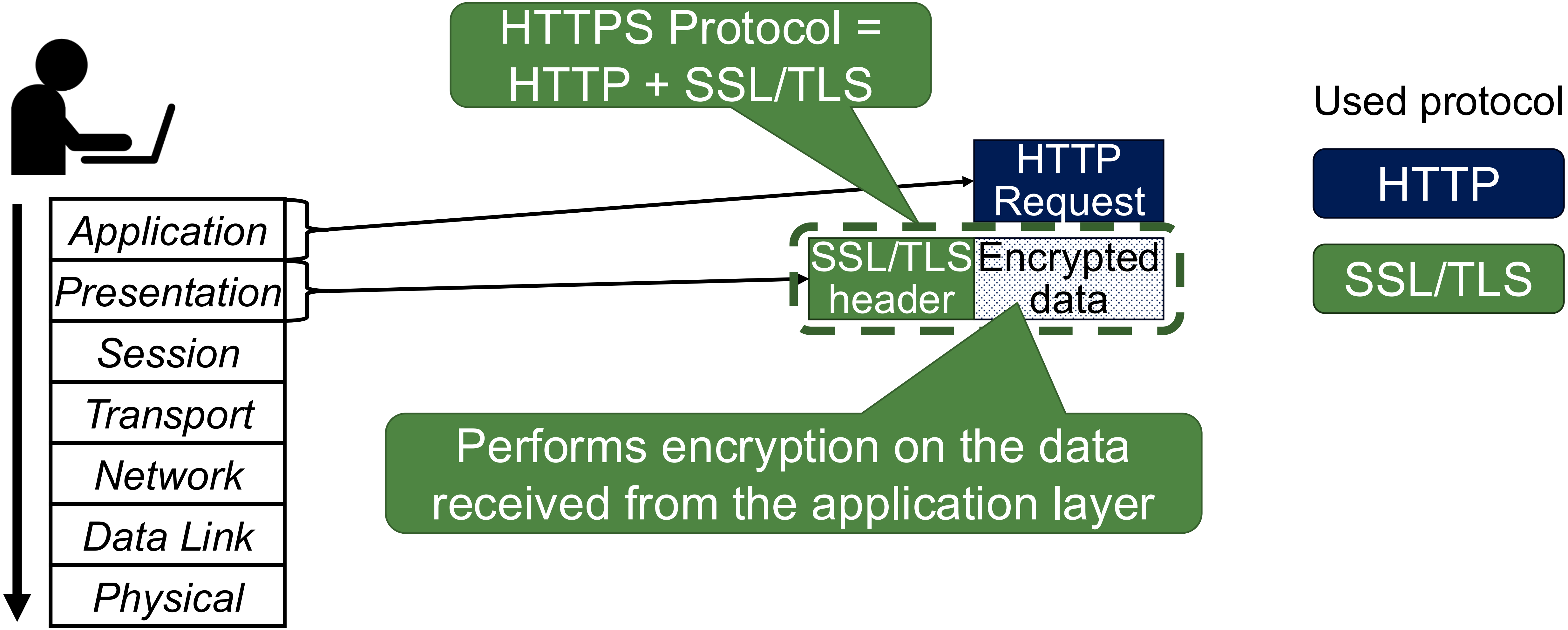
Used protocol

HTTP

SSL/TLS

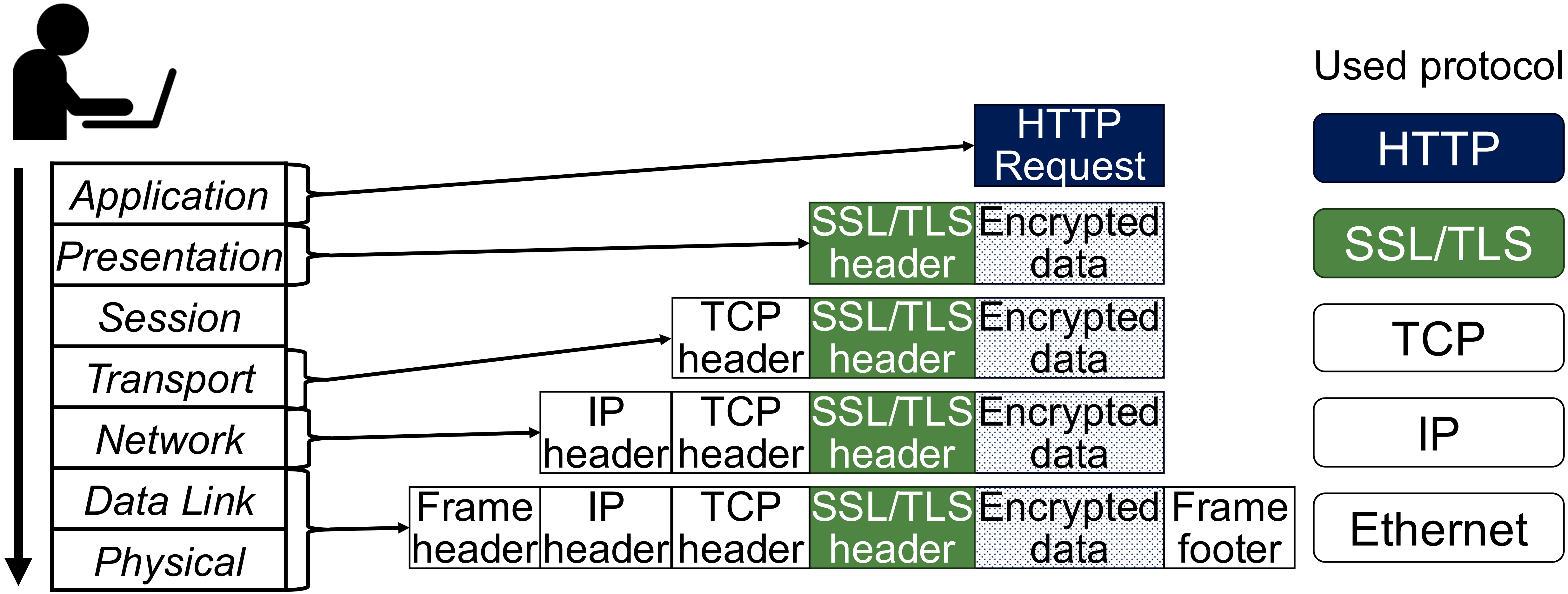
SSL/TLS Basic Idea

- Adding a protocol layer for secure communication!



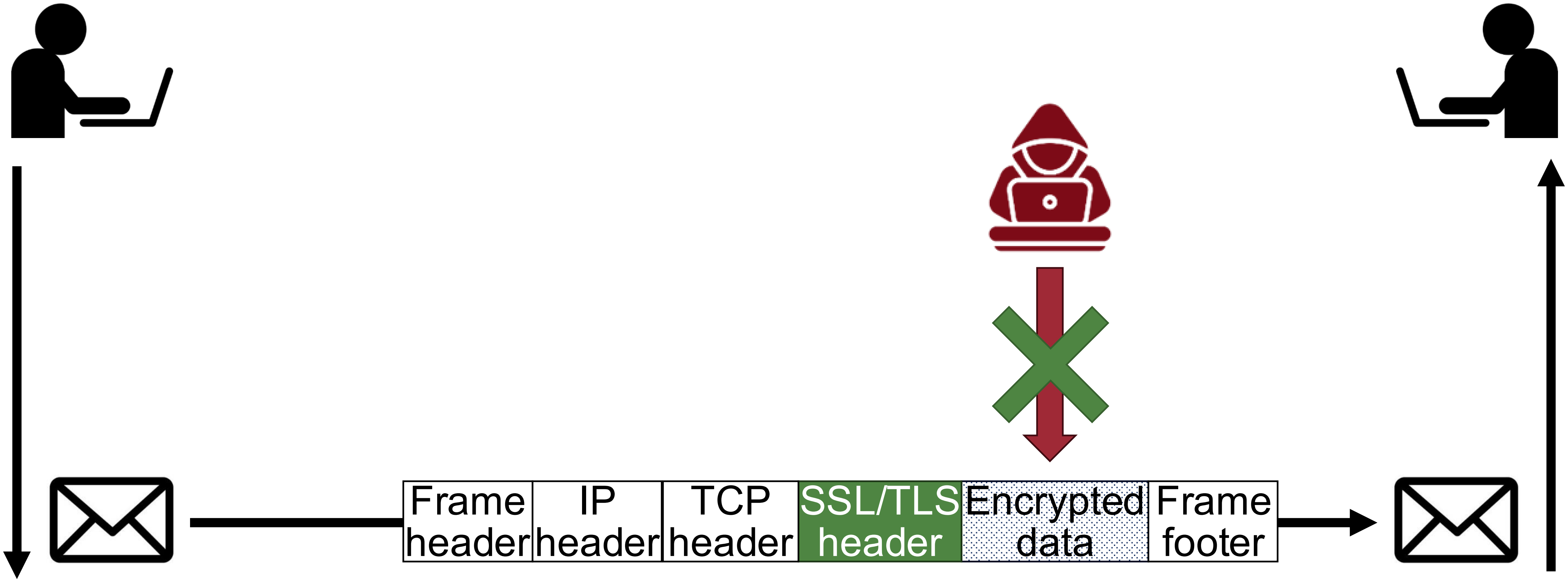
SSL/TLS Basic Idea

- Adding a protocol layer for secure communication!



SSL/TLS Basic Idea

- Adding a protocol layer for secure communication!

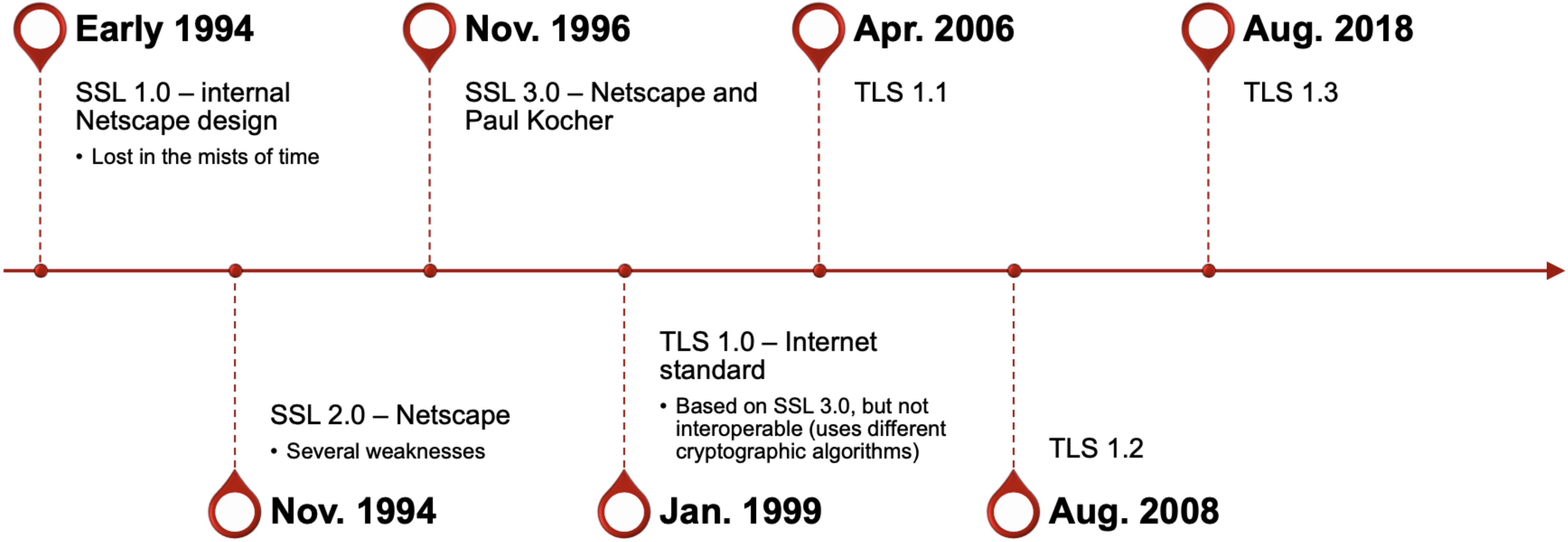


Frame header	IP header	TCP header	SSL/TLS header	Encrypted data	Frame footer

Use Cases

- Email
- Vice over IP (VoIP)
- Payment systems (transactions)
- **HTTPS**
  - The most publicly visible use case!
  - Deployed in every web browser

History of the Protocol



Early 1994

SSL 1.0 – internal Netscape design

- Lost in the mists of time

Nov. 1994

SSL 2.0 – Netscape

- Several weaknesses

Nov. 1996

SSL 3.0 – Netscape and Paul Kocher

Jan. 1999

TLS 1.0 – Internet standard

- Based on SSL 3.0, but not interoperable (uses different cryptographic algorithms)

Apr. 2006

TLS 1.1

Aug. 2008

TLS 1.2

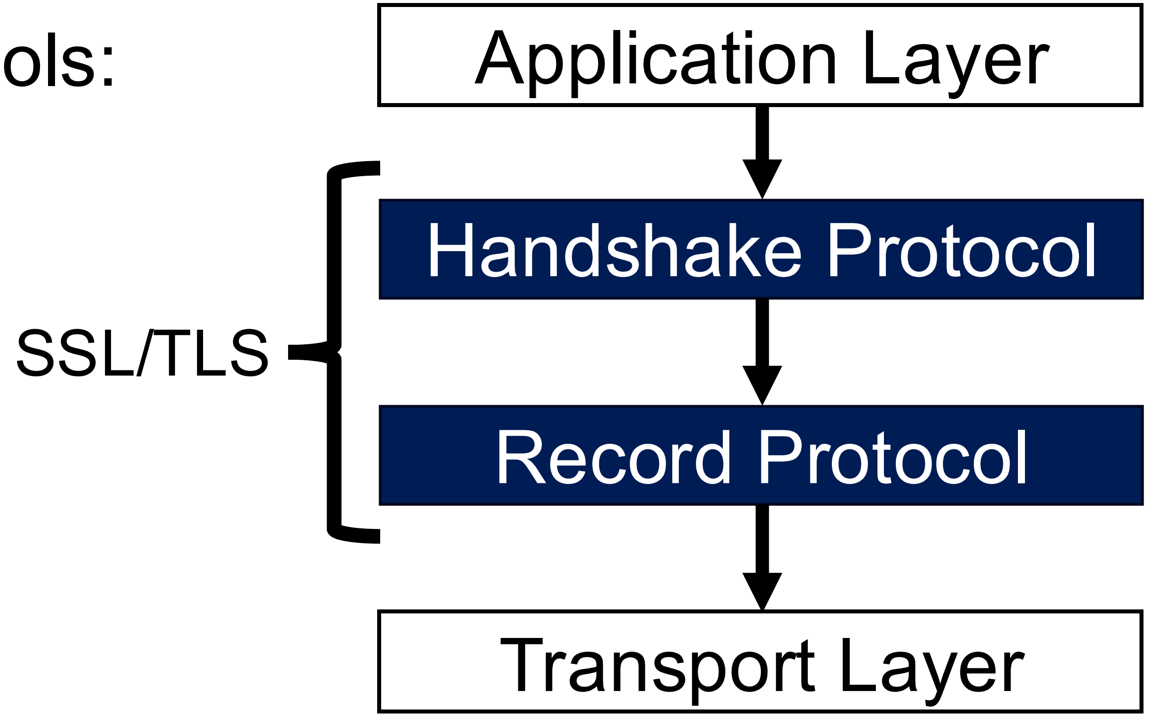
Aug. 2018

TLS 1.3

SSL/TLS Basics

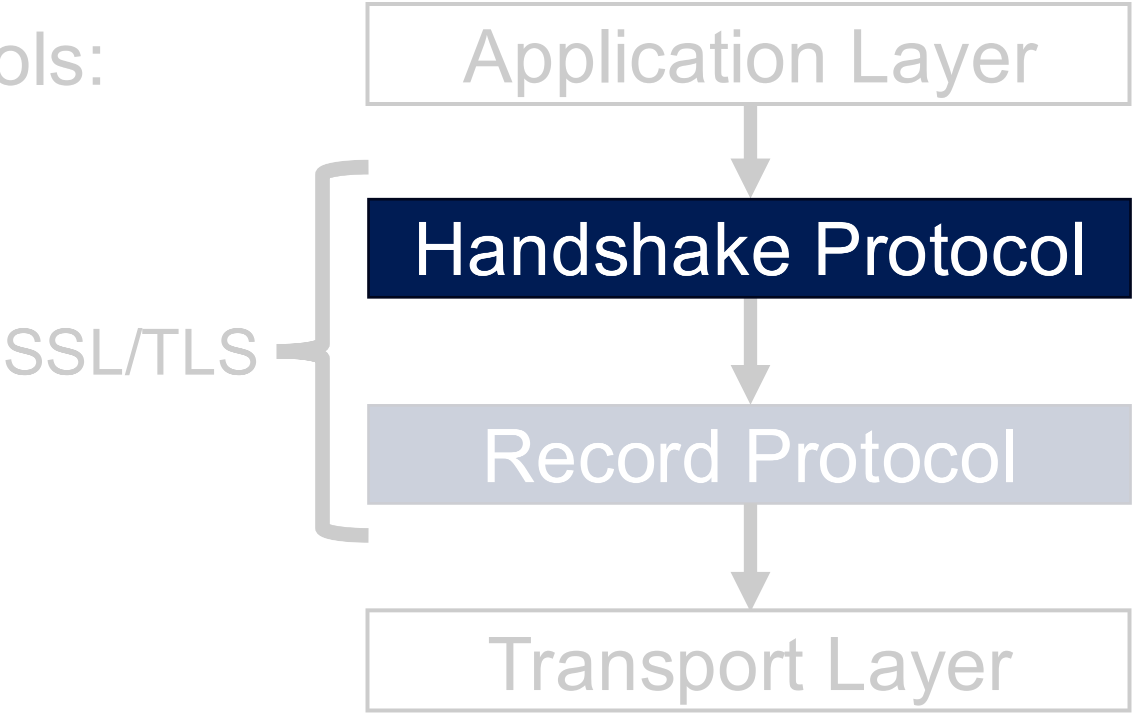
- Runs in the presentation layer
- Uses symmetric crypto, asymmetric crypto, and digital signatures

- Composed of two layers of protocols:
  1. Handshake protocol
  2. Record protocol



SSL/TLS Basics

- Runs in the presentation layer
- Uses symmetric crypto, asymmetric crypto, and digital signatures
- Composed of two layers of protocols:
  1. Handshake protocol
  2. Record protocol

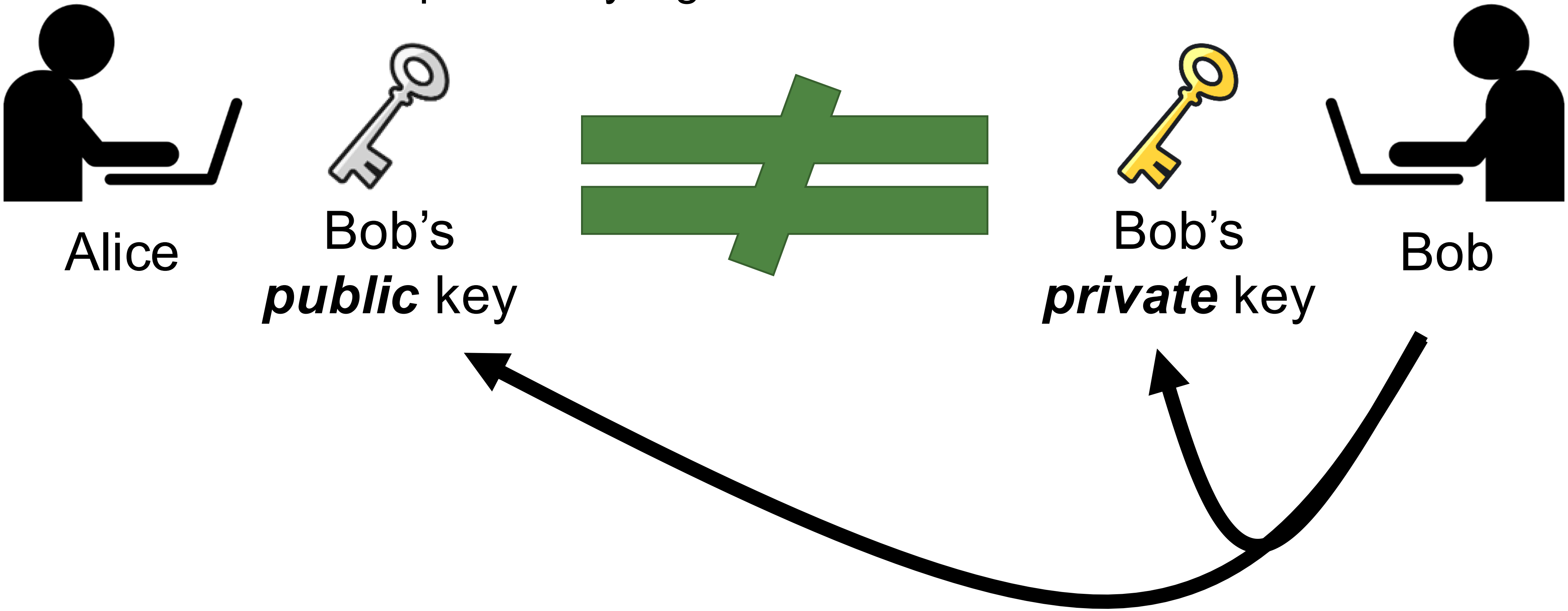


SSL/TLS Handshake Protocol

- The most complex part of SSL
- Uses asymmetric cryptography (public-key cryptography) to establish **several shared secret**

Ref: Asymmetric Key Cryptography

- Each party has two distinct keys: public key and private key
  - Also known as public-key algorithm



Alice

Bob's ***public*** key

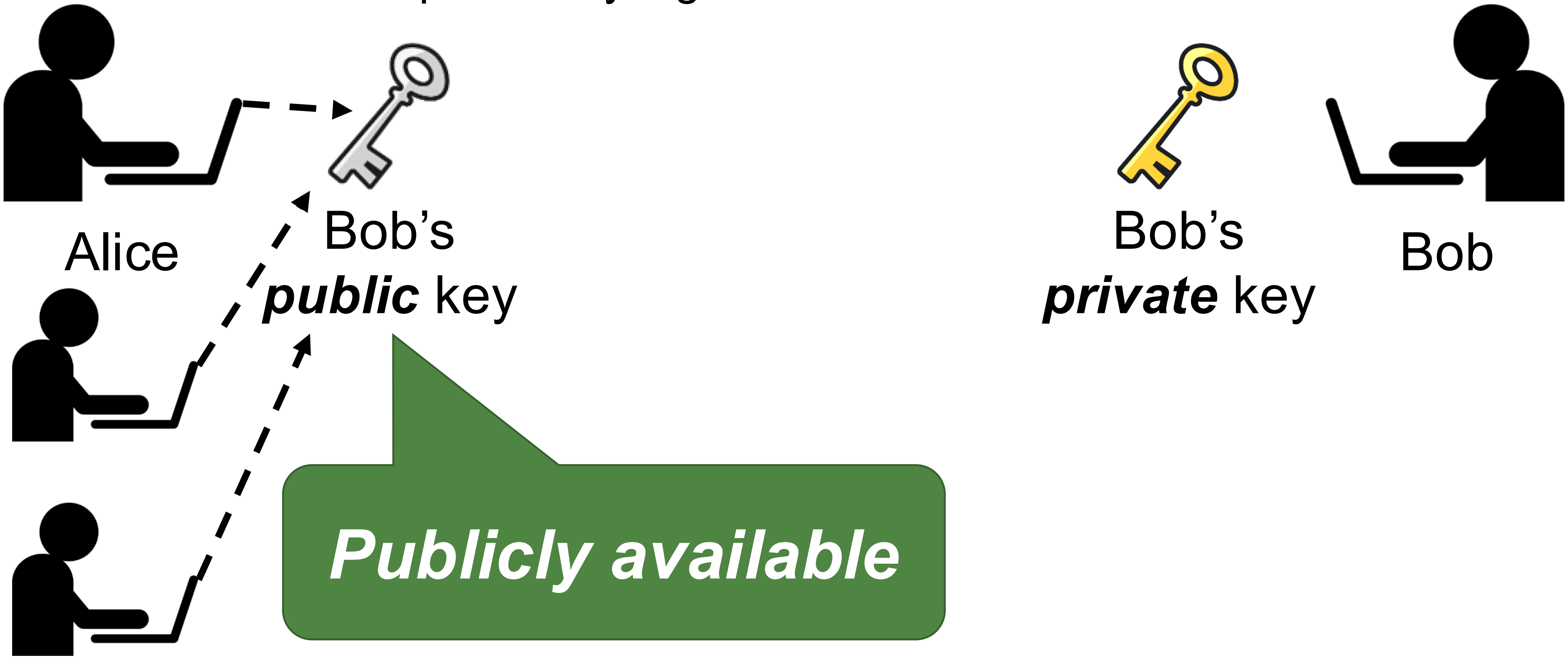
≠

Bob's ***private*** key

Bob

Ref: Asymmetric Key Cryptography

- Each party has two distinct keys: public key and private key
  - Also known as public-key algorithm



Alice

Bob's ***public*** key

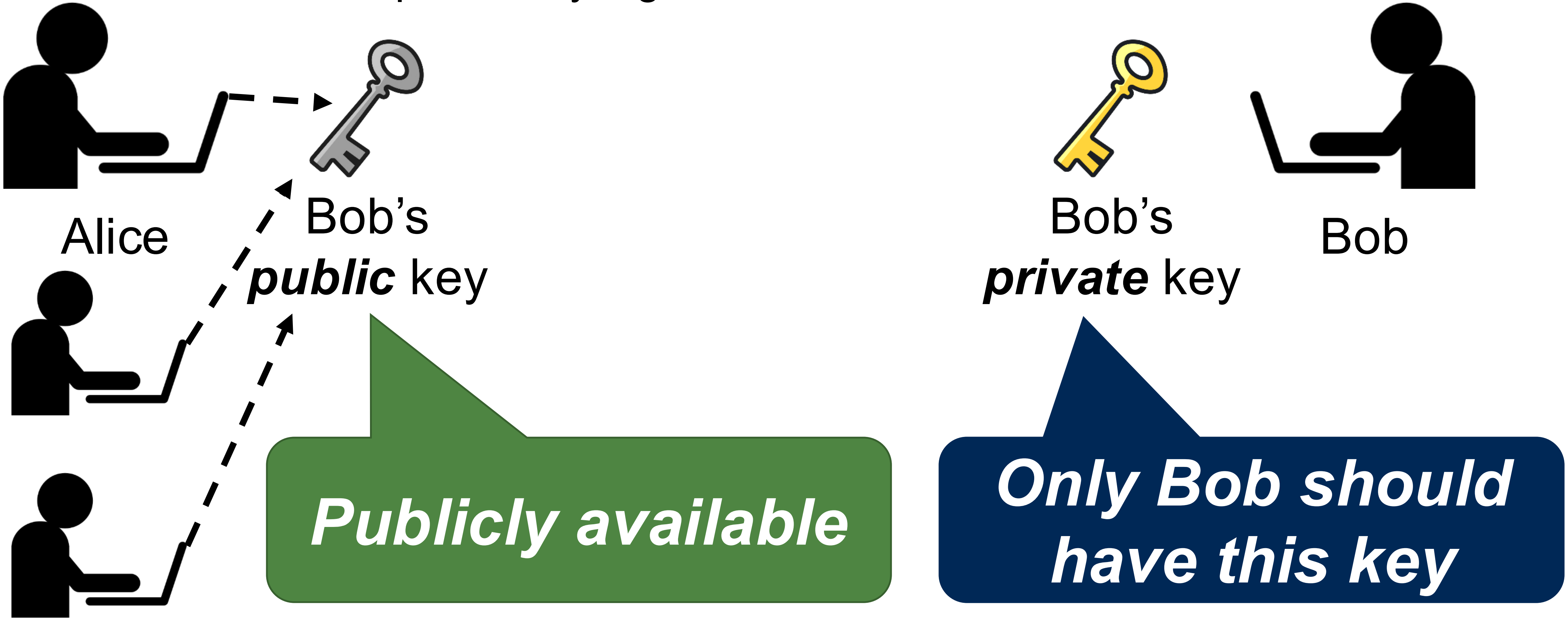
Bob's ***private*** key

Bob

Publicly available

Ref: Asymmetric Key Cryptography

- Each party has two distinct keys: public key and private key
  - Also known as public-key algorithm



Alice

Bob's ***public*** key

Bob's ***private*** key

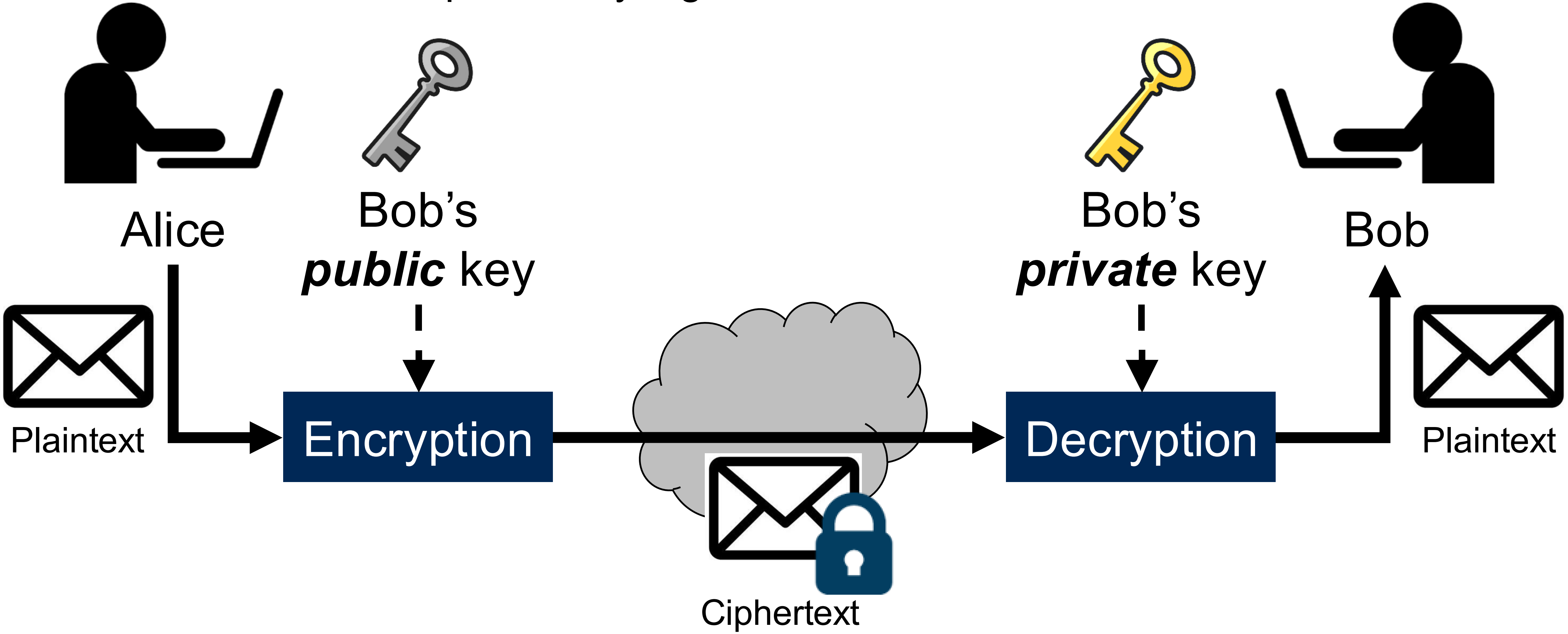
Bob

Publicly available

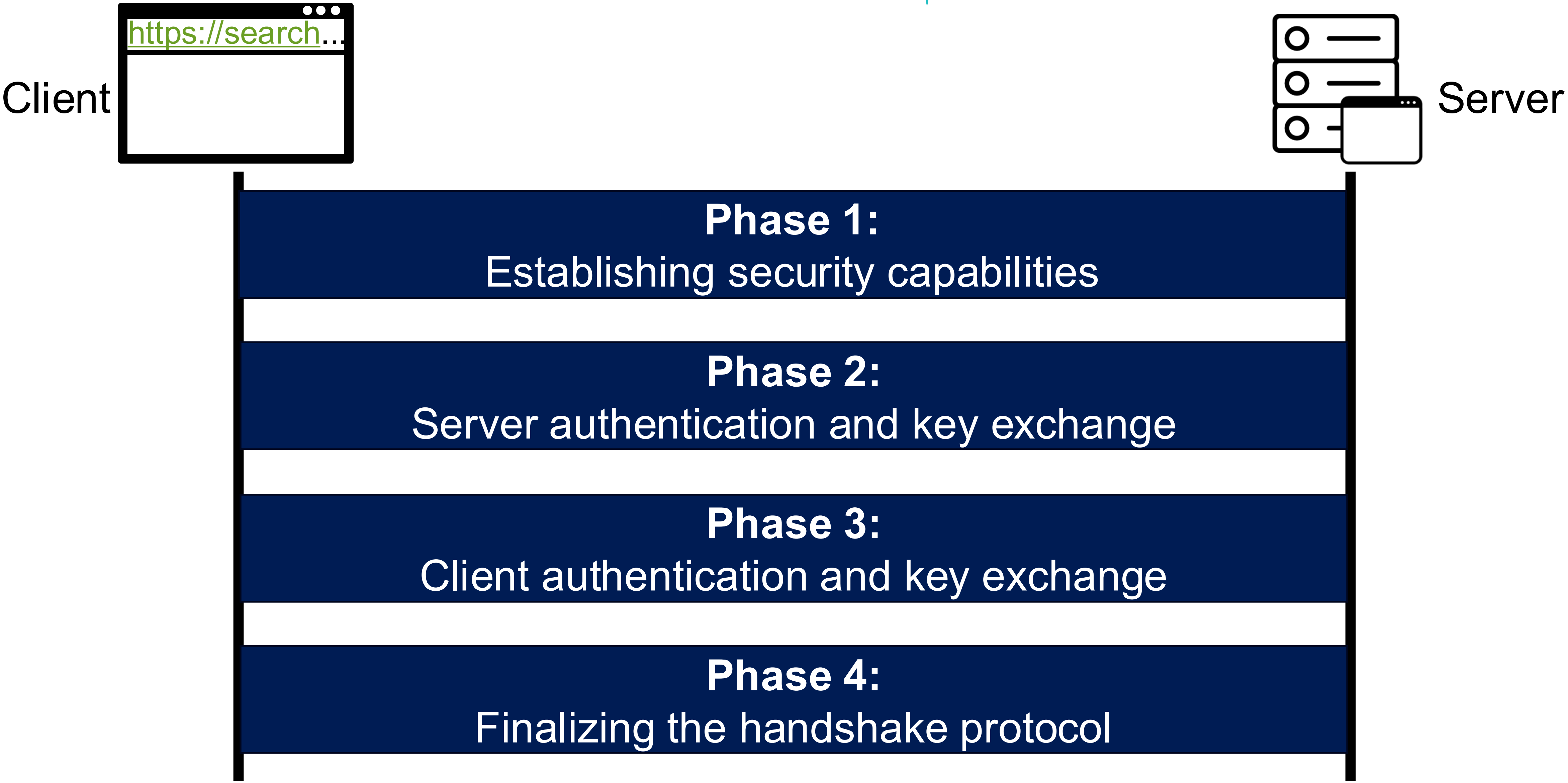
Only Bob should have this key

Ref: Asymmetric Key Cryptography

- Each party has two distinct keys: public key and private key
  - Also known as public-key algorithm



Four Phases of Handshake Protocol



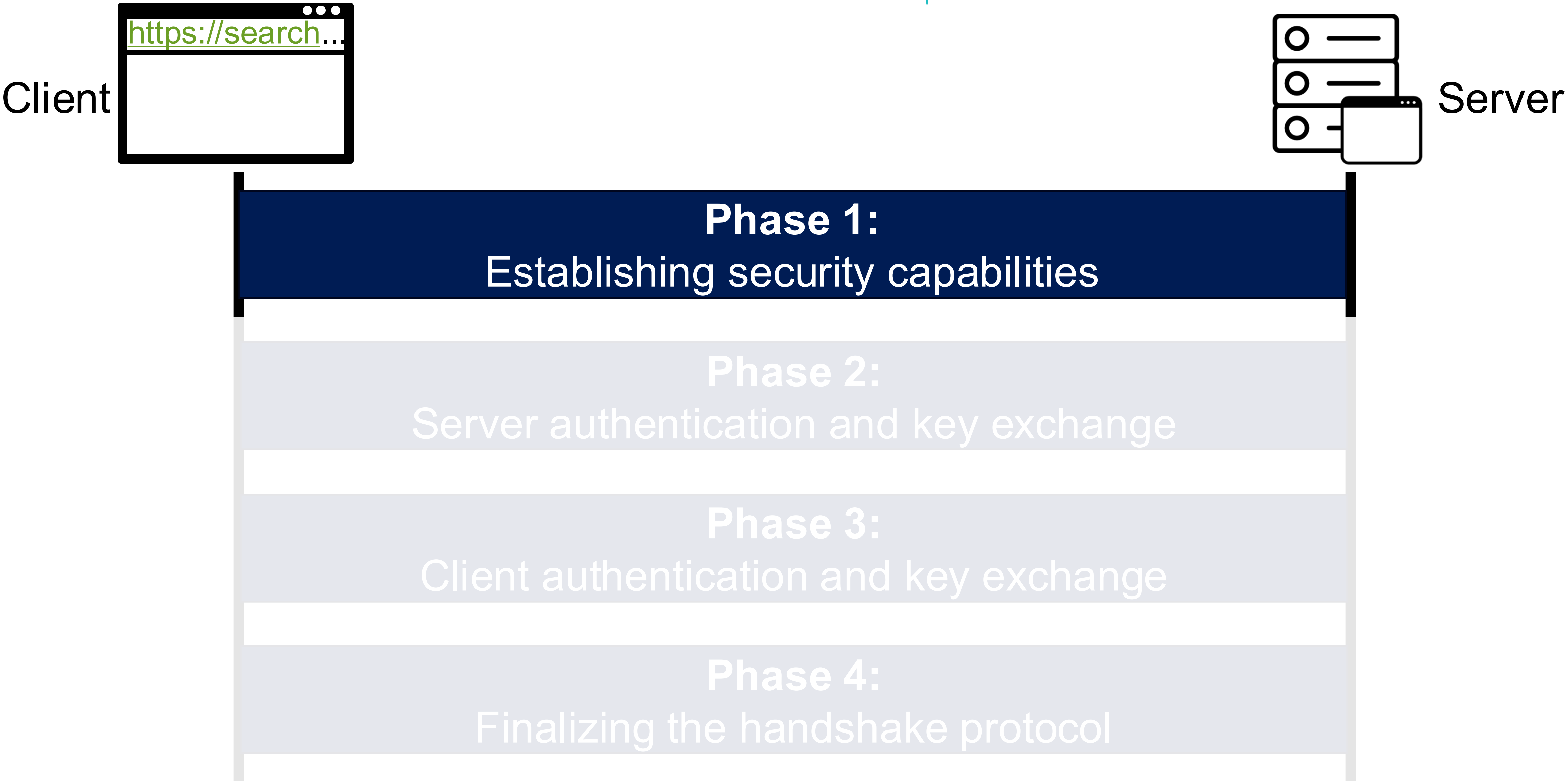
Phase 1:
Establishing security capabilities

Phase 2:
Server authentication and key exchange

Phase 3:
Client authentication and key exchange

Phase 4:
Finalizing the handshake protocol

Phase 1: Establishing Security Capabilities



Client

https://search...

Server

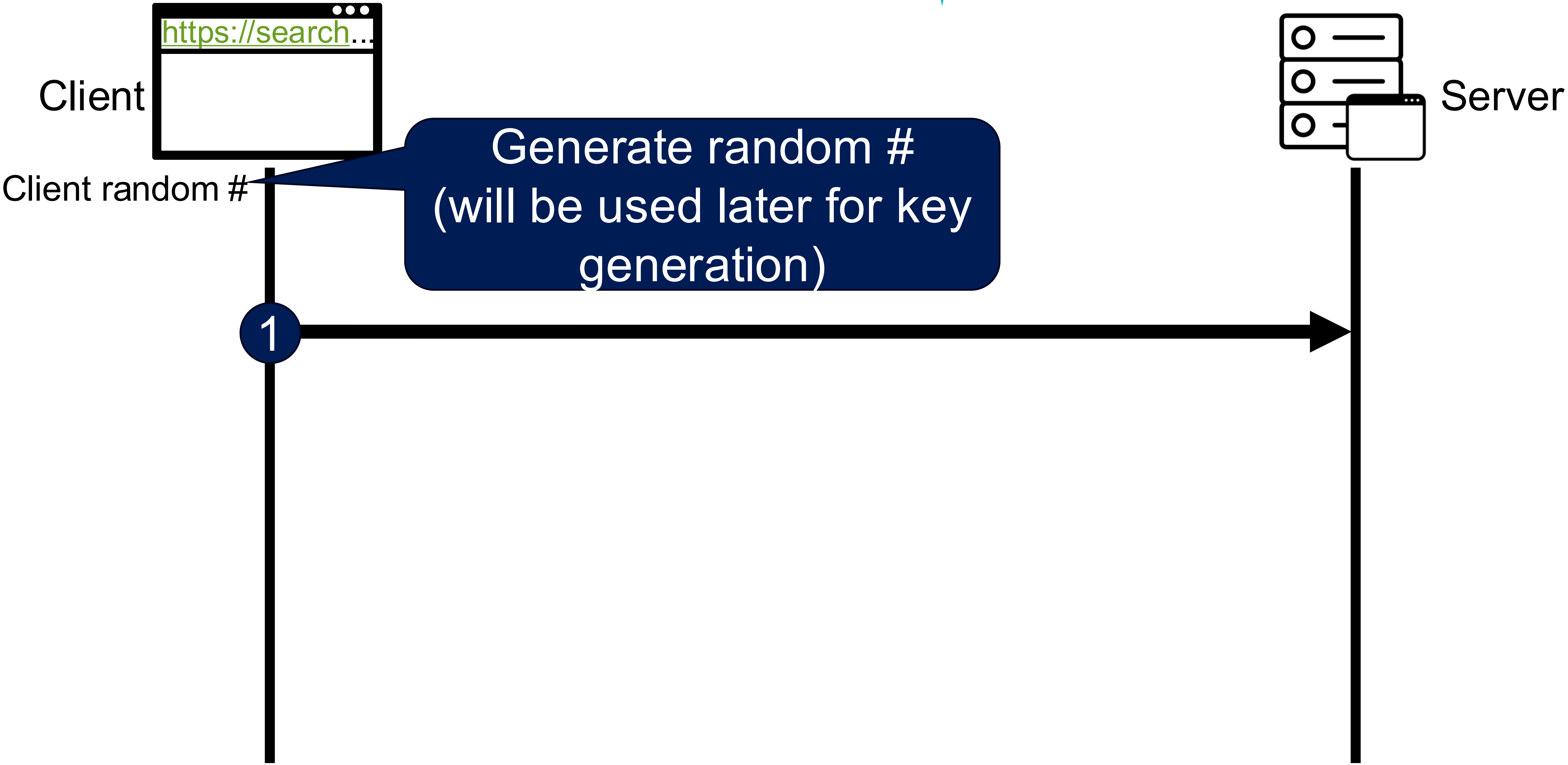
Phase 1:
Establishing security capabilities

Phase 2:
Server authentication and key exchange

Phase 3:
Client authentication and key exchange

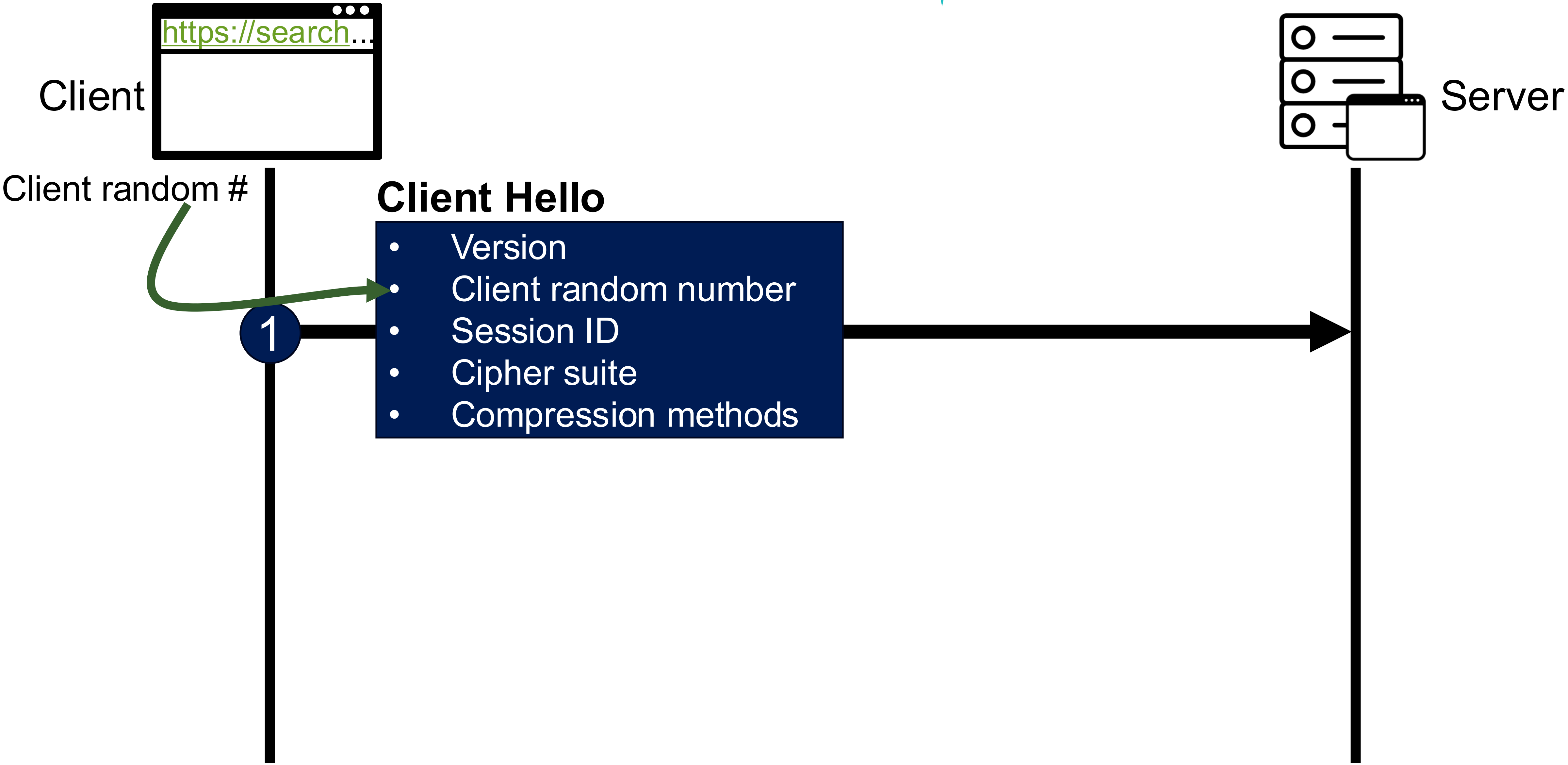
Phase 4:
Finalizing the handshake protocol

Phase 1: Establishing Security Capabilities



Generate random # (will be used later for key generation)

Phase 1: Establishing Security Capabilities



Client Hello

- Version
- Client random number
- Session ID
- Cipher suite
- Compression methods

Phase 1 – Client Hello – Details

Client Hello – Details

- **Version**
  - Highest protocol version supported by the client
- **Client random number**
  - Random 32 bit time stamp + 28 random bytes
  - It will be used later for key generation
- **Session ID**
  - 0: establish new connection on new session
  - Non-zero: resume an old session
- **Cipher suite**
  - Set of cryptographic algorithms supported by the client
- **Compression methods**
  - Sequence of compression methods

Cipher Suites

Client Hello – Details

- **Version**
  - Highest protocol version supported by the client
- **Client random number**
  - Random 32 bit time stamp + 28 random bytes
  - It will be used later for key generation
- **Session ID**
  - 0: establish new connection on new session
  - Non-zero: resume an old session
- **Cipher suite**
  - Set of cryptographic algorithms supported by the client
- **Compression methods**
  - Sequence of compression methods

Format:

```
TLS_RSA_WITH_AES_128_CBC_SHA
```

Cipher Suites

Client Hello – Details

- **Version**
  - Highest protocol version supported by the client
- **Client random number**
  - Random 32 bit time stamp +
  - It will be used later for key generation
- **Session ID**
  - 0: establish
  - Non-zero: re
- **Cipher suite**
  - Set of cryptographic algorithms supported by the client
- **Compression methods**
  - Sequence of compression methods

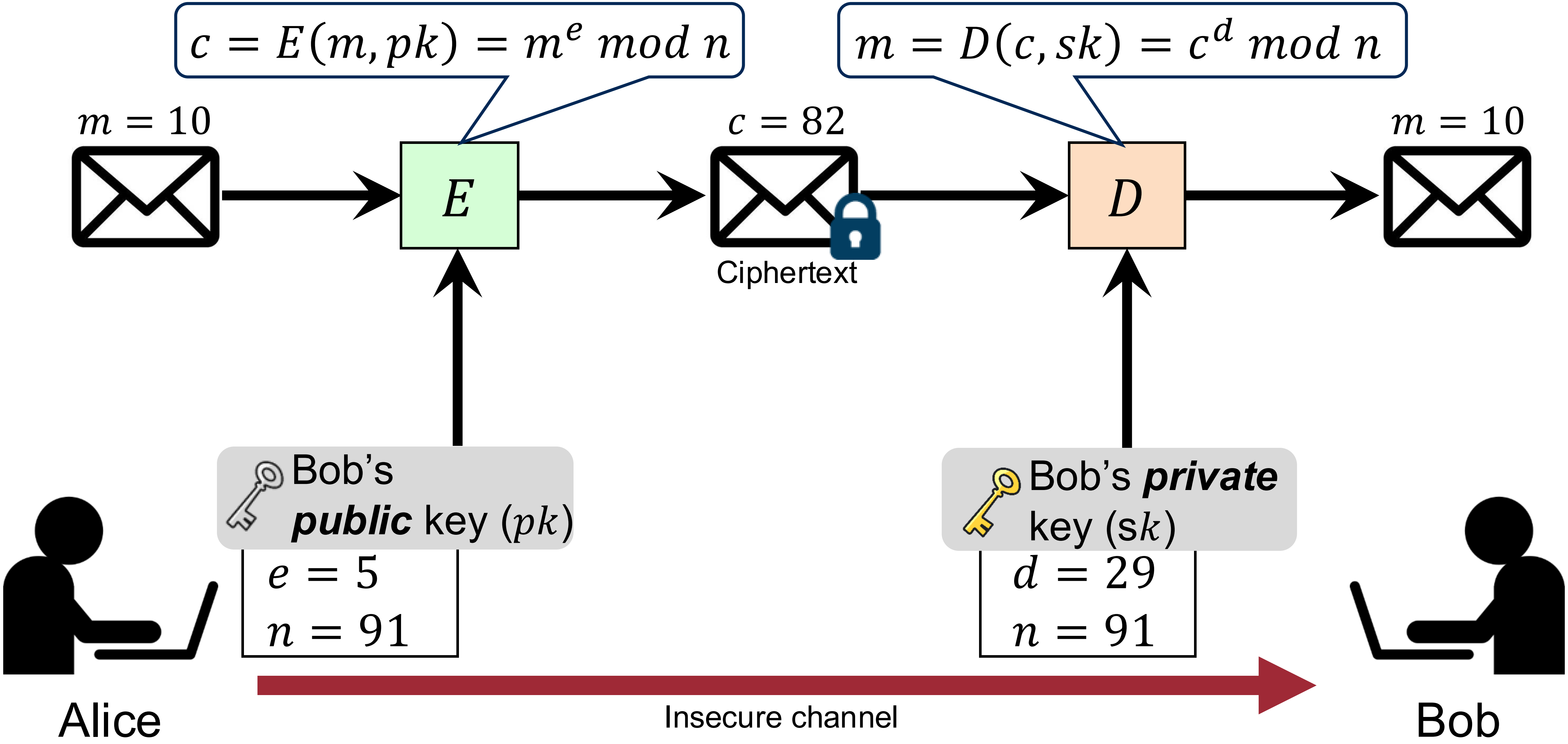
Format:

```
TLS_RSA_WITH_AES_128_CBC_SHA
```

Protocol

(Asymmetric) Encryption/decryption algorithm (for handshake protocol)

Ref: RSA Algorithm


$c = E(m, pk) = m^e \ mod \ n$
$m = D(c, sk) = c^d \ mod \ n$
$m = 10$
E
$c = 82$
Ciphertext
D
$m = 10$
Bob's **public** key (pk)
$e = 5$
$n = 91$
Bob's **private** key (sk)
$d = 29$
$n = 91$
Alice
Insecure channel
Bob

Cipher Suites

Client Hello – Details

- **Version**
  - Highest protocol version supported by the client
- **Client random number**
  - Random 32 bit time stamp +
  - It will be used later for key generation
- **Session ID**
  - 0: establish
  - Non-zero: re
- **Cipher suite**
  - Set of cryptographic algorithm
    the client
- **Compression methods**
  - Sequence of compression me

Format:

`TLS_RSA_WITH_AES_128_CBC_SHA`

Protocol

(Asymmetric)
Encryption/decryption algorithm
(for handshake protocol)

(Symmetric)
Encryption/decryption algorithm
(for record protocol)

Ref: Symmetric Key Cryptography

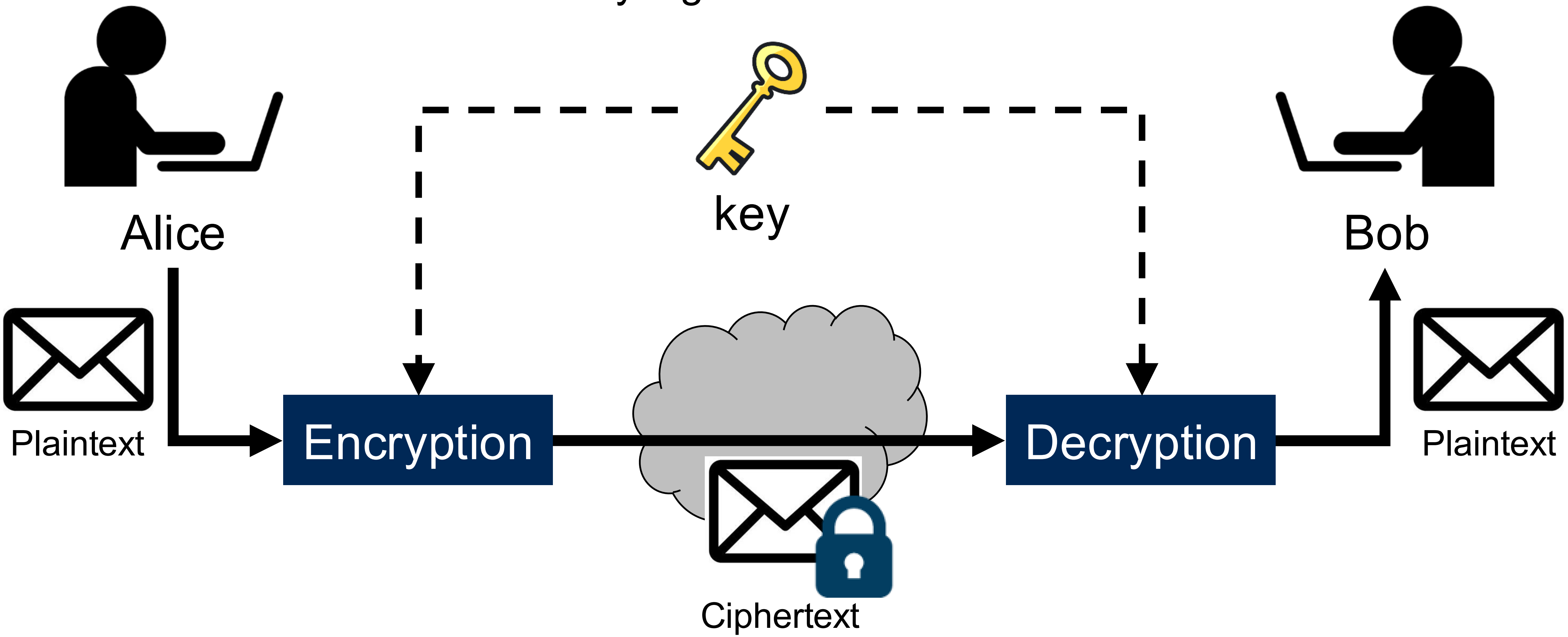
- The same key is used to encrypt/decrypt messages
  - Also known as secret key algorithm

Alice
key
Bob

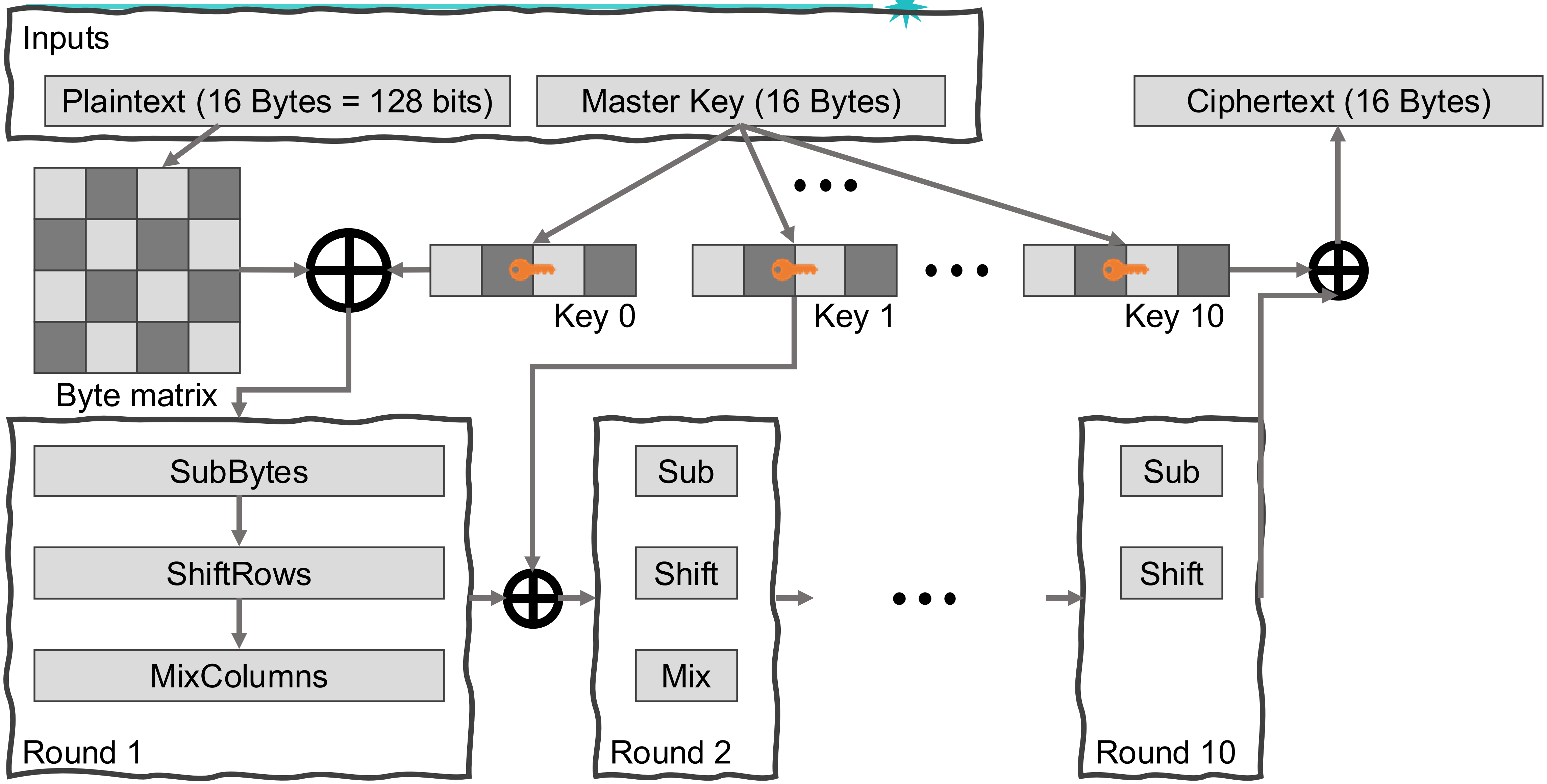
Shared secret key

Ref: Symmetric Key Cryptography

- The same key is used to encrypt/decrypt messages
  - Also known as secret key algorithm



Ref: Advanced Encryption Standard (AES)



Cipher Suites

Client Hello – Details

- **Version**
  - Highest protocol version supported by the client
- **Client random number**
  - Random 32 bit time stamp +
  - It will be used later for key generation
- **Session ID**
  - 0: establish
  - Non-zero: re
- **Cipher suite**
  - Set of cryptographic algorithm
    the client
- **Compression methods**
  - Sequence of compression me

Format:

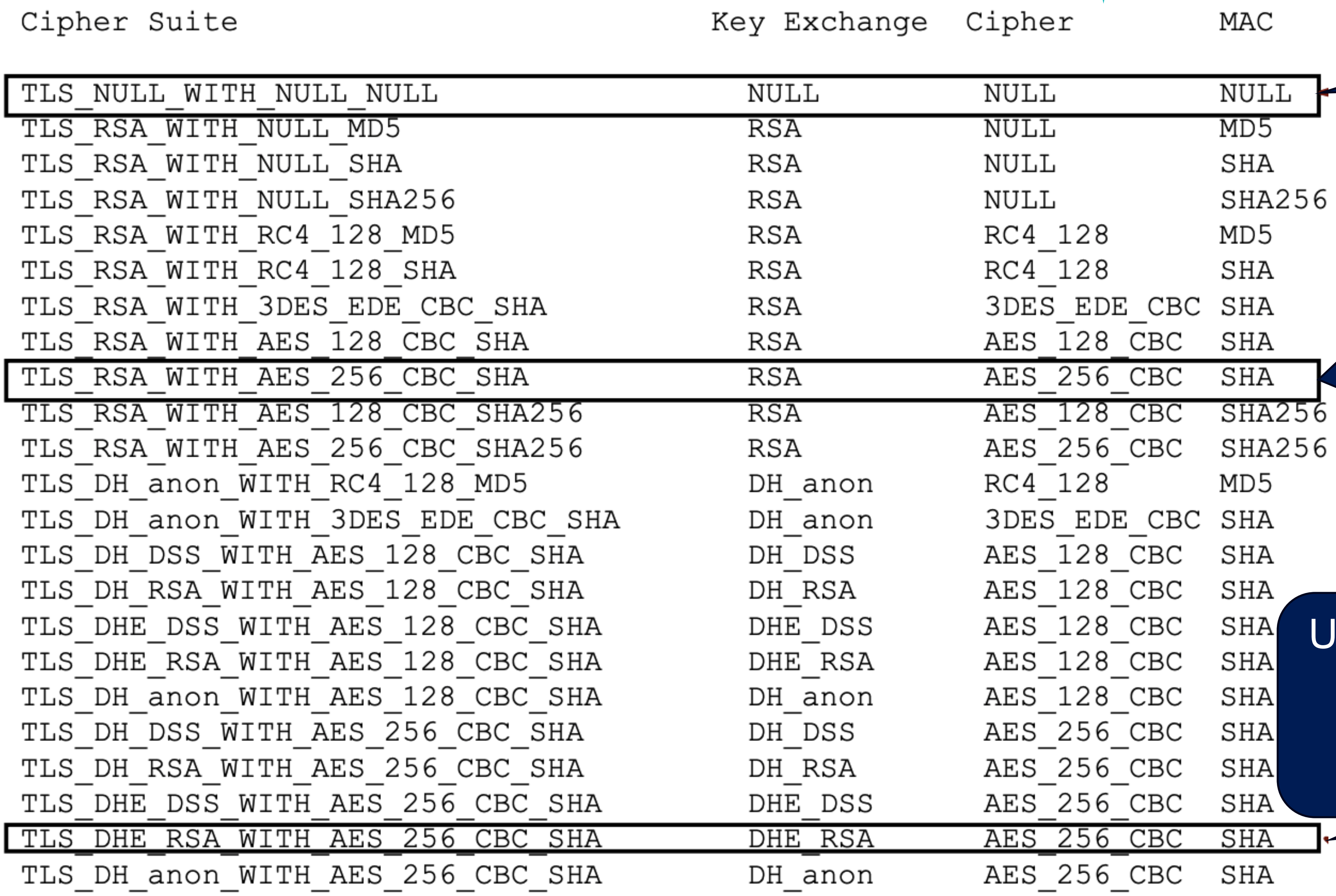
`TLS_RSA_WITH_AES_128_CBC_SHA`

Protocol

(Asymmetric) Encryption/decryption algorithm (for key exchange)

(Symmetric) Encryption/decryption algorithm (for data exchange)

Cipher Suite – Example



Cipher Suite	Key Exchange	Cipher	MAC
TLS_NULL_WITH_NULL_NULL	NULL	NULL	NULL
TLS_RSA_WITH_NULL_MD5	RSA	NULL	MD5
TLS_RSA_WITH_NULL_SHA	RSA	NULL	SHA
TLS_RSA_WITH_NULL_SHA256	RSA	NULL	SHA256
TLS_RSA_WITH_RC4_128_MD5	RSA	RC4_128	MD5
TLS_RSA_WITH_RC4_128_SHA	RSA	RC4_128	SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA	3DES_EDE_CBC	SHA
TLS_RSA_WITH_AES_128_CBC_SHA	RSA	AES_128_CBC	SHA
TLS_RSA_WITH_AES_256_CBC_SHA	RSA	AES_256_CBC	SHA
TLS_RSA_WITH_AES_128_CBC_SHA256	RSA	AES_128_CBC	SHA256
TLS_RSA_WITH_AES_256_CBC_SHA256	RSA	AES_256_CBC	SHA256
TLS_DH_anon_WITH_RC4_128_MD5	DH_anon	RC4_128	MD5
TLS_DH_anon_WITH_3DES_EDE_CBC_SHA	DH_anon	3DES_EDE_CBC	SHA
TLS_DH_DSS_WITH_AES_128_CBC_SHA	DH_DSS	AES_128_CBC	SHA
TLS_DH_RSA_WITH_AES_128_CBC_SHA	DH_RSA	AES_128_CBC	SHA
TLS_DHE_DSS_WITH_AES_128_CBC_SHA	DHE_DSS	AES_128_CBC	SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA	DHE_RSA	AES_128_CBC	SHA
TLS_DH_anon_WITH_AES_128_CBC_SHA	DH_anon	AES_128_CBC	SHA
TLS_DH_DSS_WITH_AES_256_CBC_SHA	DH_DSS	AES_256_CBC	SHA
TLS_DH_RSA_WITH_AES_256_CBC_SHA	DH_RSA	AES_256_CBC	SHA
TLS_DHE_DSS_WITH_AES_256_CBC_SHA	DHE_DSS	AES_256_CBC	SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA	DHE_RSA	AES_256_CBC	SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA	DH_anon	AES_256_CBC	SHA

No protection

Uses RSA (certificate) for key exchange, AES 256 in CBC mode for encryption and SHA256 as MAC

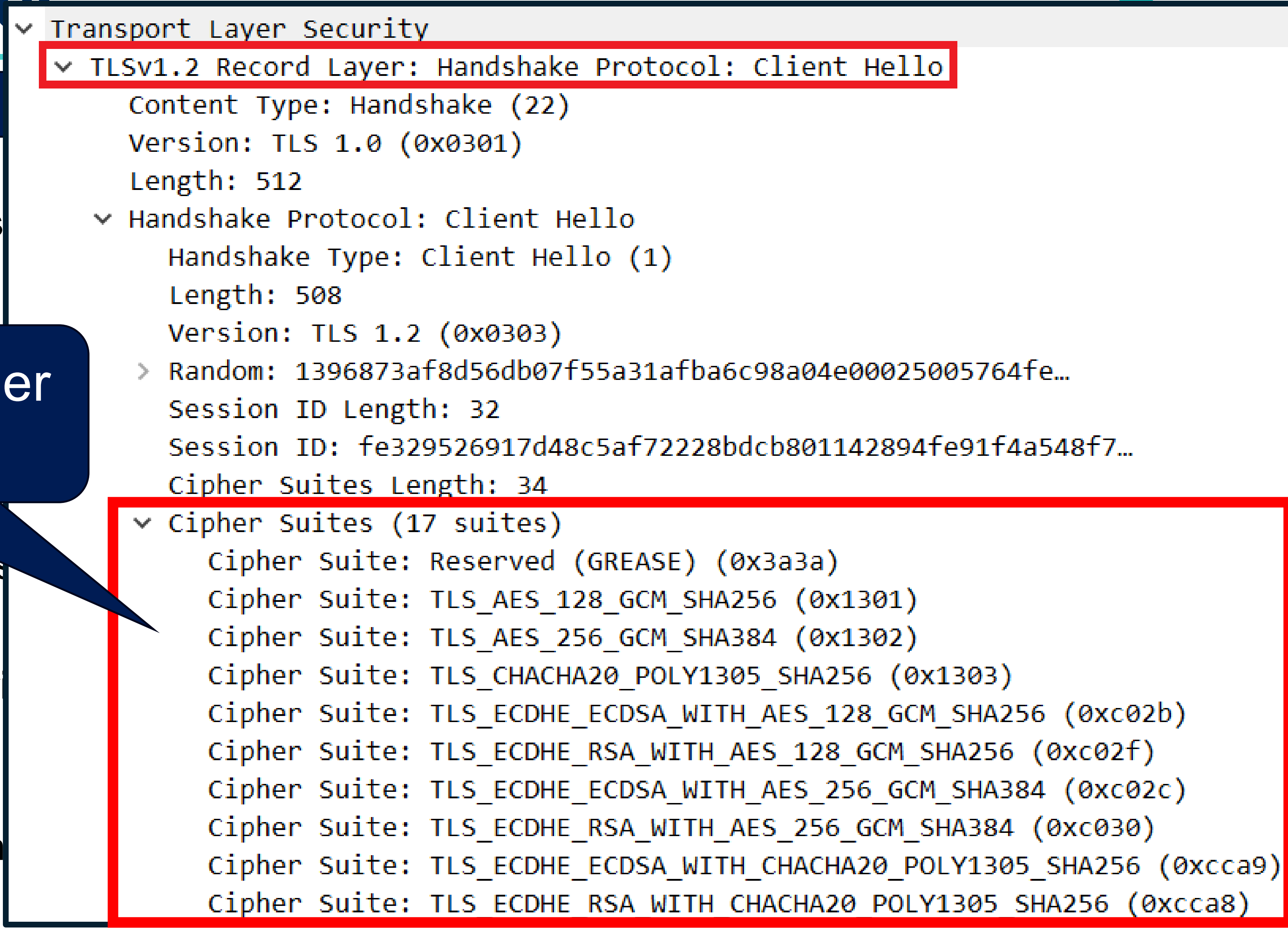
Uses ephemeral Diffie- Hellman with RSA for key exchange, AES 256 CBC for encryption and SHA256 as MAC

Cipher Suites

Client Hello –

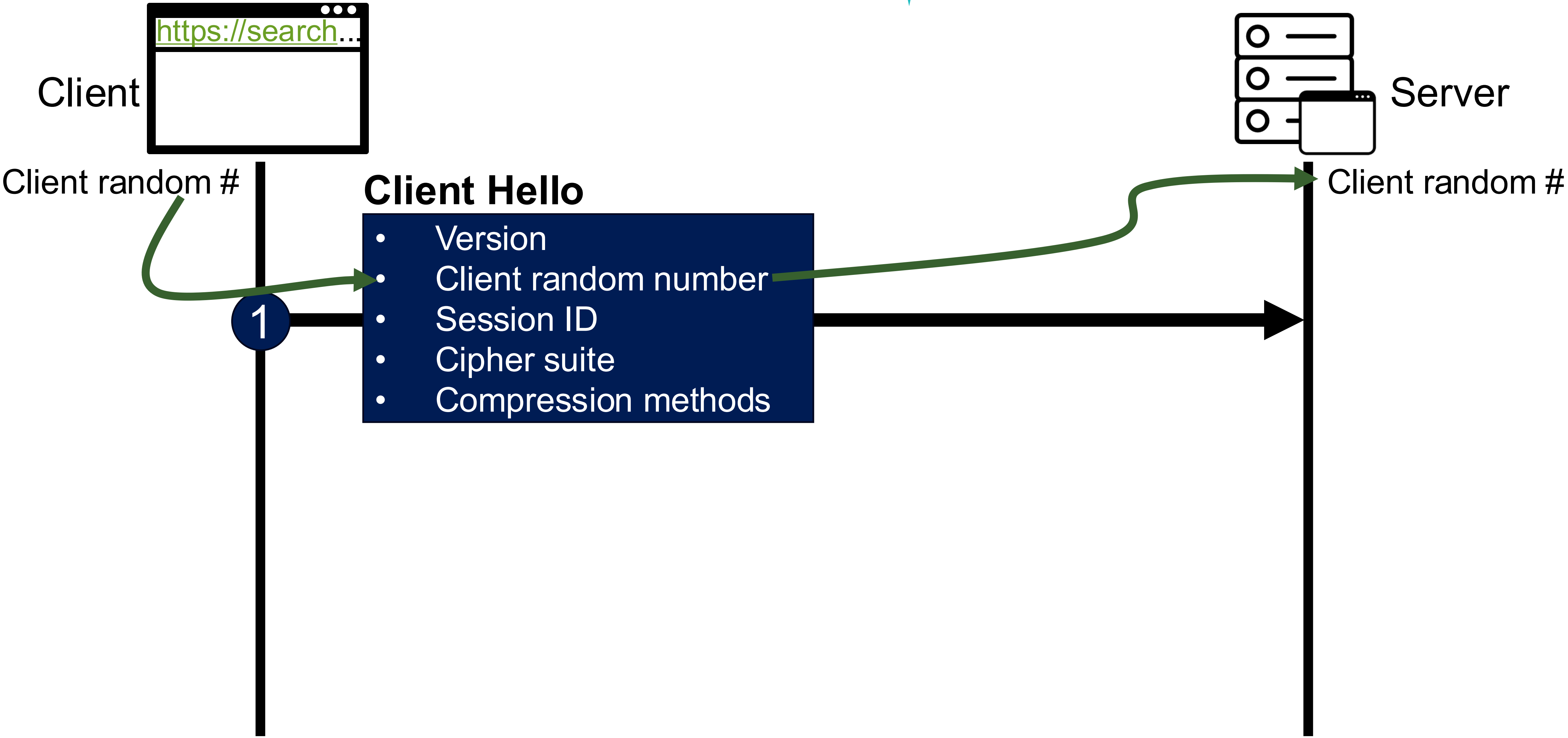
- **Version**
  - Highest protocol version s
- **Client random number**
  -
  -
- **Se**
  - 0: establish new co
  - Non-zero: resume an old s
- **Cipher suite**
  - Set of cryptographic algori the client
- **Compression methods**
  - Sequence of compression

In decreasing order of preference



```
v Transport Layer Security
  v TLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 512
    v Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 508
        Version: TLS 1.2 (0x0303)
      > Random: 1396873af8d56db07f55a31afba6c98a04e00025005764fe…
        Session ID Length: 32
        Session ID: fe329526917d48c5af72228bdcb801142894fe91f4a548f7…
        Cipher Suites Length: 34
      v Cipher Suites (17 suites)
          Cipher Suite: Reserved (GREASE) (0x3a3a)
          Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
          Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
          Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
          Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
          Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
          Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
```

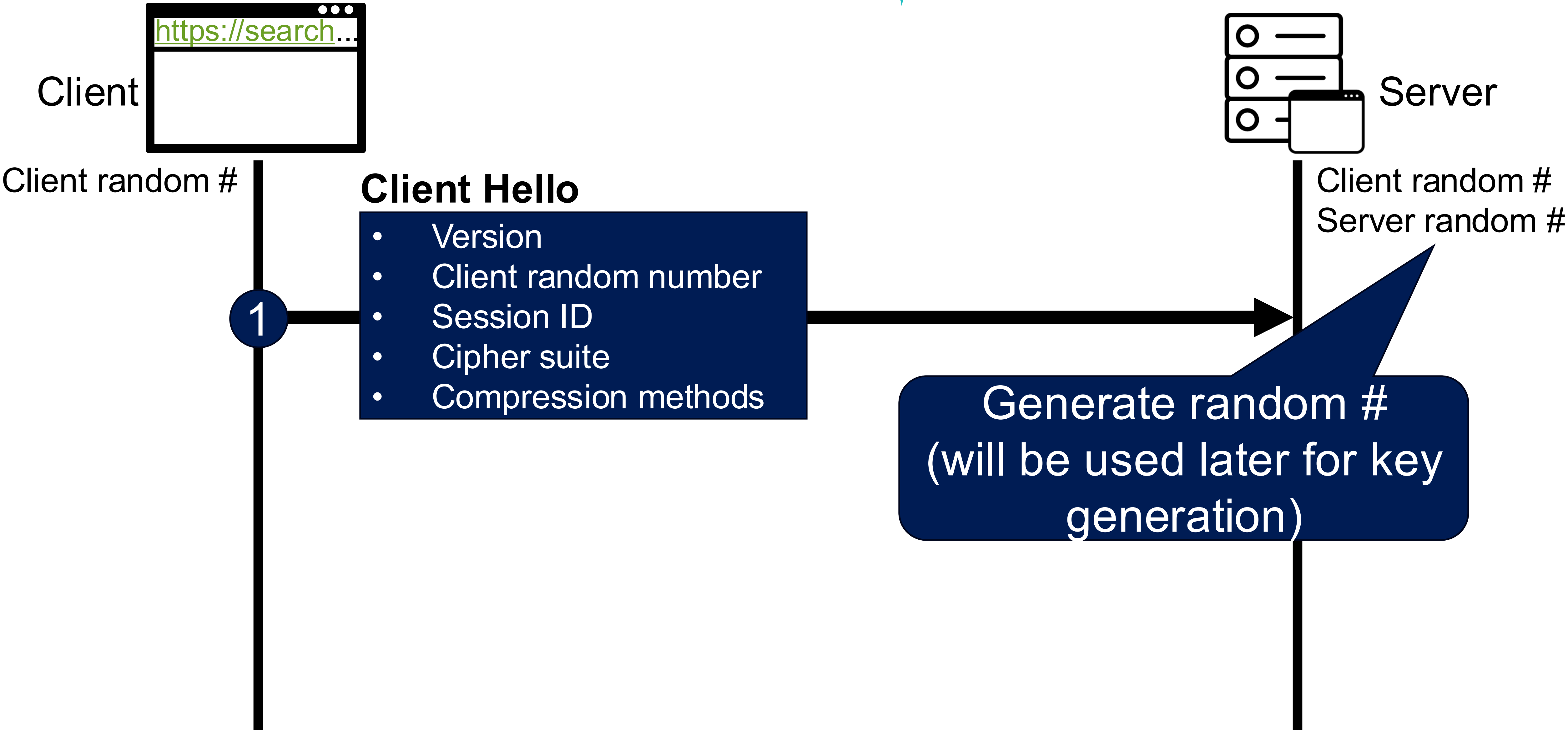
Phase 1: Establishing Security Capabilities



Client Hello

- Version
- Client random number
- Session ID
- Cipher suite
- Compression methods

Phase 1: Establishing Security Capabilities

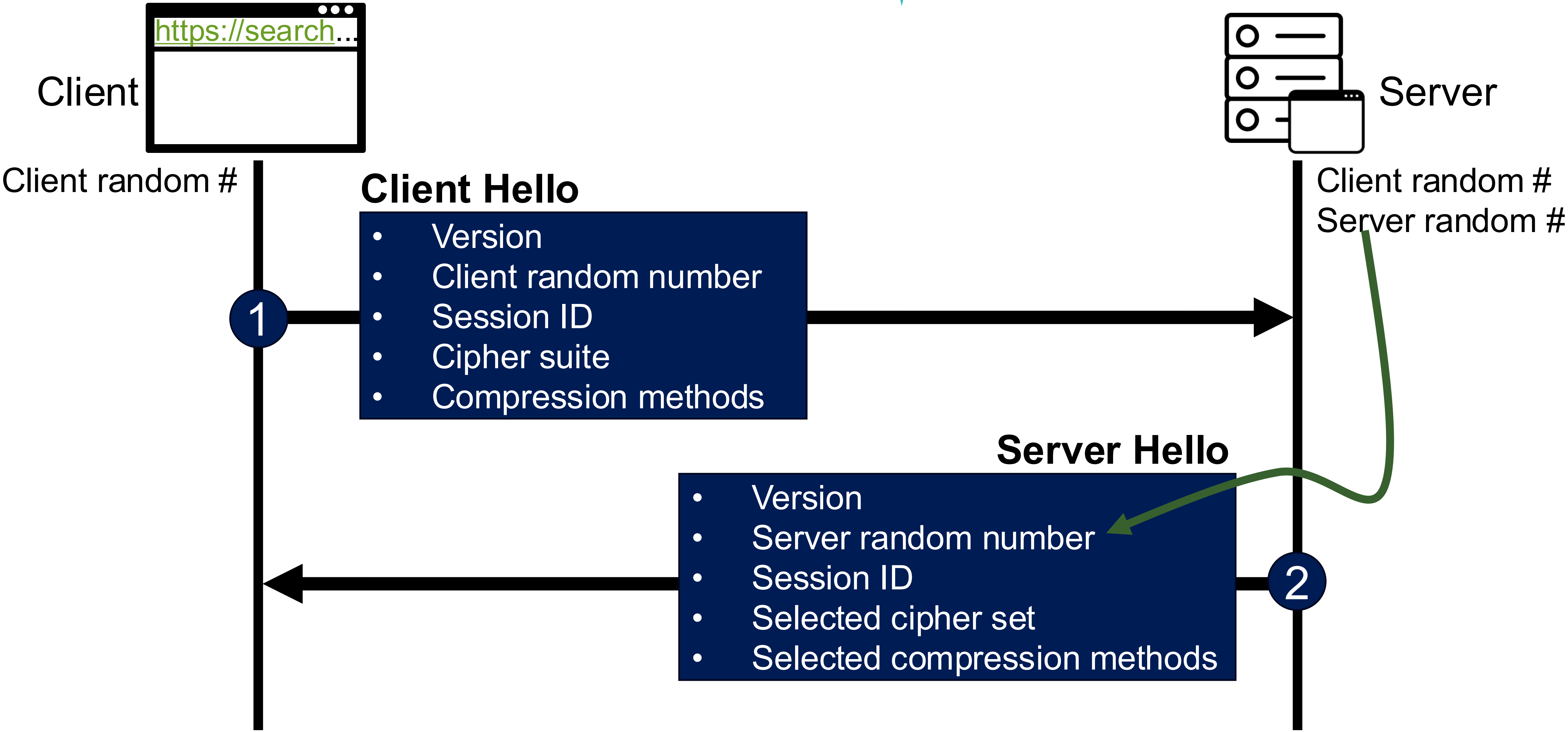


Client Hello

- Version
- Client random number
- Session ID
- Cipher suite
- Compression methods

Generate random # (will be used later for key generation)

Phase 1: Establishing Security Capabilities



Client

https://search...

Client random #

1

Client Hello

- Version
- Client random number
- Session ID
- Cipher suite
- Compression methods

Server

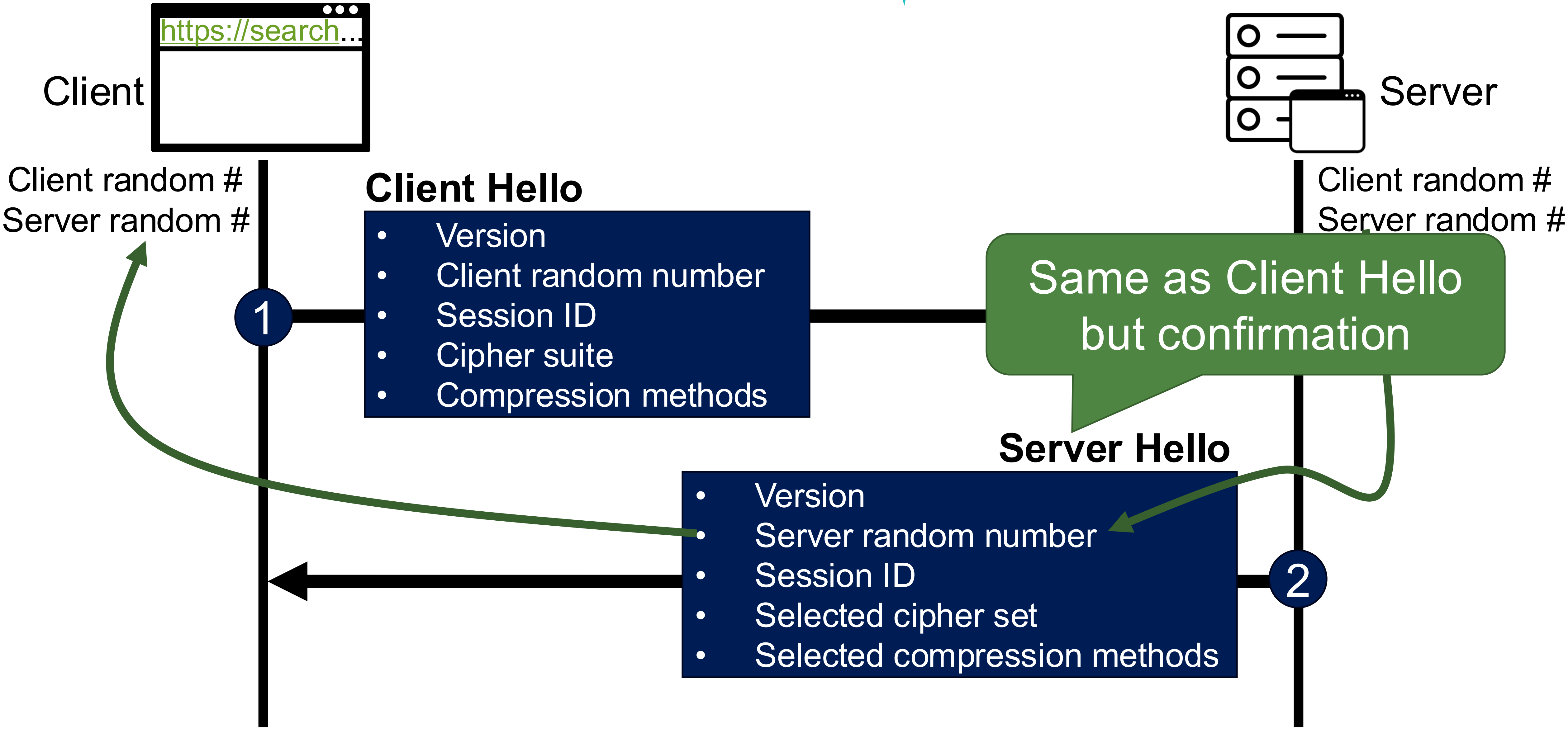
Client random #
Server random #

Server Hello

- Version
- Server random number
- Session ID
- Selected cipher set
- Selected compression methods

2

Phase 1: Establishing Security Capabilities



Client random #
Server random #

Client random #
Server random #

Client Hello

- Version
- Client random number
- Session ID
- Cipher suite
- Compression methods

Same as Client Hello but confirmation

Server Hello

- Version
- Server random number
- Session ID
- Selected cipher set
- Selected compression methods

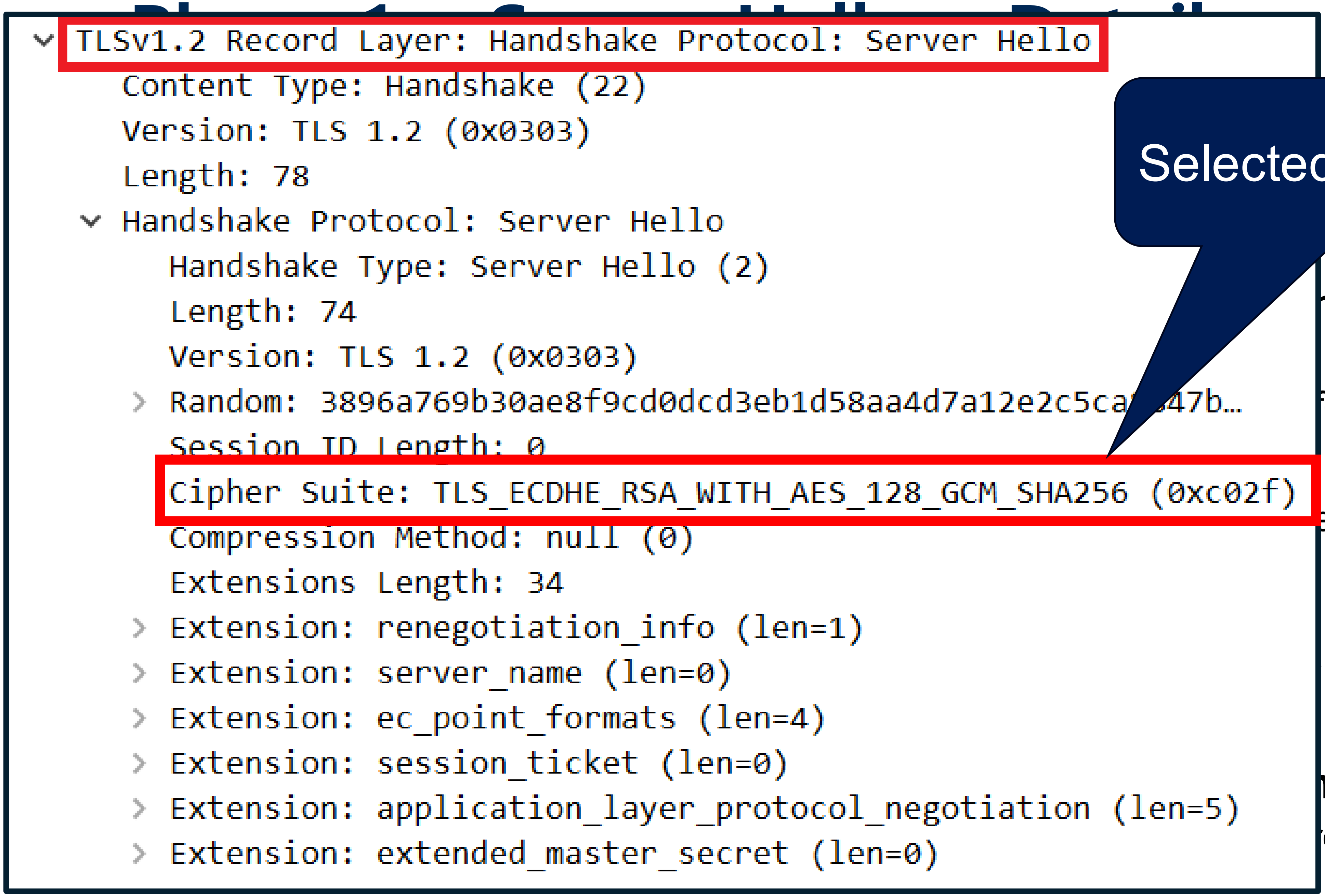
Phase 1 – Server Hello – Details

Client Hello – Details

- **Version**
  - Highest protocol version supported by the client
- **Client random number**
  - Random 32 bit time stamp + 28 random bytes
  - It will be used later for key generation
- **Session ID**
  - 0: establish new connection on new session
  - Non-zero: resume an old session
- **Cipher suite**
  - Set of cryptographic algorithms supported by the client
- **Compression methods**
  - Sequence of compression methods

Server Hello – Details

- **Version**
  - Highest common version
- **Server random number**
  - Random 32 bit time stamp + 28 random bytes
  - It will be used later for key generation
- **Session ID**
  - New session ID if zero, old session ID otherwise
- **Cipher suite**
  - The selected cipher suite
- **Compression methods**
  - The selected compression technique



```
⌄ TLSv1.2 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 78
  ⌄ Handshake Protocol: Server Hello
      Handshake Type: Server Hello (2)
      Length: 74
      Version: TLS 1.2 (0x0303)
    > Random: 3896a769b30ae8f9cd0dcd3eb1d58aa4d7a12e2c5ca 47b…
      Session ID Length: 0
      Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
      Compression Method: null (0)
      Extensions Length: 34
    > Extension: renegotiation_info (len=1)
    > Extension: server_name (len=0)
    > Extension: ec_point_formats (len=4)
    > Extension: session_ticket (len=0)
    > Extension: application_layer_protocol_negotiation (len=5)
    > Extension: extended_master_secret (len=0)
```

Selected cipher suite

rsion

mber

stamp + 28 random bytes

for key generation

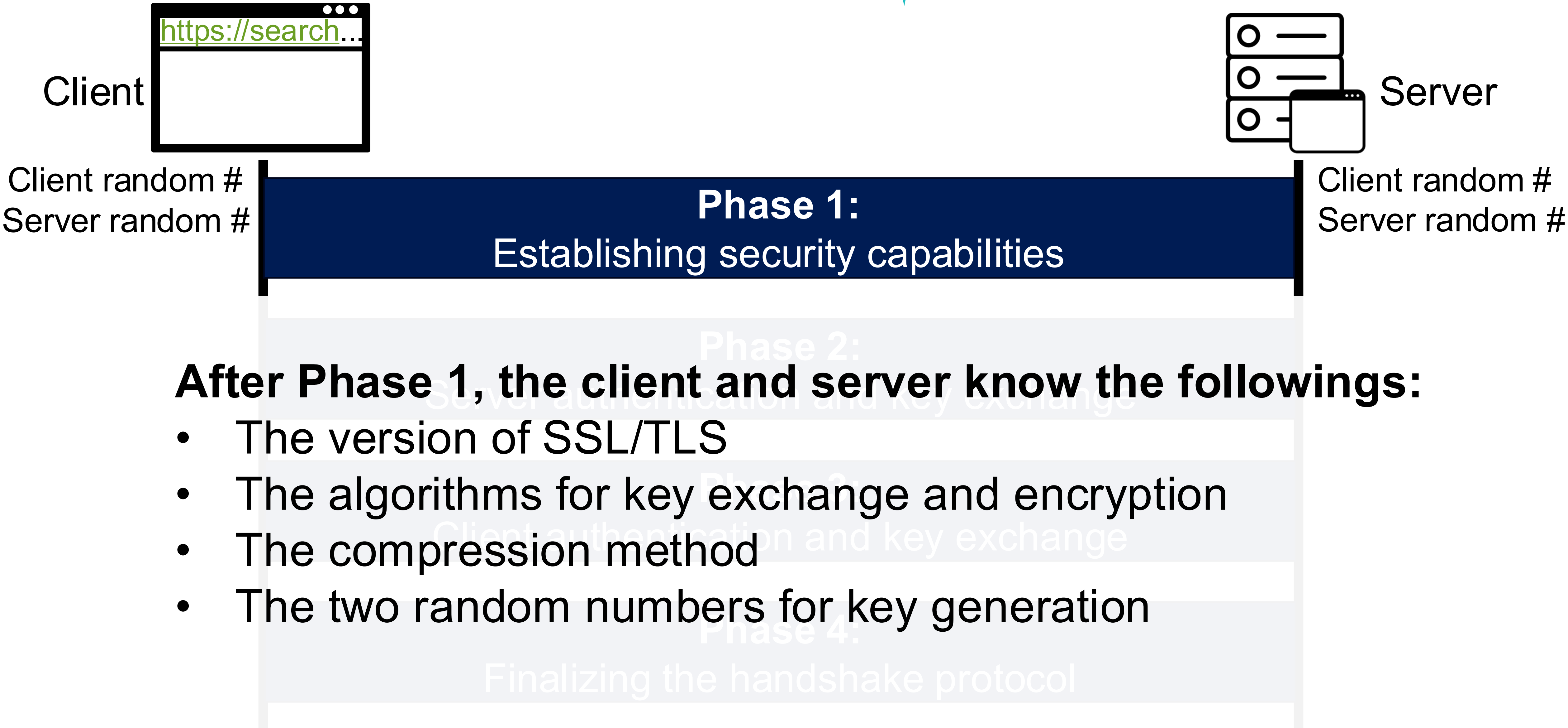
ero, old session ID

suite

hods

ession technique

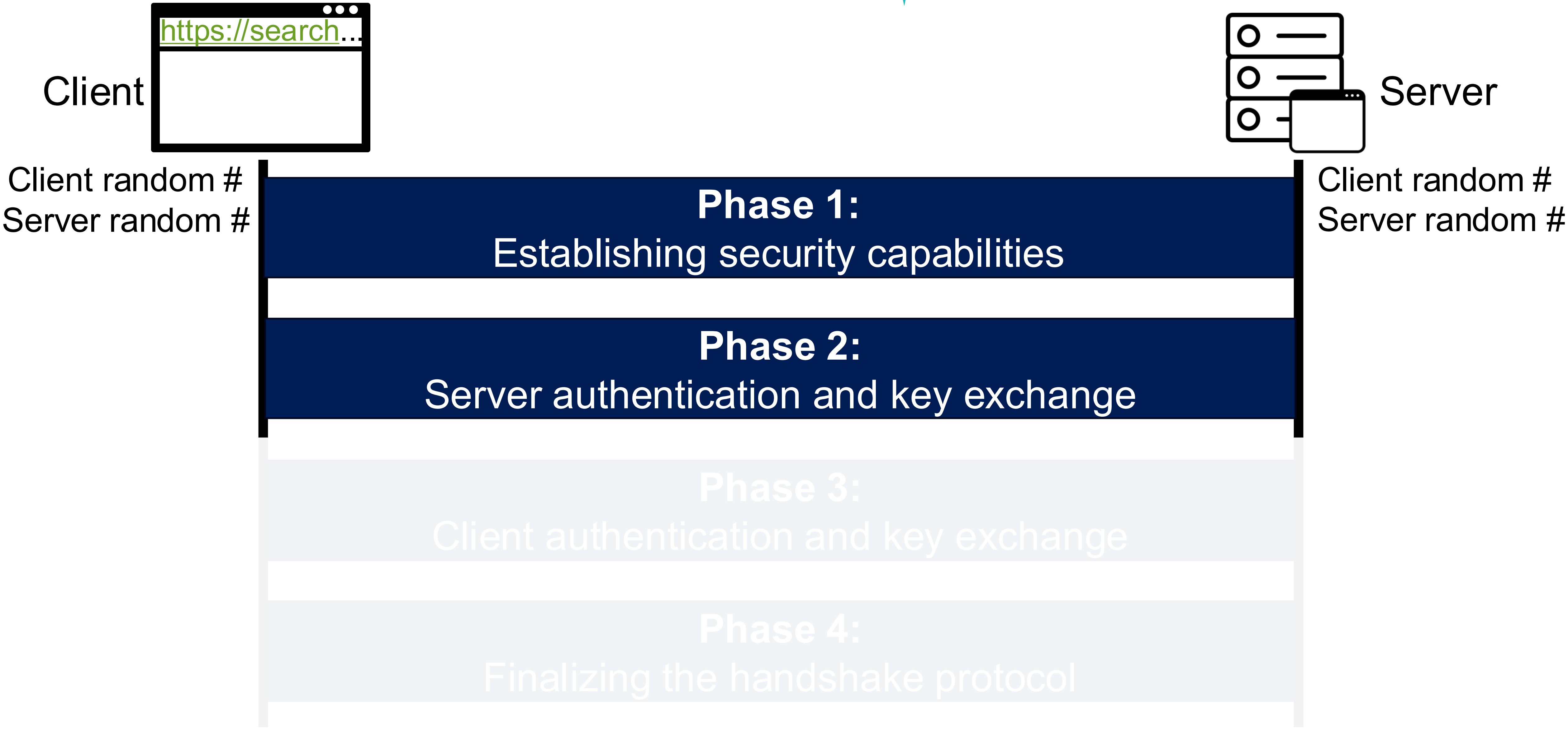
Phase 1: Establishing Security Capabilities


https://search...
Client
Server
Client random #
Server random #
Phase 1:
Establishing security capabilities
Client random #
Server random #
Phase 2:
Server authentication and key exchange
Phase 3:
Client authentication and key exchange
Phase 4:
Finalizing the handshake protocol

After Phase 1, the client and server know the followings:

- The version of SSL/TLS
- The algorithms for key exchange and encryption
- The compression method
- The two random numbers for key generation

Phase 2: Server Auth. and Key Exchange



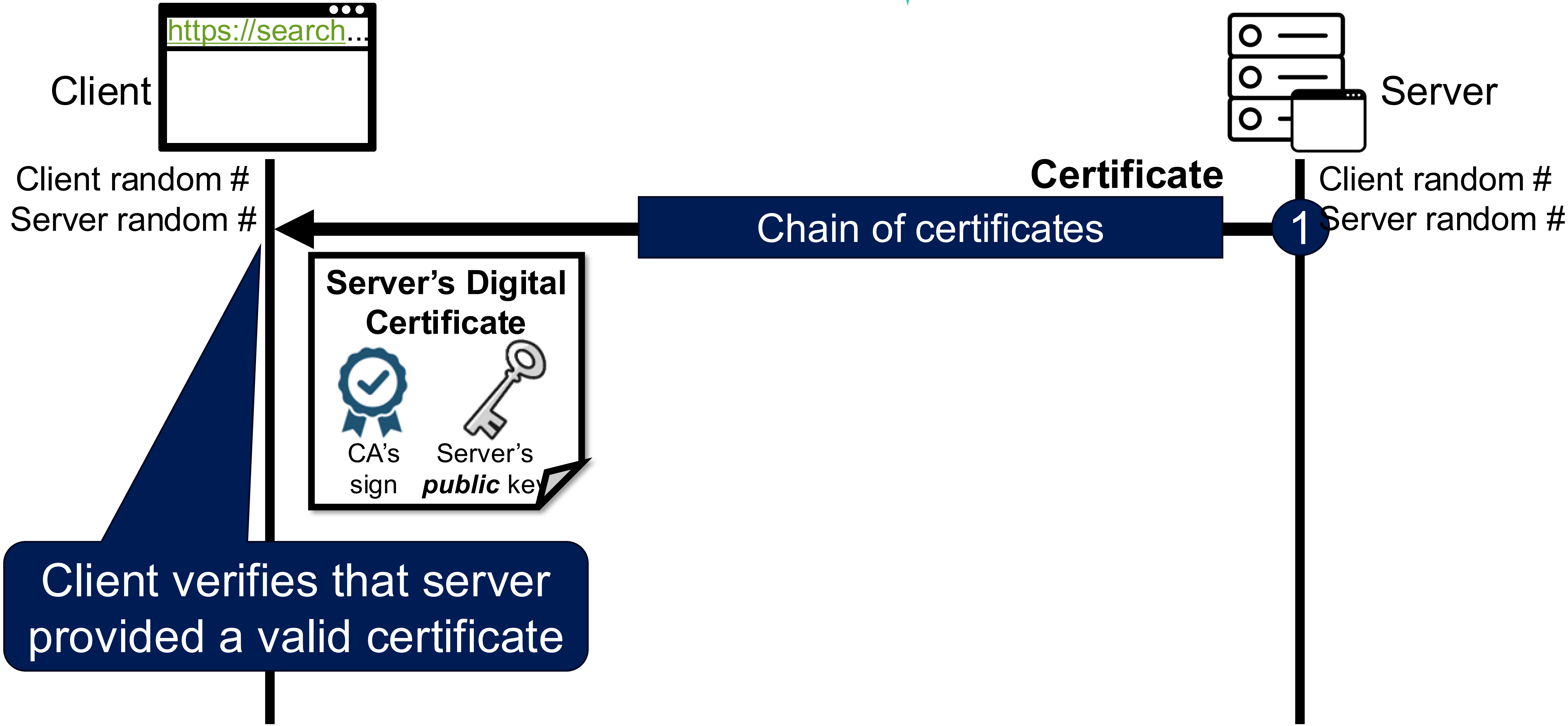
Phase 1:
Establishing security capabilities

Phase 2:
Server authentication and key exchange

Phase 3:
Client authentication and key exchange

Phase 4:
Finalizing the handshake protocol

Phase 2: Server Auth. and Key Exchange



Client

https://search...

Client random #
Server random #

Server

Client random #
Server random #

Certificate

1 Chain of certificates

Server's Digital Certificate

CA's sign

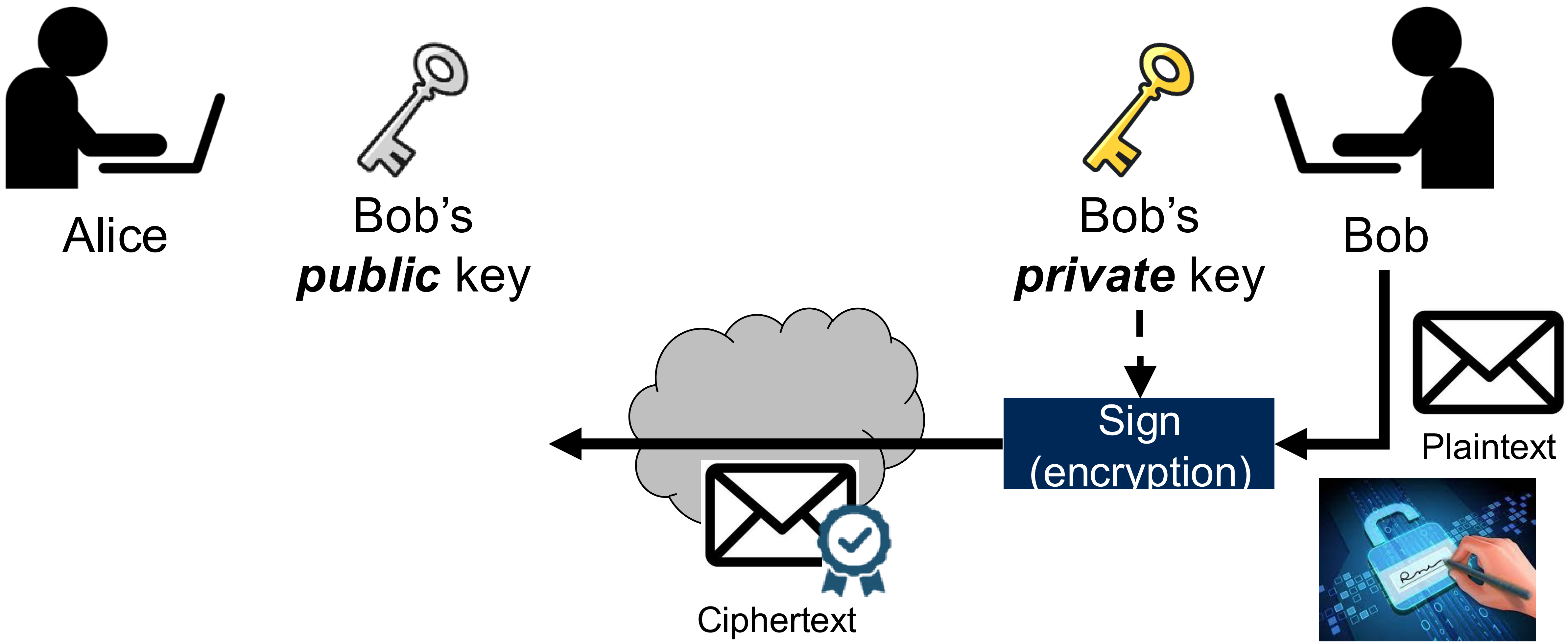
Server's *public* key

Client verifies that server provided a valid certificate

Digital Certificate

- A document certifying that the public key included inside does **belong to the identity described in the document**

Digital Signature



Alice

Bob's
public key

Bob's
private key

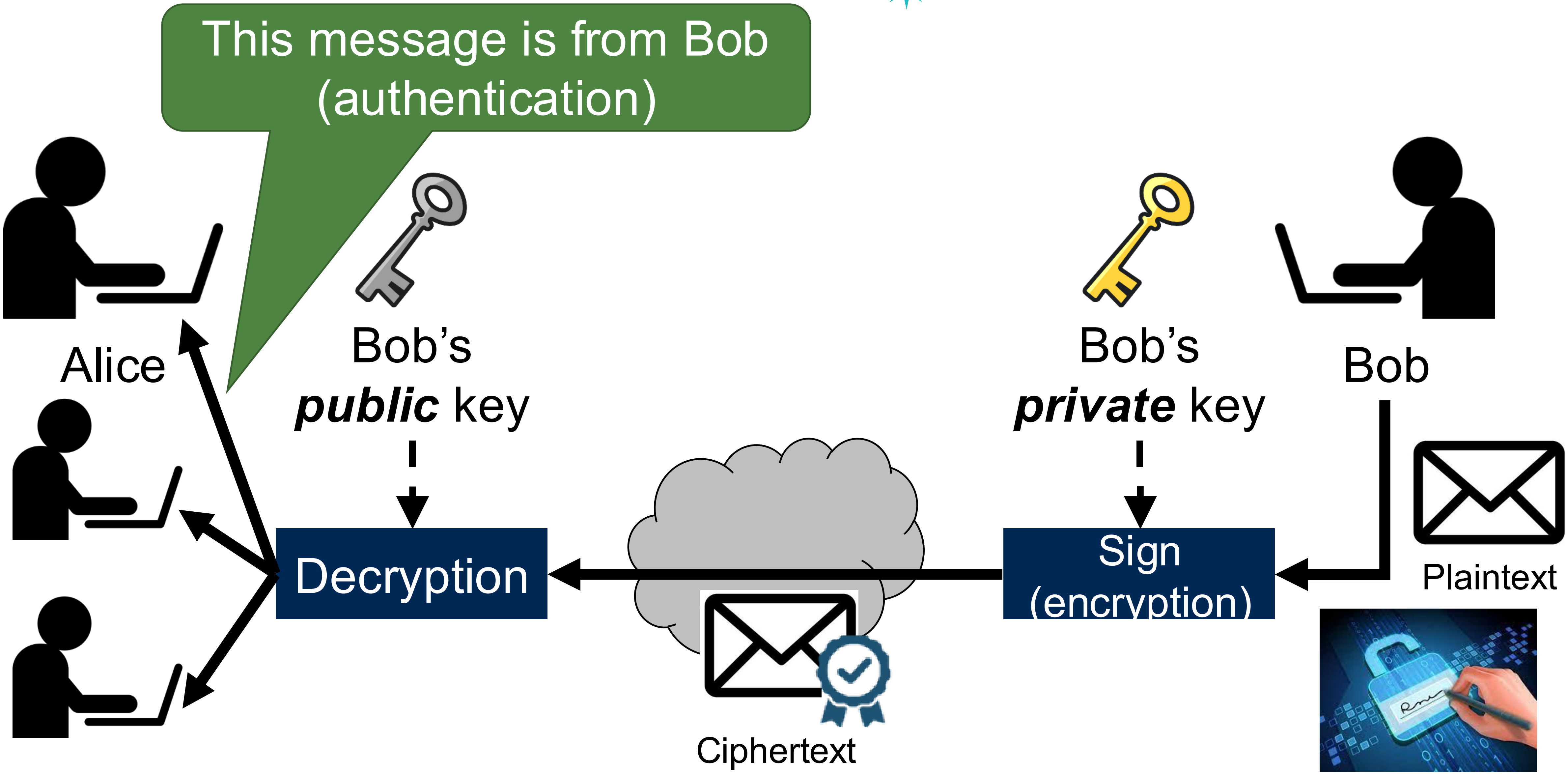
Bob

Sign
(encryption)

Plaintext

Ciphertext

Digital Signature



This message is from Bob (authentication)

Alice

Bob's *public* key

Decryption

Ciphertext

Sign (encryption)

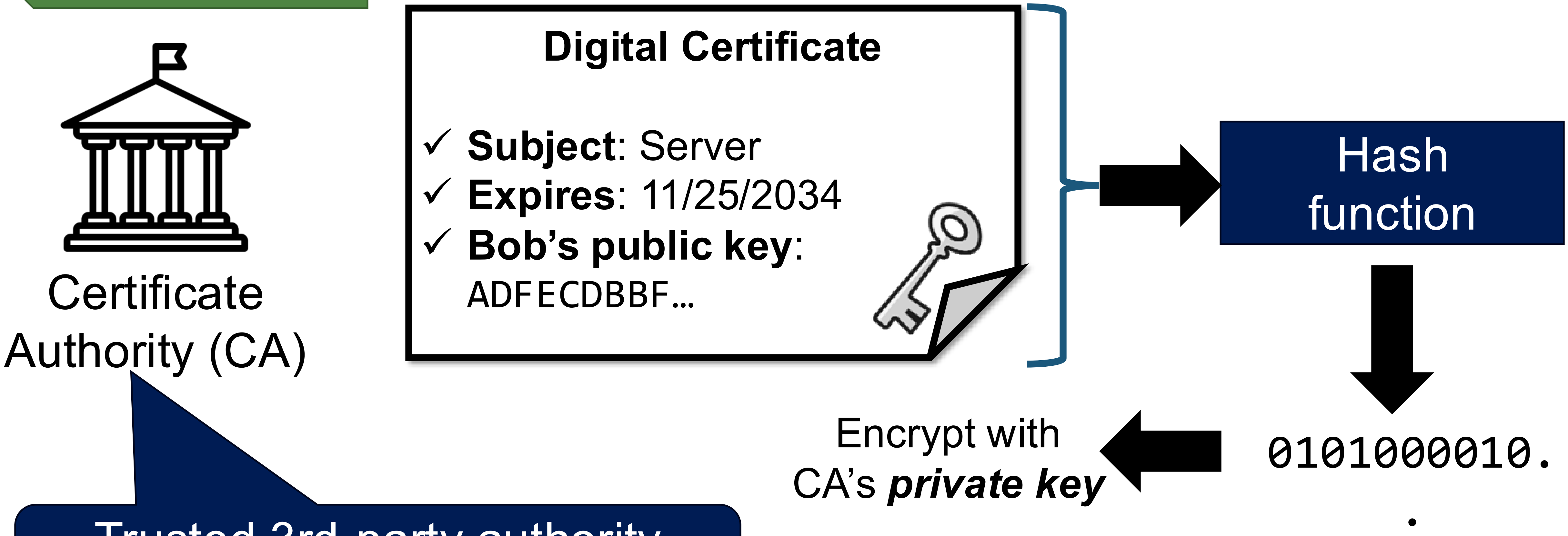
Bob's *private* key

Bob

Plaintext

Digital Certificate

Signing



Certificate Authority (CA)

Digital Certificate

- ✓ **Subject**: Server
- ✓ **Expires**: 11/25/2034
- ✓ **Bob's public key**: ADFECDBBF…

Hash function

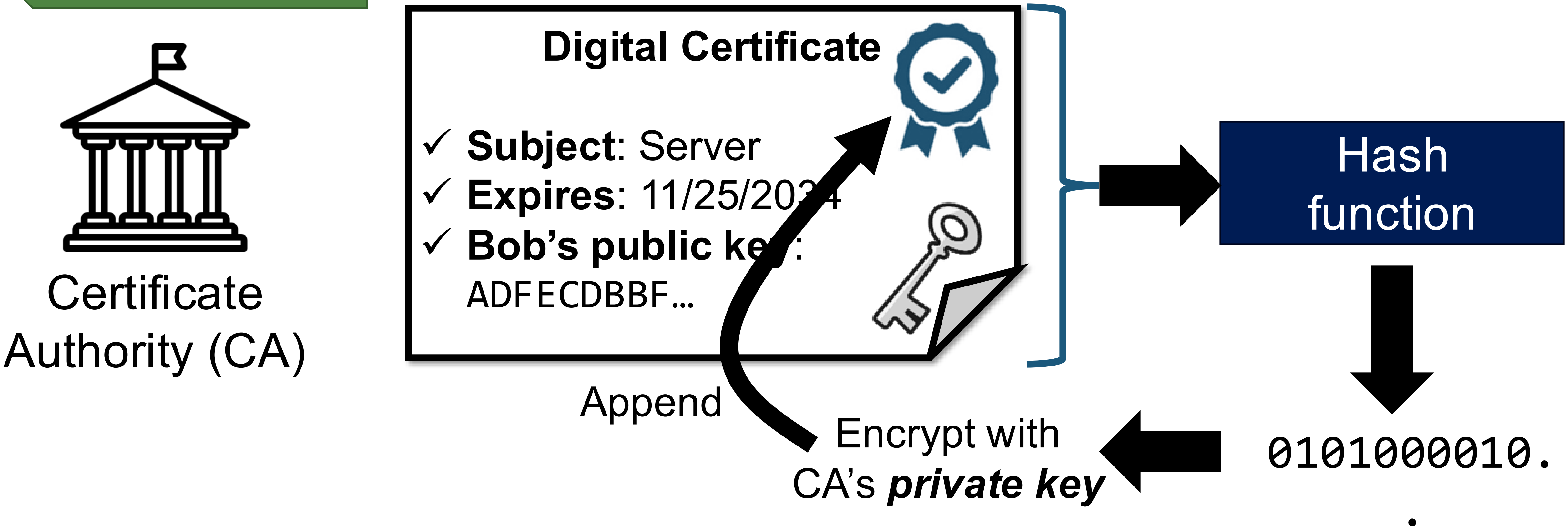
0101000010.
.

Encrypt with CA's ***private key***

Trusted 3rd-party authority (KISA, yesSign, Verisign …)

Digital Certificate

Signing



Certificate Authority (CA)

Digital Certificate

- ✓ **Subject**: Server
- ✓ **Expires**: 11/25/2024
- ✓ **Bob's public key**:
ADFECDBBF…

Hash function

0101000010. .

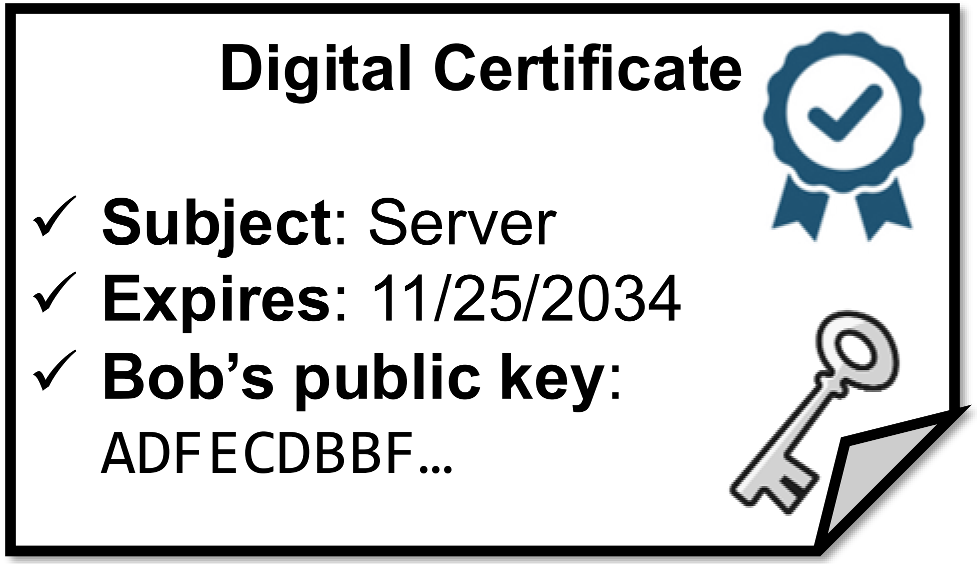
Encrypt with CA's ***private key***

Append

Hash-based Digital Signature

Verification

Alice

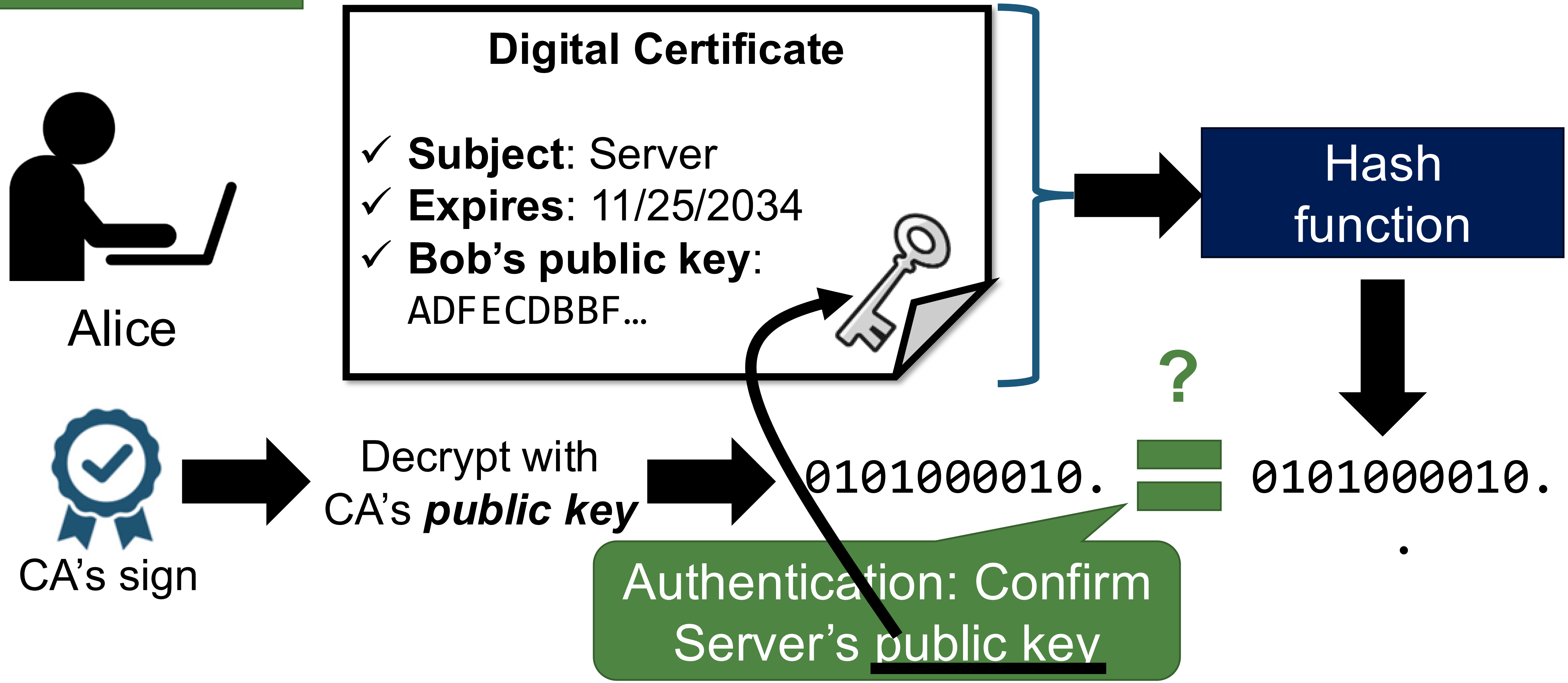


Digital Certificate

- ✓ **Subject**: Server
- ✓ **Expires**: 11/25/2034
- ✓ **Bob’s public key**: ADFECDBBF…

Hash-based Digital Signature

Verification



Alice

Digital Certificate

✓ **Subject**: Server
✓ **Expires**: 11/25/2034
✓ **Bob's public key**:
ADFECDBBF…

Hash function

CA's sign

Decrypt with CA's ***public key***

0101000010.

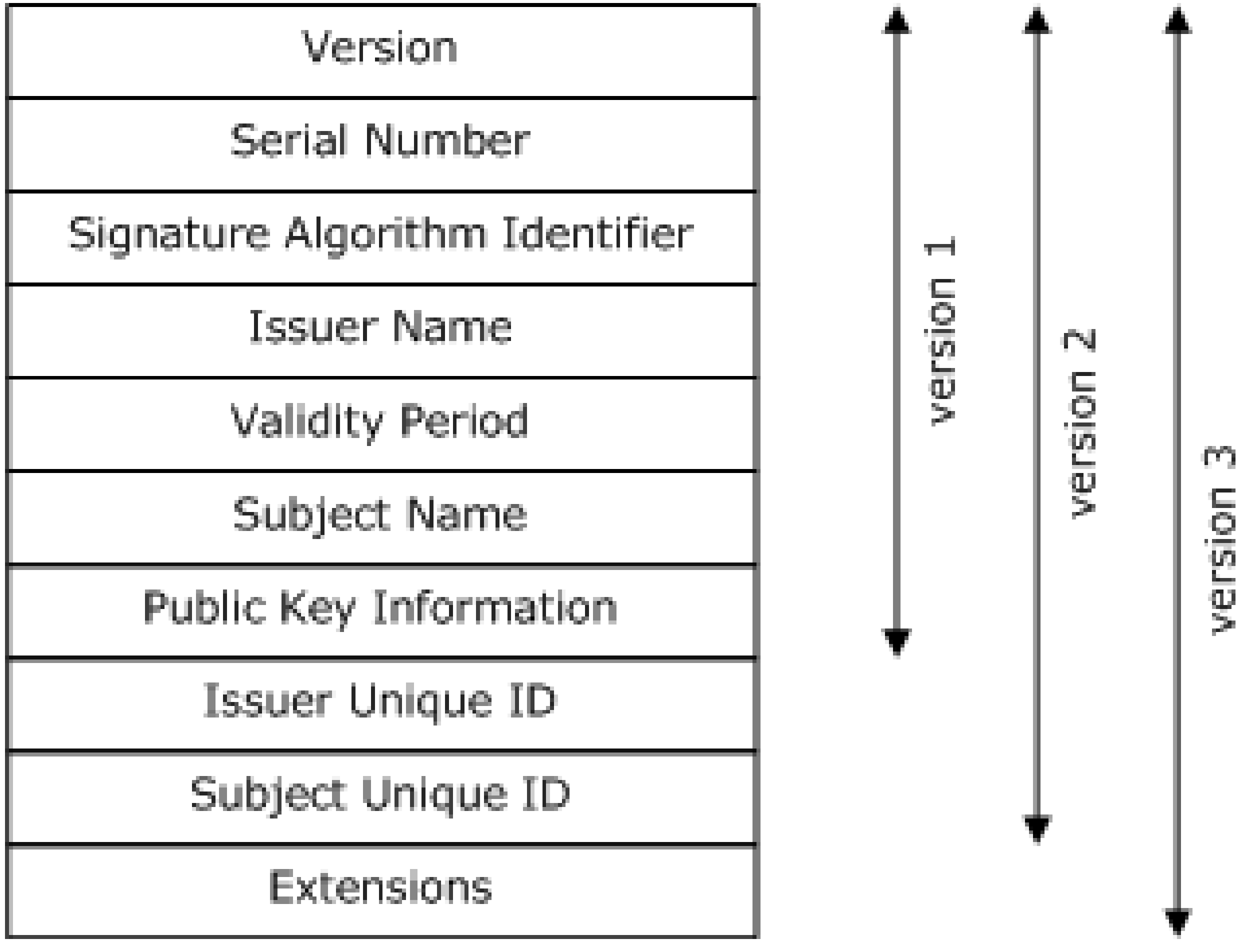
? =

0101000010.

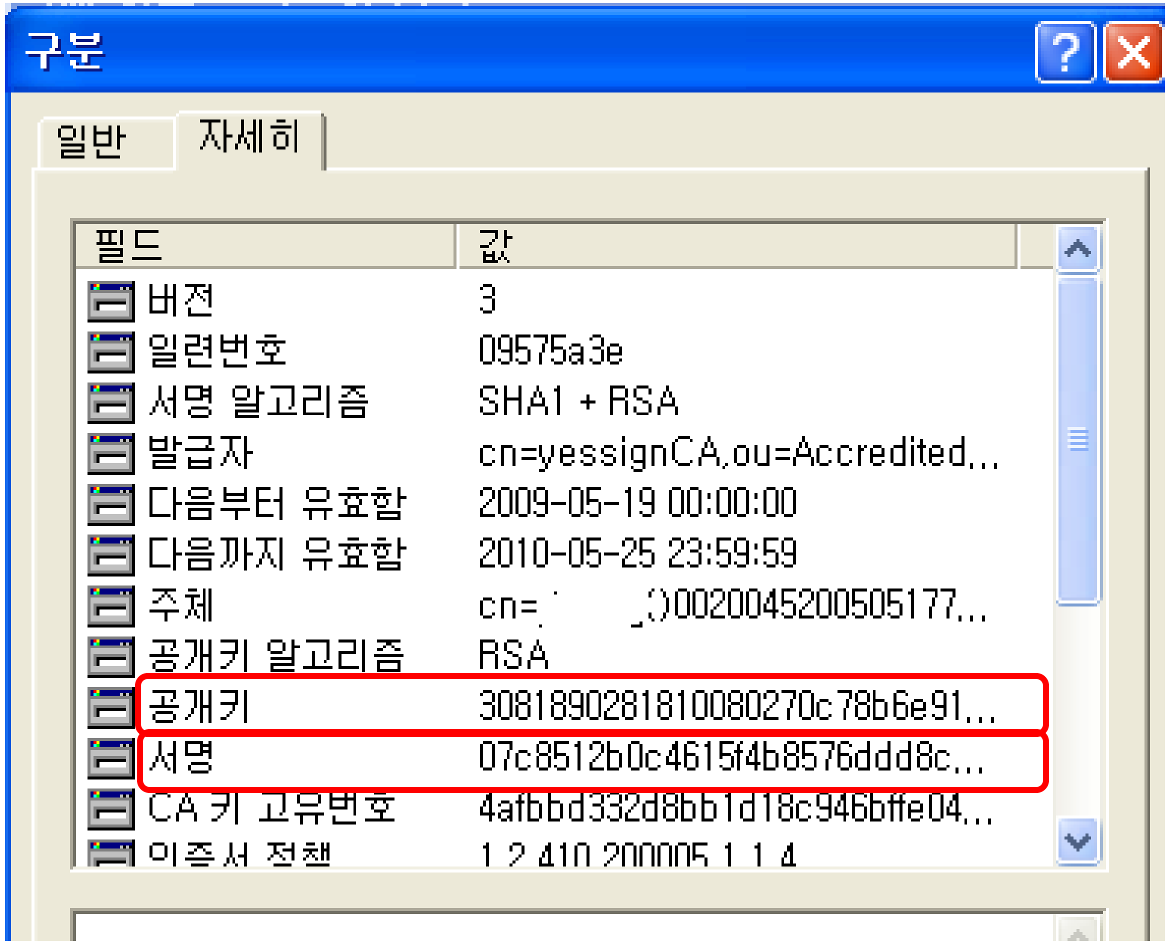
.

Authentication: Confirm Server's public key

X.509 Certificate

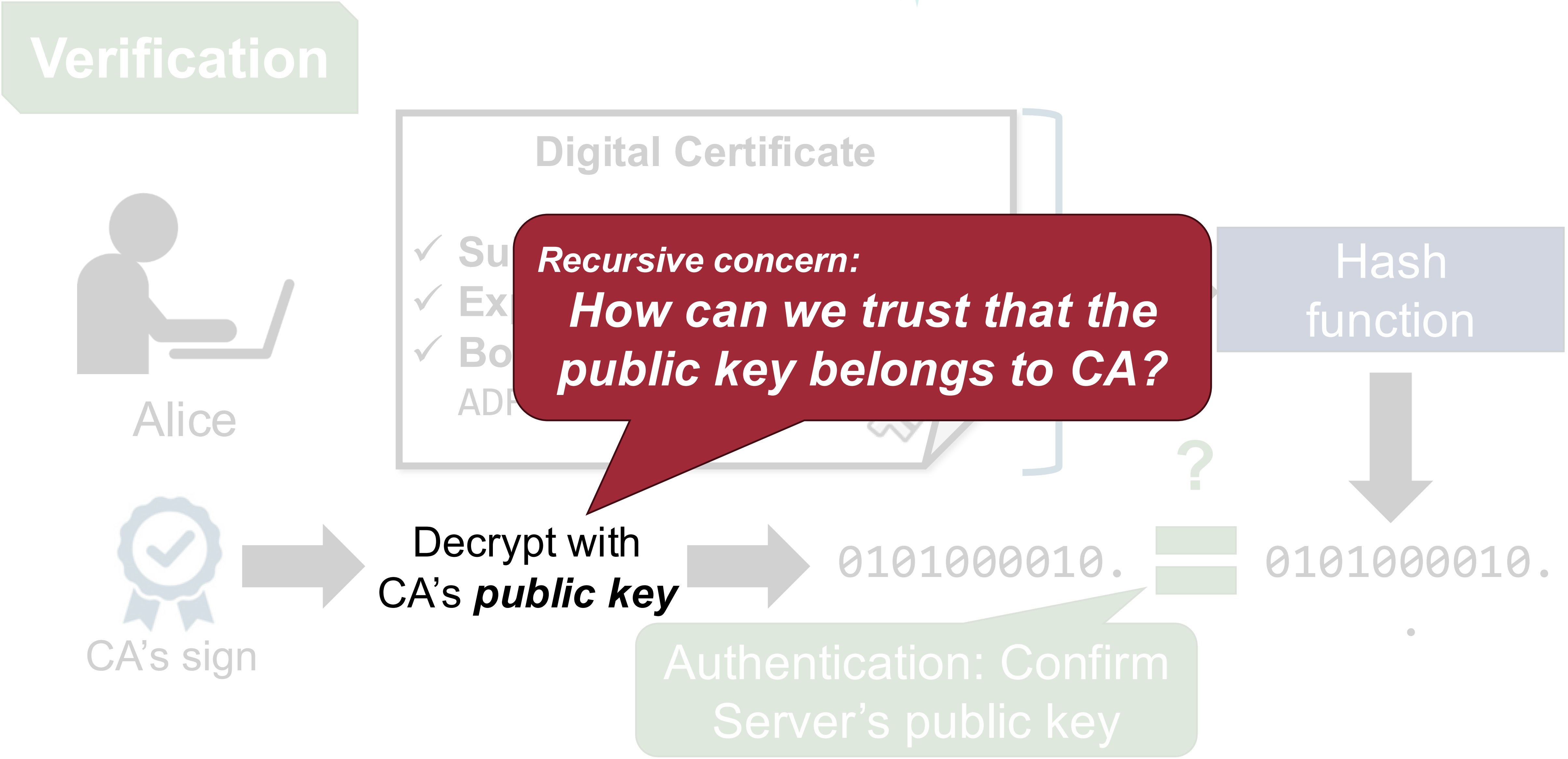


Version
Serial Number
Signature Algorithm Identifier
Issuer Name
Validity Period
Subject Name
Public Key Information
Issuer Unique ID
Subject Unique ID
Extensions



필드	값
버전	3
일련번호	09575a3e
서명 알고리즘	SHA1 + RSA
발급자	cn=yessignCA,ou=Accredited,...
다음부터 유효함	2009-05-19 00:00:00
다음까지 유효함	2010-05-25 23:59:59
주체	cn= ()0020045200505177...
공개키 알고리즘	RSA
공개키	30818902818100 80270c78b6e91...
서명	07c8512b0c4615f4b8576ddd8c...
CA 키 고유번호	4afbbd332d8bb1d18c946bffe04...
인증서 정책	1.2.410.200005.1.1.4

One Concern:

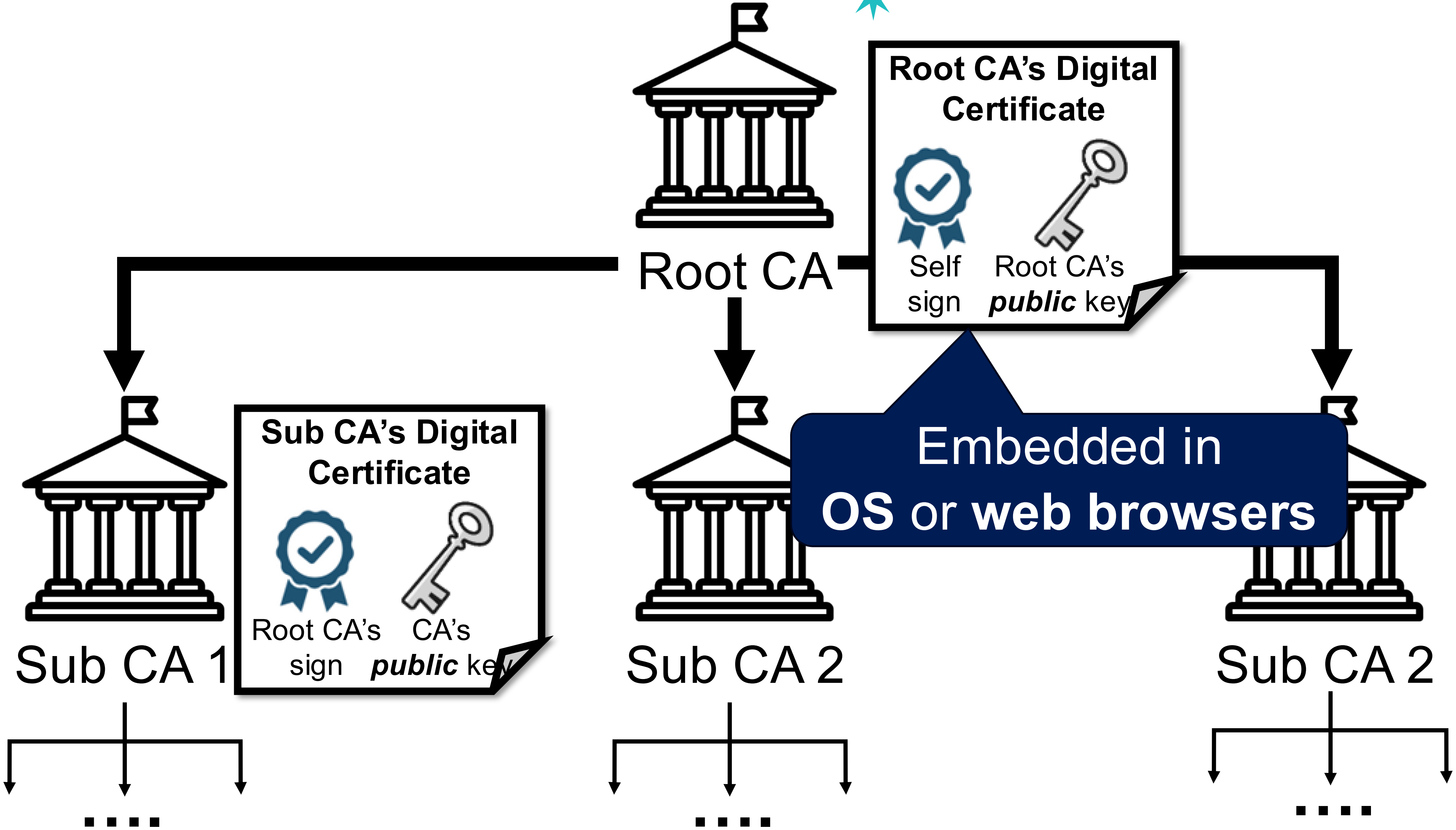


Recursive concern:

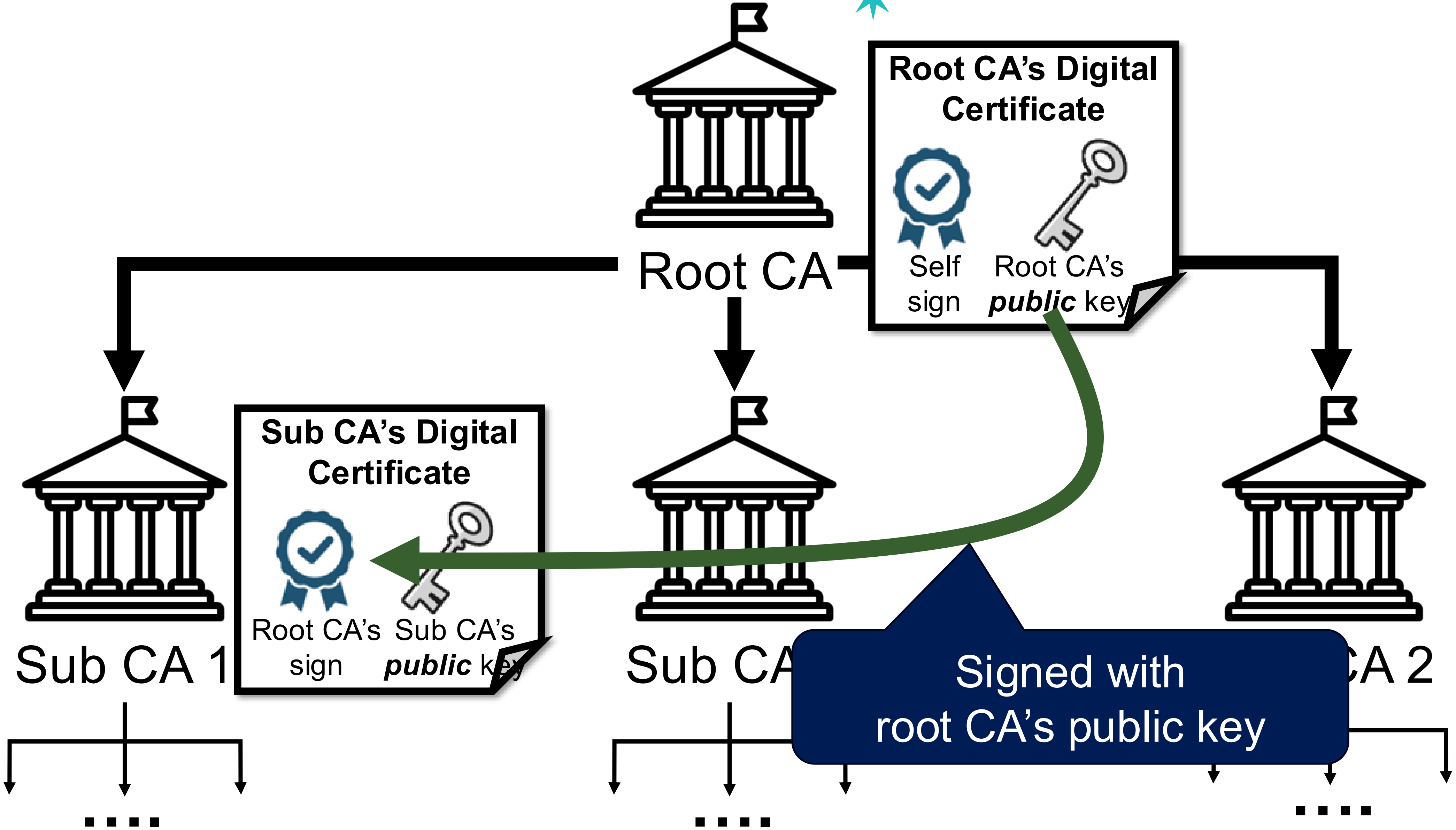
How can we trust that the public key belongs to CA?

Decrypt with CA's ***public key***

Chain of Trust

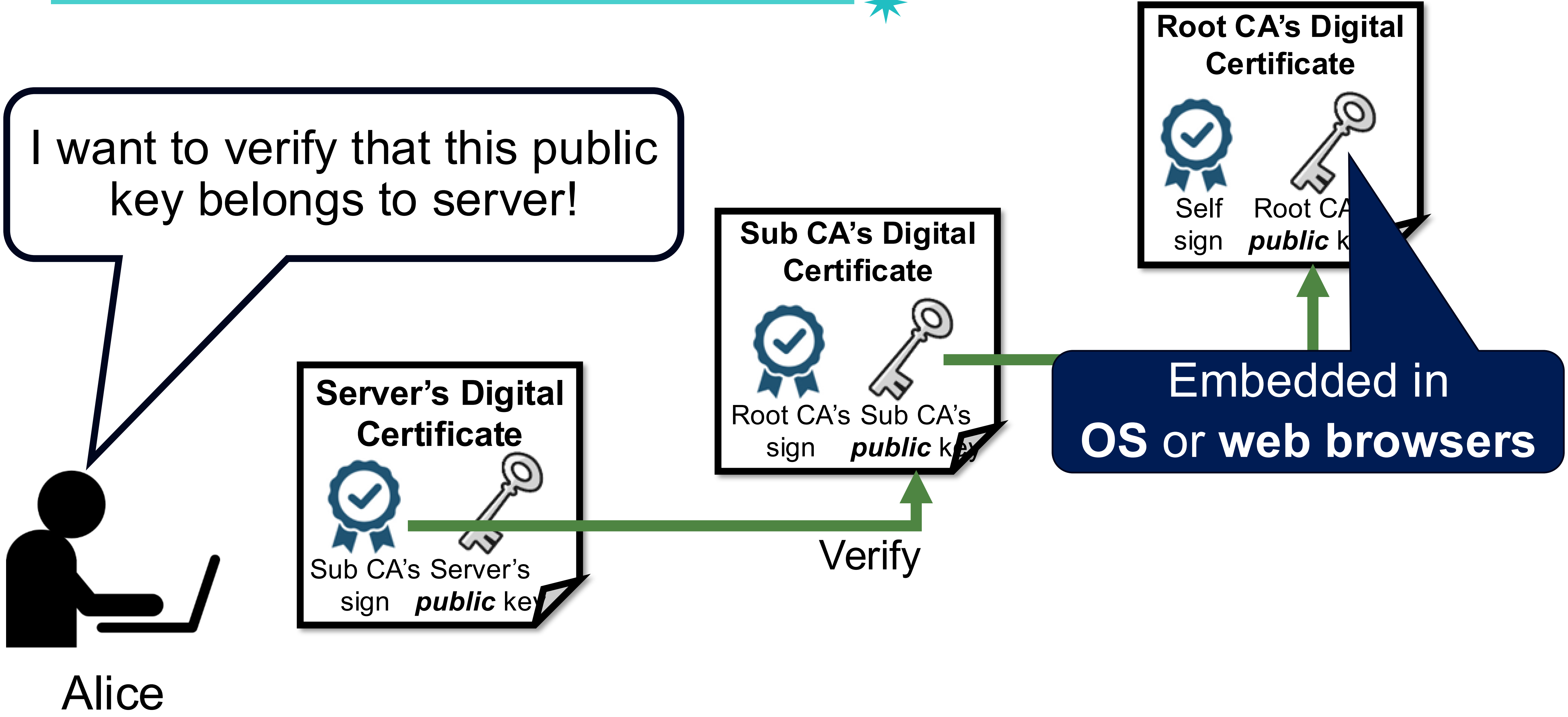


Chain of Trust

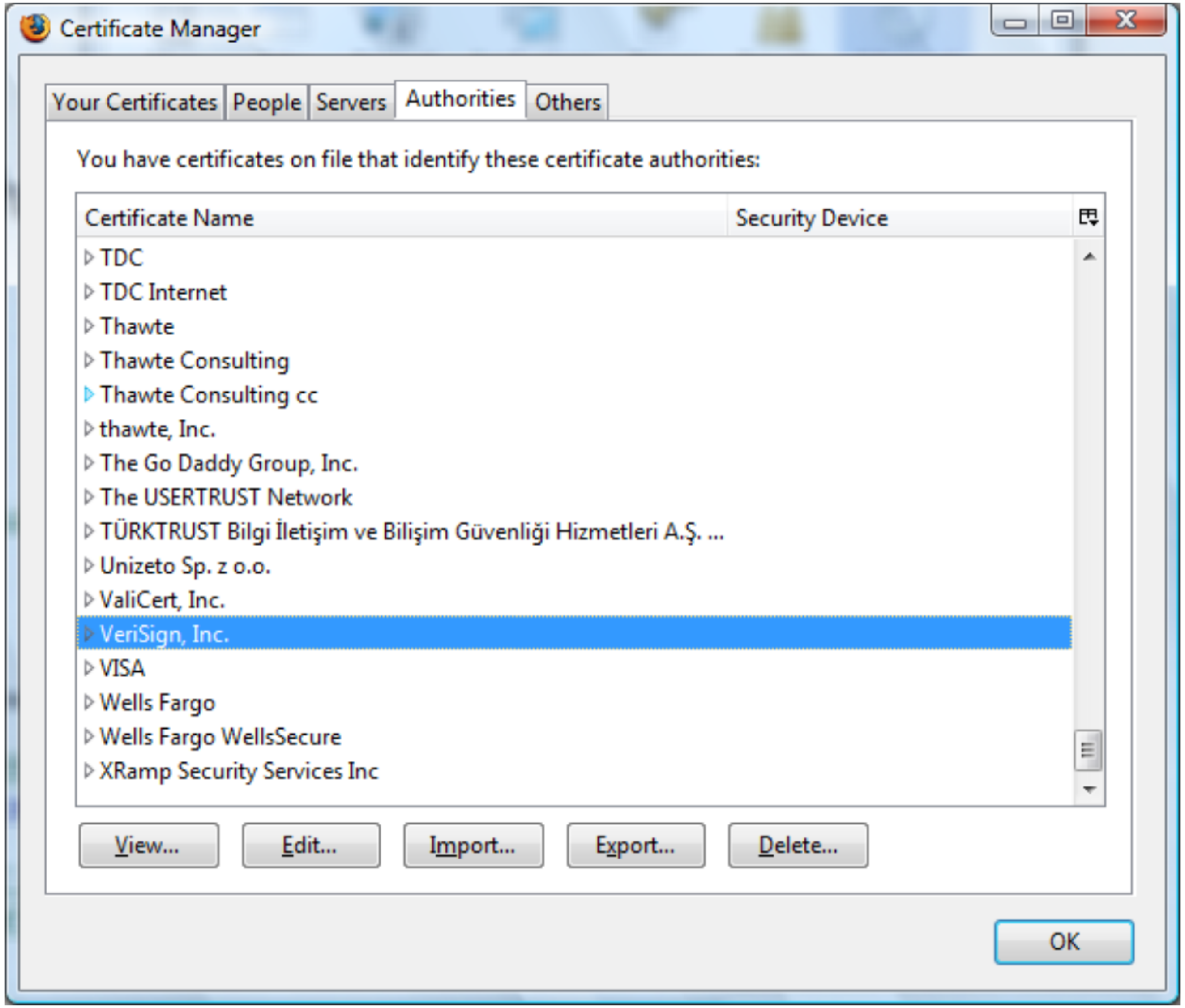


Signed with
root CA's public key

Recap: Chain of Trust


I want to verify that this public key belongs to server!
Alice
Server's Digital Certificate
Sub CA's sign
Server's public key
Verify
Sub CA's Digital Certificate
Root CA's sign
Sub CA's public key
Root CA's Digital Certificate
Self sign
Root CA public k
Embedded in OS or web browsers

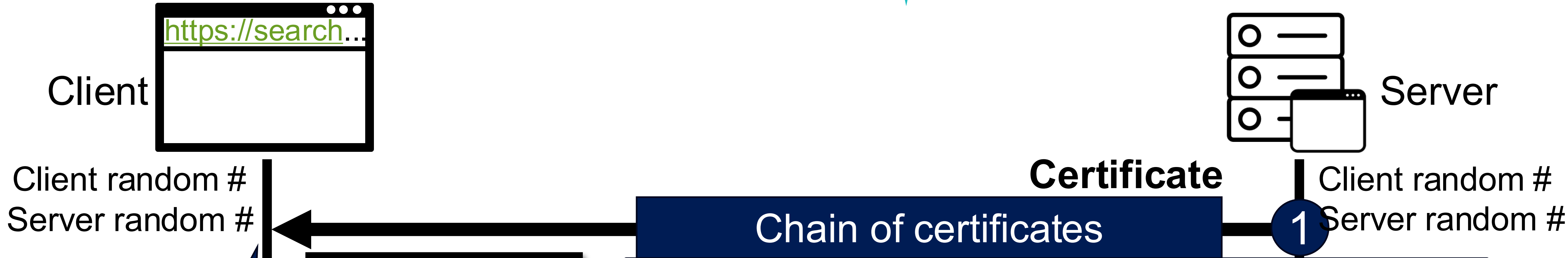
Browsers are Pre-configured with 100+ Trusted CAs



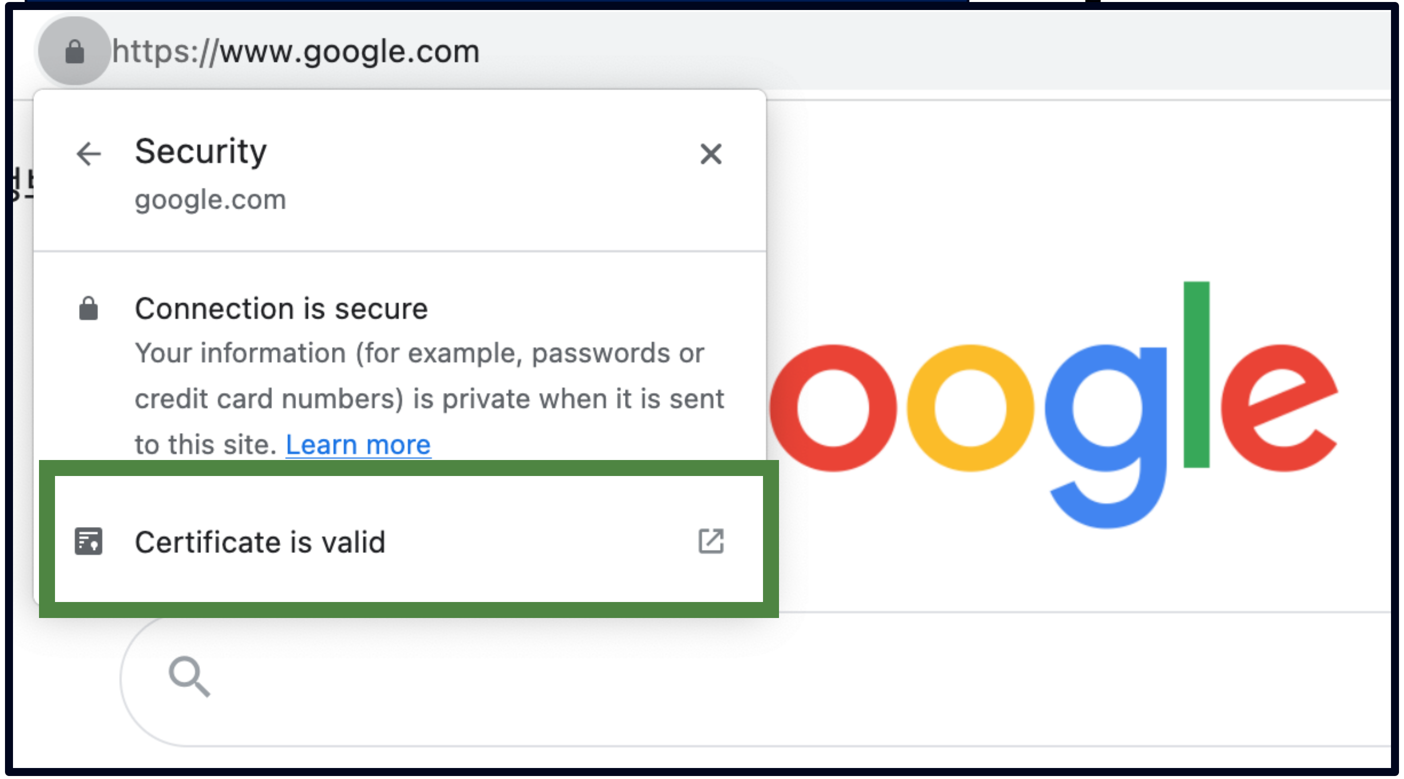
You have certificates on file that identify these certificate authorities:

Certificate Name	Security Device
TDC	
TDC Internet	
Thawte	
Thawte Consulting	
Thawte Consulting cc	
thawte, Inc.	
The Go Daddy Group, Inc.	
The USERTRUST Network	
TÜRKTRUST Bilgi İletişim ve Bilişim Güvenliği Hizmetleri A.Ş. ...	
Unizeto Sp. z o.o.	
ValiCert, Inc.	
VeriSign, Inc.	
VISA	
Wells Fargo	
Wells Fargo WellsSecure	
XRamp Security Services Inc	

Phase 2: Server Auth. and Key Exchange



Server's Digital Certificate
CA's sign
Server's *public* key


https://www.google.com

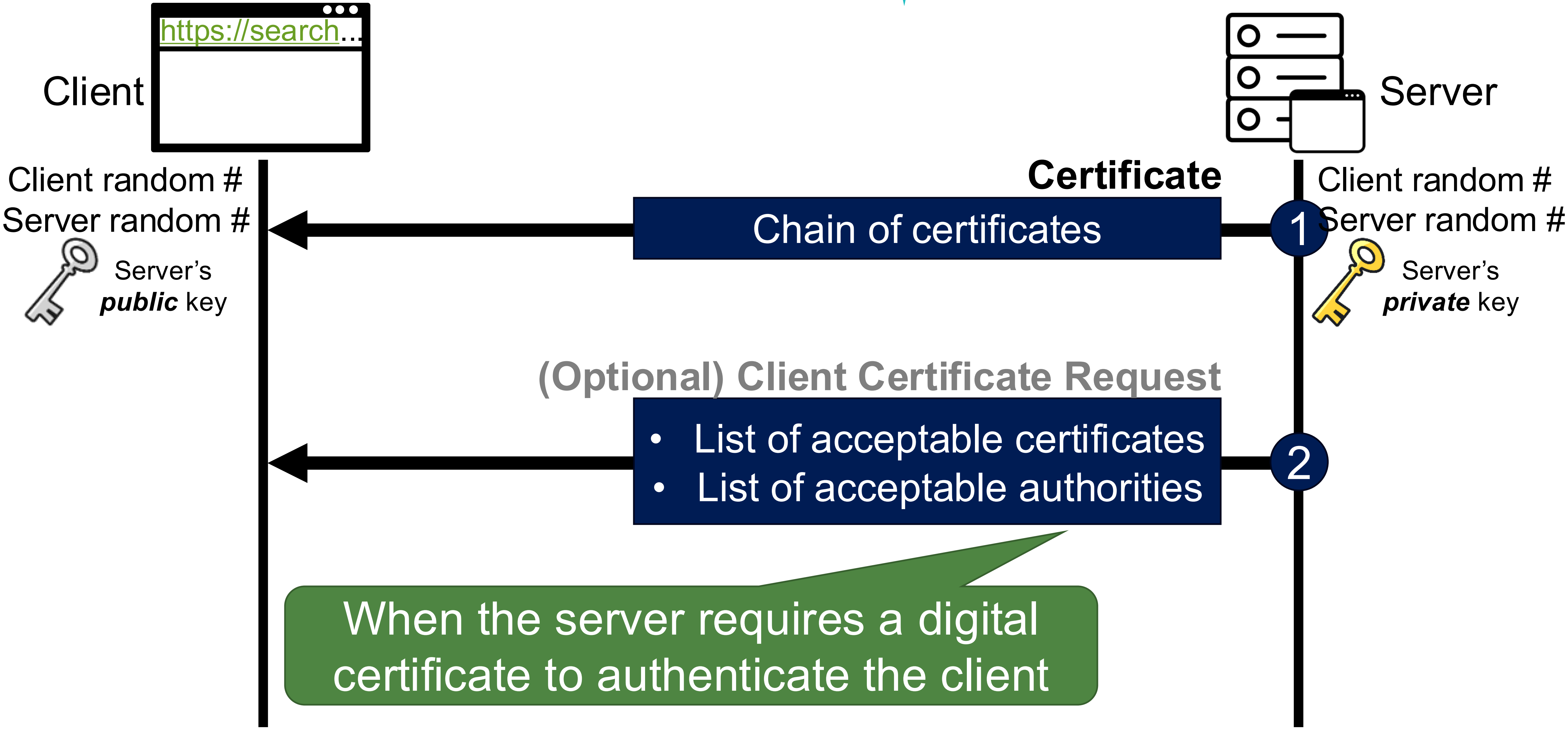
Security
google.com

Connection is secure
Your information (for example, passwords or credit card numbers) is private when it is sent to this site. Learn more

Certificate is valid

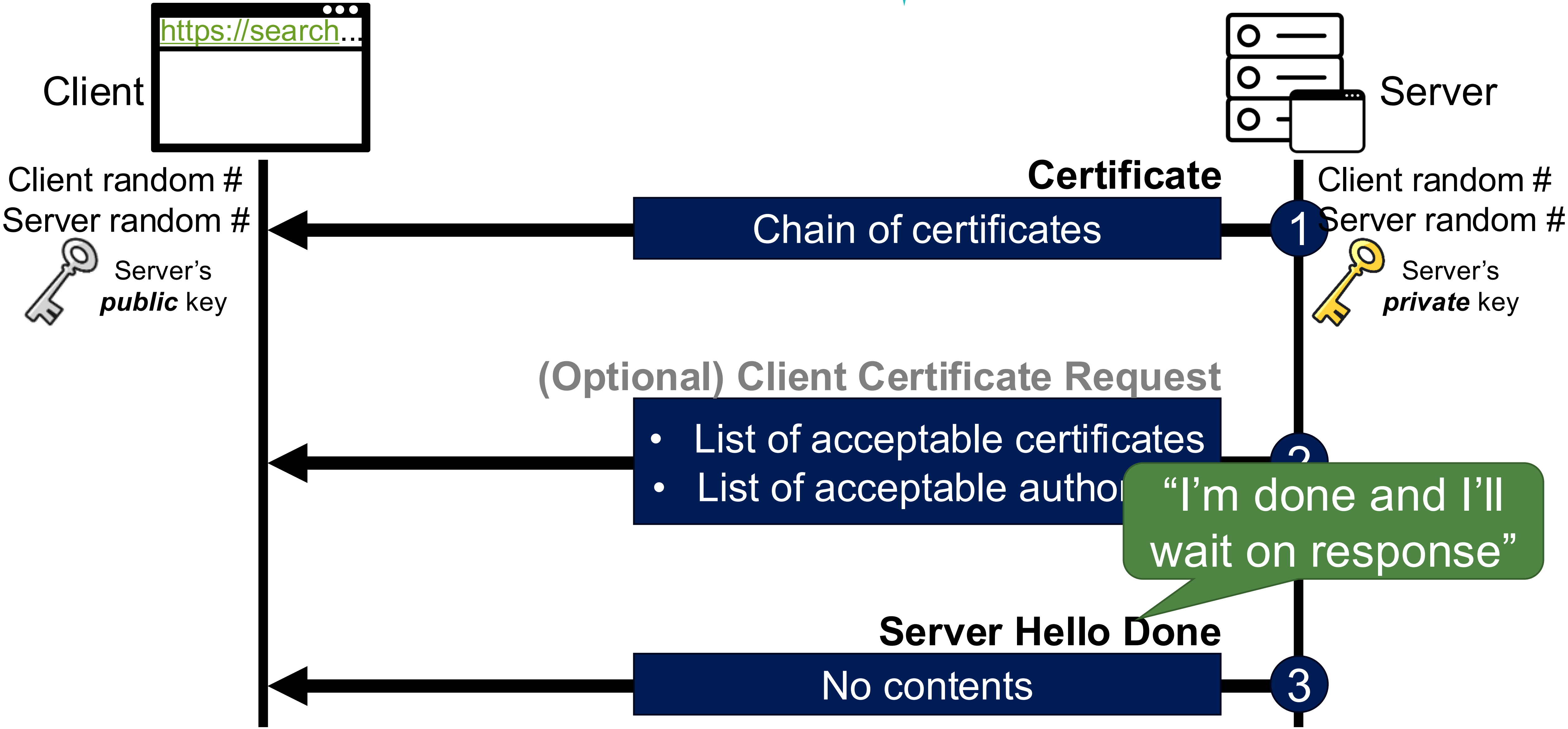
Client verifies that server provided a valid certificate

Phase 2: Server Auth. and Key Exchange



When the server requires a digital certificate to authenticate the client

Phase 2: Server Auth. and Key Exchange



Client

https://search...

Server

Client random #
Server random #
Server's **public** key

Client random #
Server random #
Server's **private** key

Certificate

1 Chain of certificates

(Optional) Client Certificate Request

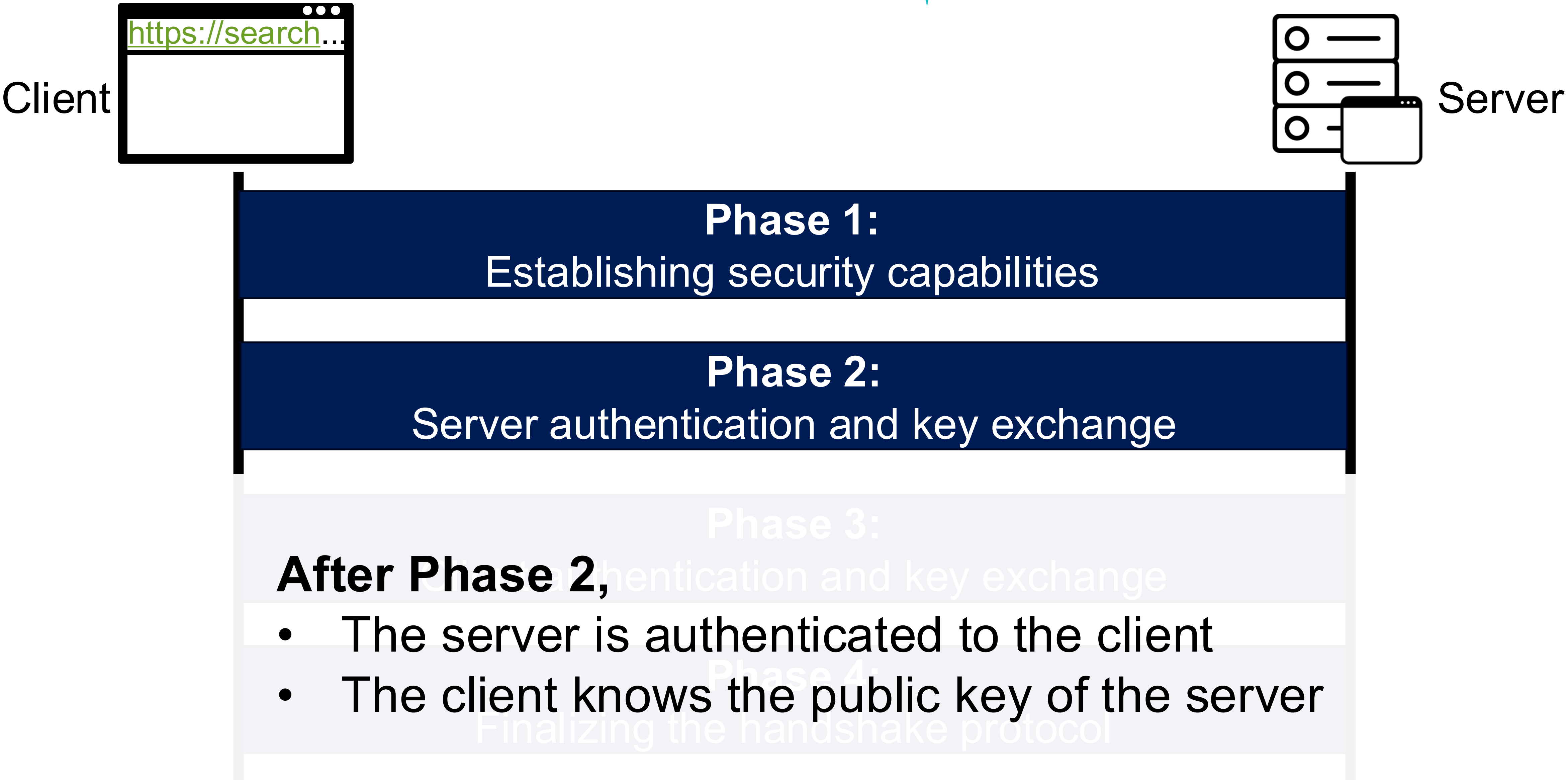
2
- List of acceptable certificates
- List of acceptable author

"I'm done and I'll wait on response"

Server Hello Done

3 No contents

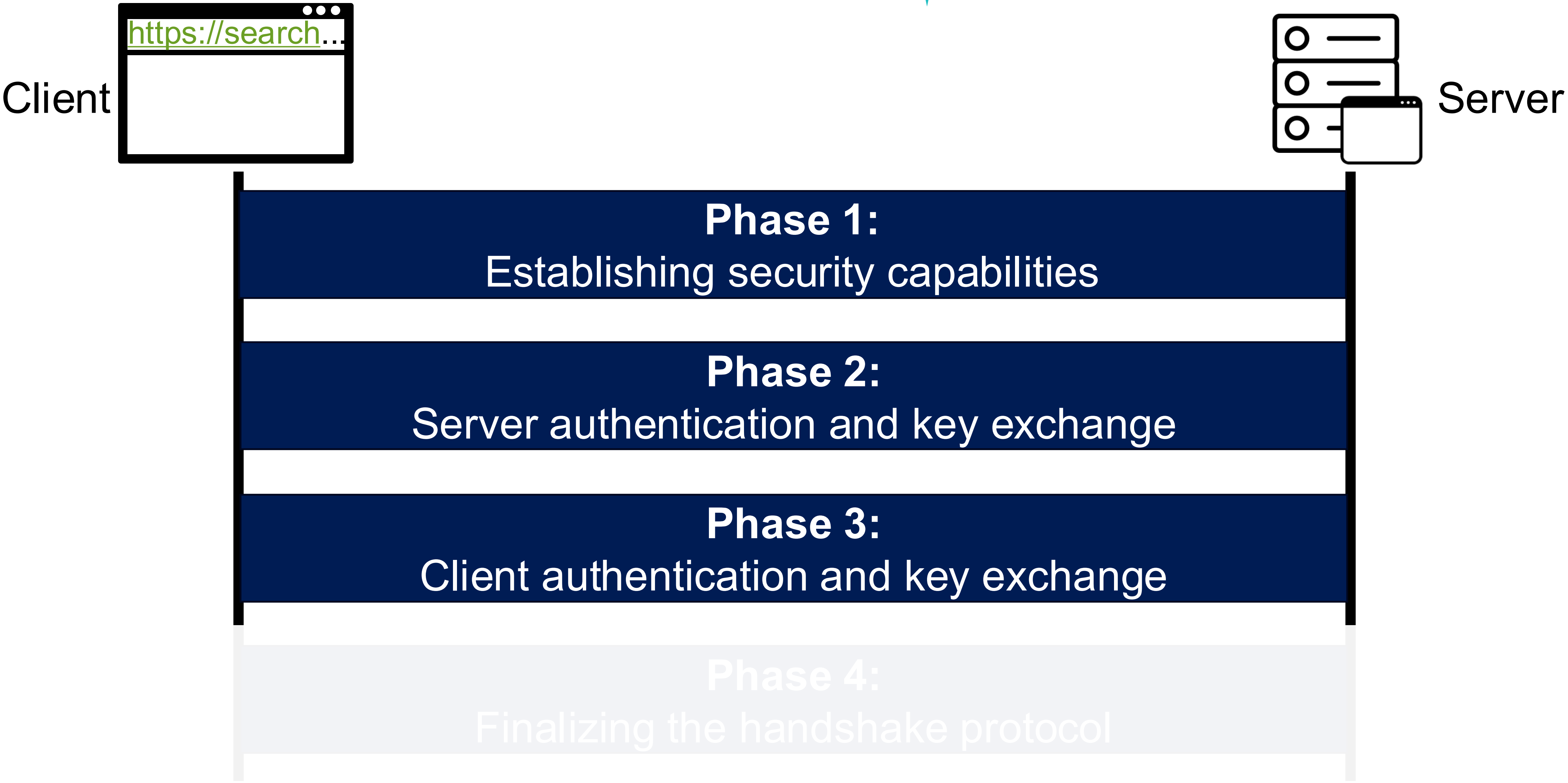
Phase 1: Establishing Security Capabilities


https://search...
Client
Server
Phase 1:
Establishing security capabilities
Phase 2:
Server authentication and key exchange
Phase 3:
Client authentication and key exchange
Phase 4:
Finalizing the handshake protocol

After Phase 2,

- The server is authenticated to the client
- The client knows the public key of the server

Phase 3: Client Auth. and Key Exchange



Phase 1:
Establishing security capabilities

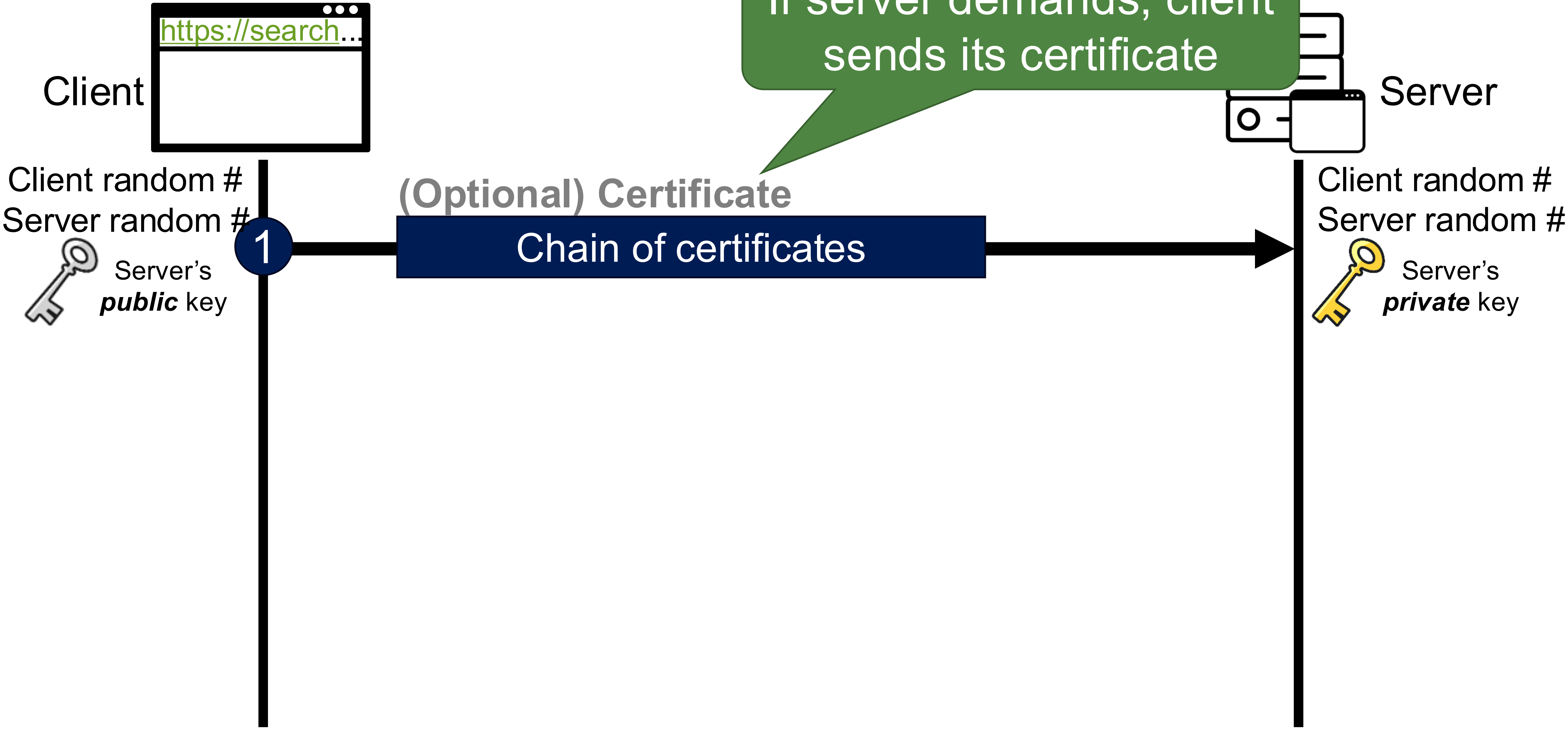
Phase 2:
Server authentication and key exchange

Phase 3:
Client authentication and key exchange

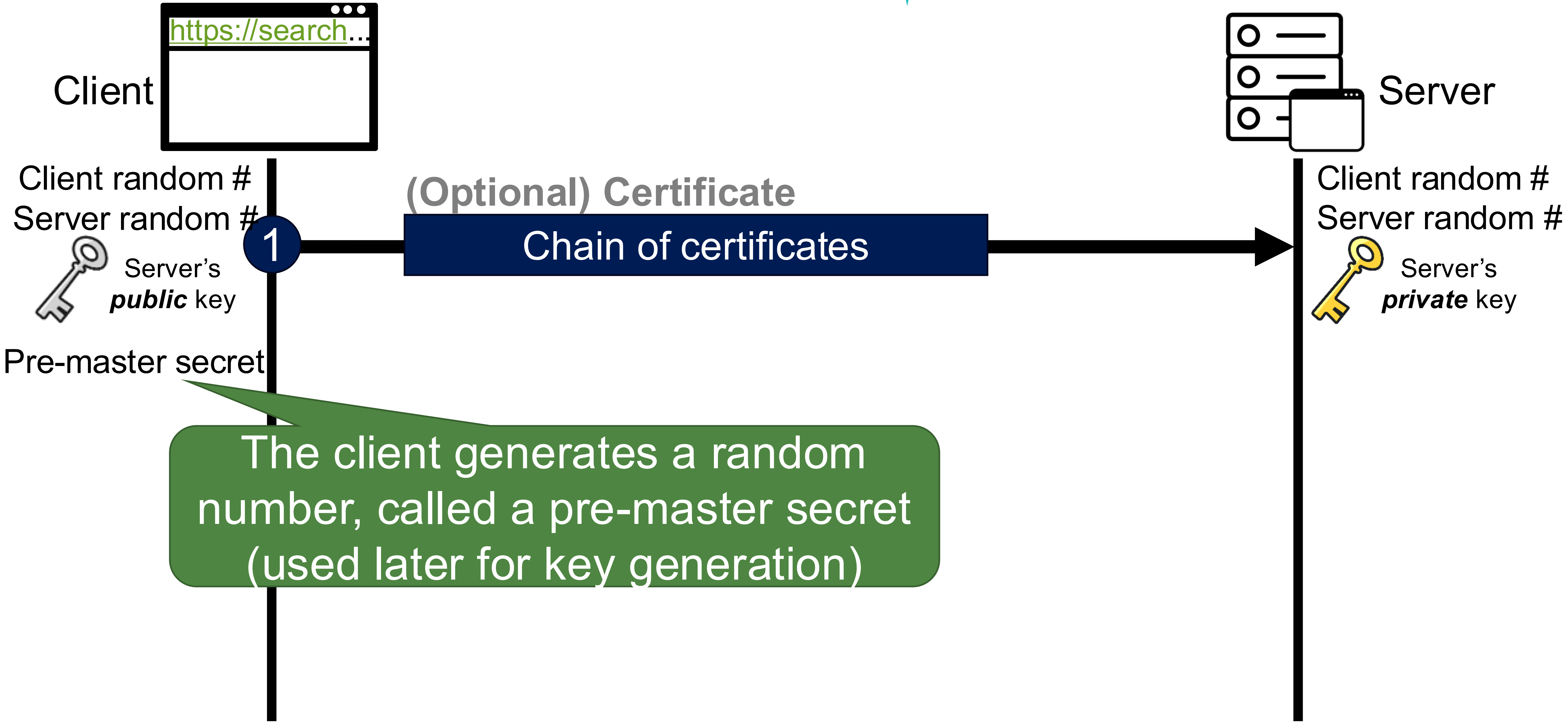
Phase 4:
Finalizing the handshake protocol

Phase 3: Client Auth. and Key Exchange

If server demands, client sends its certificate



Phase 3: Client Auth. and Key Exchange



Client

Server

Client random #
Server random #

Server's **public** key

Pre-master secret

(Optional) Certificate

1

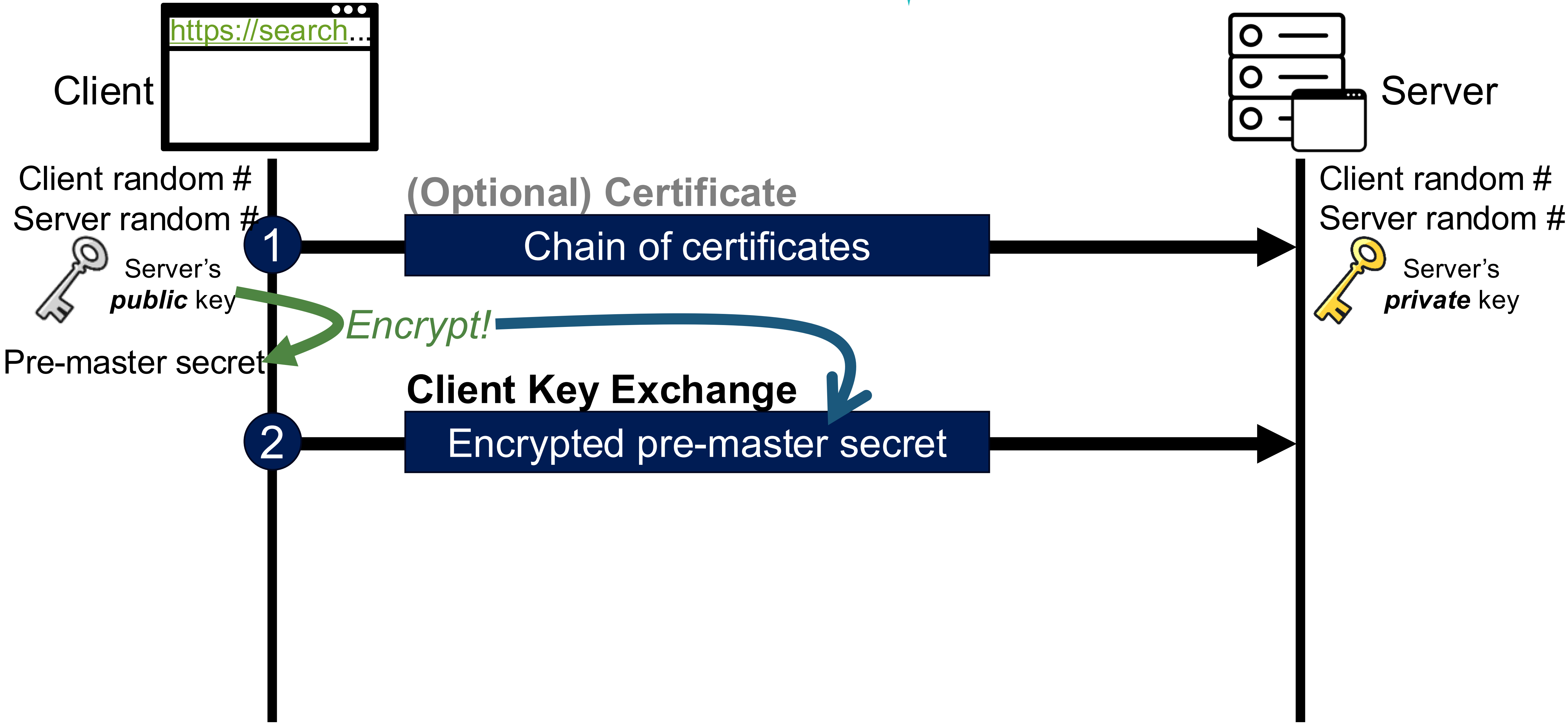
Chain of certificates

Client random #
Server random #

Server's **private** key

The client generates a random number, called a pre-master secret (used later for key generation)

Phase 3: Client Auth. and Key Exchange



Client

https://search...

Server

Client random #
Server random #

Server's **public** key

Pre-master secret

Encrypt!

(Optional) Certificate

1 Chain of certificates

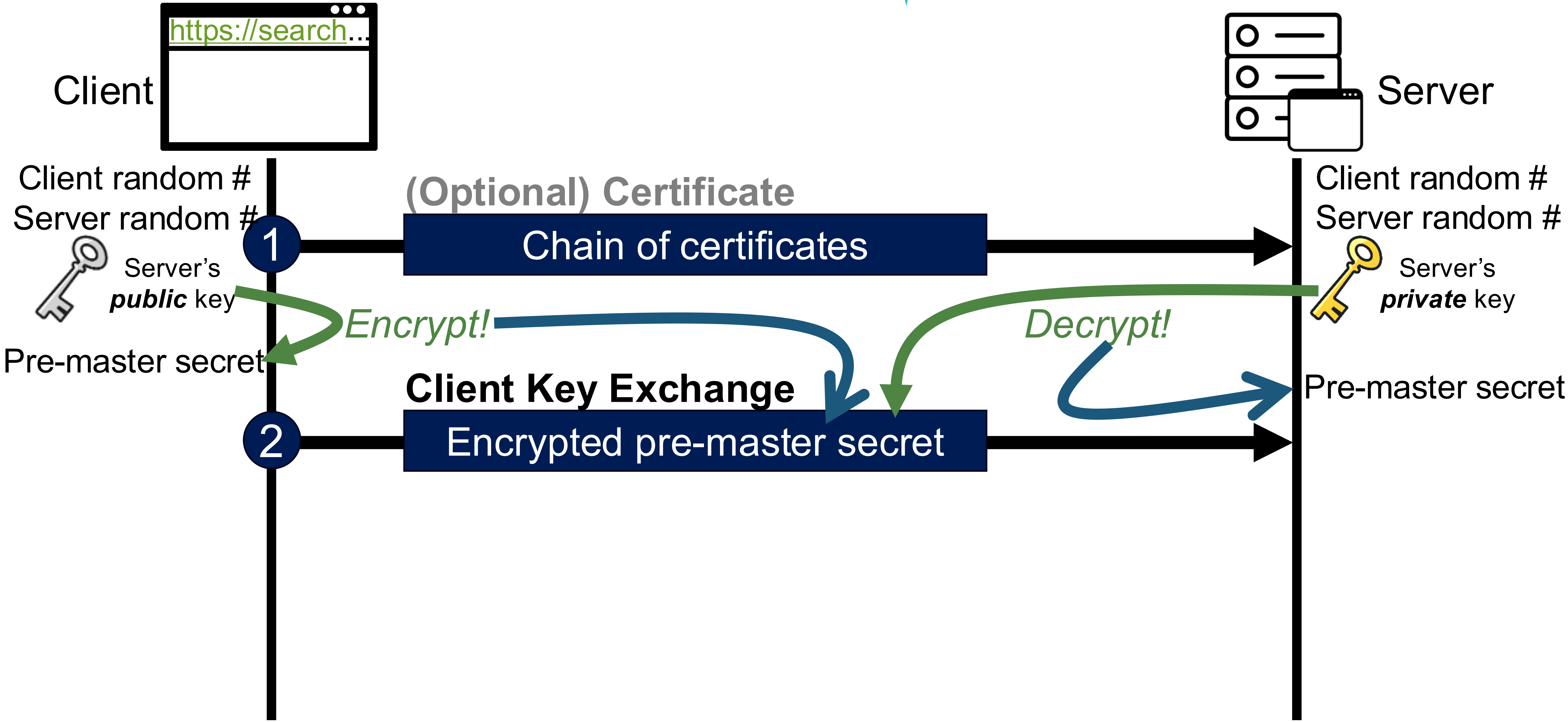
Client Key Exchange

2 Encrypted pre-master secret

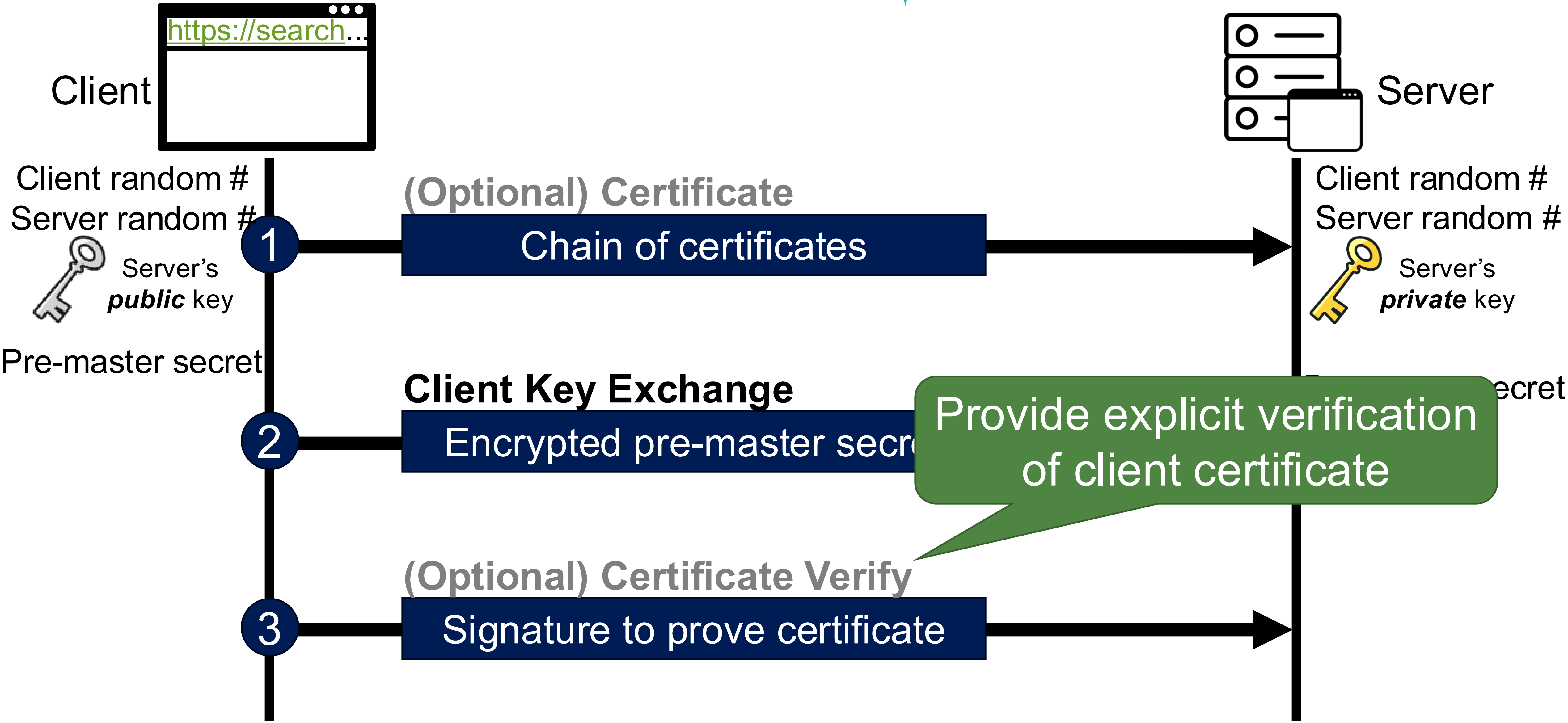
Client random #
Server random #

Server's **private** key

Phase 3: Client Auth. and Key Exchange



Phase 3: Client Auth. and Key Exchange



Client

https://search...

Server

Client random #

Server random #

Server's **_public_** key

Pre-master secret

Client random #

Server random #

Server's **_private_** key

ecret

(Optional) Certificate

1 Chain of certificates

Client Key Exchange

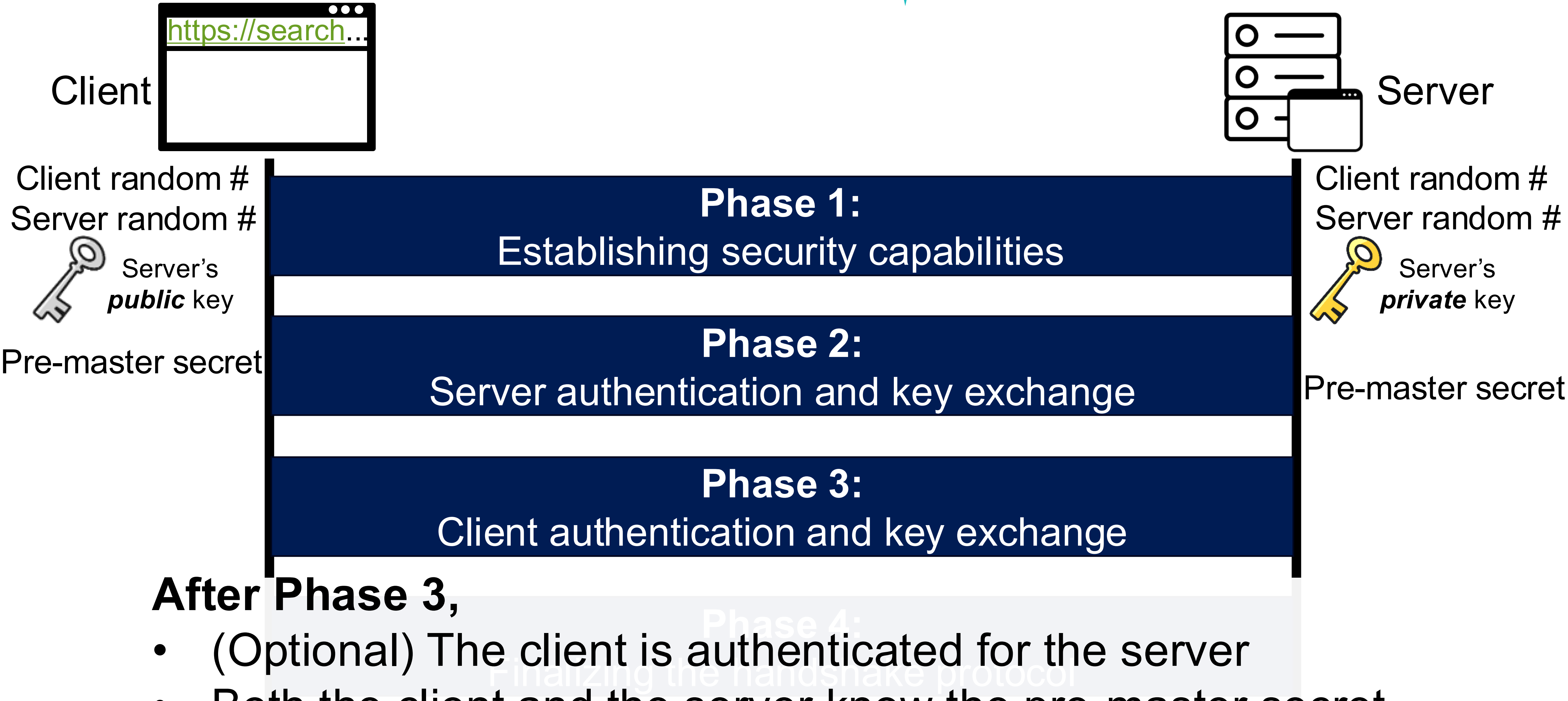
2 Encrypted pre-master secr

(Optional) Certificate Verify

3 Signature to prove certificate

Provide explicit verification of client certificate

Phase 3: Client Auth. and Key Exchange



Client

https://search...

Server

Client random #
Server random #

Server's ***public*** key

Pre-master secret

Client random #
Server random #

Server's ***private*** key

Pre-master secret

Phase 1:
Establishing security capabilities

Phase 2:
Server authentication and key exchange

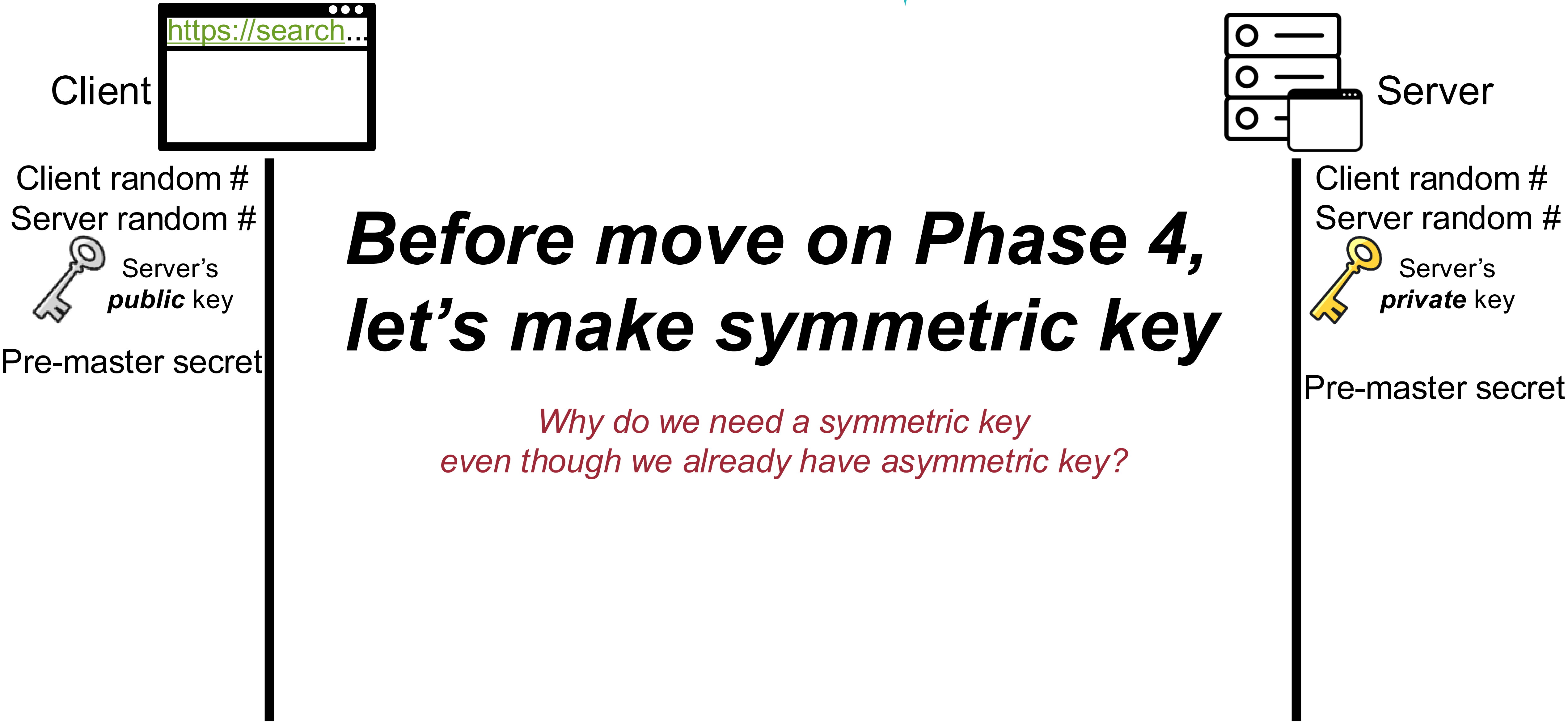
Phase 3:
Client authentication and key exchange

Phase 4:
Finalizing the handshake protocol

After Phase 3,

- (Optional) The client is authenticated for the server
- Both the client and the server know the pre-master secret

Phase 3: Client Auth. and Key Exchange



Client random #
Server random #
Server's **public** key

Pre-master secret

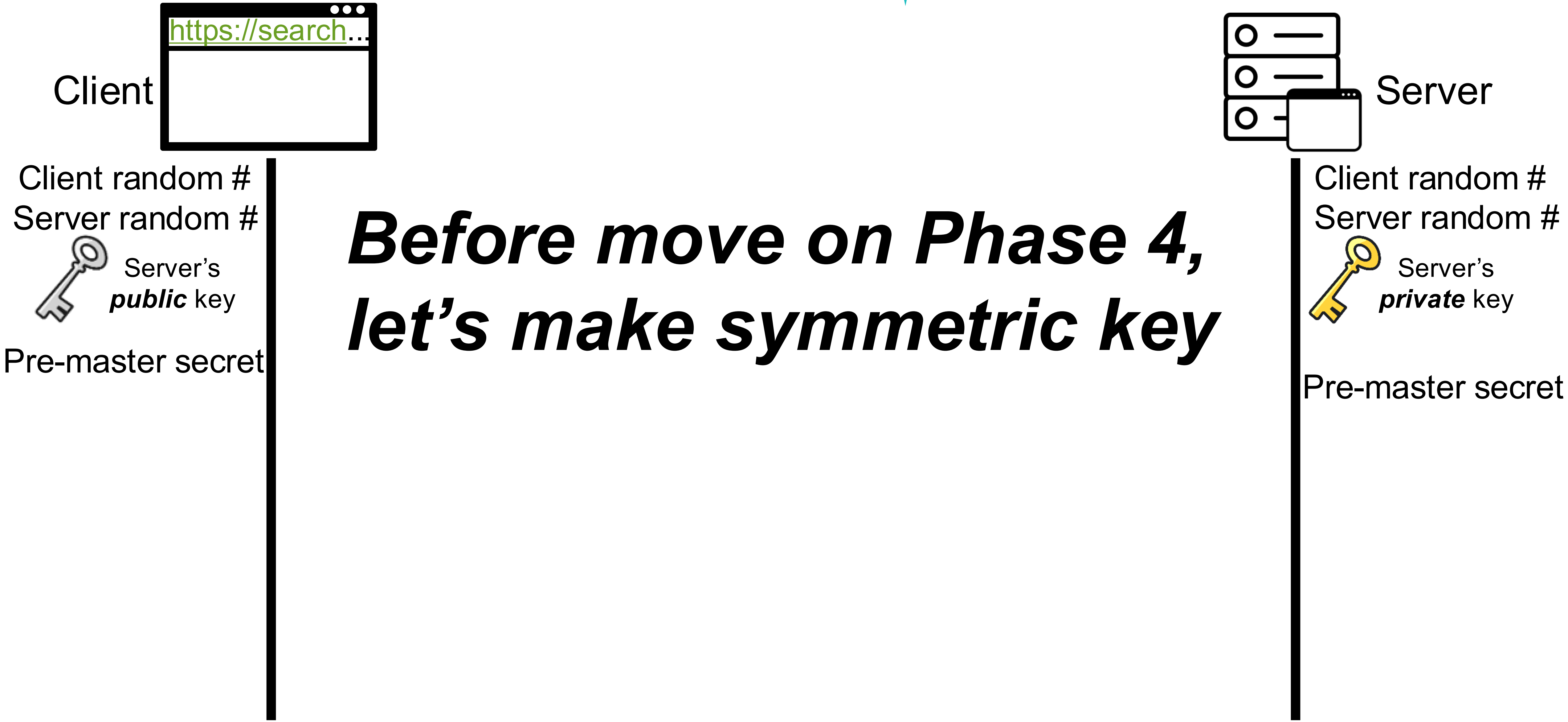
Before move on Phase 4, let's make symmetric key

Why do we need a symmetric key even though we already have asymmetric key?

Client random #
Server random #
Server's **private** key

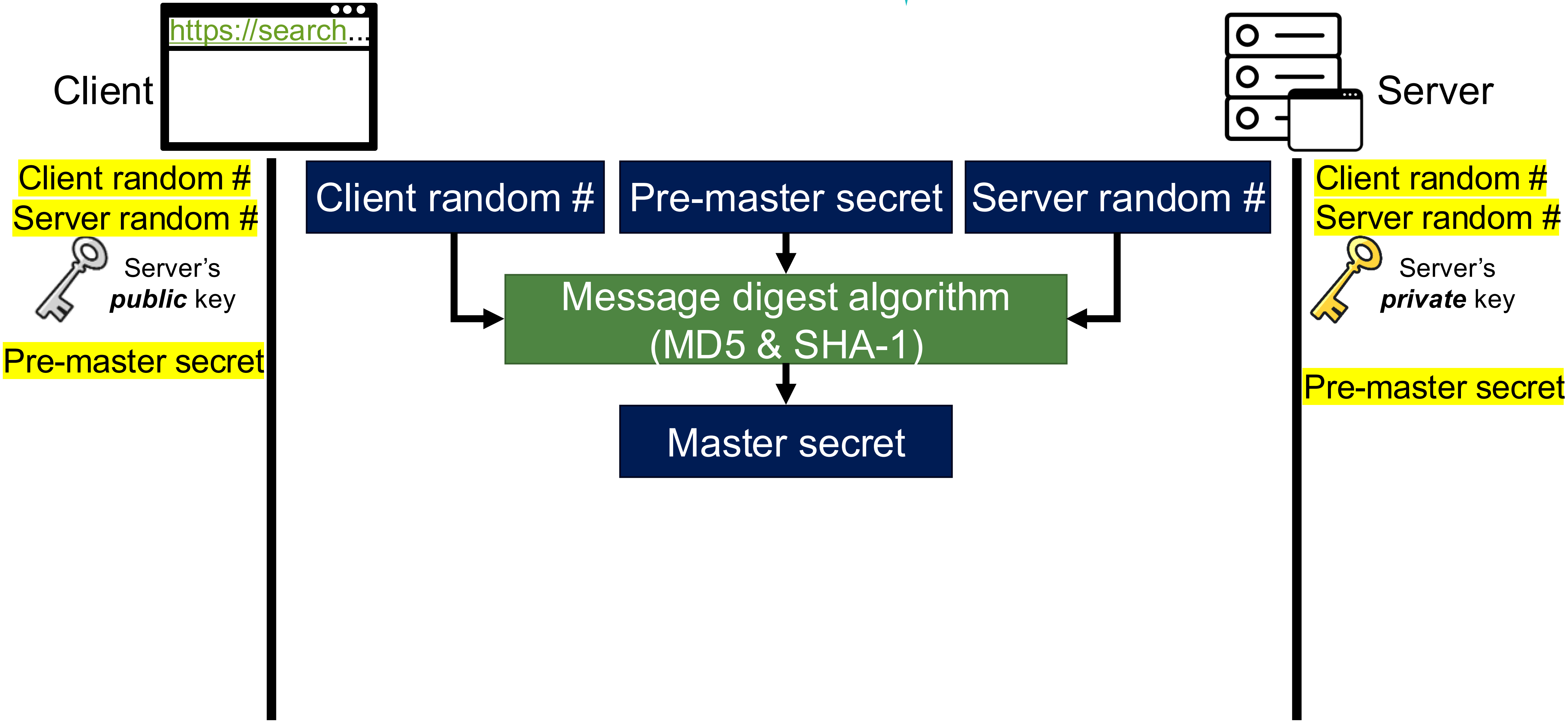
Pre-master secret

Phase 3: Client Auth. and Key Exchange



Before move on Phase 4, let's make symmetric key

Calculation of Master Secret



Client random #
Server random #
Server's **public** key

Pre-master secret

Client random #
Pre-master secret
Server random #

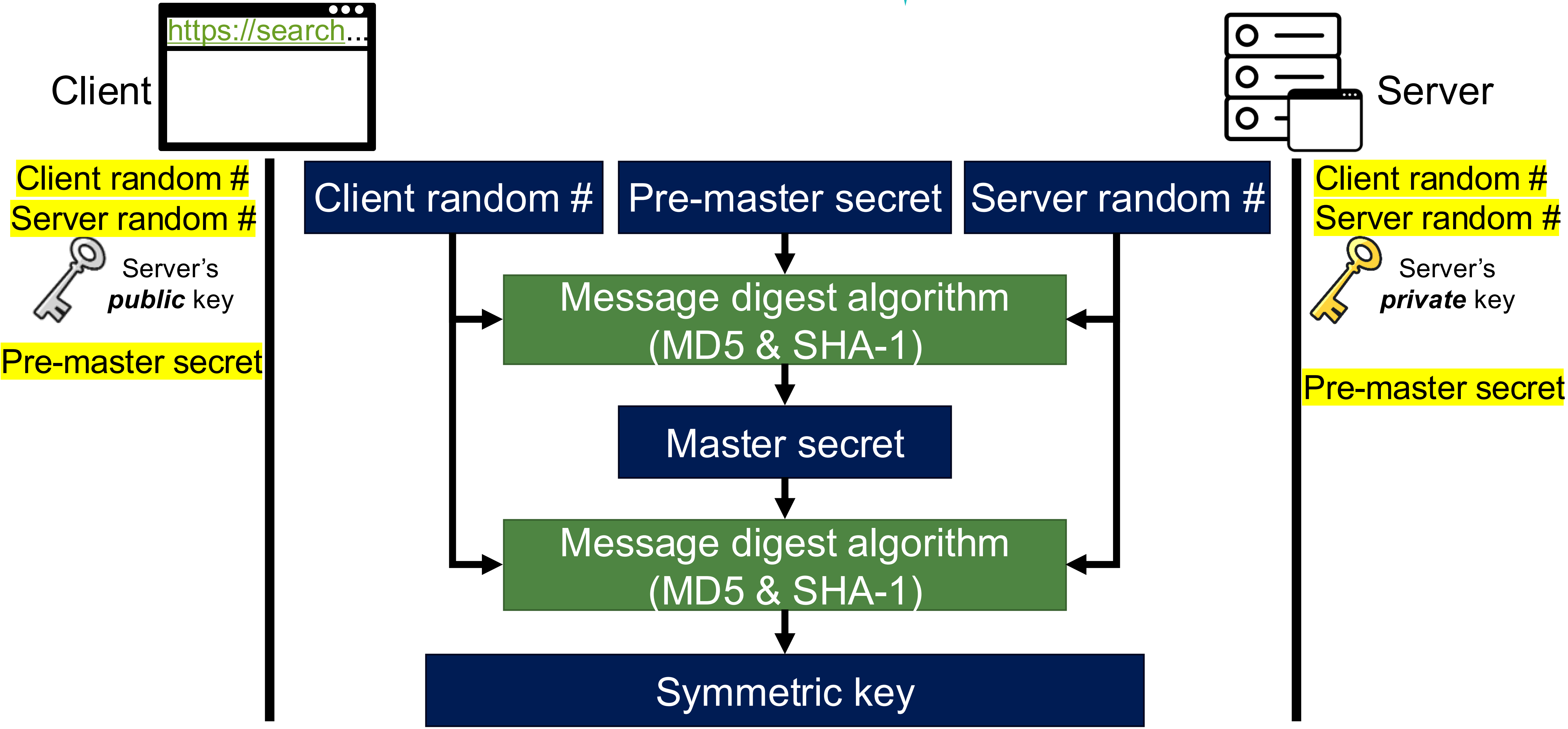
Message digest algorithm (MD5 & SHA-1)

Master secret

Client random #
Server random #
Server's **private** key

Pre-master secret

Calculation of Symmetric Key



Client

https://search...

Client random #

Server random #

Server's **_public_** key

Pre-master secret

Client random #

Pre-master secret

Server random #

Message digest algorithm (MD5 & SHA-1)

Master secret

Message digest algorithm (MD5 & SHA-1)

Symmetric key

Server

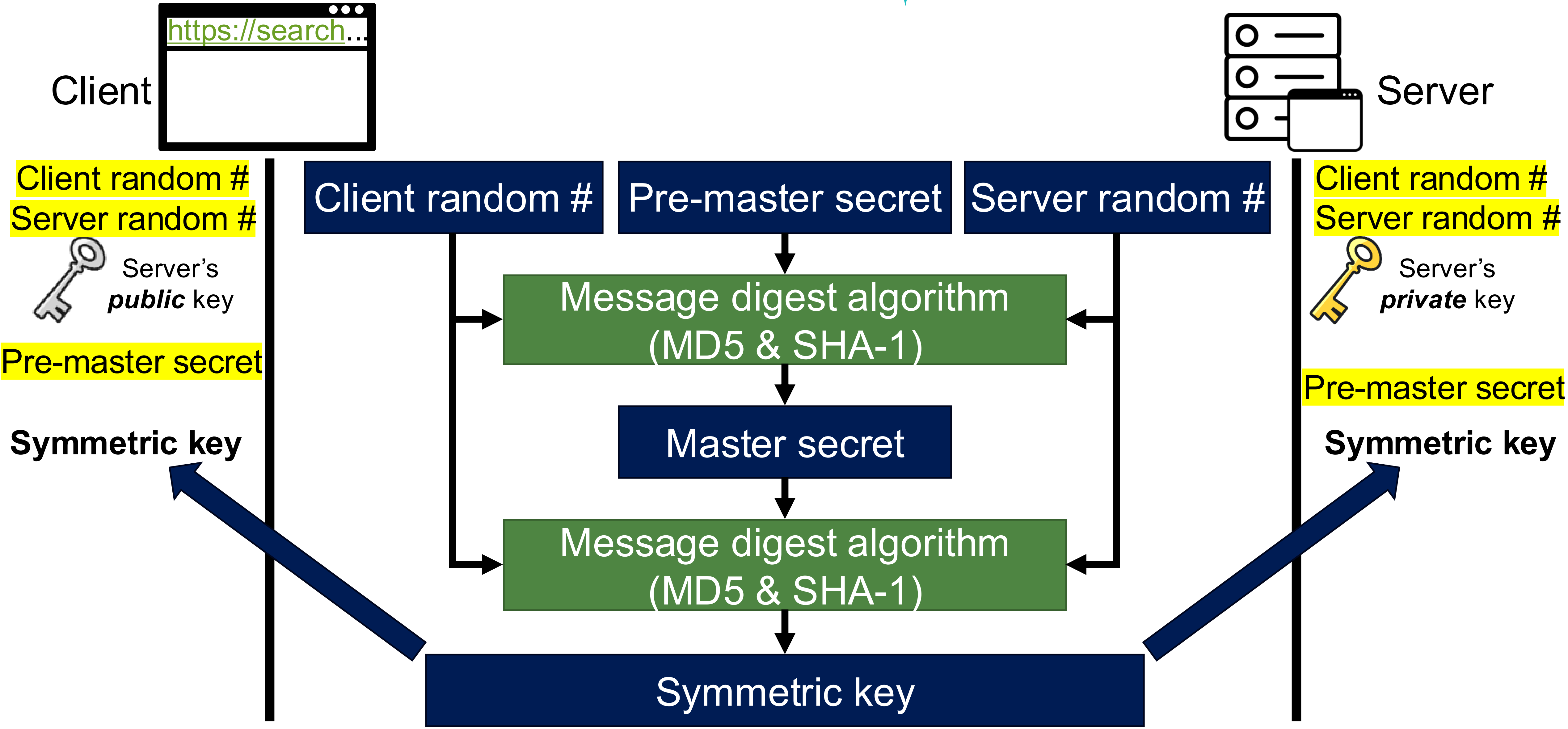
Client random #

Server random #

Server's **_private_** key

Pre-master secret

Calculation of Symmetric Key



Client

https://search...

Server

Client random #
Server random #

Server's **_public_** key

Pre-master secret

Symmetric key

Client random #

Pre-master secret

Server random #

Message digest algorithm (MD5 & SHA-1)

Master secret

Message digest algorithm (MD5 & SHA-1)

Symmetric key

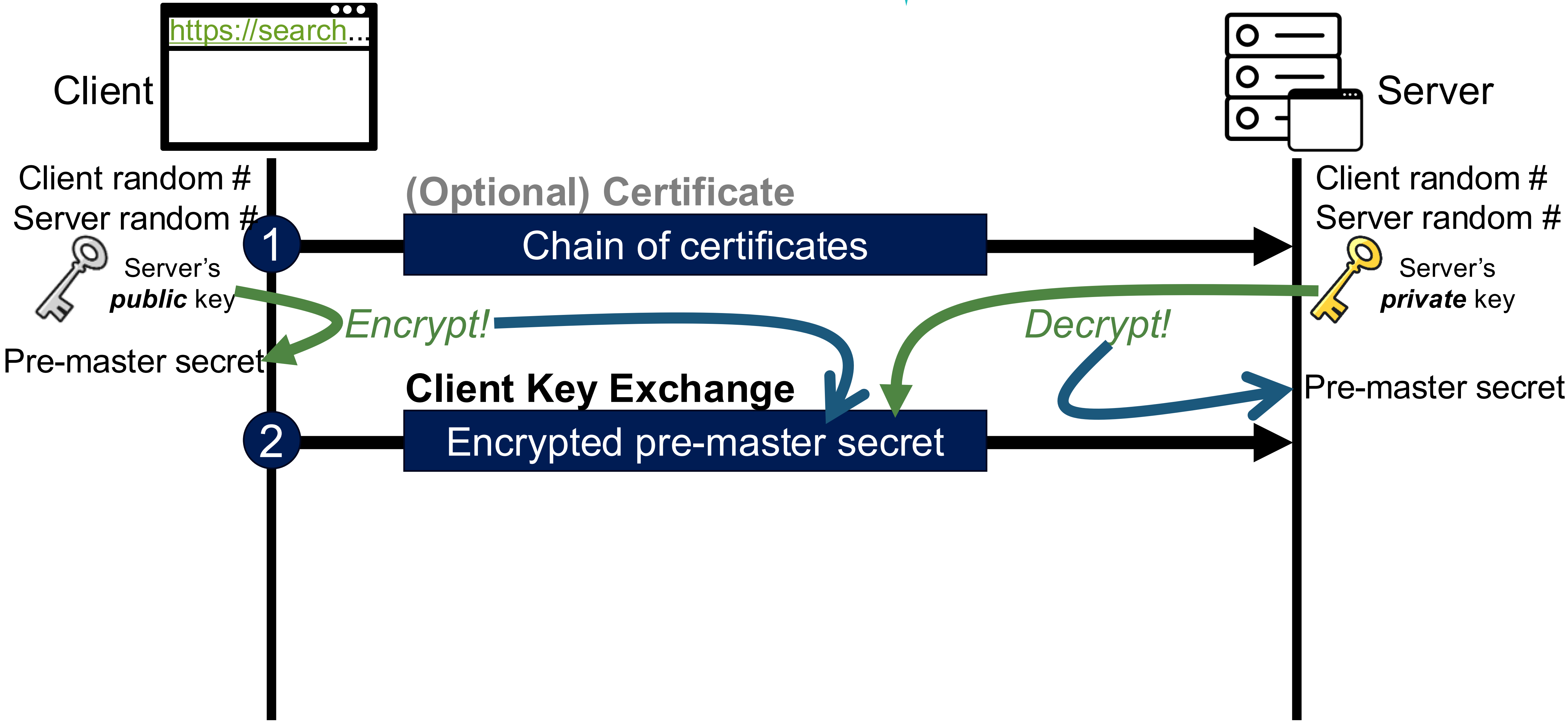
Client random #
Server random #

Server's **_private_** key

Pre-master secret

Symmetric key

Recap: Client Auth. and Key Exchange



ClientKeyExchange (RFC)

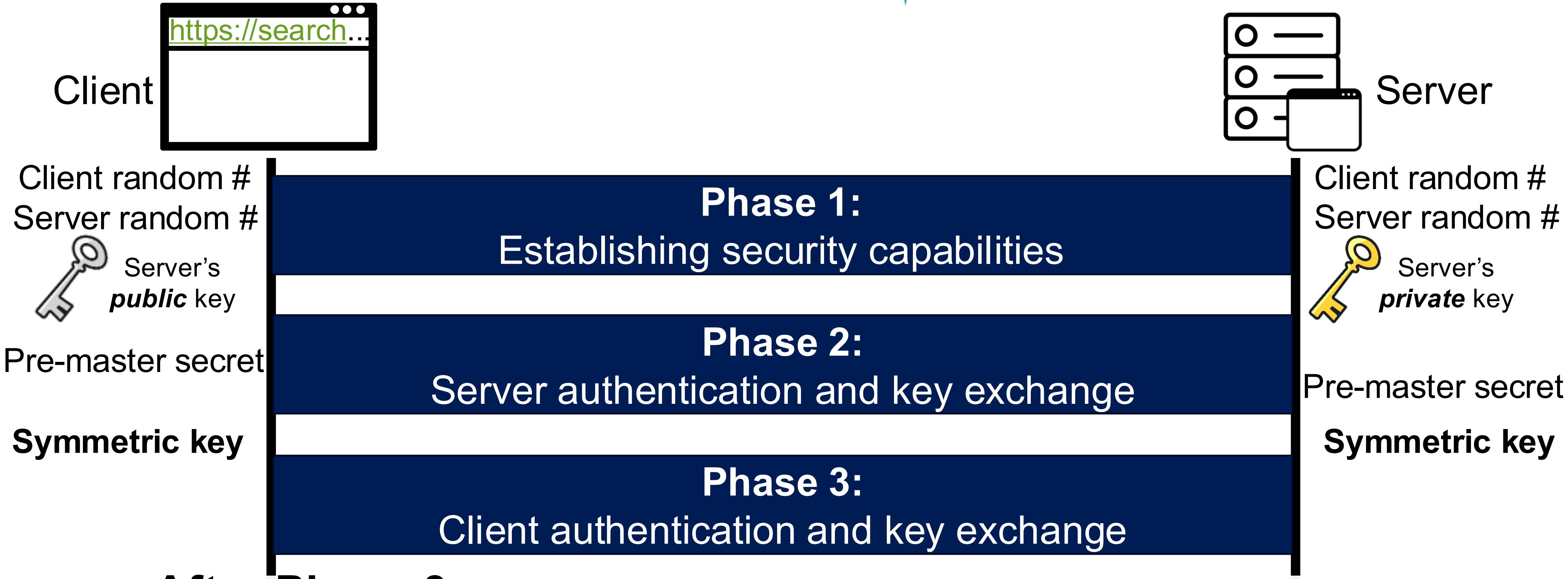
```
struct {
    ProtocolVersion client_version;
    opaque random[46];
} PreMasterSecret
```

Where do random bits come from?

Debian Linux (2006-08)

- A line of code commented out from `md_rand` (Developer's mistake!)
  - `MD_Update(&m,buf,j); /* purify complains */`
- Without this line, the seed for the pseudo-random generator is **derived only from process ID**
  - Default maximum on Linux = 32768
- Result: all keys generated using Debian-based OpenSSL package in 2006-08 are <u>predictable</u>
  - *"Affected keys include SSH keys, OpenVPN keys, DNSSEC keys, and key material for use in X.509 certificates and session keys used in SSL/TLS connections"*

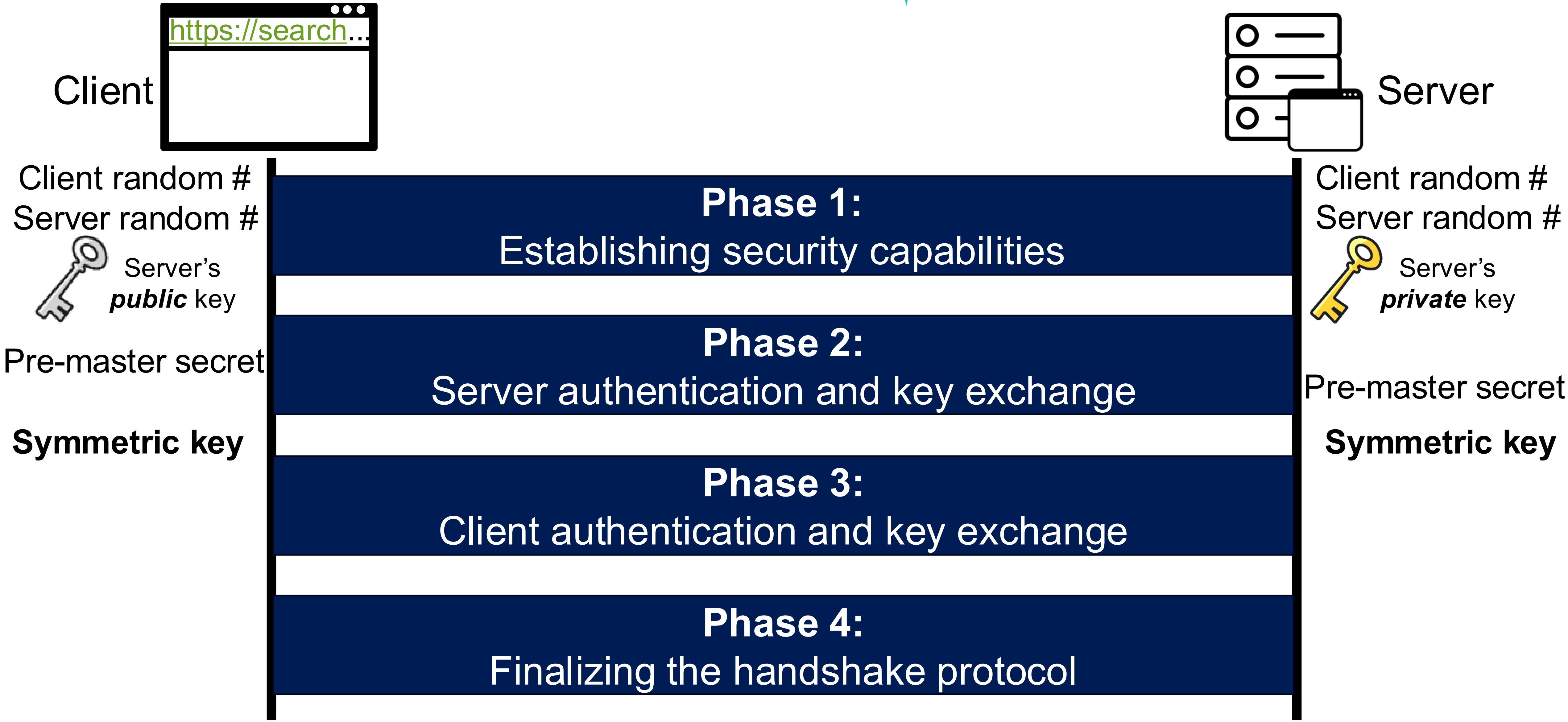
Phase 3: Client Auth. and Key Exchange



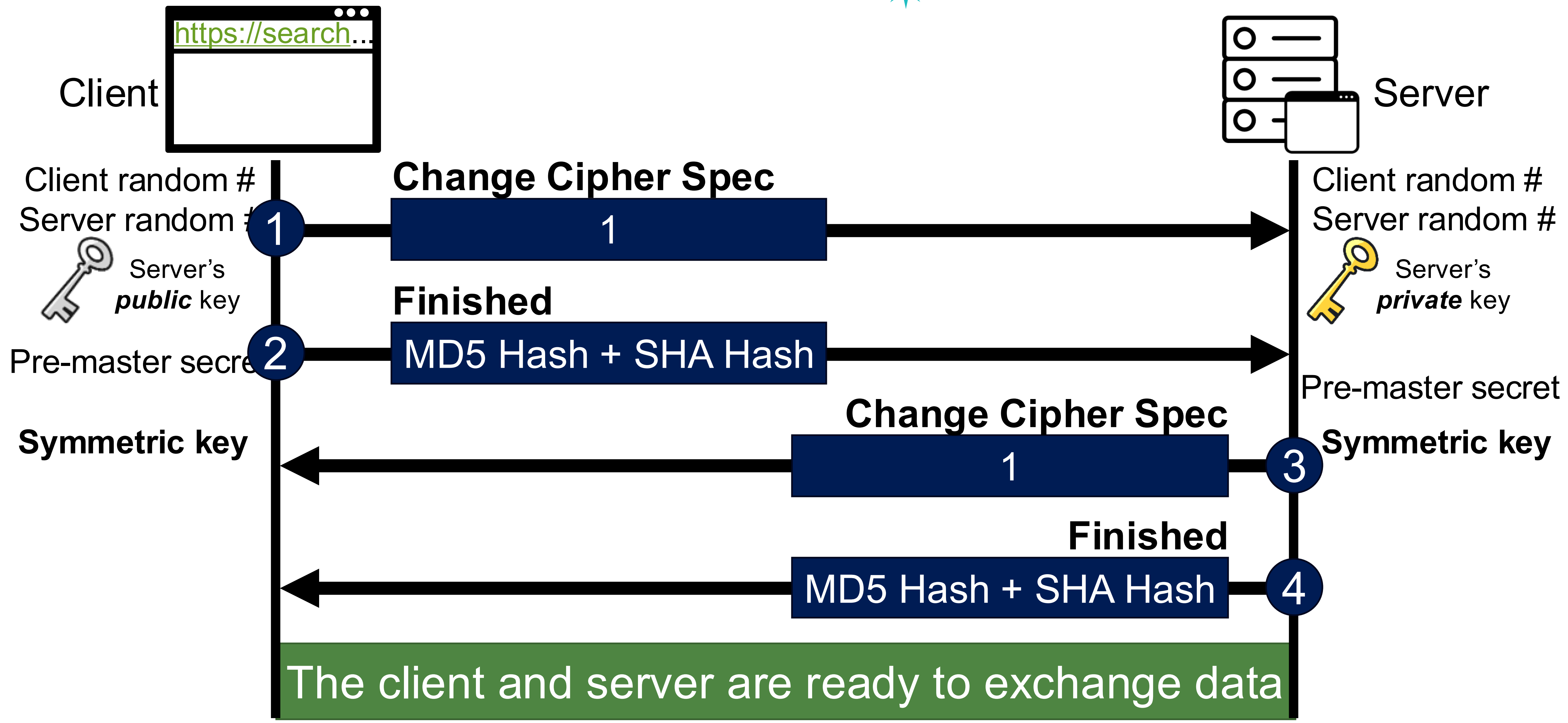
After Phase 3,

- (Optional) The client is authenticated for the server
- Both the client and the server know the pre-master secret

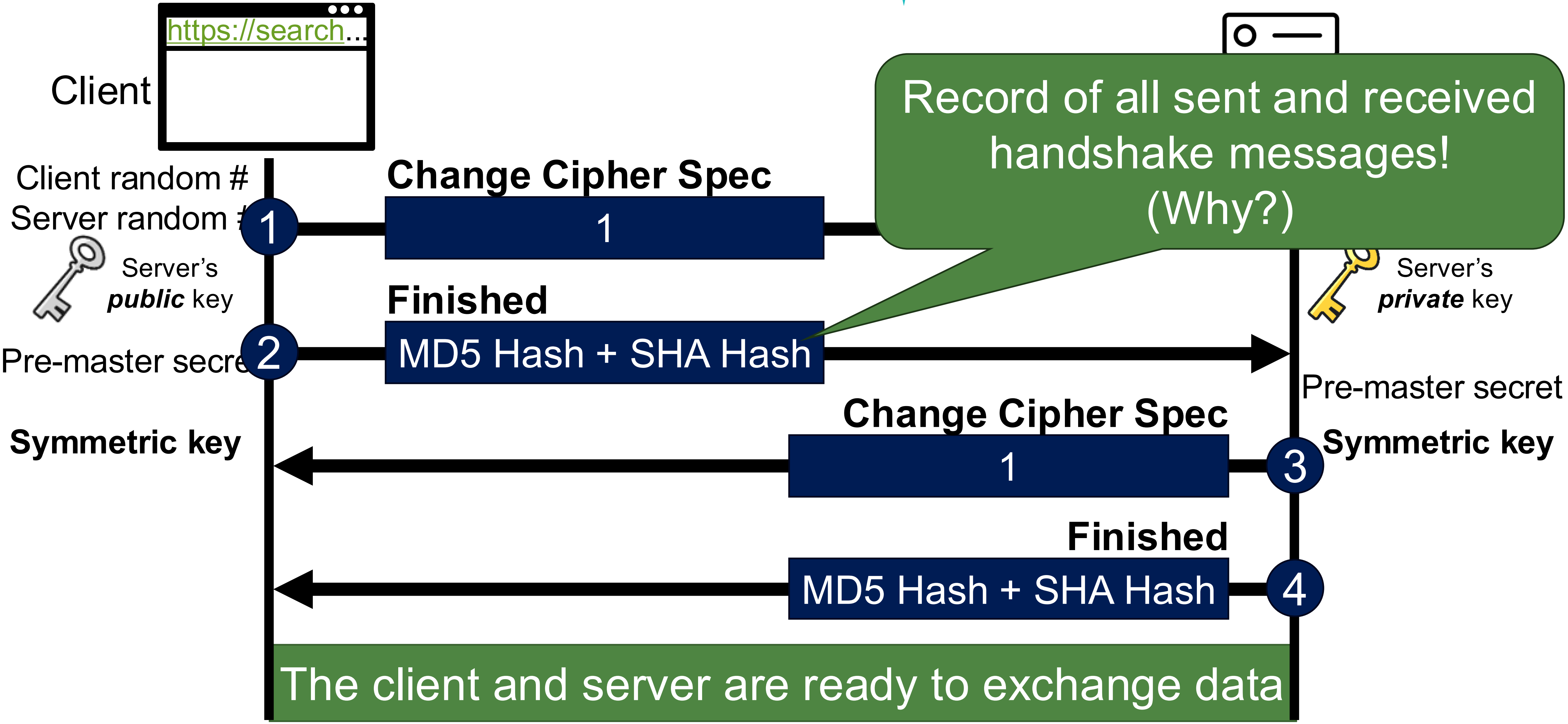
Phase 4: Finalizing the Handshake Protocol



Phase 4: Finalizing the Handshake Protocol



Phase 4: Finalizing the Handshake Protocol



Client

https://search...

Client random #

Server random #

Server's **public** key

Pre-master secre

Symmetric key

Change Cipher Spec

1 — 1

Finished

2 — MD5 Hash + SHA Hash

Record of all sent and received handshake messages! (Why?)

Server's **private** key

Pre-master secret

Symmetric key

Change Cipher Spec

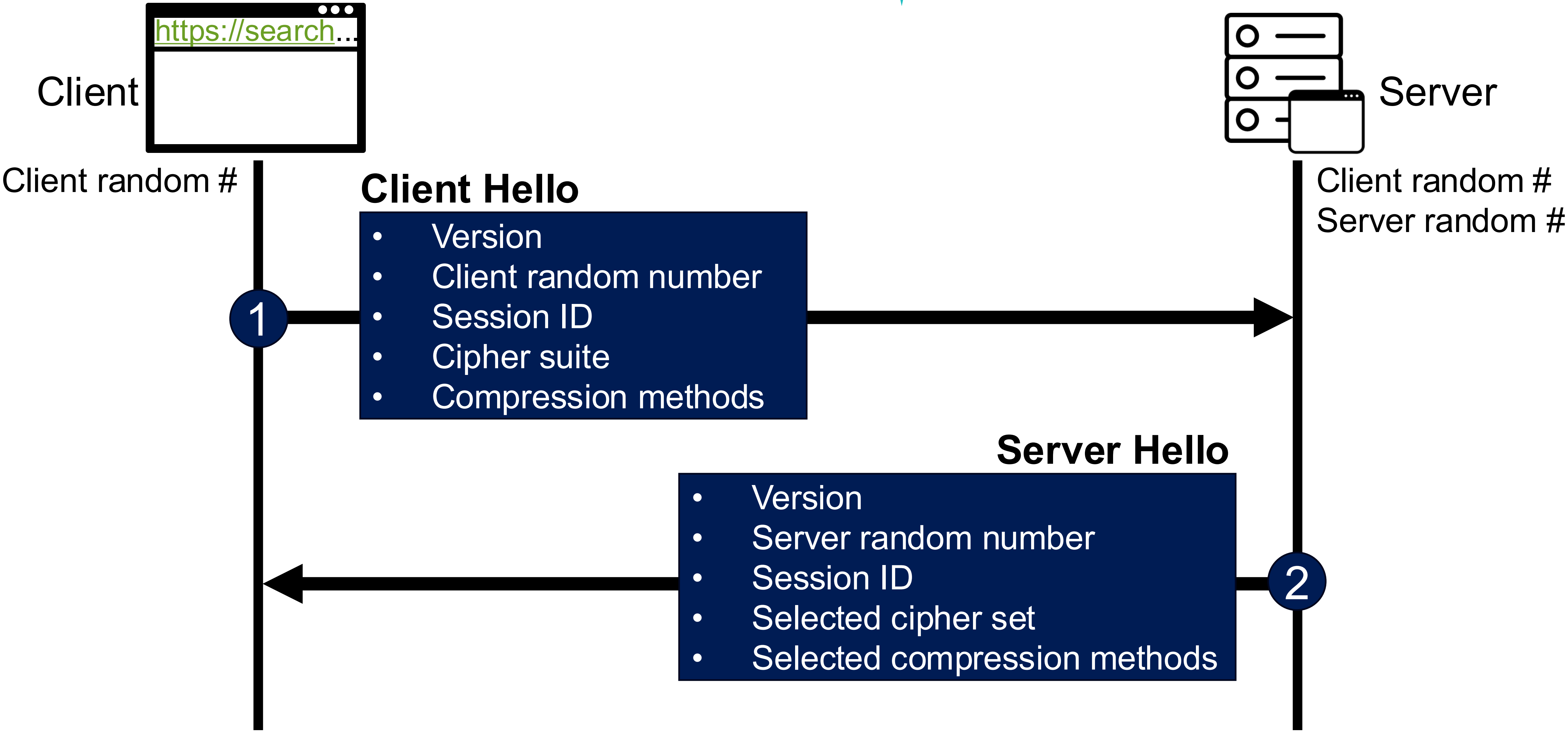
3 — 1

Finished

4 — MD5 Hash + SHA Hash

The client and server are ready to exchange data

Recap: Establishing Security Capabilities



Client

https://search...

Client random #

Server

Client random #
Server random #

1

Client Hello

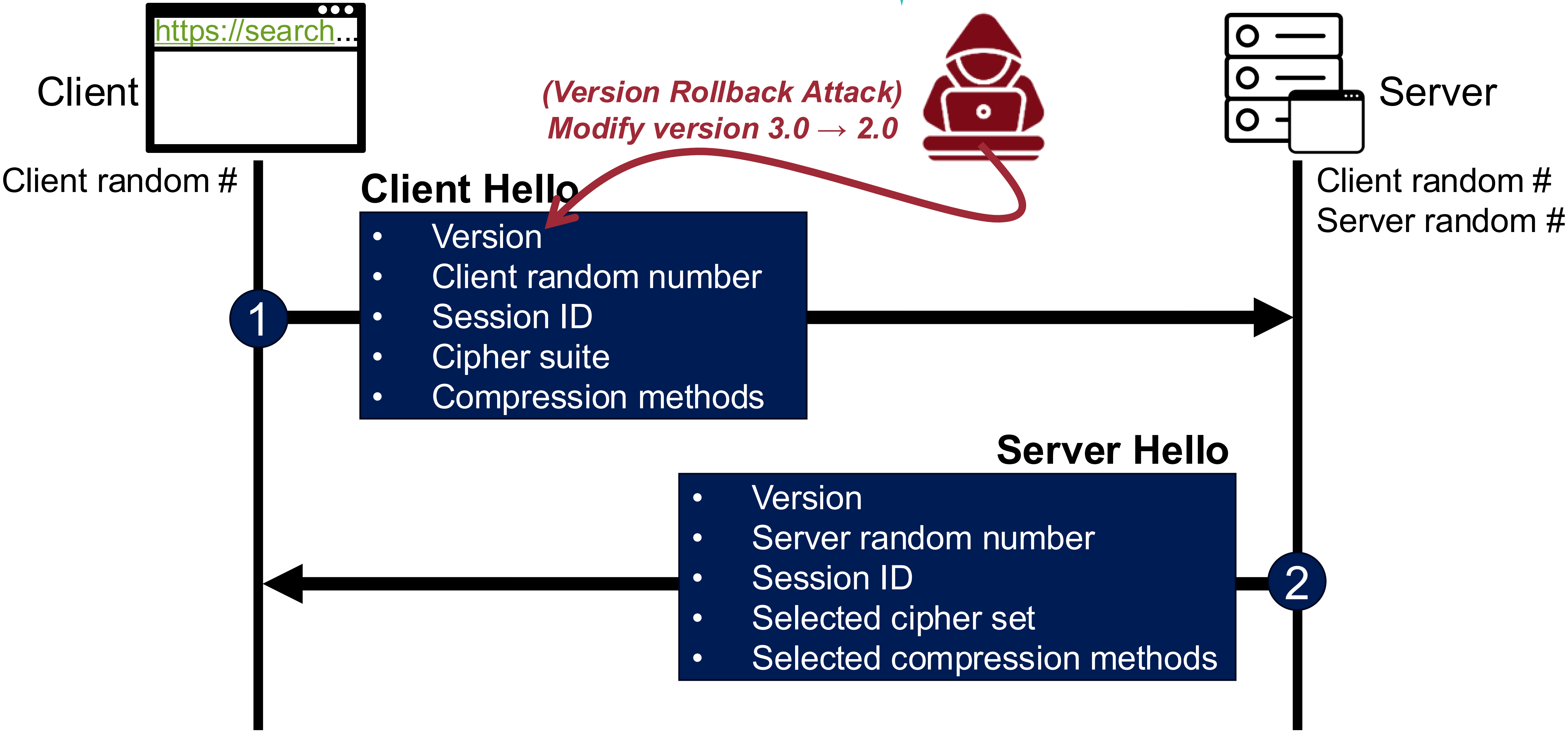
- Version
- Client random number
- Session ID
- Cipher suite
- Compression methods

2

Server Hello

- Version
- Server random number
- Session ID
- Selected cipher set
- Selected compression methods

Version Rollback Attack



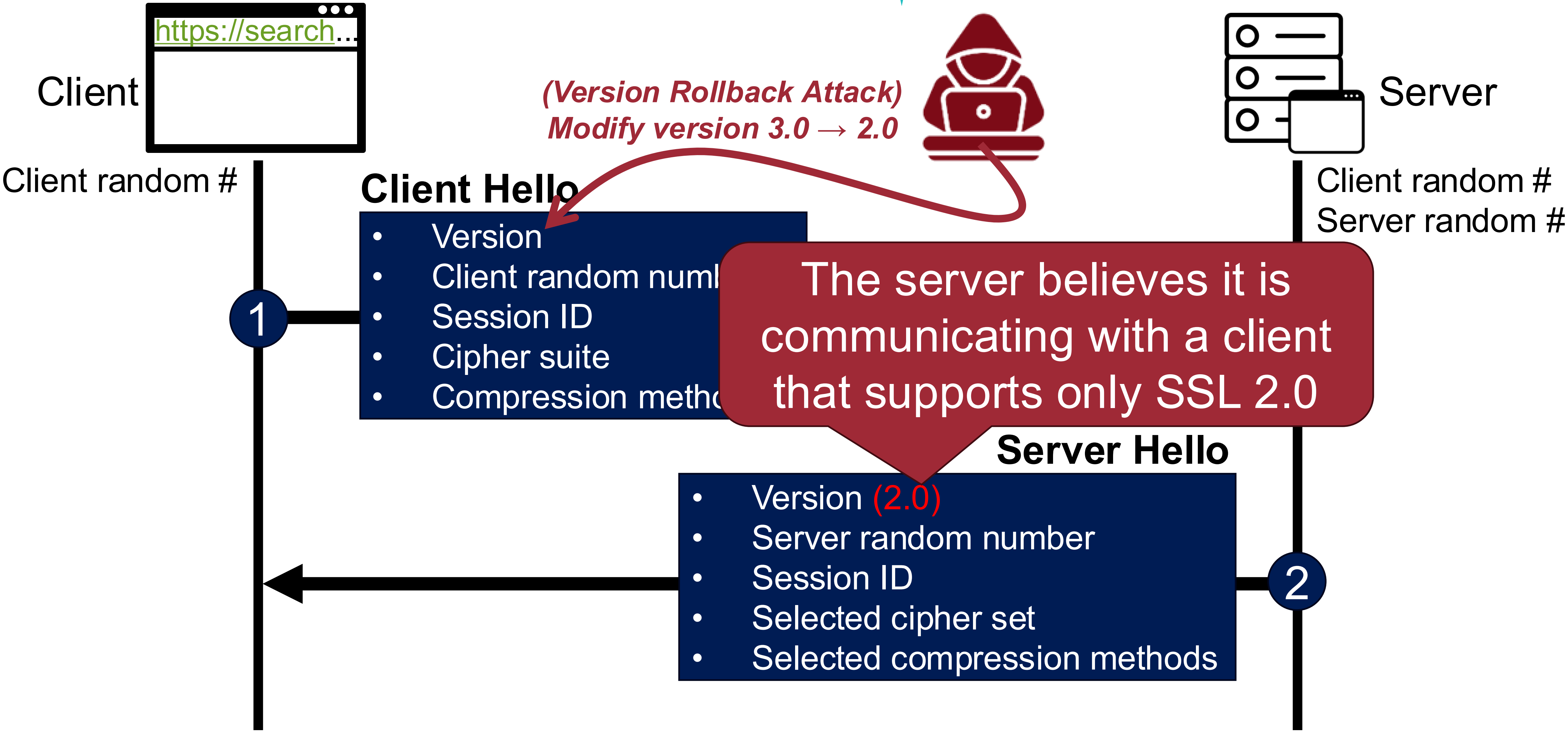
Client Hello

- Version
- Client random number
- Session ID
- Cipher suite
- Compression methods

Server Hello

- Version
- Server random number
- Session ID
- Selected cipher set
- Selected compression methods

Version Rollback Attack



(Version Rollback Attack)
Modify version 3.0 → 2.0

Client Hello

- Version
- Client random numb
- Session ID
- Cipher suite
- Compression metho

The server believes it is communicating with a client that supports only SSL 2.0

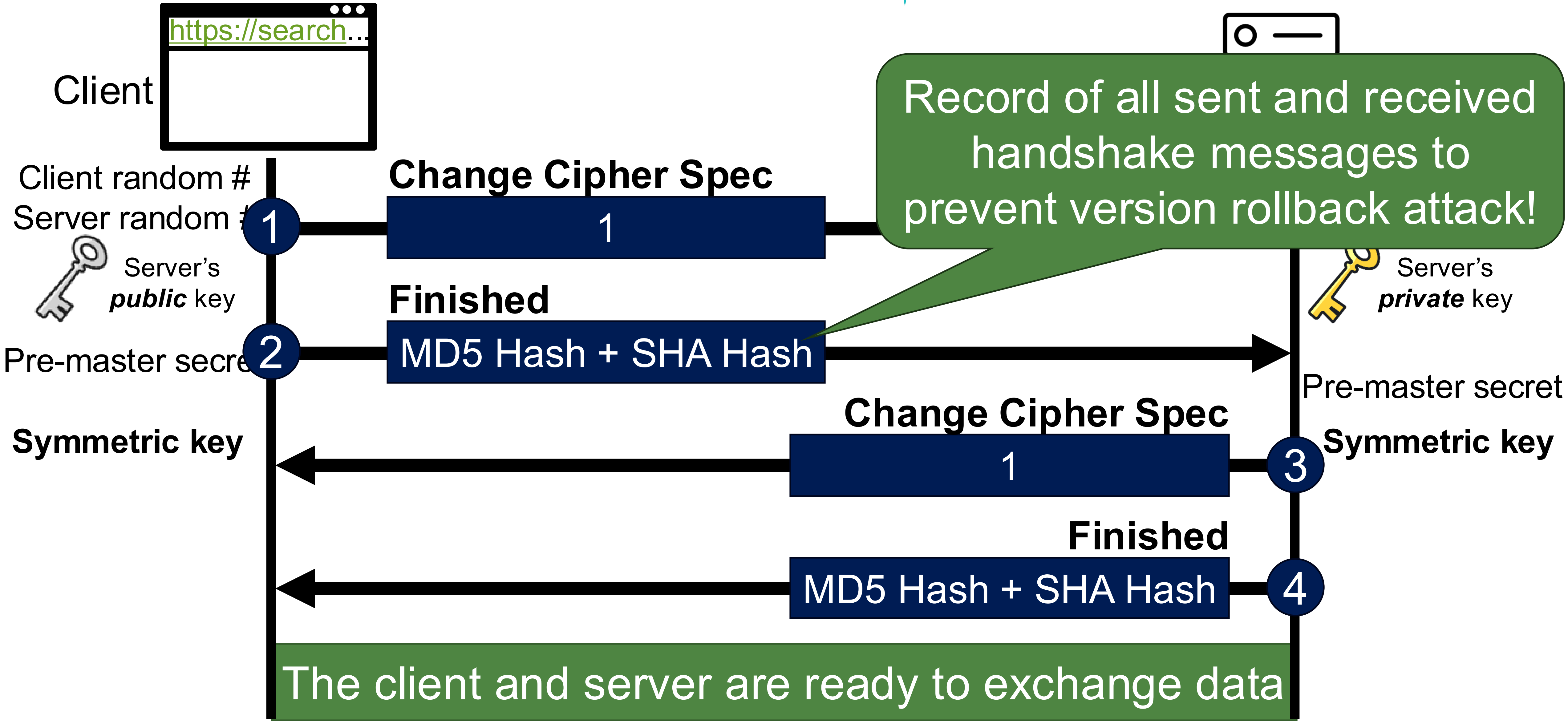
Server Hello

- Version (2.0)
- Server random number
- Session ID
- Selected cipher set
- Selected compression methods

SSL 2.0 Weaknesses (Fixed in 3.0)

- Cipher suite preferences are not authenticated
  - “Cipher suite rollback” attack is possible
- Weak MAC construction, MAC hash uses only 40 bits in export mode (TLS 1.3 AES_256_CBC_SHA256, MAC keysize: 64 bytes)
- SSL 2.0 uses padding when computing MAC in block cipher modes, but padding length field is not authenticated
  - Attacker can delete bytes from the end of messages
- No support for certificate chains or non-RSA algorithms

Phase 4: Finalizing the Handshake Protocol



https://search...

Client

Client random #

Server random #

Server's **public** key

Pre-master secret

Symmetric key

1

Change Cipher Spec

1

2

Finished

MD5 Hash + SHA Hash

Record of all sent and received handshake messages to prevent version rollback attack!

Server's **private** key

Pre-master secret

Symmetric key

3

Change Cipher Spec

1

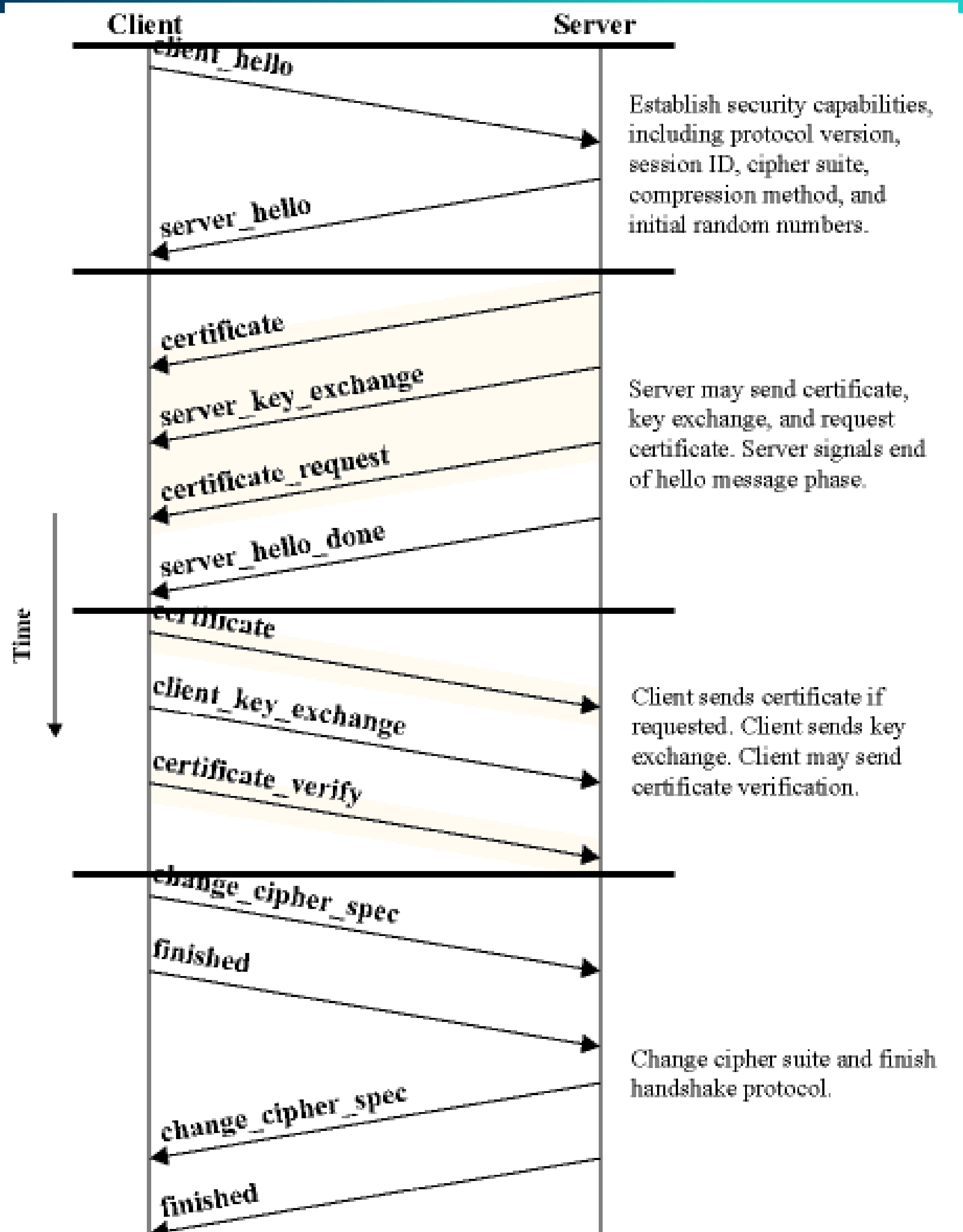
4

Finished

MD5 Hash + SHA Hash

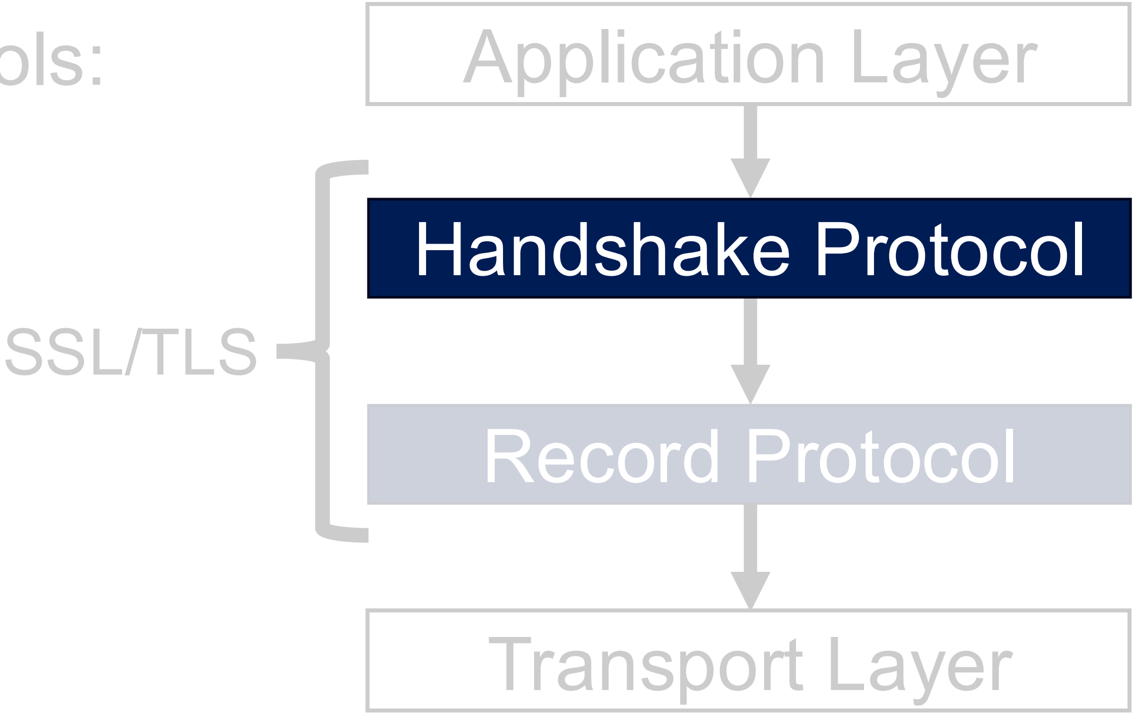
The client and server are ready to exchange data

Handshake Protocol Summary


Client
Server
client_hello
server_hello
Establish security capabilities, including protocol version, session ID, cipher suite, compression method, and initial random numbers.
certificate
server_key_exchange
certificate_request
server_hello_done
Server may send certificate, key exchange, and request certificate. Server signals end of hello message phase.
Time
certificate
client_key_exchange
certificate_verify
Client sends certificate if requested. Client sends key exchange. Client may send certificate verification.
change_cipher_spec
finished
change_cipher_spec
finished
Change cipher suite and finish handshake protocol.

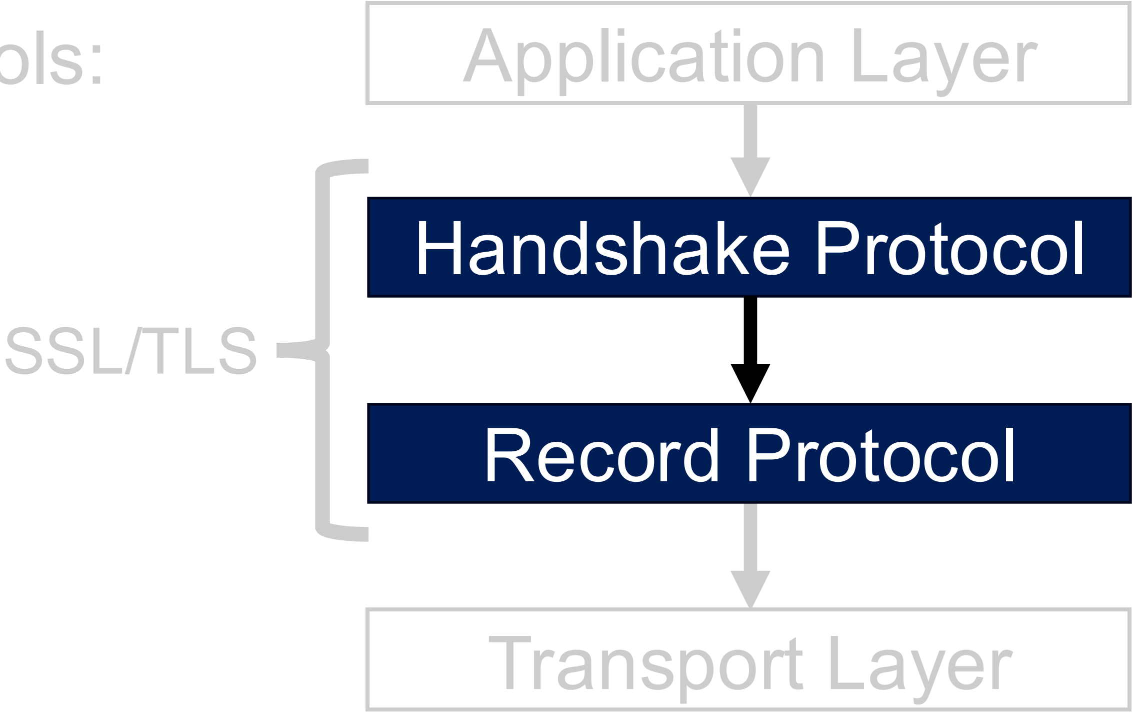
SSL/TLS Basics

- Runs in the presentation layer
- Uses symmetric crypto, asymmetric crypto, and digital signatures
- Composed of two layers of protocols:
  1. Handshake protocol
  2. Record protocol



SSL/TLS Basics

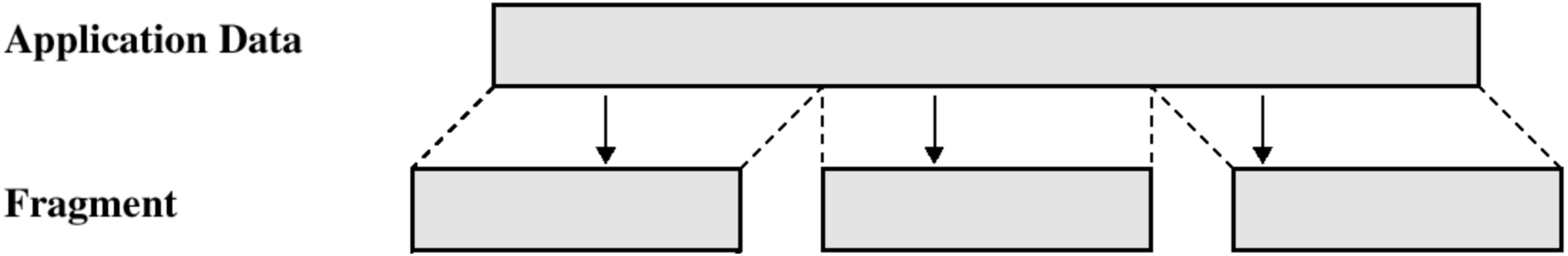
- Runs in the presentation layer
- Uses symmetric crypto, asymmetric crypto, and digital signatures
- Composed of two layers of protocols:
  1. Handshake protocol
  2. Record protocol



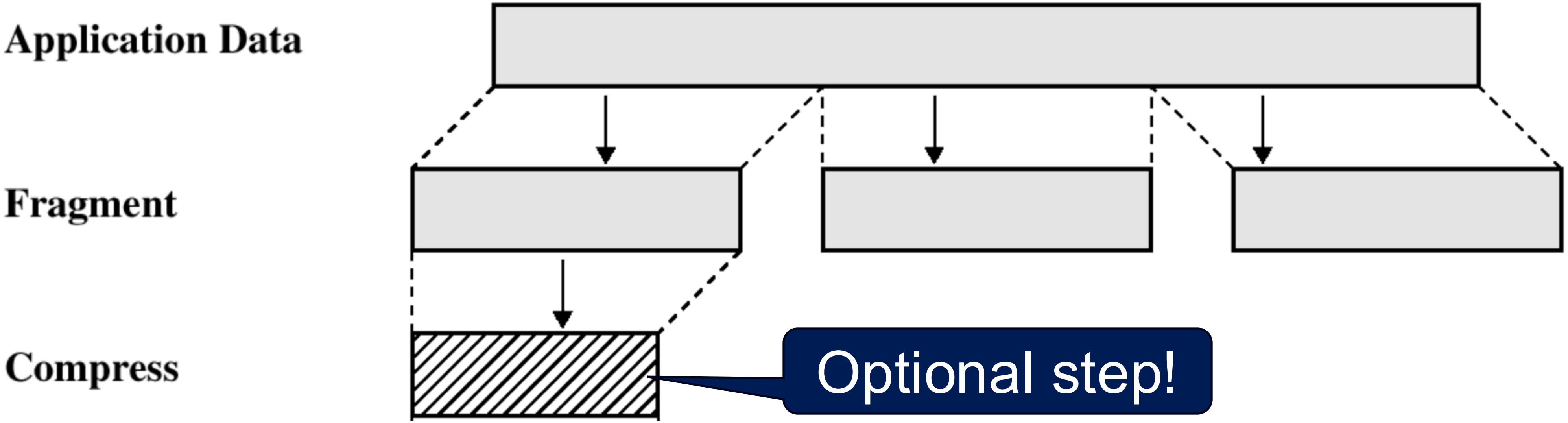
SSL Record Protocol Operation

Application Data

SSL Record Protocol Operation

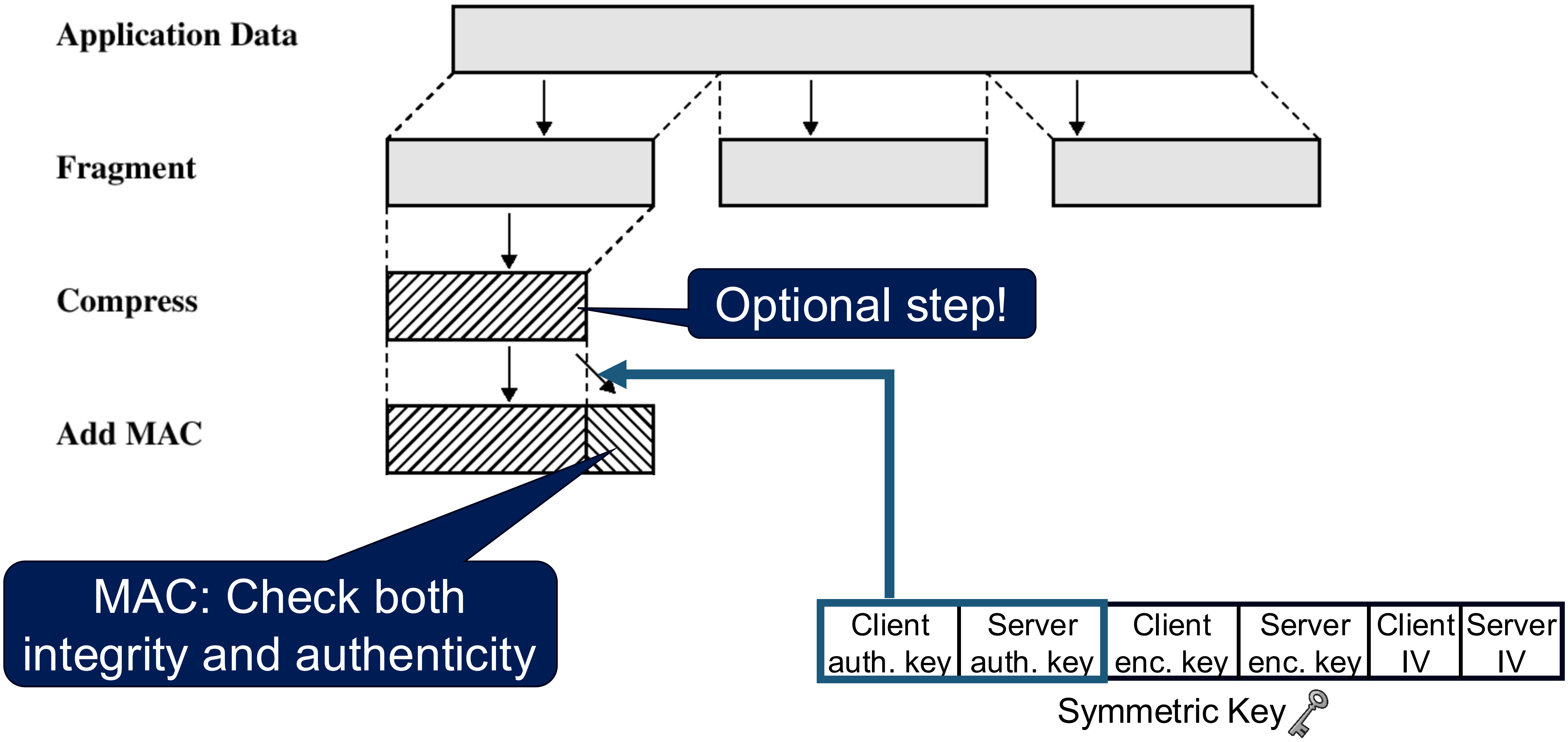


SSL Record Protocol Operation



Optional step!

SSL Record Protocol Operation



Application Data

Fragment

Compress

Add MAC

Optional step!

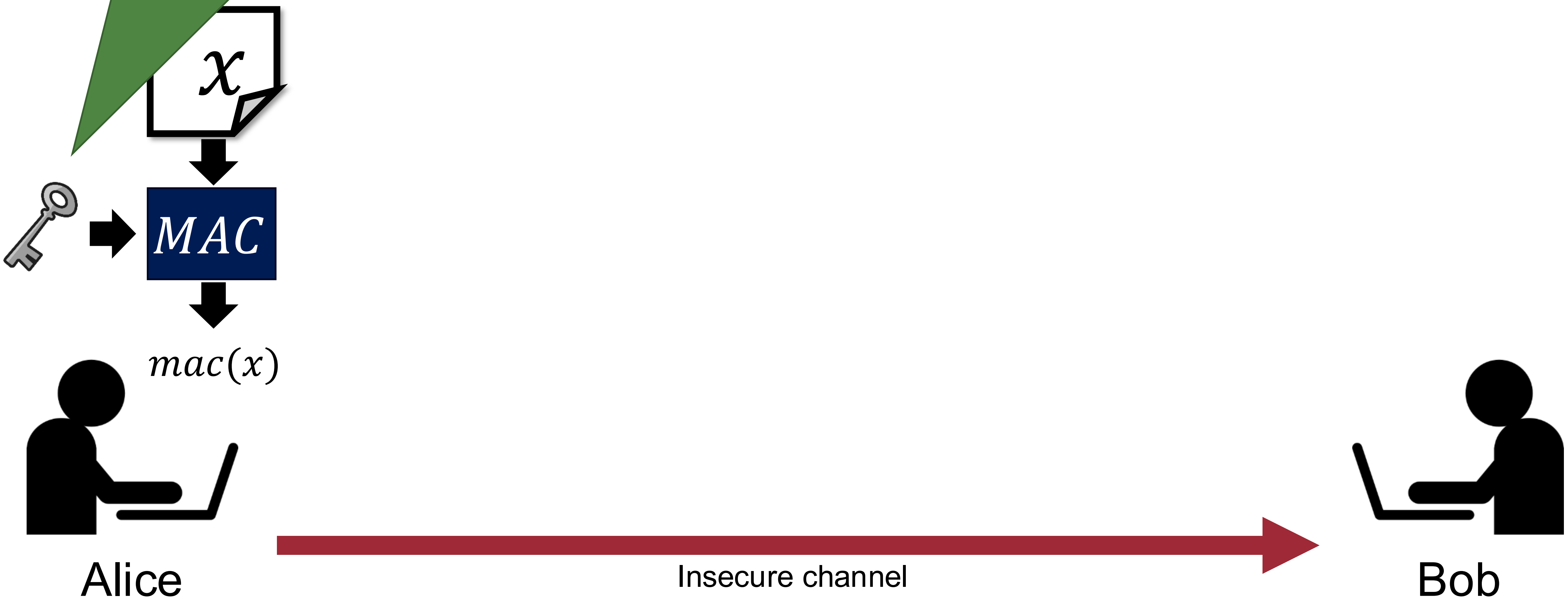
MAC: Check both integrity and authenticity

Client auth. key	Server auth. key	Client enc. key	Server enc. key	Client IV	Server IV

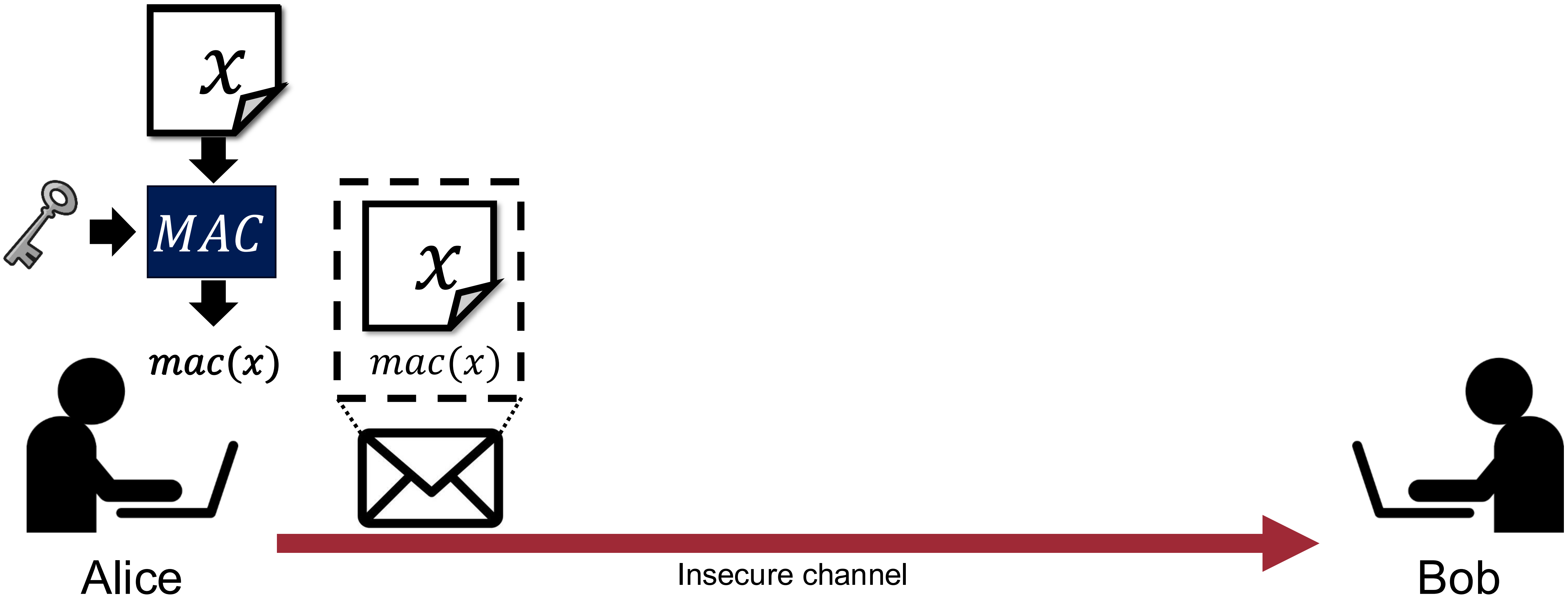
Symmetric Key

Ref: Message Authentication Codes (MAC)

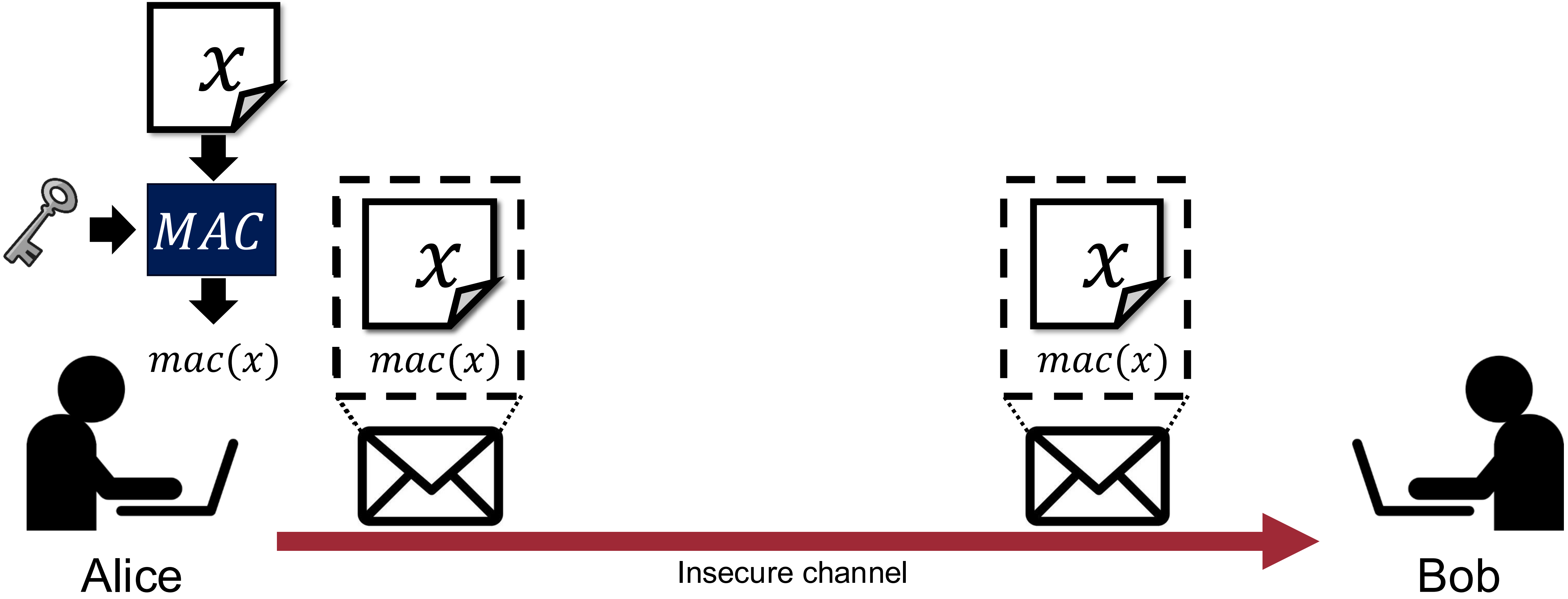
Use the symmetric key!



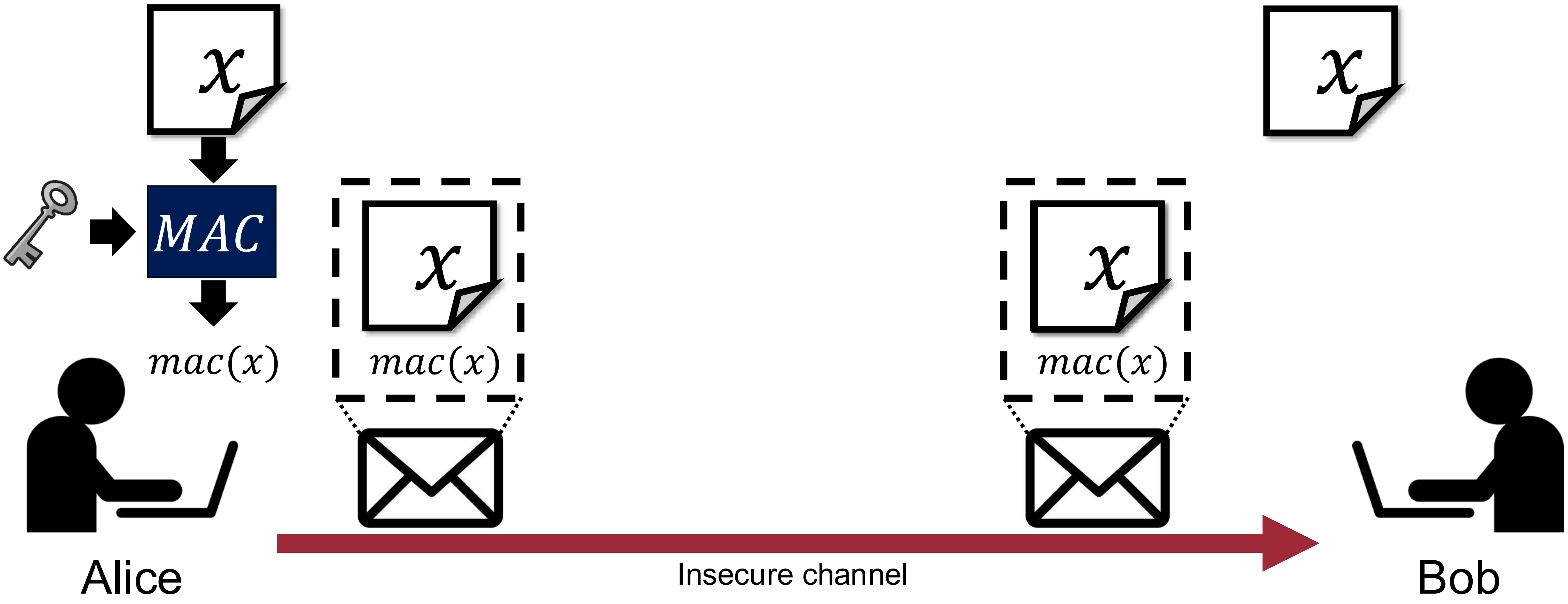
Ref: Message Authentication Codes (MAC)



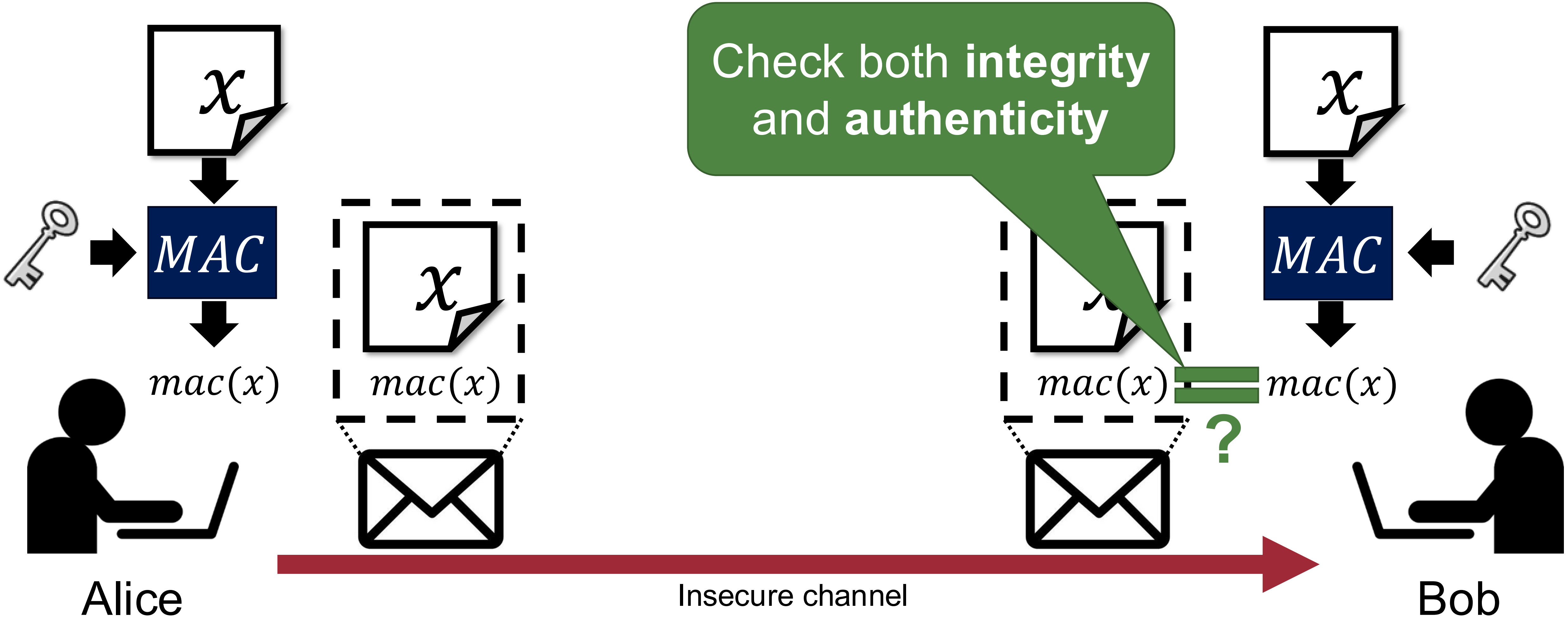
Ref: Message Authentication Codes (MAC)



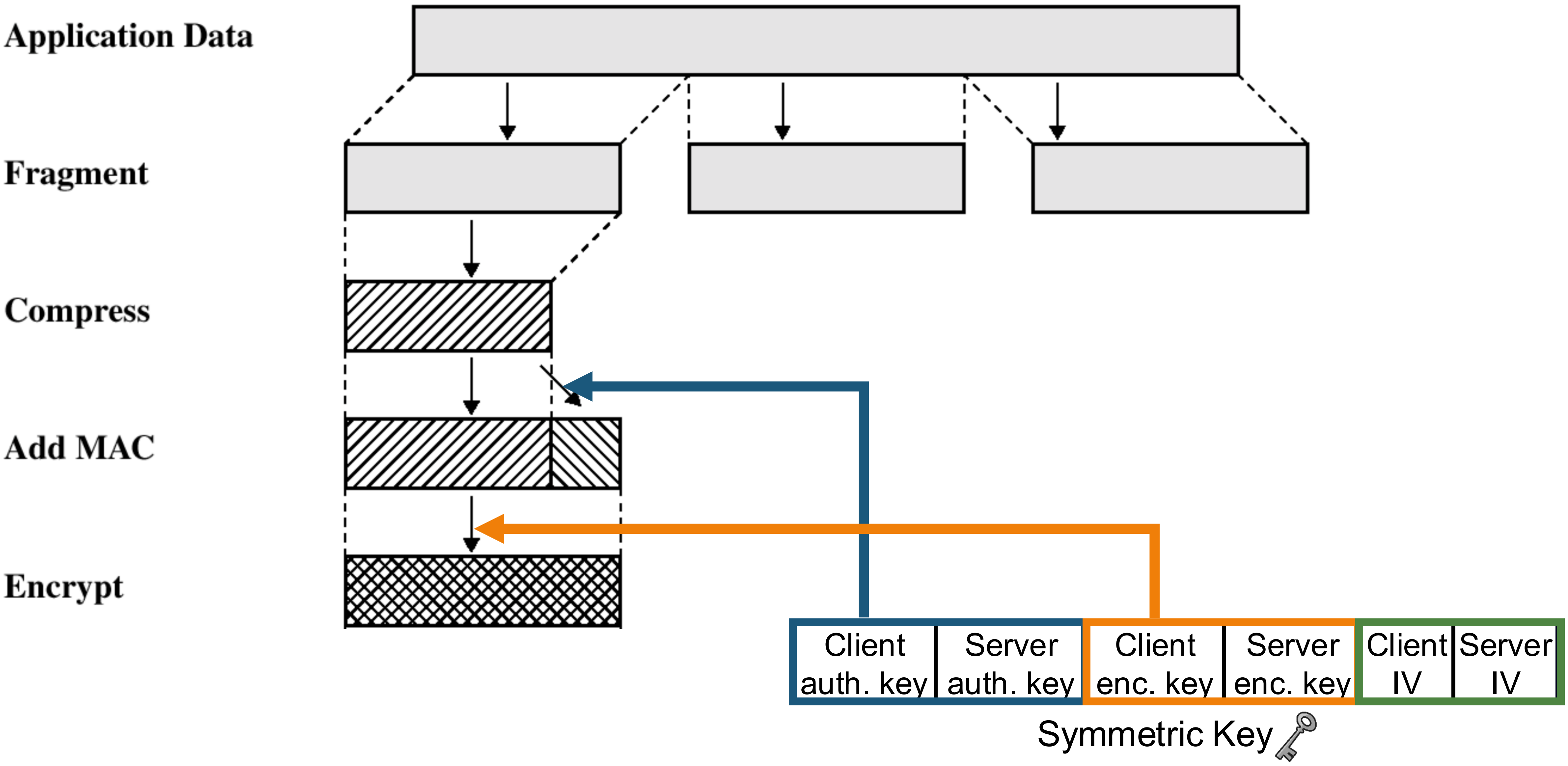
Ref: Message Authentication Codes (MAC)



Ref: Message Authentication Codes (MAC)



SSL Record Protocol Operation



Application Data

Fragment

Compress

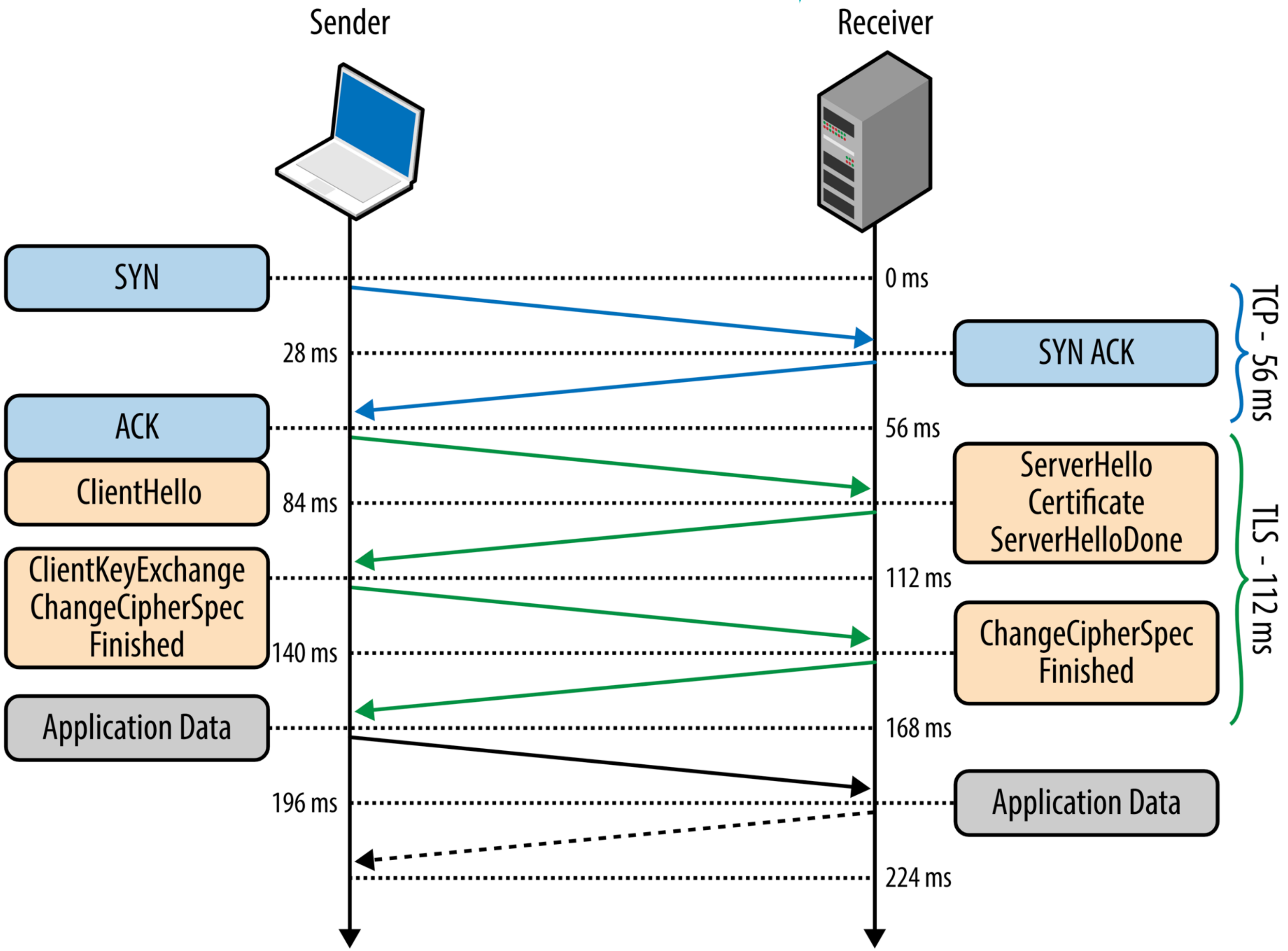
Add MAC

Encrypt

Client auth. key | Server auth. key | Client enc. key | Server enc. key | Client IV | Server IV

Symmetric Key

SSL/TLS Final Overview



How SSL/TLS Provides Security Properties?

- Security goals: achieving confidentiality, integrity, and authentication
  - **Confidentiality**
    - Asymmetric-key algorithm for key exchange (pre-master key)
    - Symmetric-key algorithm for data exchange
  - **Integrity**:
    - MAC (with hash algorithm)
    - If an attacker modifies the message, the recipient can detect the modification
  - **Authentication**
    - Authenticate the identity of the server using the server’s certificate

How SSL/TLS Provides Security Properties?

- Security goals: achieving confidentiality, integrity, and authentication
  - **Confidentiality**
    - Asymmetric-key algorithm for key exchange (pre-master key)

Are we safe now?

  - **Integrity**
    - MAC (with hash algorithm)
    - If an attacker modifies the message, the recipient can detect the modification
  - **Authentication**
    - Authenticate the identity of the server using the server’s certificate

SSL/TLS Implementations

- Many open-source implementations of SSL/TLS are available for developers

OpenSSL
Cryptography and SSL/TLS Toolkit

NSS LABS

POLARSSL
Straightforward, Secure Communication

yaSSL

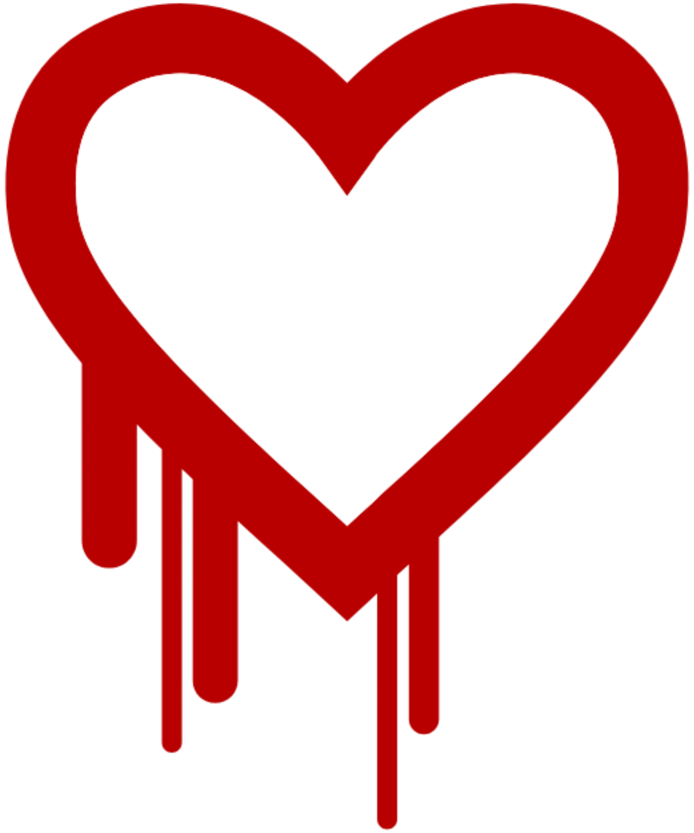
matrixSSL

Can We Believe the SSL/TLS Implementations?

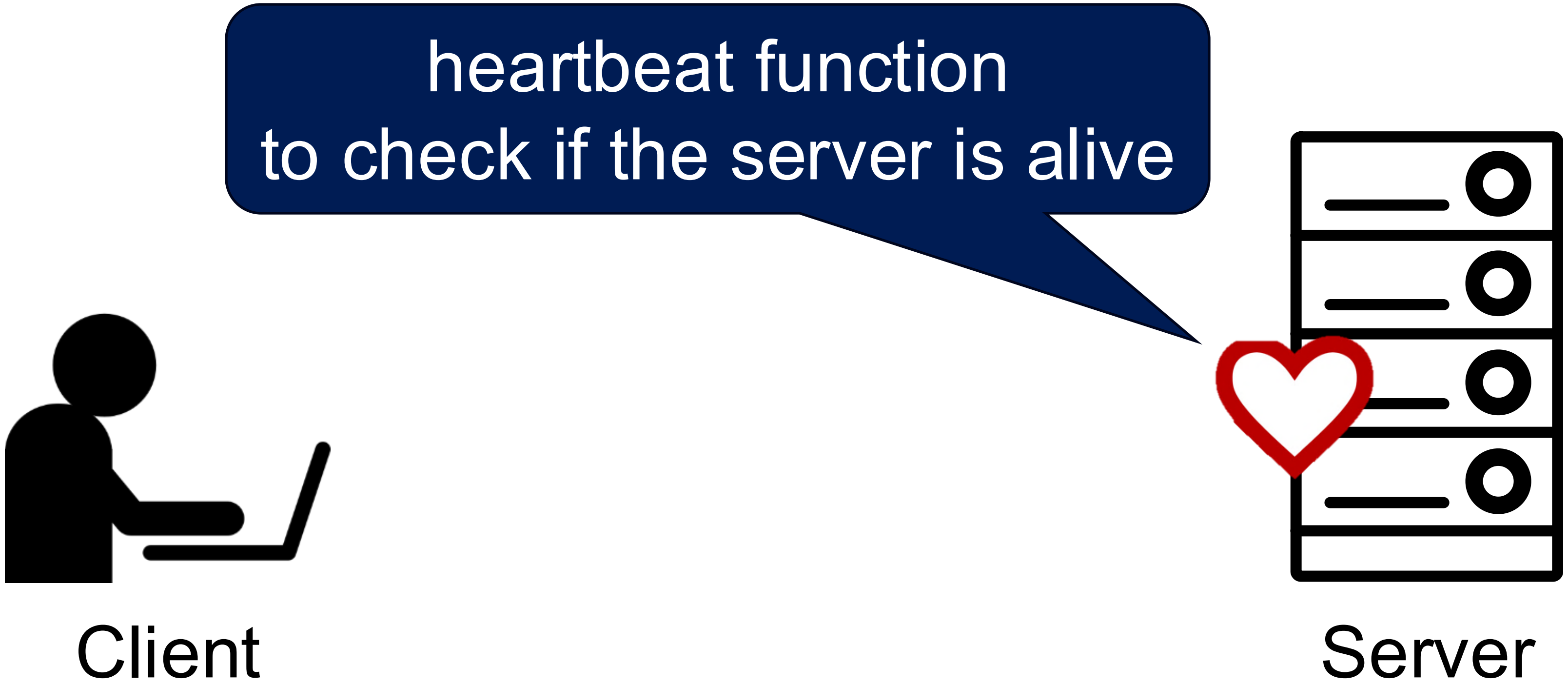
Heartbleed Bug (in 2014)

- Famous bug in OpenSSL (in TLS ***heartbeat***)

- An attacker can steal private keys



Heartbleed Bug: High-level Workflow

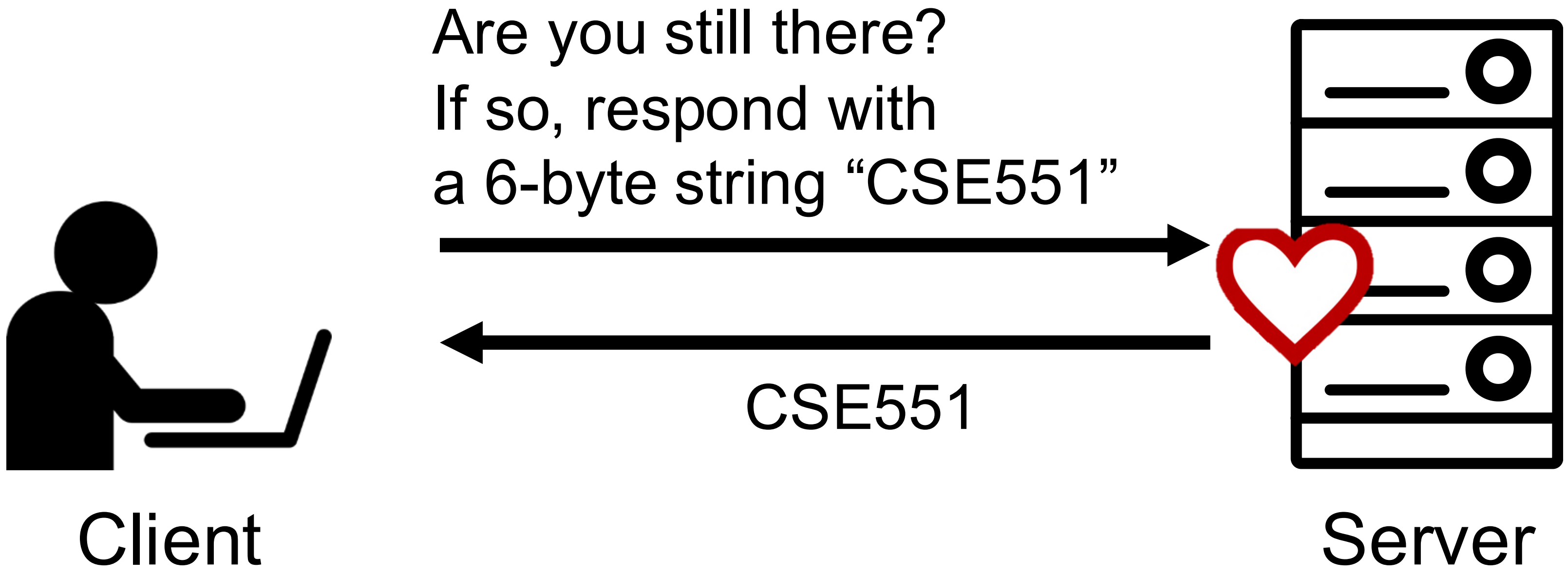


heartbeat function
to check if the server is alive

Client

Server

Heartbleed Bug: High-level Workflow



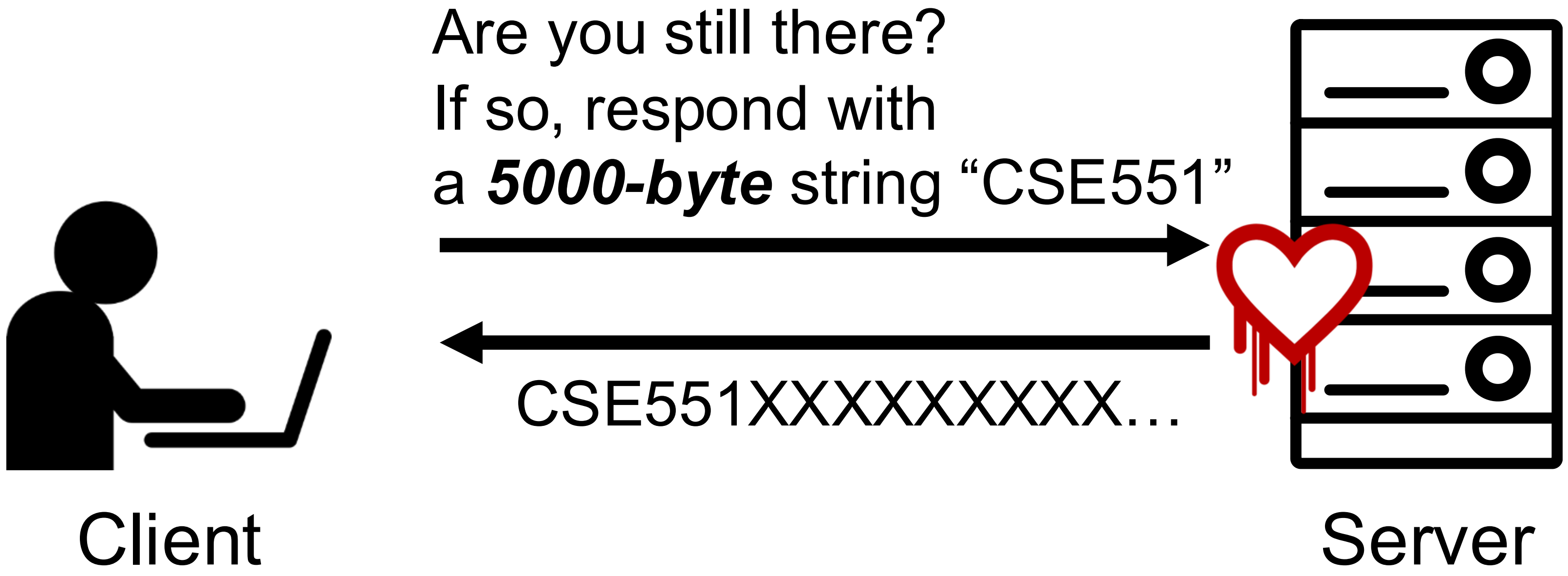
Are you still there?
If so, respond with
a 6-byte string “CSE551”

CSE551

Client

Server

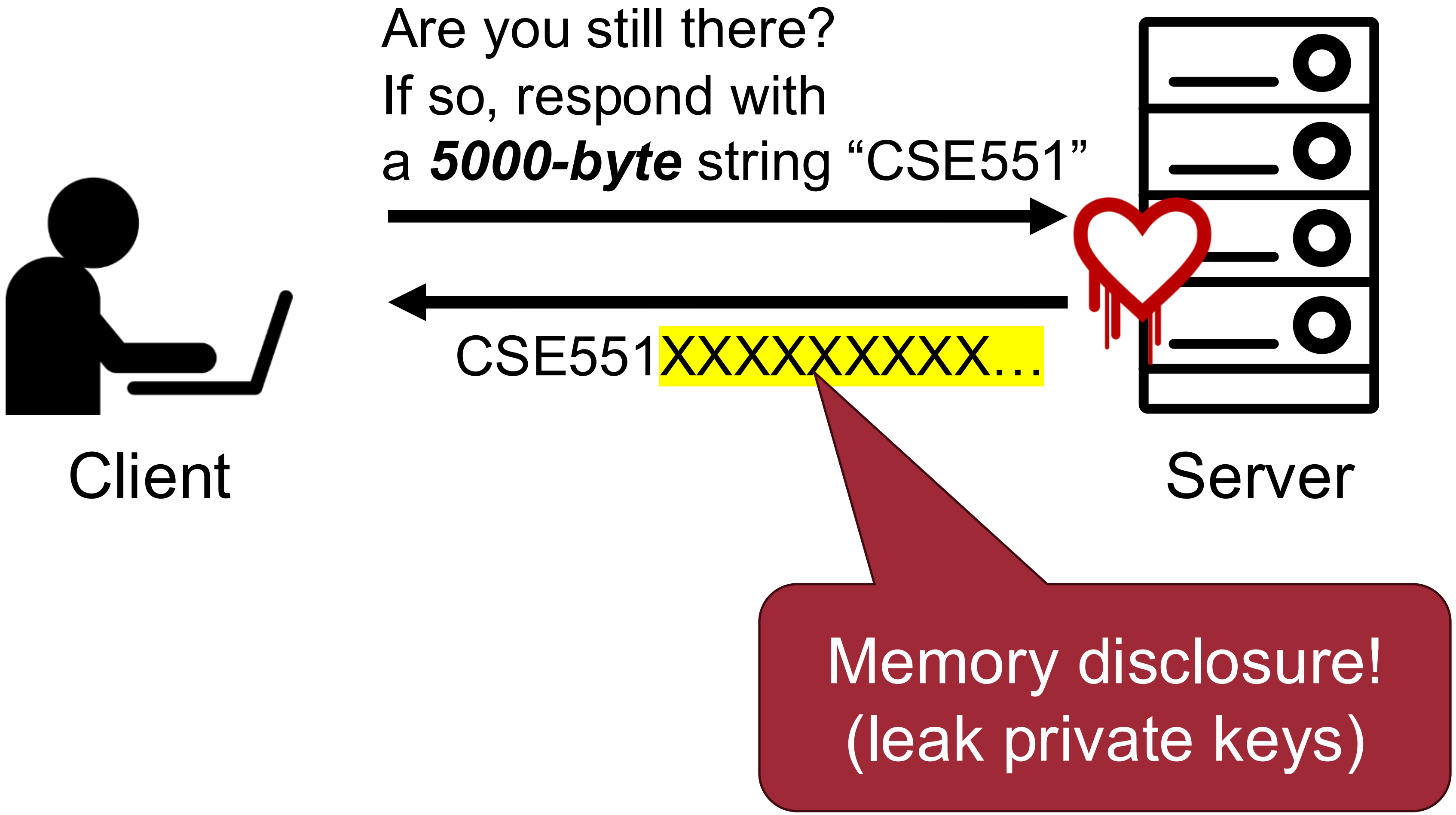
Heartbleed Bug: High-level Workflow



Are you still there?
If so, respond with
a ***5000-byte*** string “CSE551”

CSE551XXXXXXXXXX…

Heartbleed Bug: High-level Workflow



Are you still there?
If so, respond with
a ***5000-byte*** string “CSE551”

CSE551XXXXXXXXX…

Client

Server

Memory disclosure!
(leak private keys)

The Bug

```
struct {
    HeartbeatMessageType type;
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque padding[padding_length];
} HeartbeatMessage;

struct {
    unsigned int length;
    unsigned char *data;
    ...
} SSL3_RECORD;
```

The Bug

```
struct {
    HeartbeatMessageType typ
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque paddin
} HeartbeatMessa

struct {
    unsigned int length;
    unsigned char *data;
    ...
} SSL3_RECORD;

memcpy(bp, pl, length);  // vulnerable spot!
```

Calculated from the user's payload (i.e., 6)

Payload obtained from `HeartbeatMessage` (i.e., CSE467)

Obtained from the user's input (i.e., 5000)

Copy arbitrary memory contents of a server! TLS secret key may be available

The Bug

```
struct {
    HeartbeatMessageType typ
    uint16 payload_length;
    opaque payload[HeartbeatMessage.payload_length];
    opaque paddi
} HeartbeatMessa

struct {
    unsigned int length;
    unsigned char *data;
    ...
} SSL3_RECORD;

memcpy(bp, pl, length);  // vulnerable spot!
```

Calculated from the user's payload (i.e., 6)

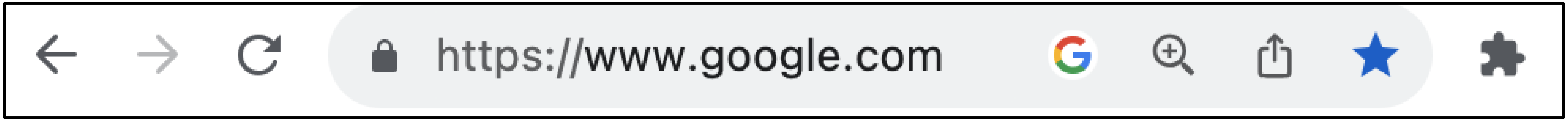
Payload obtained from `HeartbeatMessage` (i.e., CSE467)

Obtained from the user's input (i.e., 5000)

Root cause:
Did not check the consistency of the values of the two variables!

Copy arbitrary memory contents of a server! TLS secret key may be available

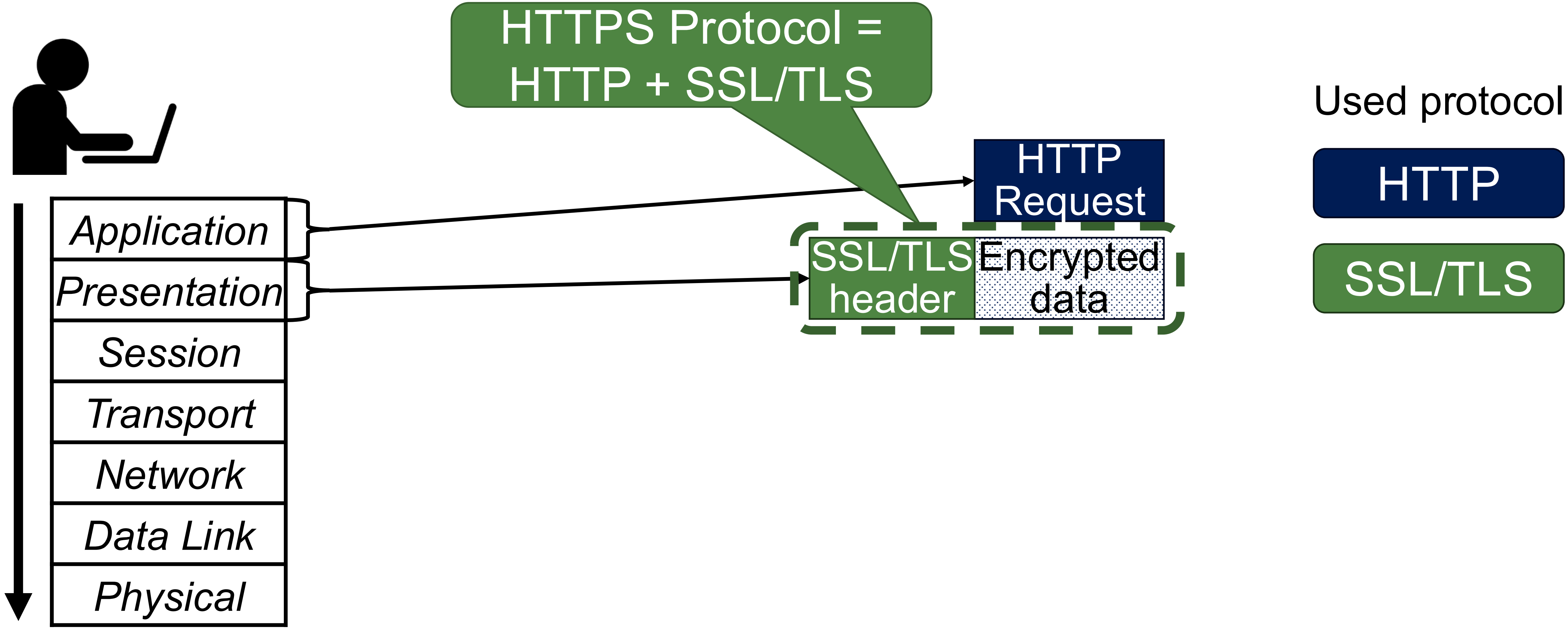
HTTPS *(Most common use of SSL/TLS)*



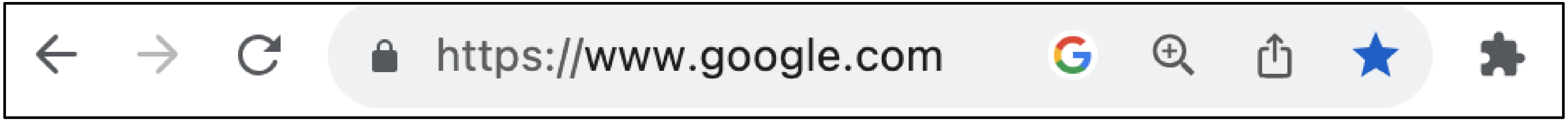
https://www.google.com

HTTPS

- Adding a protocol layer for secure communication!



HTTPS – The Lock Icon

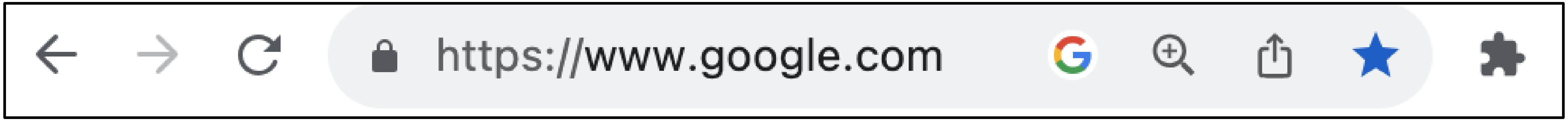


https://www.google.com

- Goal: the client (Human) can identify secure connection
  - SSL/TLS is being used to protect against active network attacker

- Lock icon should only be show when the page is secure against network attacker
  - All elements on the page fetched using HTTPS
  - Contents of the page have not been viewed or modified by an attacker
  - HTTPS certificate is valid – “This webpage is really comes from google.com server!”

HTTPS – The Lock Icon



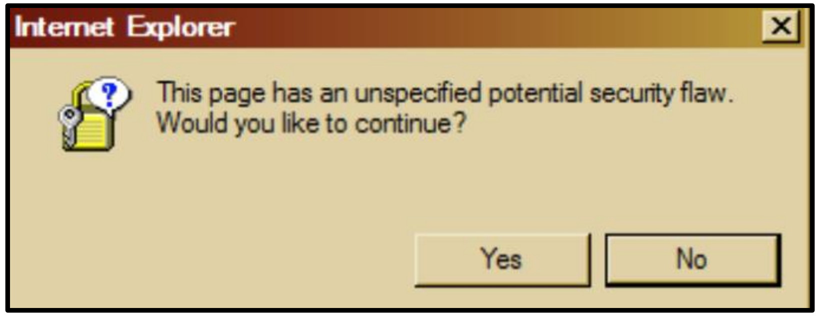
https://www.google.com

What happens if page served over HTTPS but contains HTTP?

- Goal: the client (Human) can identify s
  - SSL/TLS is being used to protect against
- Lock icon should only be show when the ge is secure against network attacker
  - All elements on the page fetched using HTTPS
  - Contents of the page have not been viewed or modified by an attacker
  - HTTPS certificate is valid – "This webpage is really comes from google.com server!"

Mixed Content: Combining HTTPS and HTTP

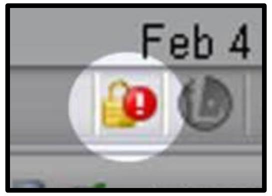
- Page served over HTTPS but contains HTTP
  - IE 7: no lock, warning



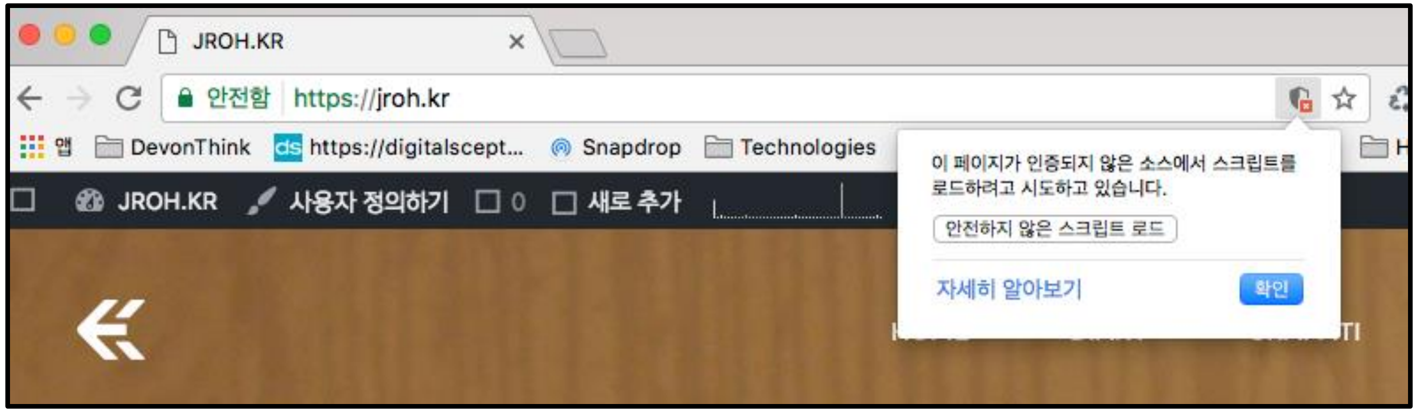
Internet Explorer

This page has an unspecified potential security flaw.
Would you like to continue?

  - Firefox: “!” over lock, no warning by default

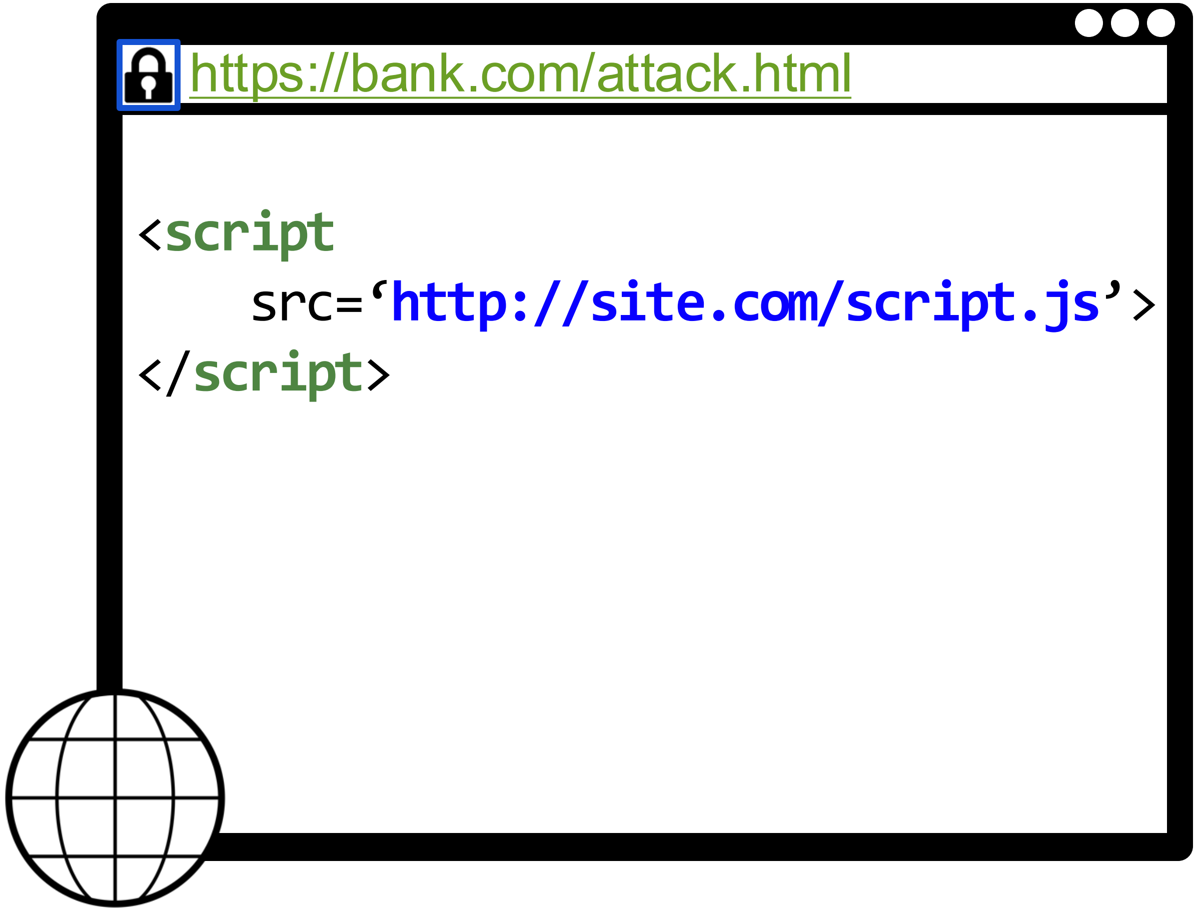


  - Safari: does not detect mixed content
  - Chrome: lock icon, warning



이 페이지가 인증되지 않은 소스에서 스크립트를 로드하려고 시도하고 있습니다.

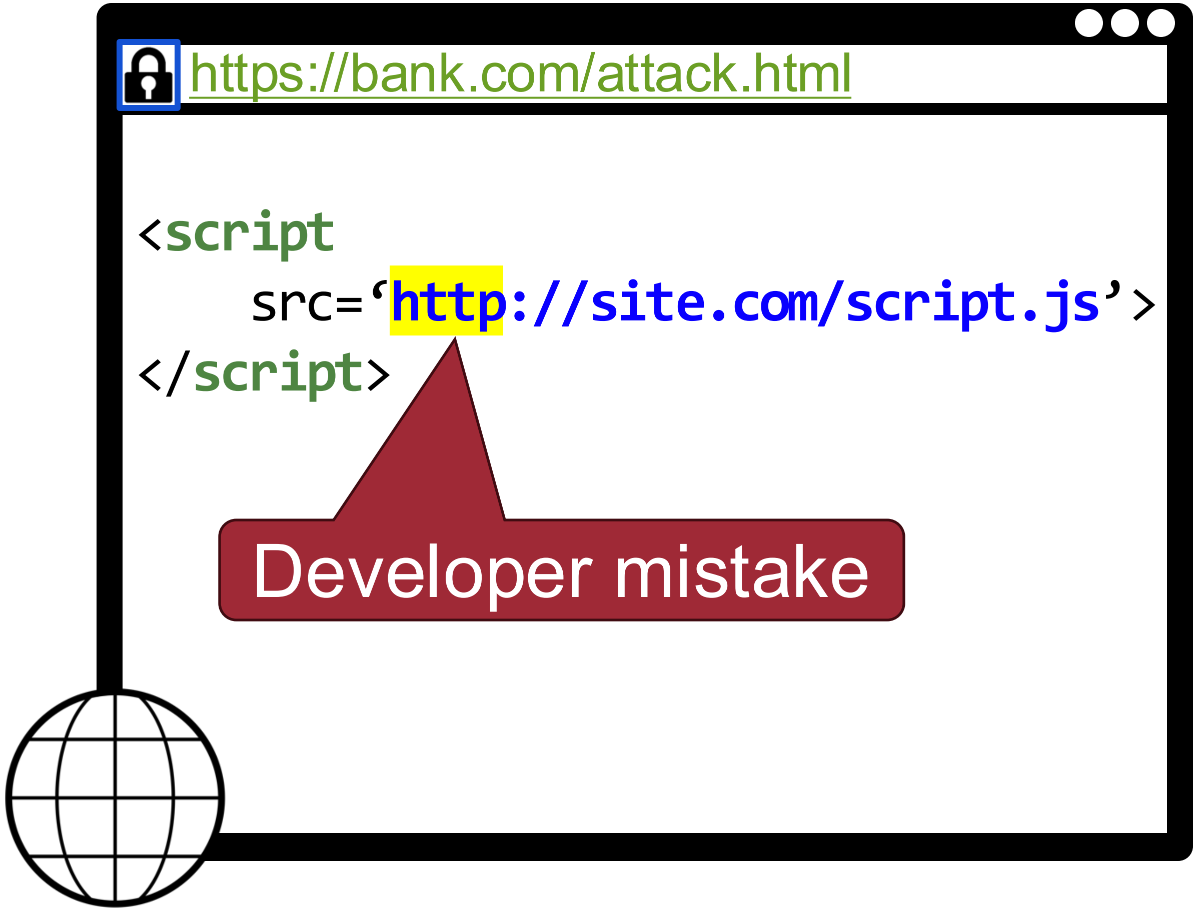
Mixed Content and Network Attacks



https://bank.com/attack.html

```
<script
    src='http://site.com/script.js'>
</script>
```

Mixed Content and Network Attacks

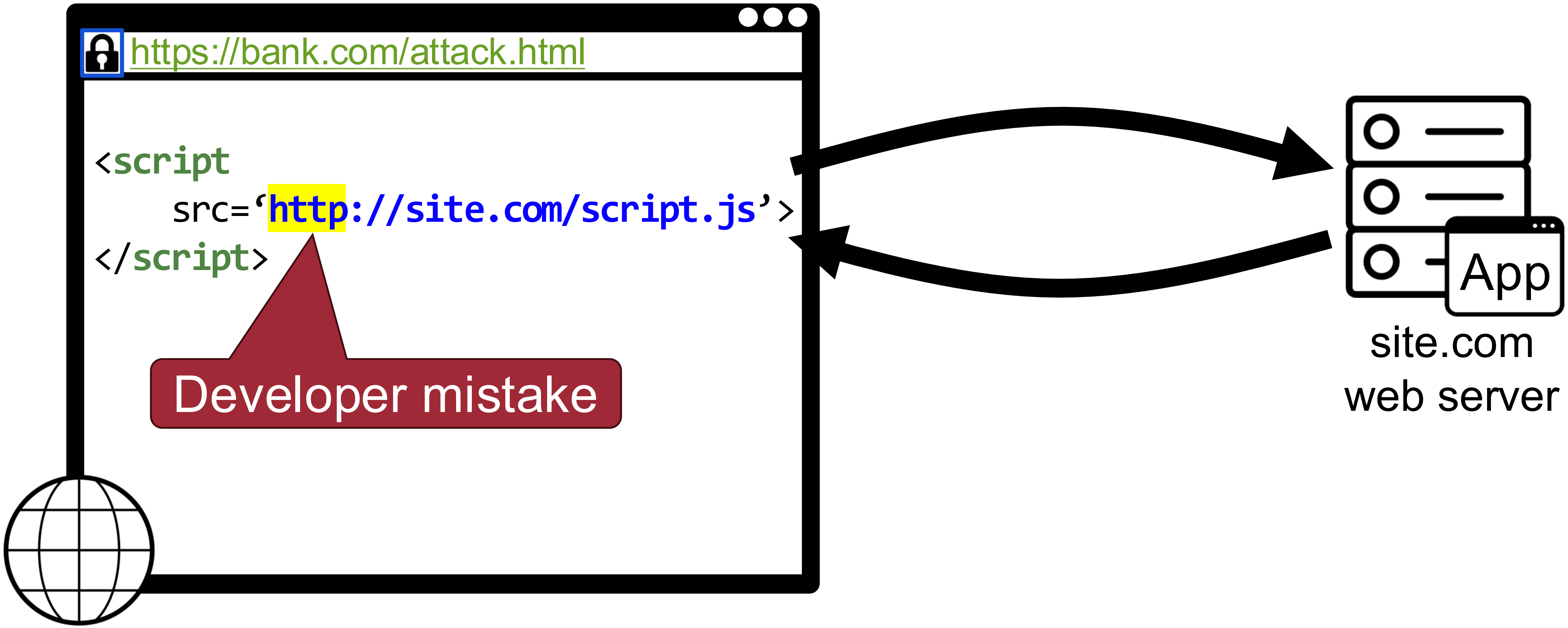


https://bank.com/attack.html

```
<script
    src=‘http://site.com/script.js’>
</script>
```

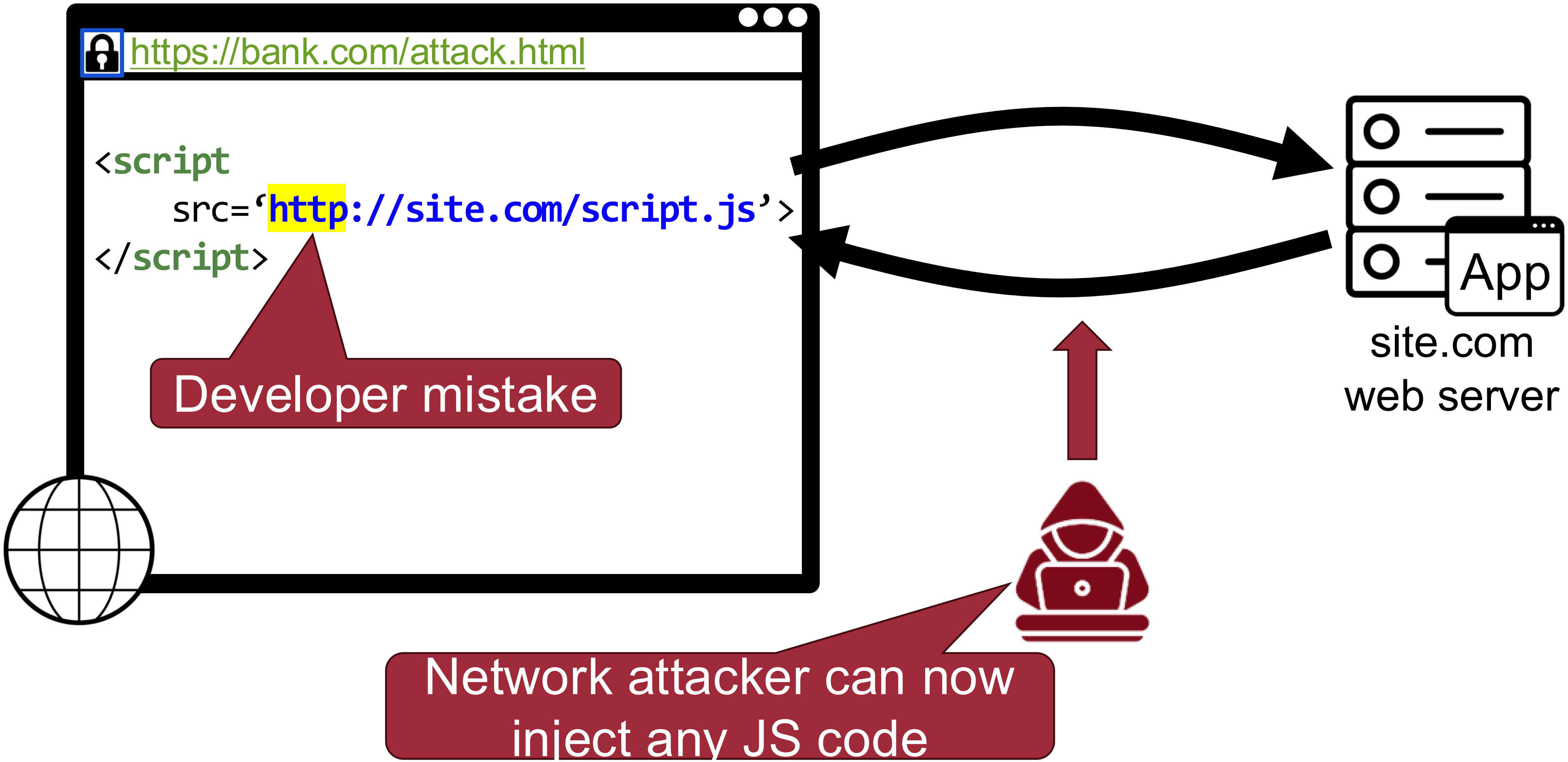
Developer mistake

Mixed Content and Network Attacks



```
<script
      src='http://site.com/script.js'>
</script>
```

Mixed Content and Network Attacks



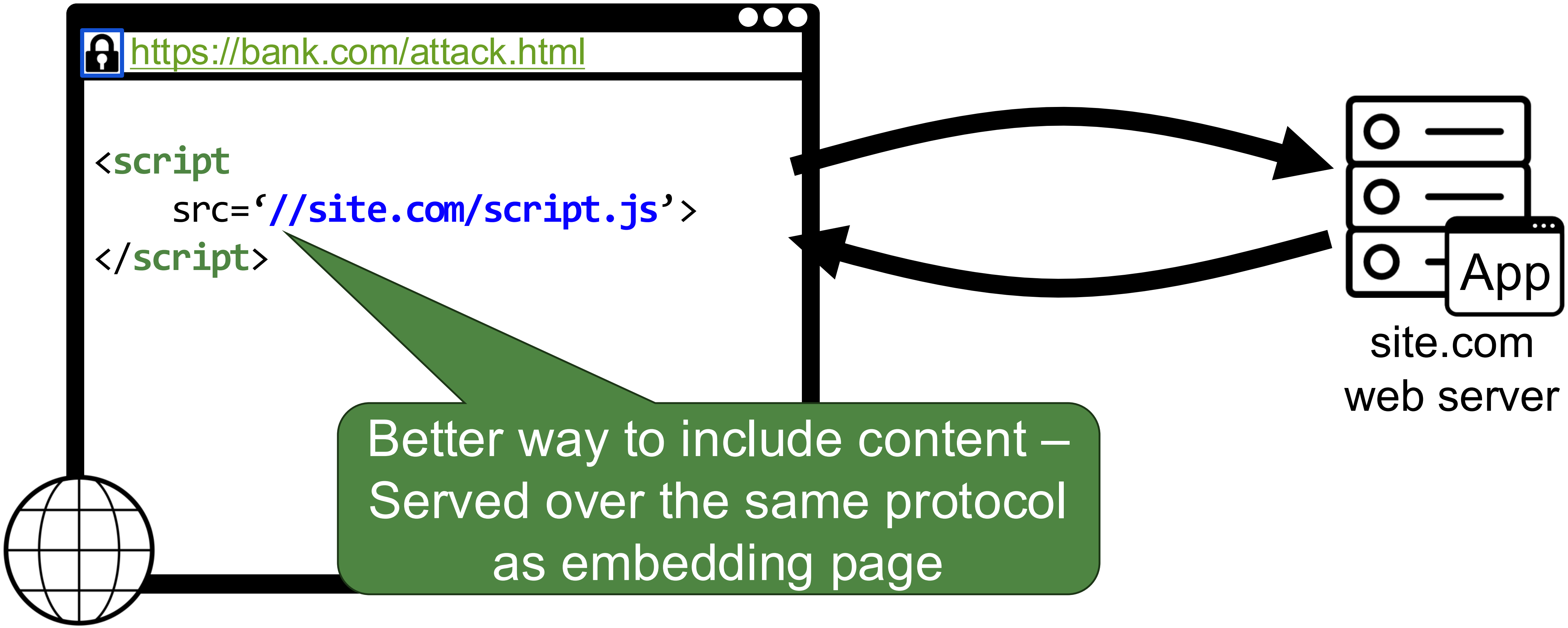
https://bank.com/attack.html

```
<script
    src='http://site.com/script.js'>
</script>
```

Developer mistake

Network attacker can now inject any JS code

Mixed Content and Network Attacks



https://bank.com/attack.html

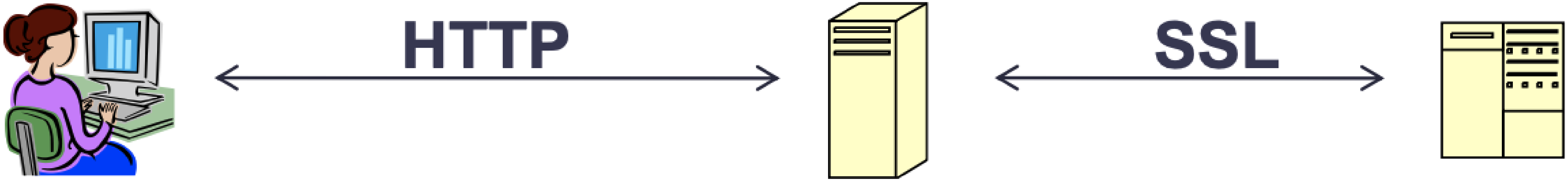
```
<script
    src='//site.com/script.js'>
</script>
```

site.com
web server

Better way to include content –
Served over the same protocol
as embedding page

HTTPS – Upgrade

- Come to site over HTTP (Port no. 80), redirect to HTTPS (Port no. 443)!



Apache configuration

```
<VirtualHost *:80>
    ServerName [Domain]
    Redirect permanent / https://[Domain]/
</VirtualHost>
```

Forcing HTTPs: HTTP Strict Transport Security

- HTTP header (`Strict-Transport-Security`) send by server
  - Only valid if sent via HTTPS
  - `Strict-Transport-Security: max-age=<expiry in seconds>`
    - `includeSubDomains`: header is valid for all subdomains
    - `preload`: allows for inclusion in preload list
  - Ensures that site cannot be loaded via HTTP until expiry is reached

Certificate Revocation

Certificate

- Revocation is very important

- Many valid reasons to revoke a certificate
  - Private key corresponding to the certified public key has been compromised
  - User stopped paying his certification fee to the CA and the CA no longer wishes to certify him
  - CA’s certificate has been compromised!

Revoking certificates with CRLs

- Certificate Revocation Lists (CRLs)
  - frequently updated by CAs
  - contains list of all certificates which have been revoked
    - e.g., because of compromised keys
  - downloaded by browsers in regular intervals

- Several issues
  - interval of updates by CAs
  - interval of updates by browsers

- CRLs are (being) deprecated from browsers!

Replacement Technologies for CRLs

- OCSP (Online Certificate Status Protocol): Real-time status checking for individual certificates
- OCSP Stapling
- Browser-driven Revocation
  - Chrome: CRLSet
  - Mozilla Firefox: OneCRL

Summary

- SSL/TLS protocol
  - Satisfy confidentiality
  - Satisfy integrity
  - Satisfy authentication
- HTTPS: HTTP + SSL/TLS protocol
- Certificate revocation

Question?