CSE467: Computer Security

21. Network Security: SSL/TLS & HTTPS

Seongil Wi

Department of Computer Science and Engineering

Notification: Hack Class101

- Find unknown security issues on Class101 websites!

- Instruction: https://bounty.class101.net/
  - Foreigners should use a translator

- Activity period: 03/03 ~ 06/18

- DO NOT try anything illegal!

Notification: Homework #3

- Hacking practice: Capture the Flag (CTF)
- Software/system hacking competition

- Challenge open (competition start): 5/28 (Wed)
- Due date (writeup report): 6/11 (Wed)

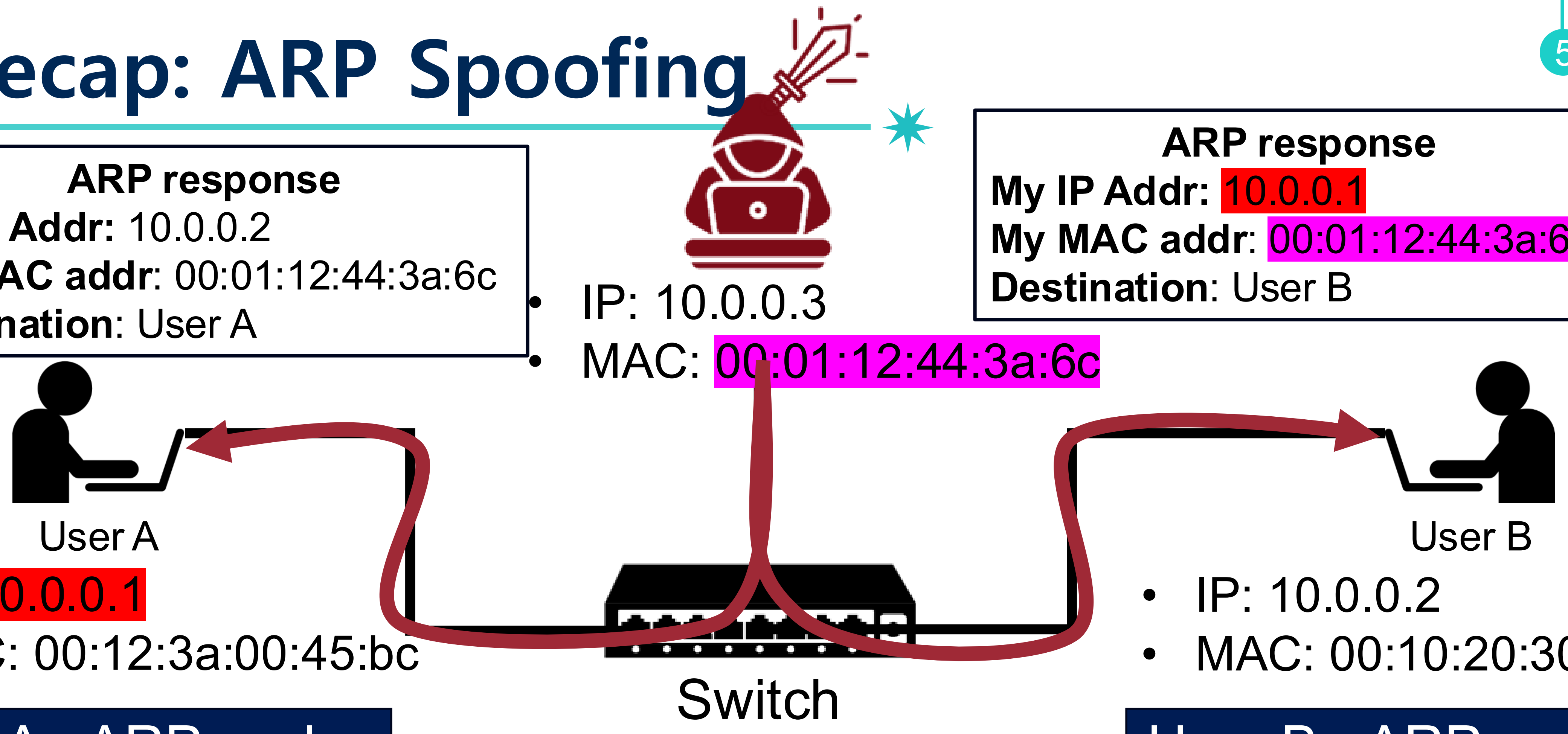
Notification: Quiz #2

- Date: 6/4 (Wed.), Class time

- Scope
  - Everything learned in Network Security, including today’s material

- T/F problems
- Computation problems
- Bring your own pen!

Recap: ARP Spoofing



ARP response
My IP Addr: 10.0.0.2
My MAC addr: 00:01:12:44:3a:6c
Destination: User A

- IP: 10.0.0.3
- MAC: 00:01:12:44:3a:6c

ARP response
My IP Addr: 10.0.0.1
My MAC addr: 00:01:12:44:3a:6c
Destination: User B

User A

- IP: 10.0.0.1
- MAC: 00:12:3a:00:45:bc

Switch

User B

- IP: 10.0.0.2
- MAC: 00:10:20:30:ac:06

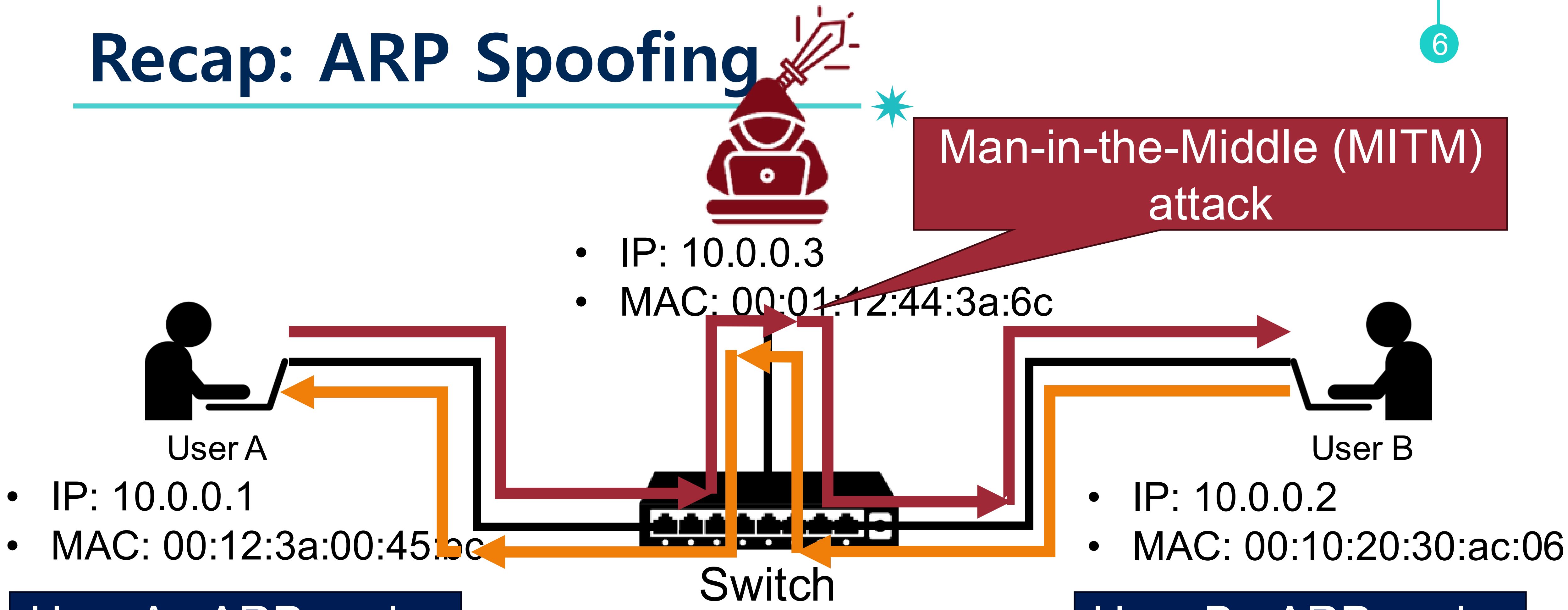
User A - ARP cache

IP Addr	Mac Addr
10.0.0.2	~~00:10:20:30:ac:06~~ 00:01:12:44:3a:6c

User B - ARP cache

IP Addr	Mac Addr
10.0.0.1	~~00:12:3a:00:45:bc~~ 00:01:12:44:3a:6c

Recap: ARP Spoofing



Man-in-the-Middle (MITM) attack

- IP: 10.0.0.3
- MAC: 00:01:12:44:3a:6c

User A

- IP: 10.0.0.1
- MAC: 00:12:3a:00:45:bc

Switch

User B

- IP: 10.0.0.2
- MAC: 00:10:20:30:ac:06

User A - ARP cache

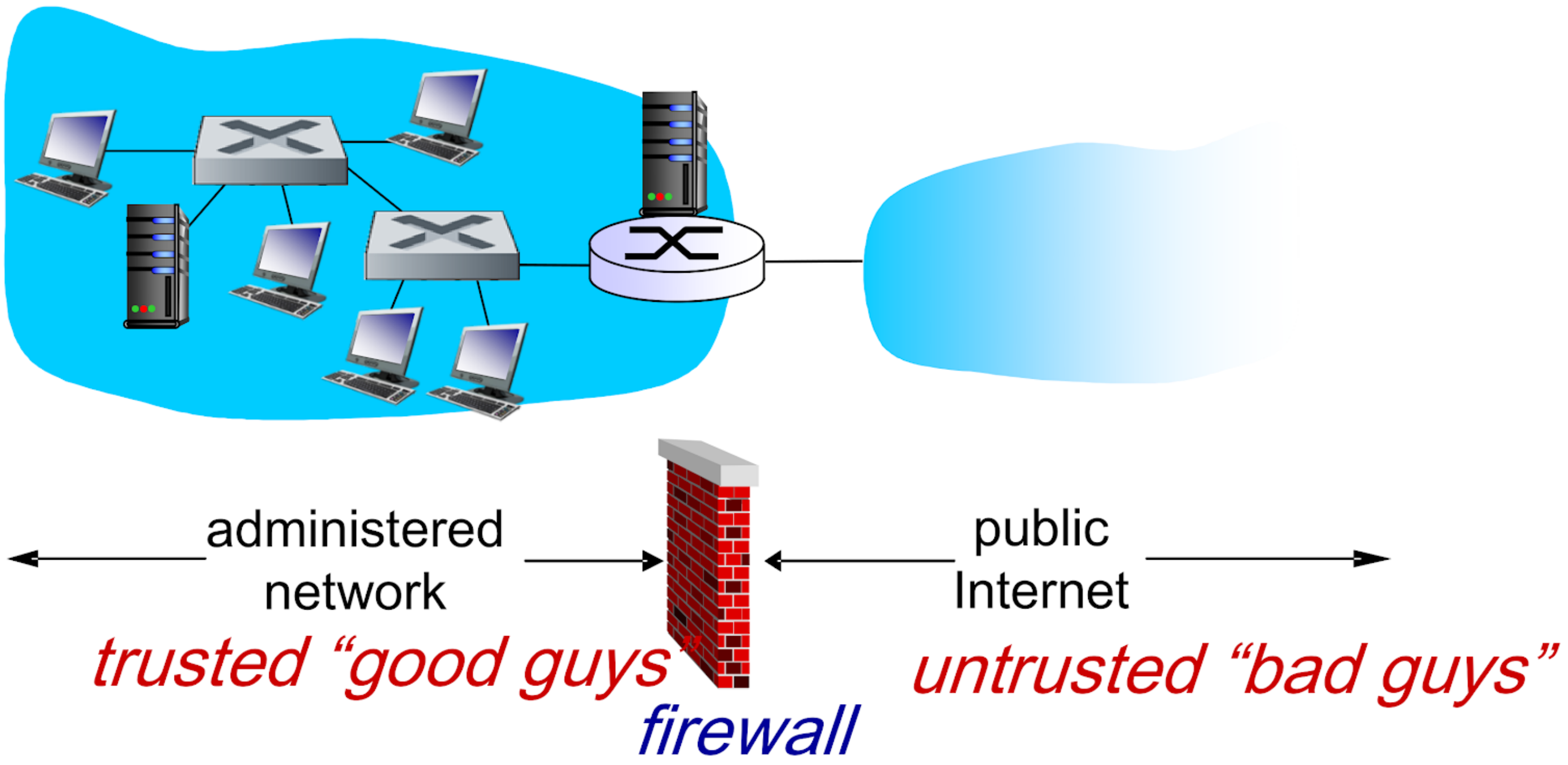
IP Addr	Mac Addr
10.0.0.2	~~00:10:20:30:ac:06~~ 00:01:12:44:3a:6c

User B - ARP cache

IP Addr	Mac Addr
10.0.0.1	~~00:12:3a:00:45:bc~~ 00:01:12:44:3a:6c

Recap: Firewalls

- Isolate organization's internal net from larger Internet, allowing some packets to pass, blocking others



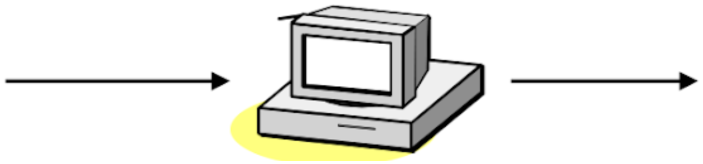
Recap: Intrusion Detection

- Intrusion
  - A set of actions aimed to compromise the security goals
- Intrusion detection
  - The process of identifying and responding to intrusion activities

Recap: Firewall vs. IDS

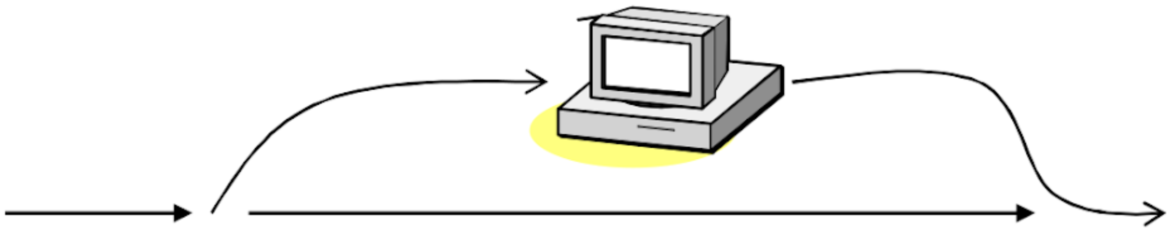
- Firewall
  - Active filtering (prevent intrusion)
  - Location: Between networks (if an attack is from inside the network it doesn’t signal)

- IDS
  - Passive monitoring (detect intrusion)
  - Location: Inside the network



Firewall

NIDS

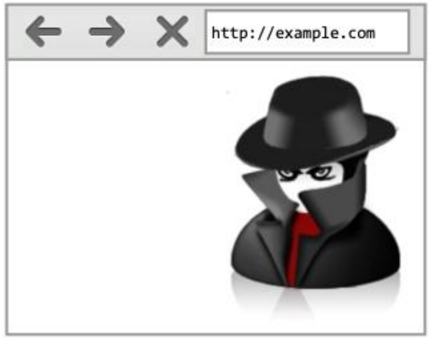


Recap: Threat Models

- **Network attacker**: resides somewhere in the communication link between client and server
  - Passive: evasdropping
  - Active: modification of messages, replay…

- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server

- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com

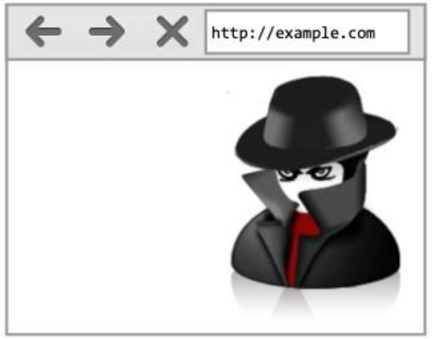


Today’s Topic

- **Network attacker**: resides somewhere in the communication link between client and server
  - Passive: evasdropping
  - Active: modification of messages, replay…

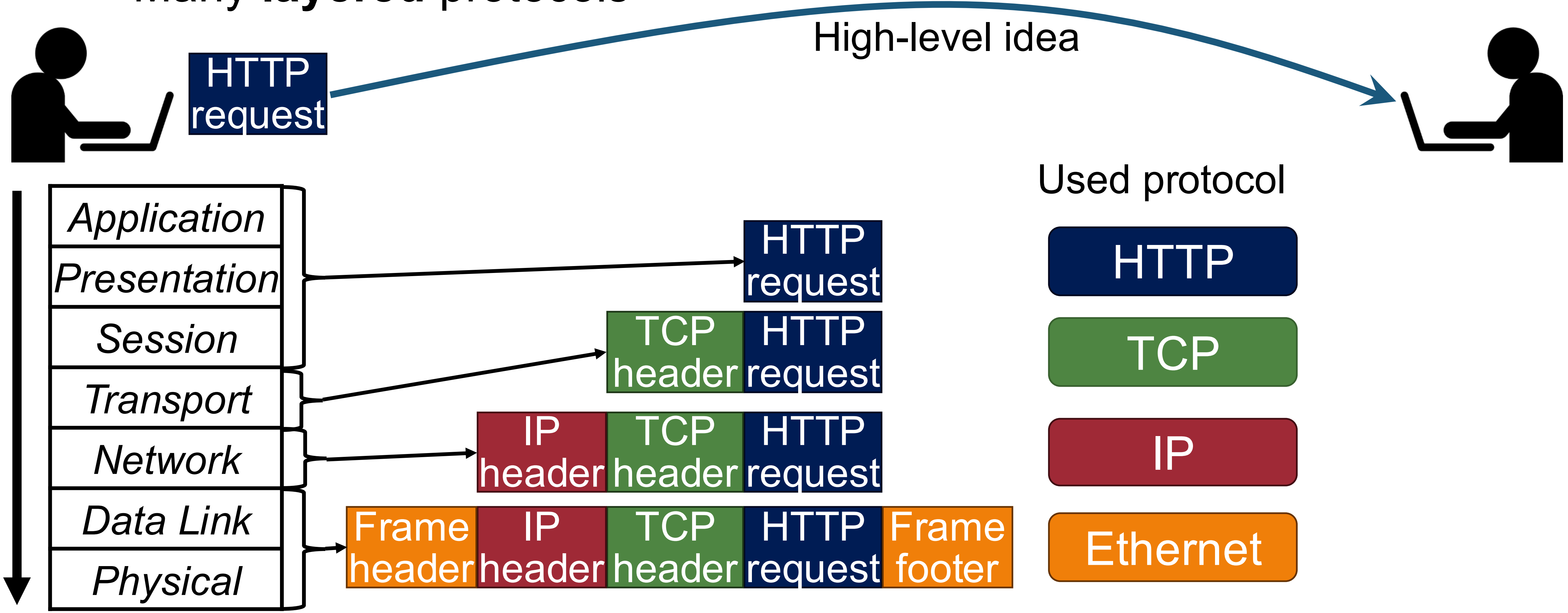
- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server

- **Web attacker**: controls attacker.com
  - Can obtain SSL/TLS certificates for attacker.com
  - Users can visit attacker.com



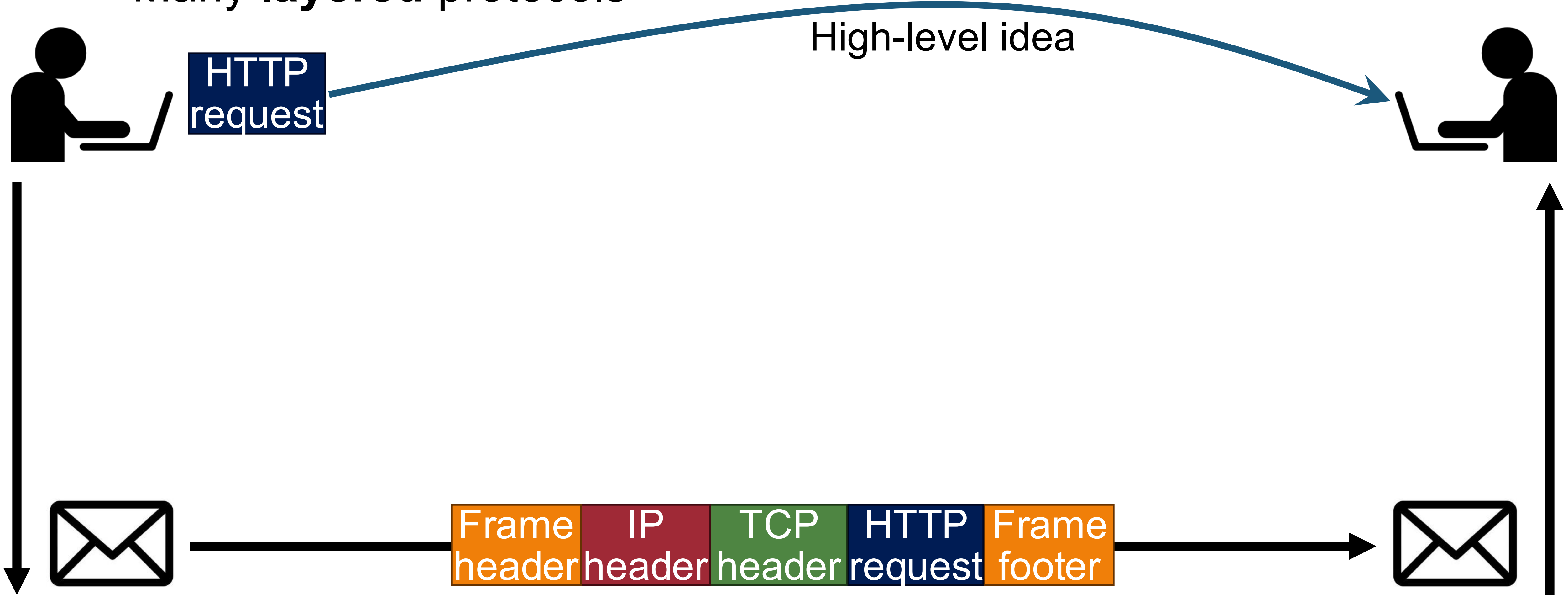
Recap: Protocol

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



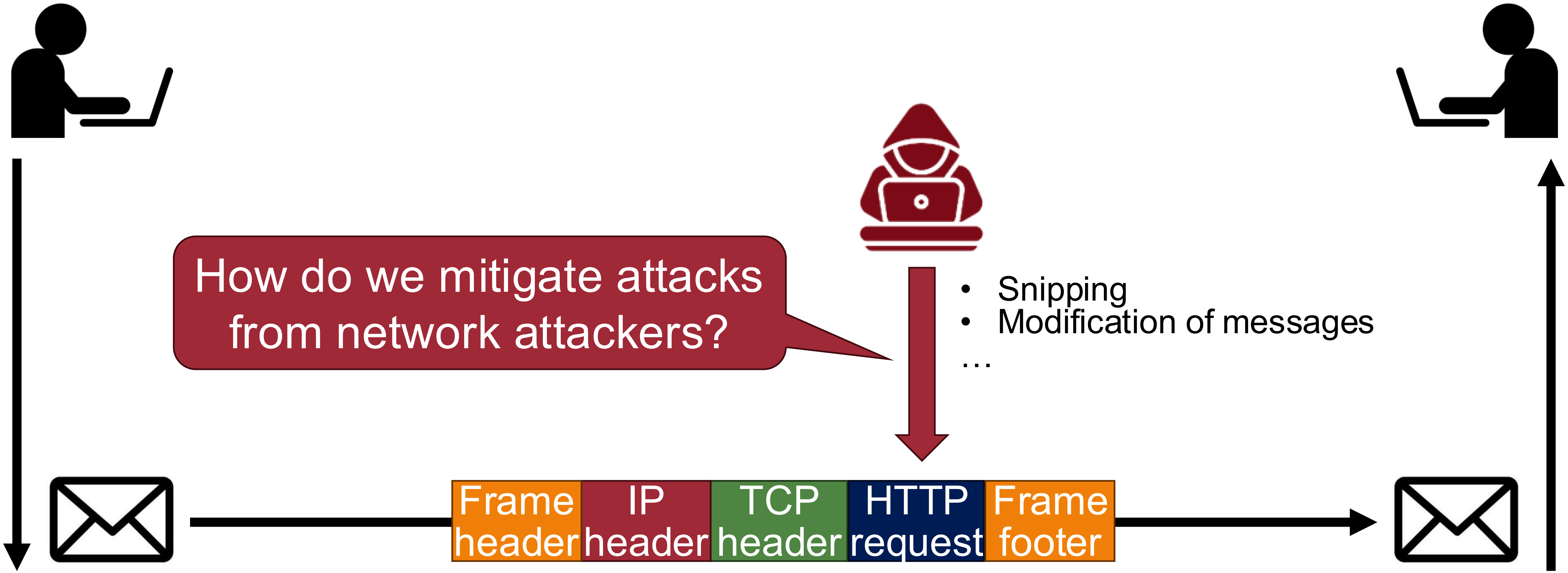
Recap: Protocol

- 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?

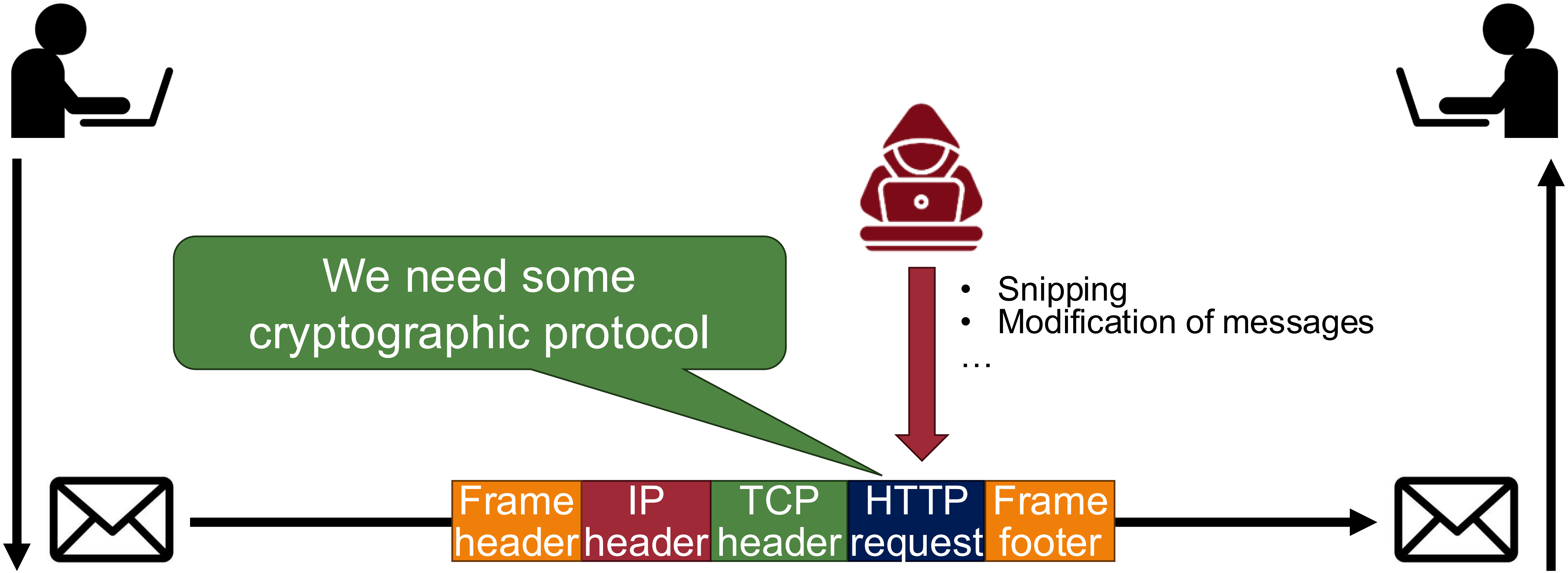
- Snipping
- Modification of messages

…

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

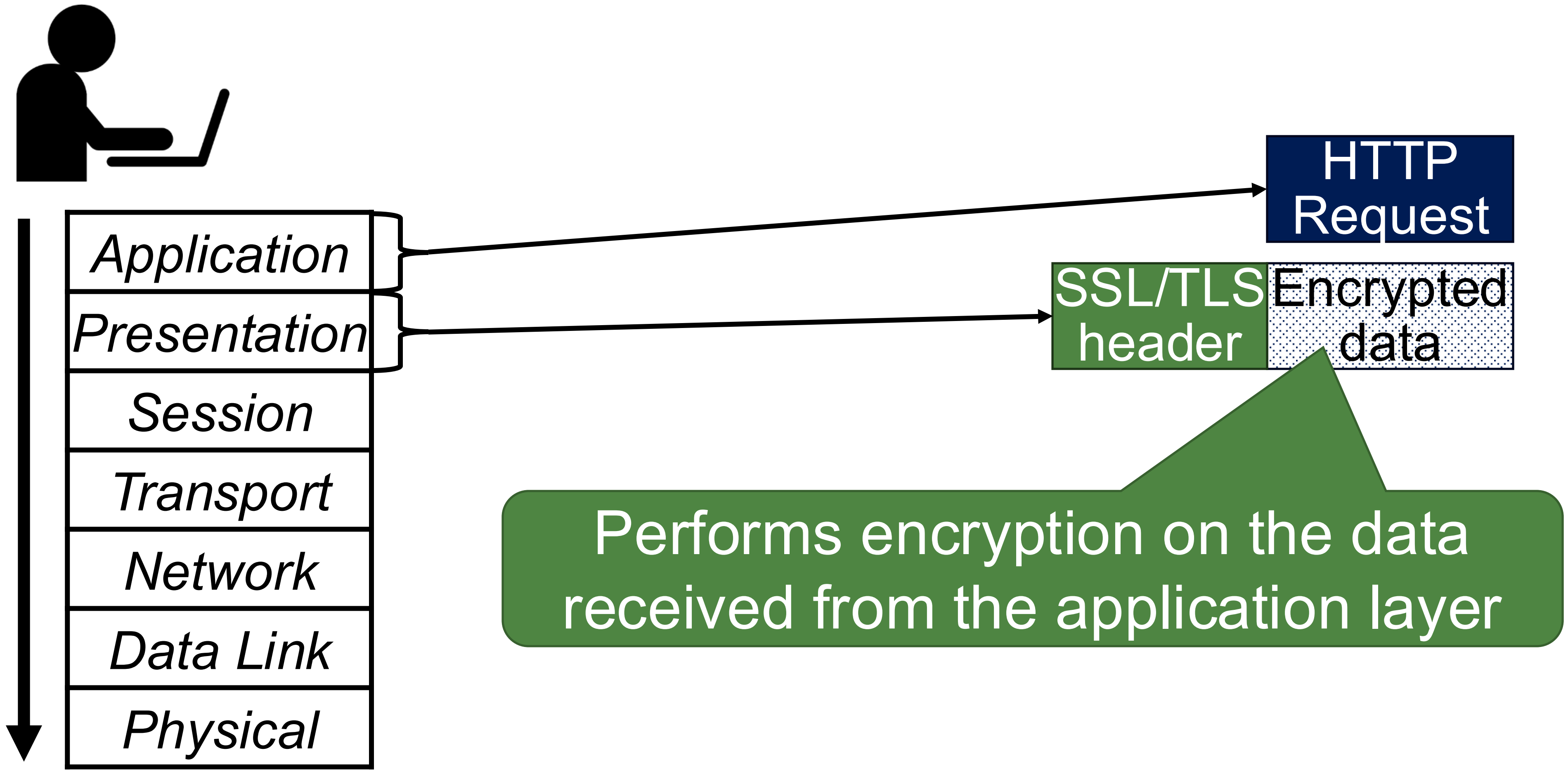
Related to cryptography, network security, web security, and software security!

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!

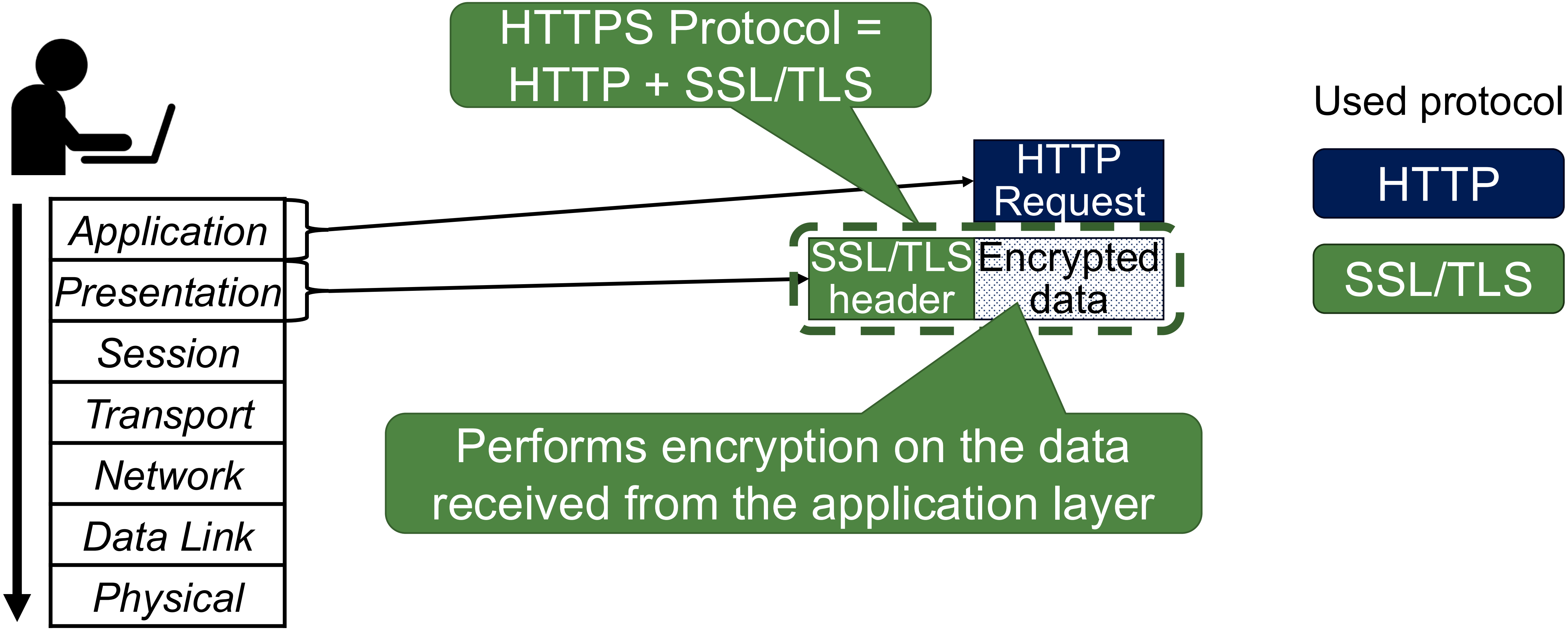


Performs encryption on the data received from the application layer

Used protocol

SSL/TLS Basic Idea

- Adding a protocol layer for secure communication!



HTTPS Protocol = HTTP + SSL/TLS

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

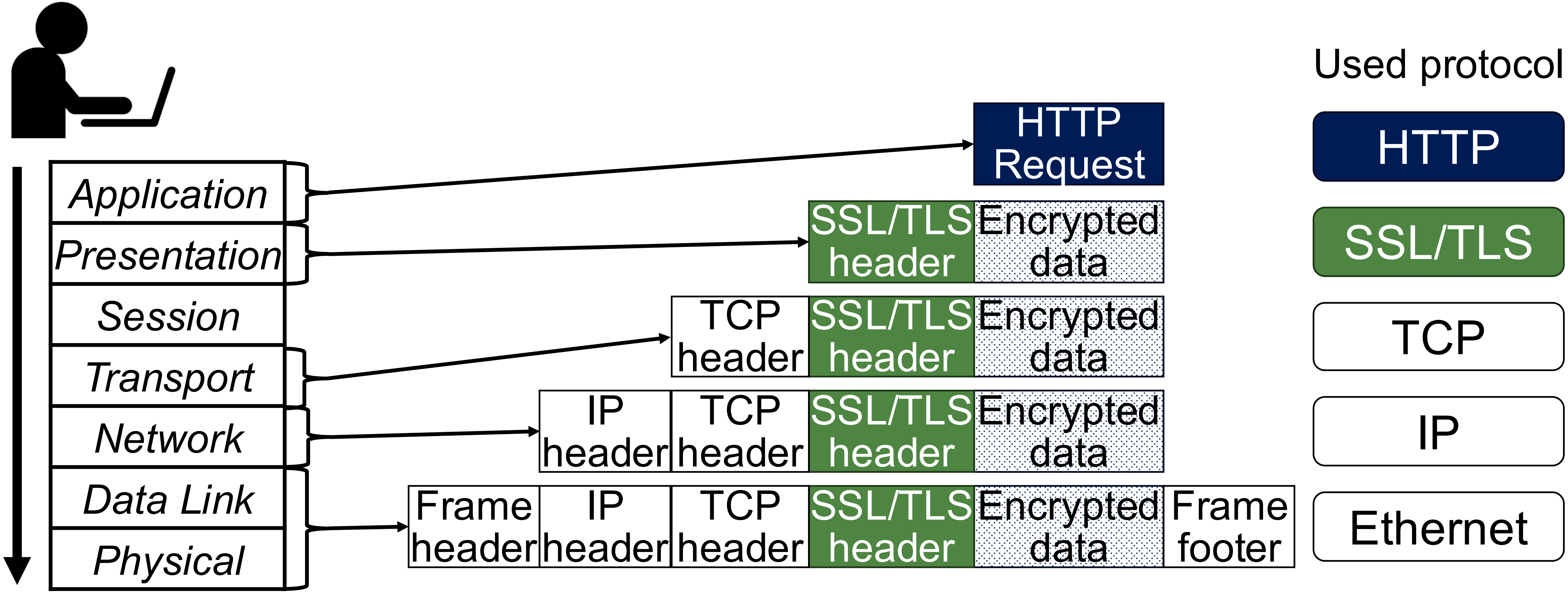
Used protocol

HTTP

SSL/TLS

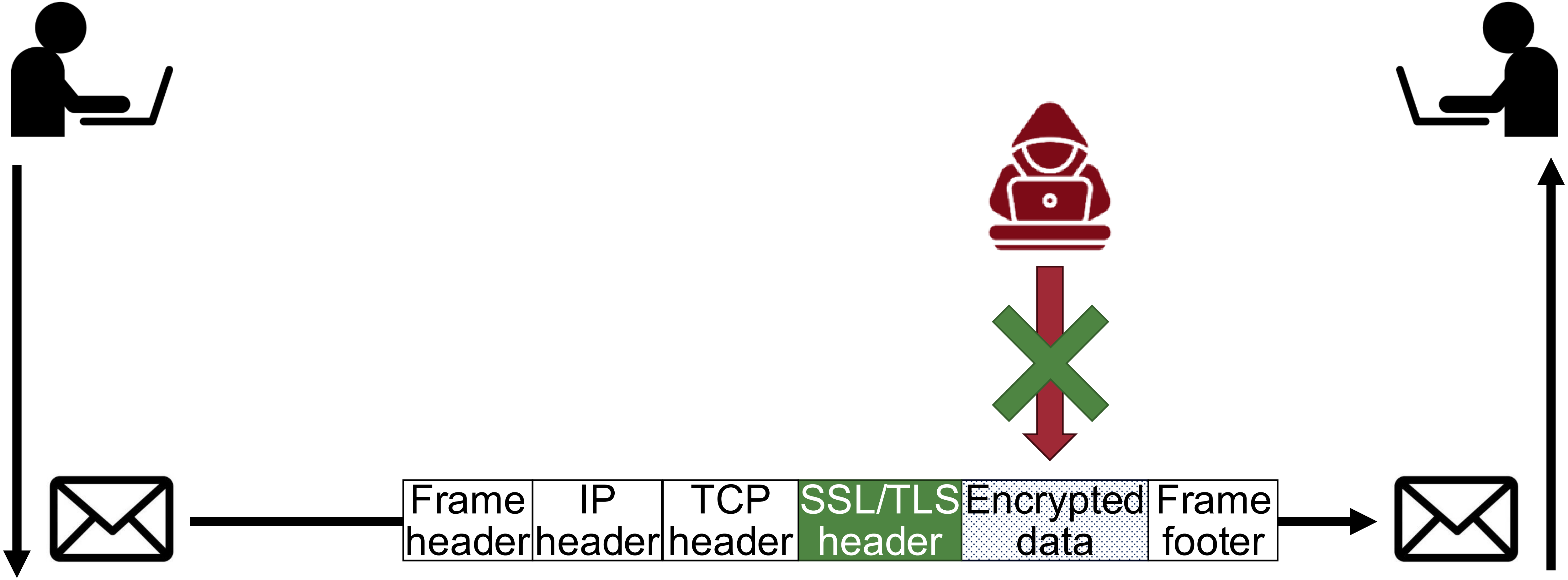
SSL/TLS Basic Idea

- Adding a protocol layer for secure communication!



SSL/TLS Basic Idea

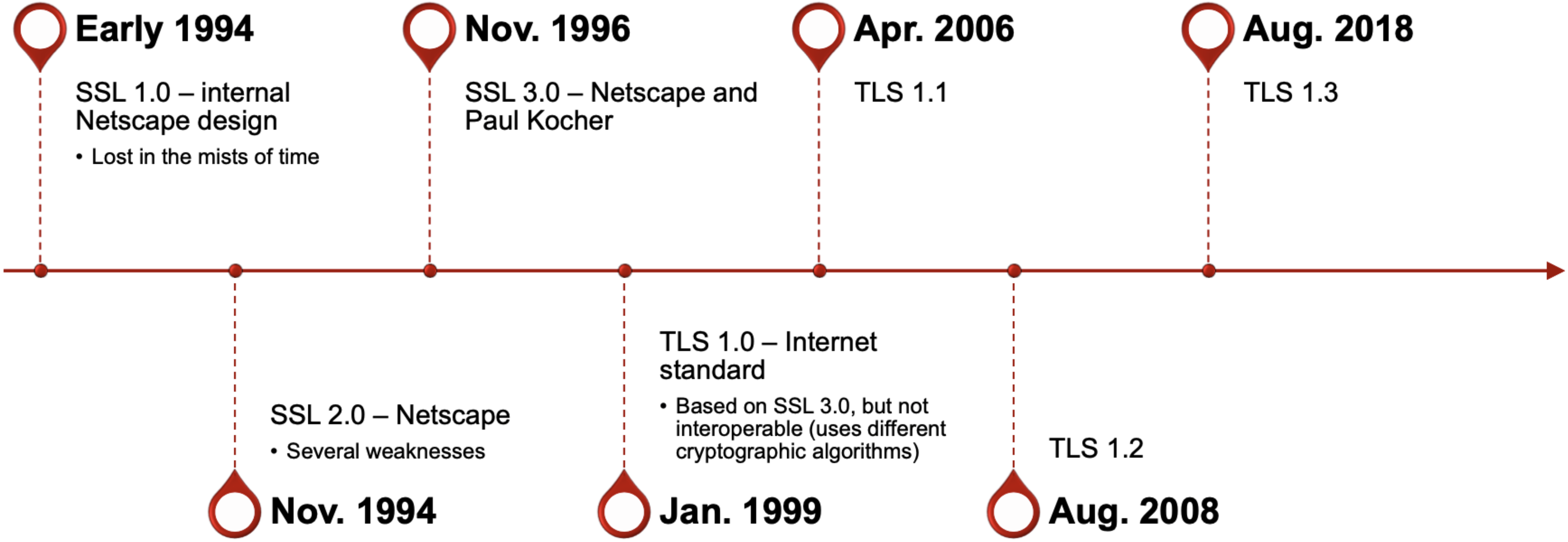
- Adding a protocol layer for secure communication!



SSL/TLS Use Cases

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

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

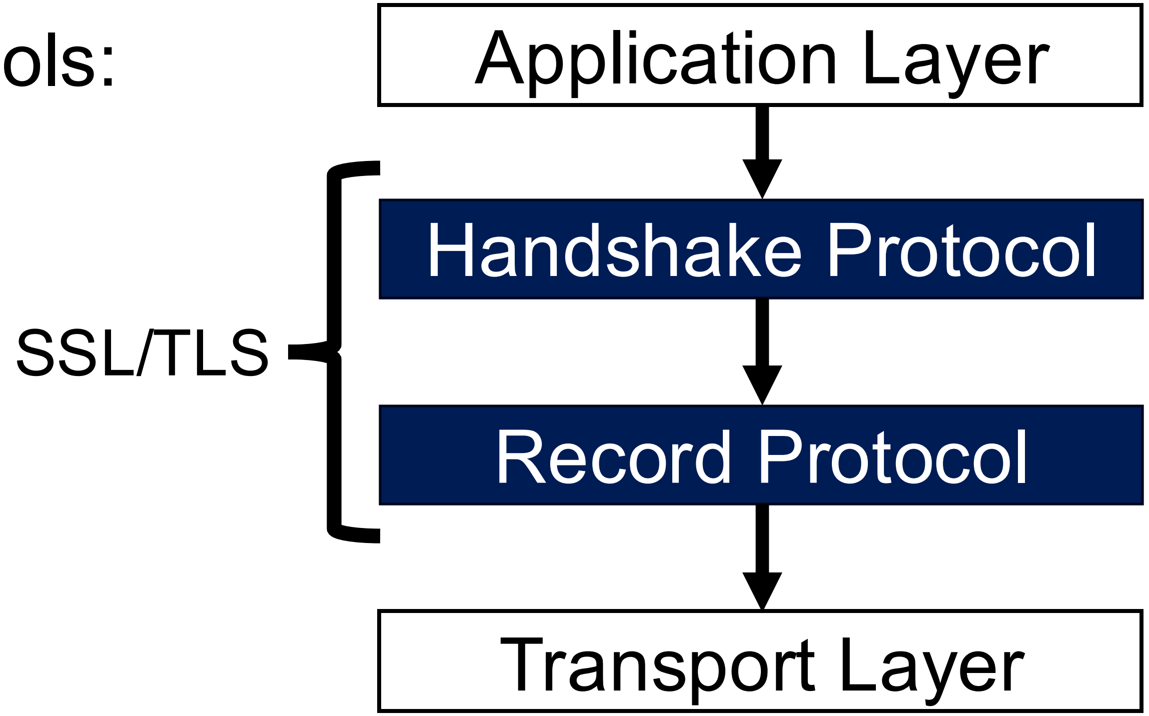
TLS 1.2

Aug. 2018

TLS 1.3

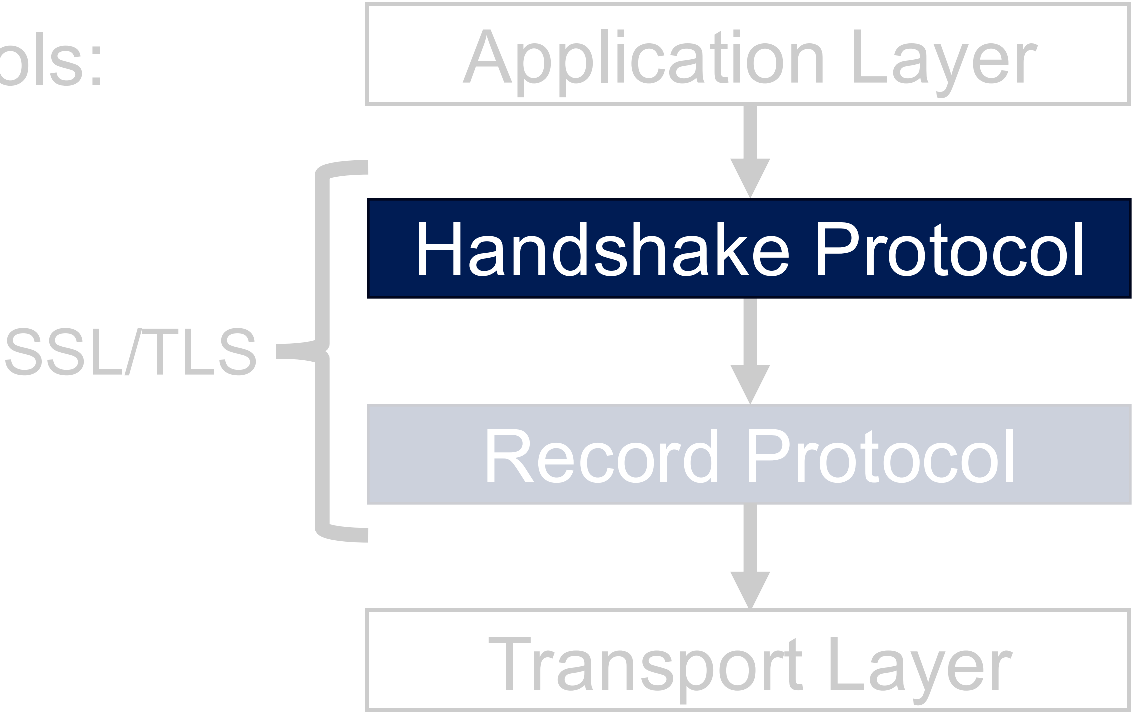
SSL/TLS Basics

- 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

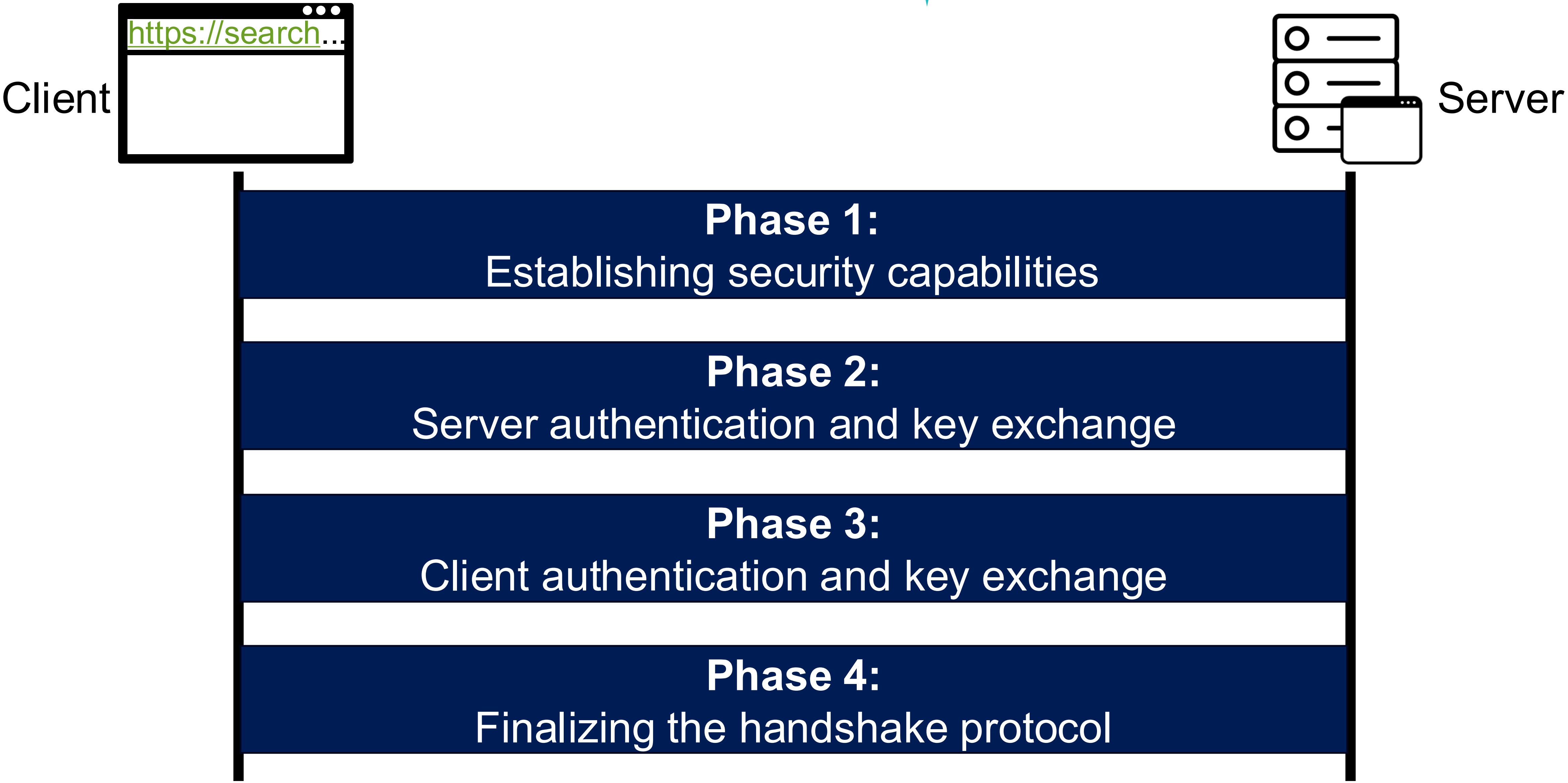
- Runs in the presentation layer
- Uses symmetric crypto, asymmetric crypto, and digital signatures
- Composed of two layers of protocols:
  1. Handshake protocol
  2. Record protocol



SSL/TLS Handshake Protocol

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

Four Phases of Handshake Protocol



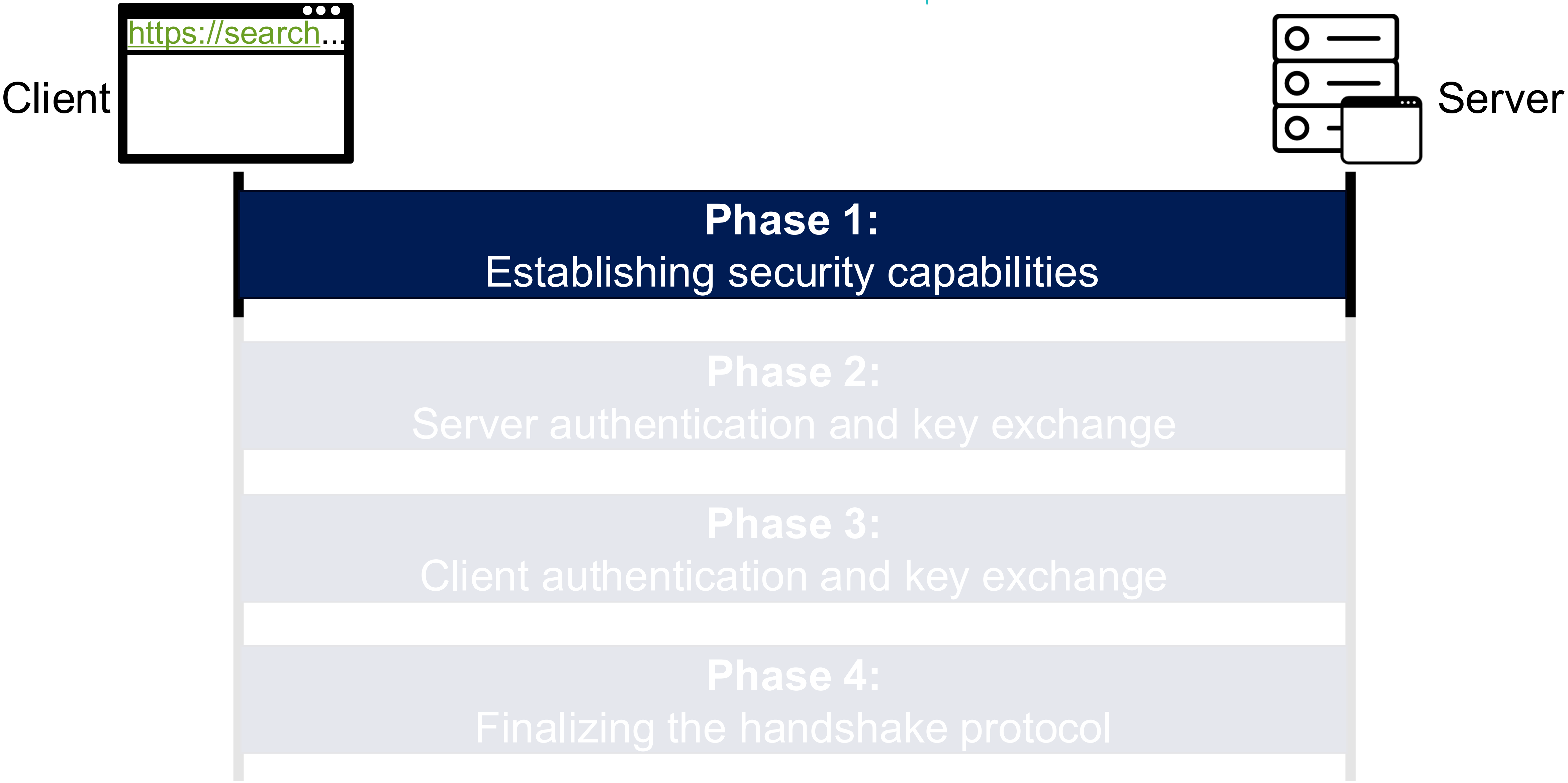
Phase 1:
Establishing security capabilities

Phase 2:
Server authentication and key exchange

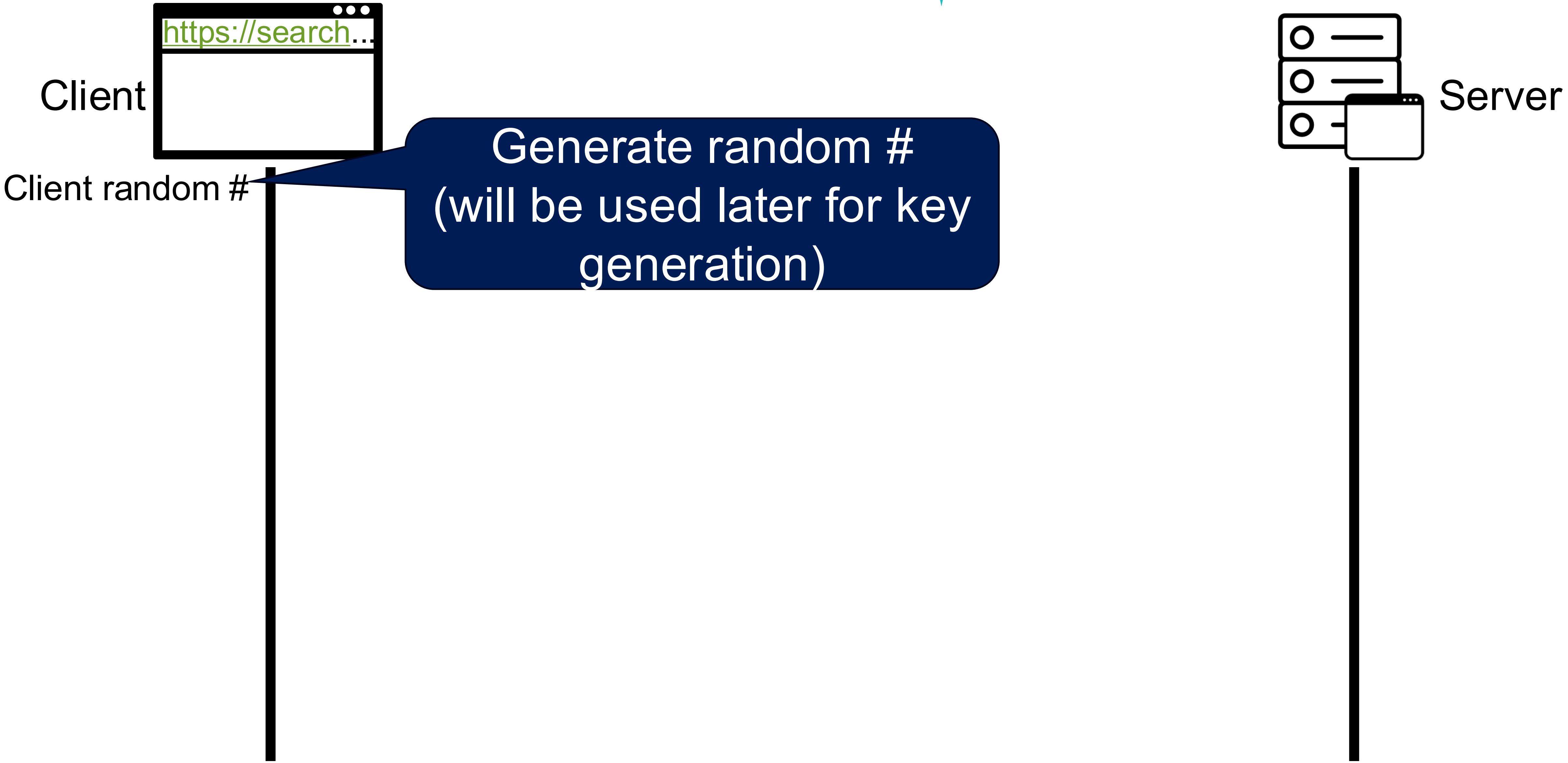
Phase 3:
Client authentication and key exchange

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



https://search...

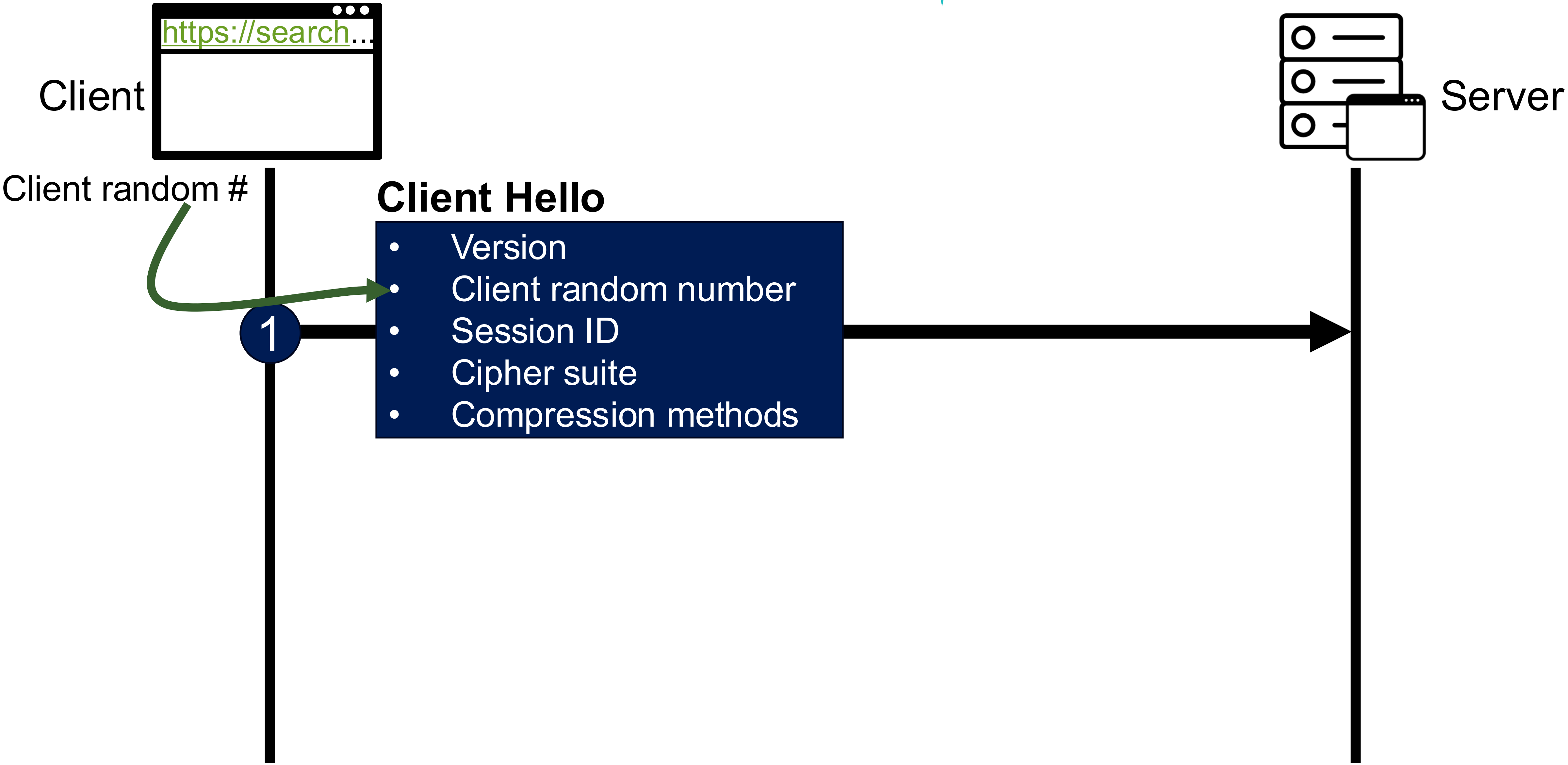
Client

Client random #

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

Server

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

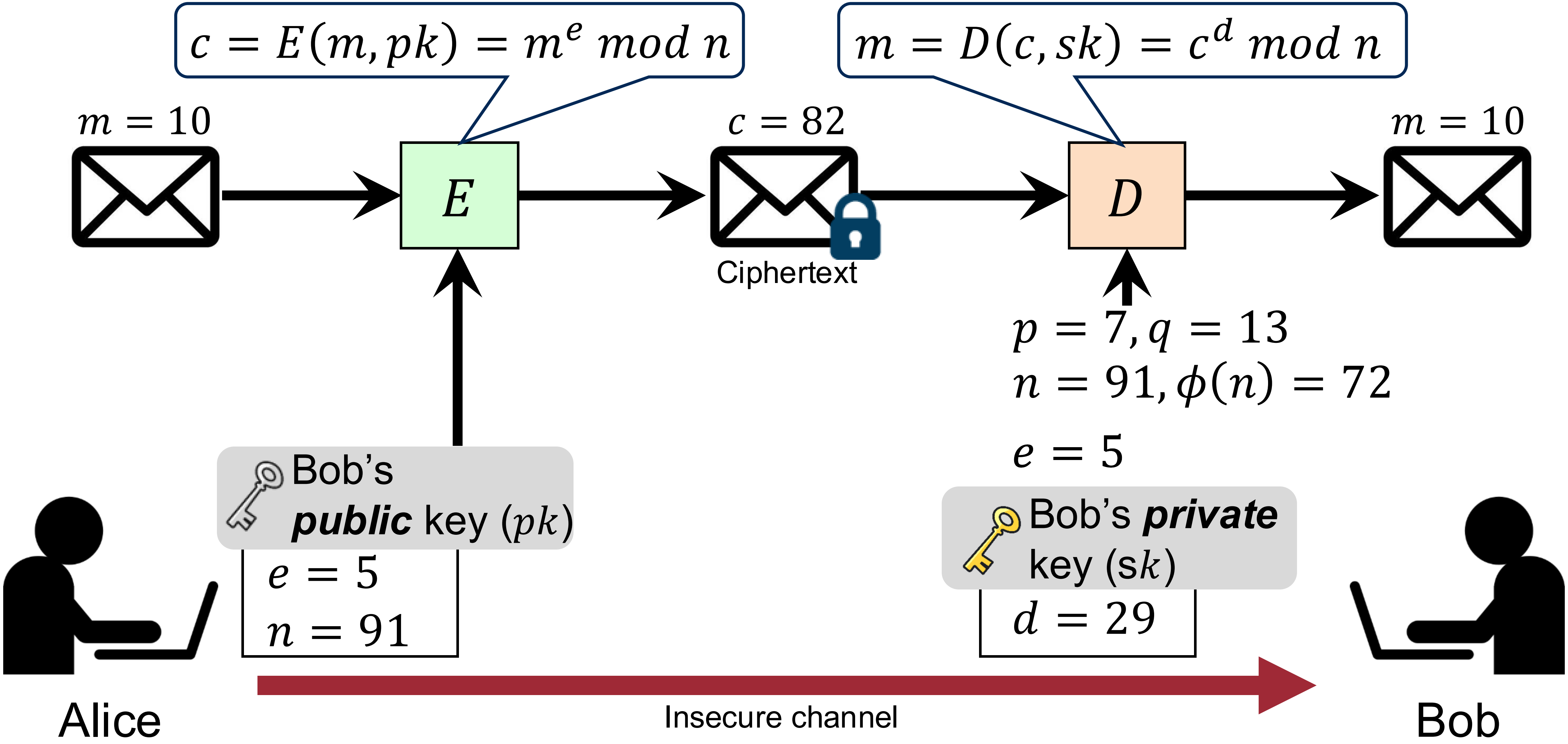
Format:

`TLS_RSA_WITH_AES_128_CBC_SHA`

Protocol

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

Recap: RSA Algorithm



$c = E(m, pk) = m^e \mod n$

$m = D(c, sk) = c^d \mod n$

Recap: Diffie-Hellman Key Exchange

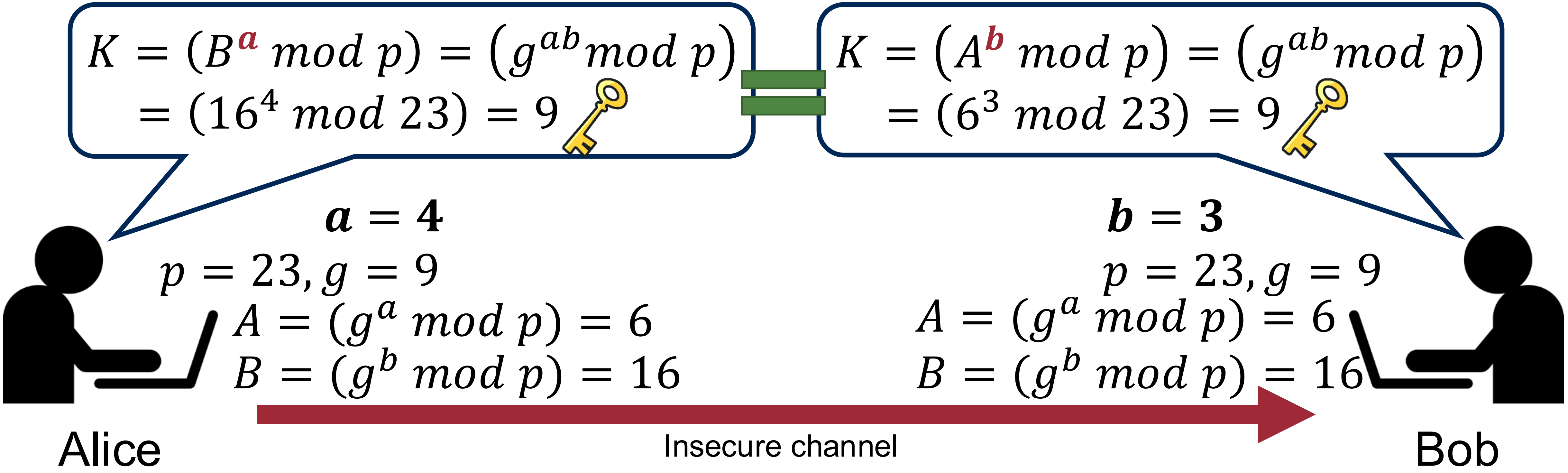
Symmetric key:

$$K = g^{ab} \ mod \ p$$

$p = 23, g = 9$

$A = (g^a \ mod \ p) = 6$

$B = (g^b \ mod \ p) = 16$



$K = (B^{a} \ mod \ p) = (g^{ab} mod \ p)$

$= (16^4 \ mod \ 23) = 9$

$K = (A^{b} \ mod \ p) = (g^{ab} mod \ p)$

$= (6^3 \ mod \ 23) = 9$

$\boldsymbol{a = 4}$

$p = 23, g = 9$

$A = (g^a \ mod \ p) = 6$

$B = (g^b \ mod \ p) = 16$

$\boldsymbol{b = 3}$

$p = 23, g = 9$

$A = (g^a \ mod \ p) = 6$

$B = (g^b \ mod \ p) = 16$

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:

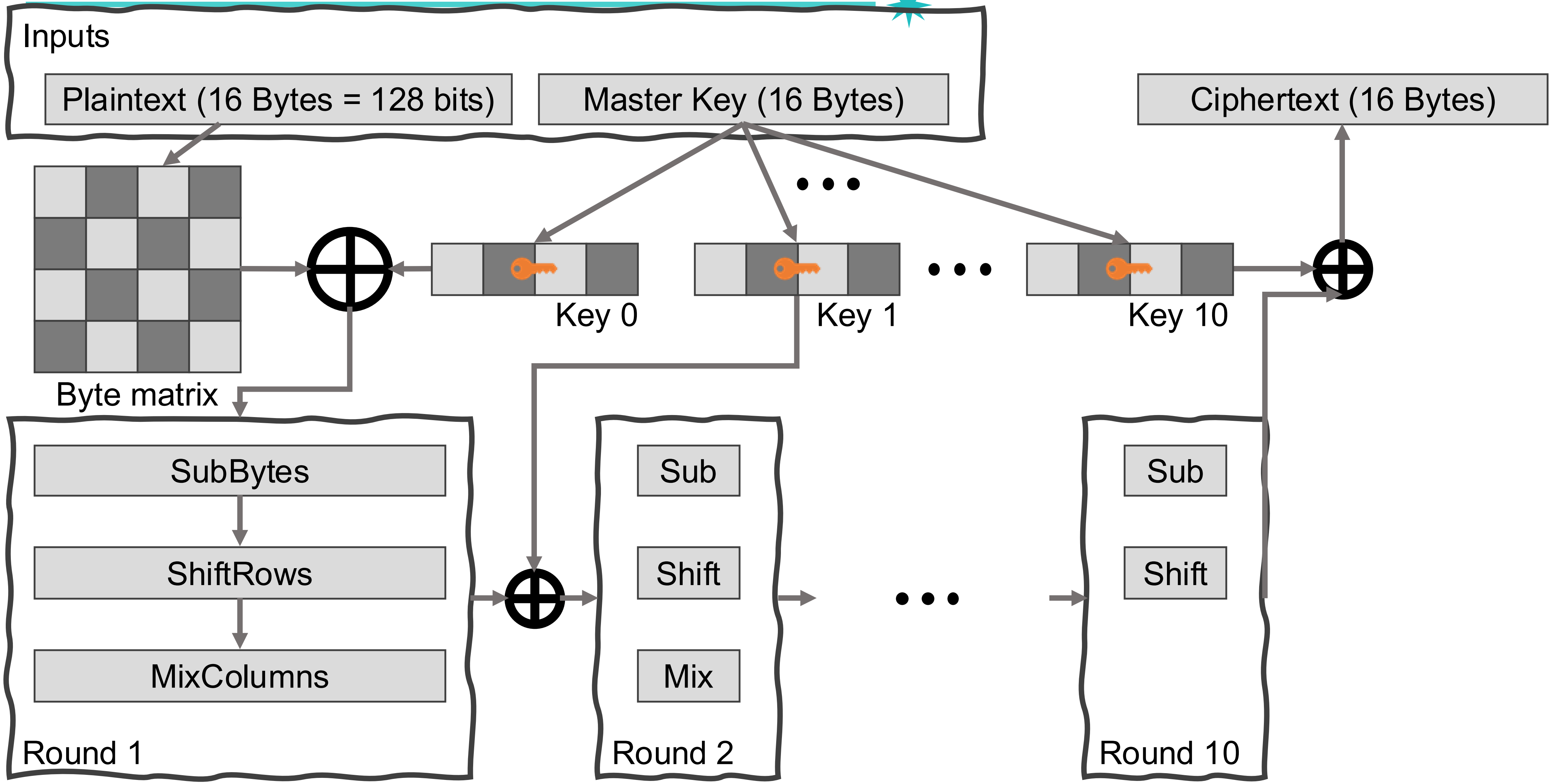
```
TLS_RSA_WITH_AES_128_CBC_SHA
```

Protocol

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

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

Recap: 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

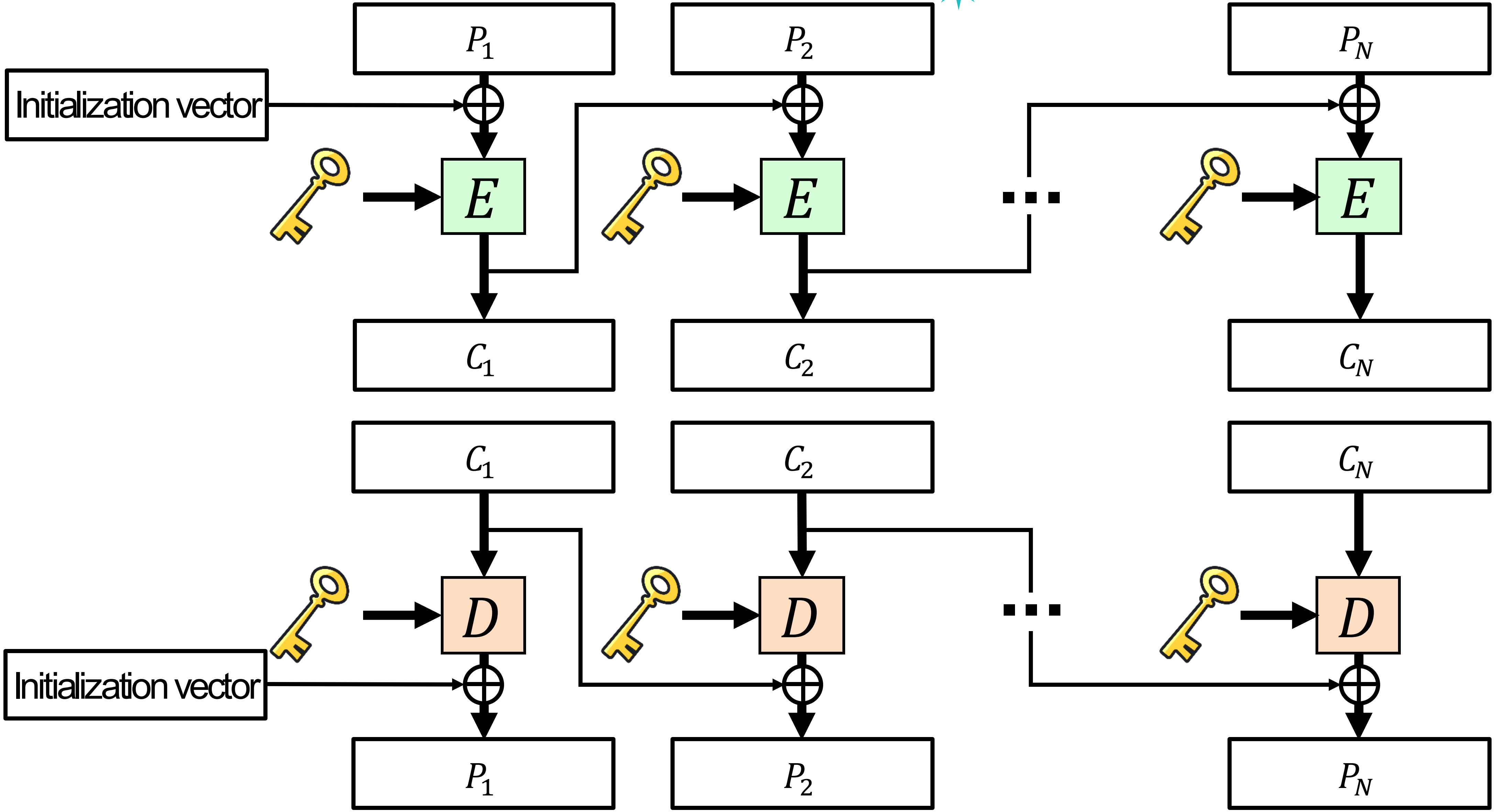
(Asymmetric) Encryption/decryption algorithm (for key exchange)

Block size

Modes of Operation

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

Recap: Cipher Block Chaining (CBC)



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)

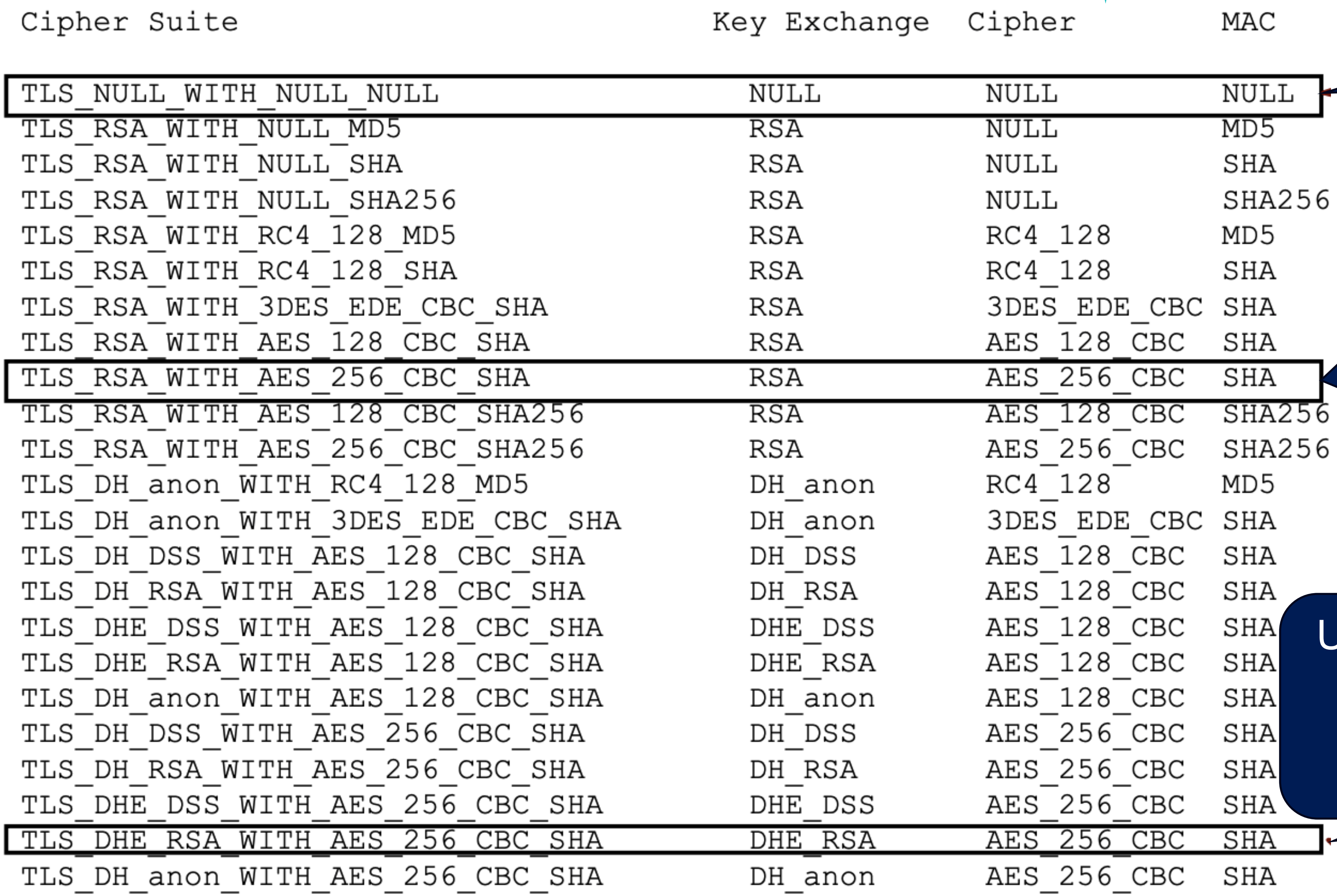
(Symmetric) Encryption/decryption algorithm (for data exchange)

Block size

Modes of Operation

MAC algorithm

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

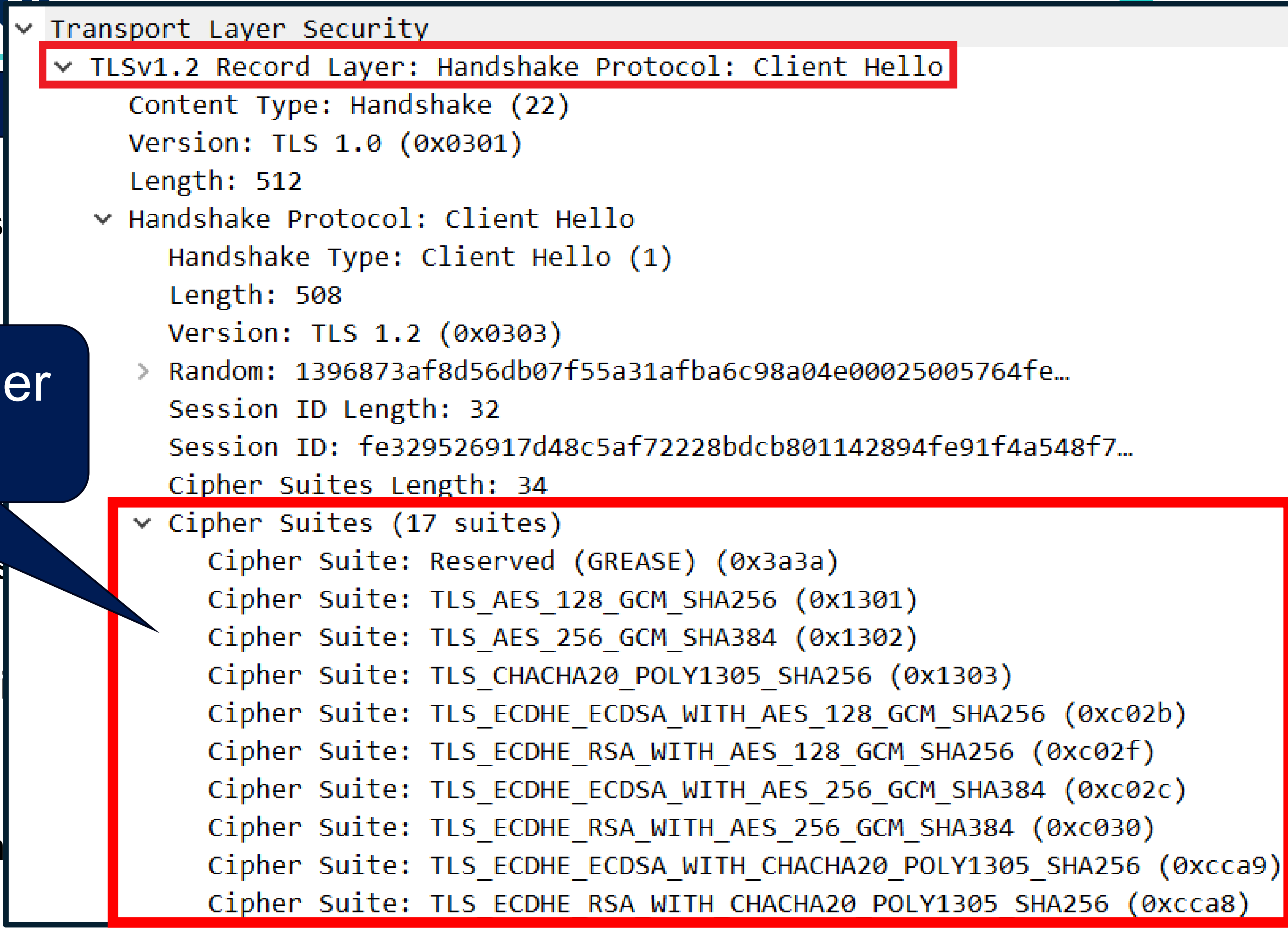
Uses ephemeral Diffie-Hellman with RSA for key exchange, AES 256 CBC for encryption and SHA256 as MAC

Cipher Suite

Client Hello –

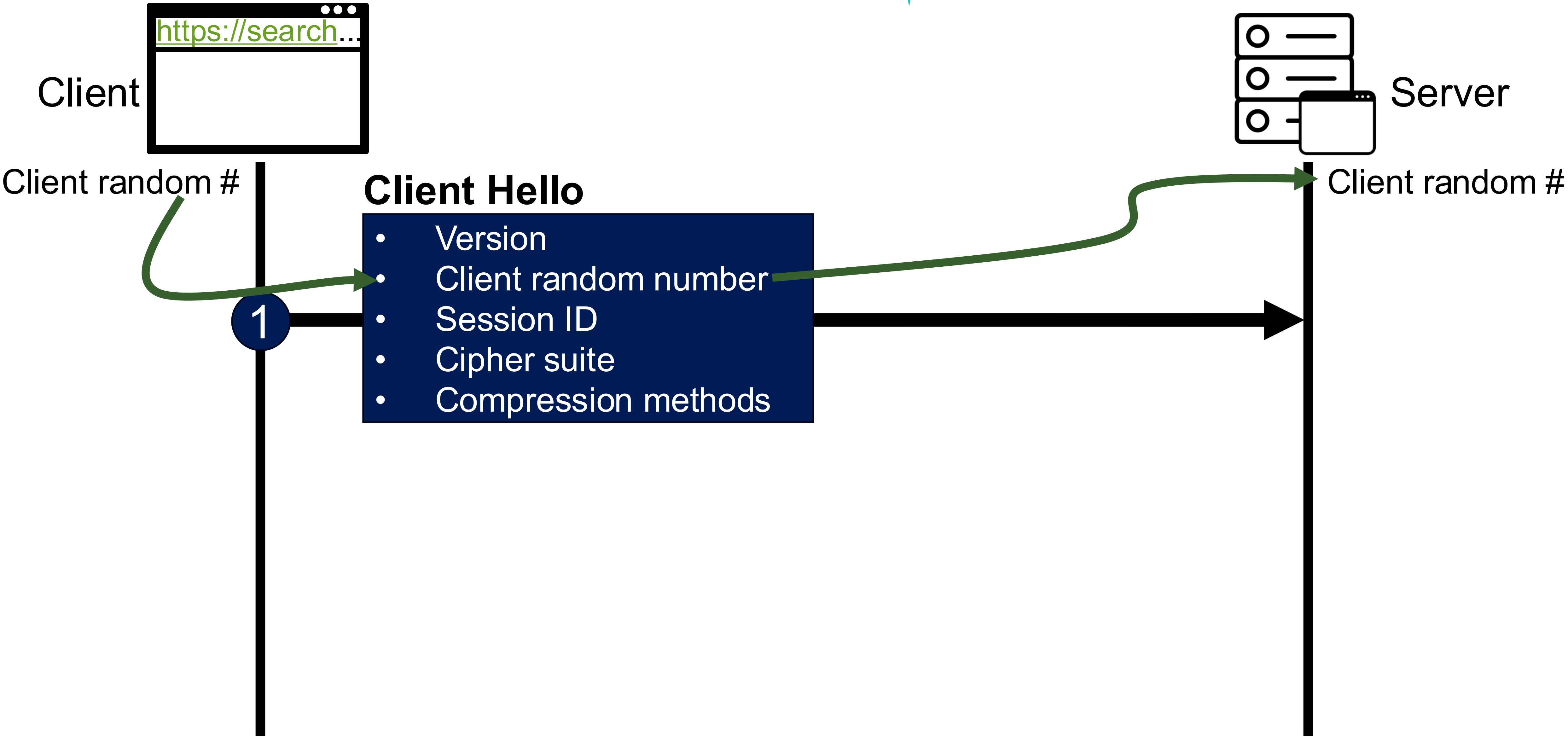
- **Version**
  - Highest protocol version s
- **Client random number**
  -
  -
- **Se**
  - 0: establish new co
  - Non-zero: resume an old s
- **Cipher suite**
  - Set of cryptographic algori the client
- **Compression methods**
  - Sequence of compression

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

https://search...

Server

Client random #

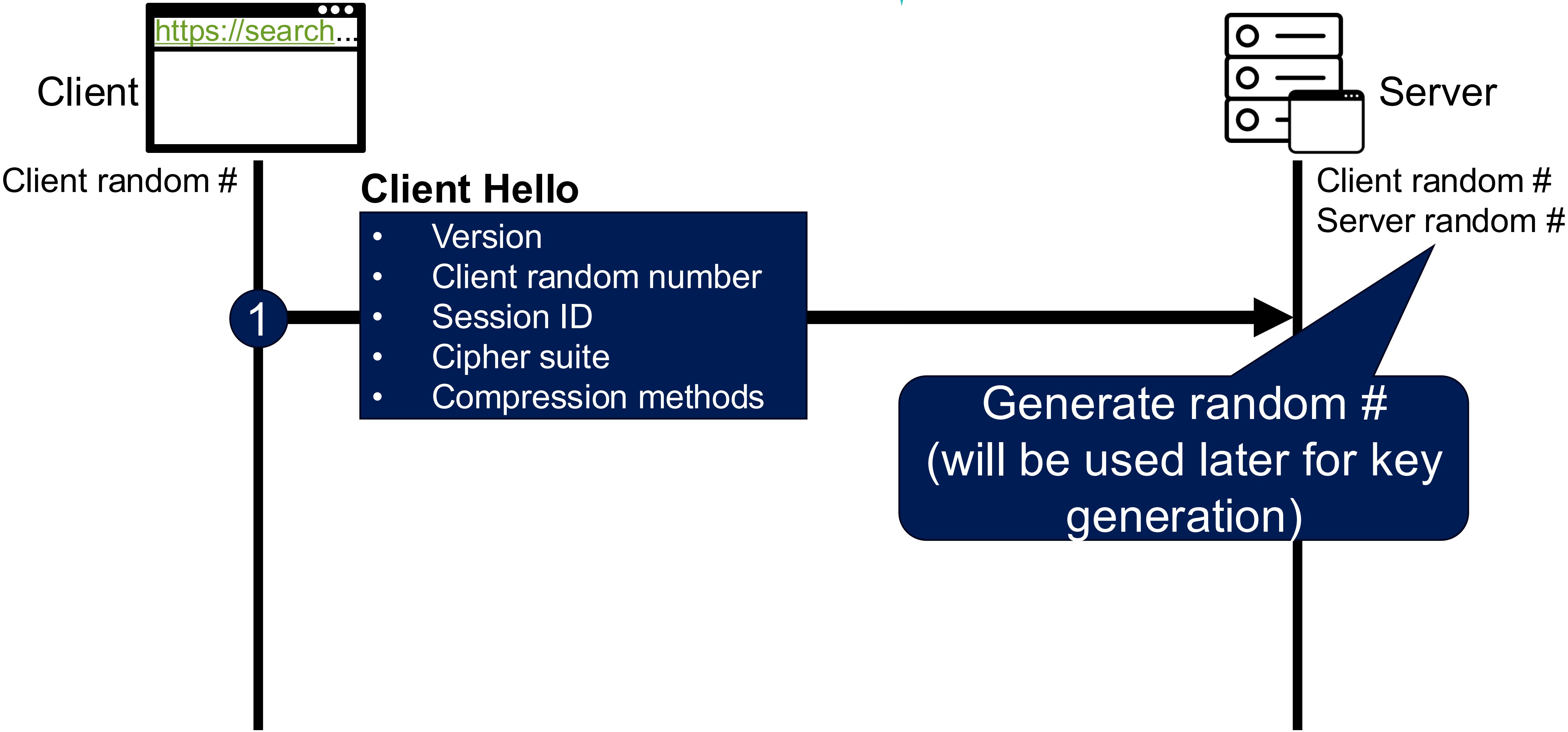
Client Hello

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

1

Client random #

Phase 1: Establishing Security Capabilities

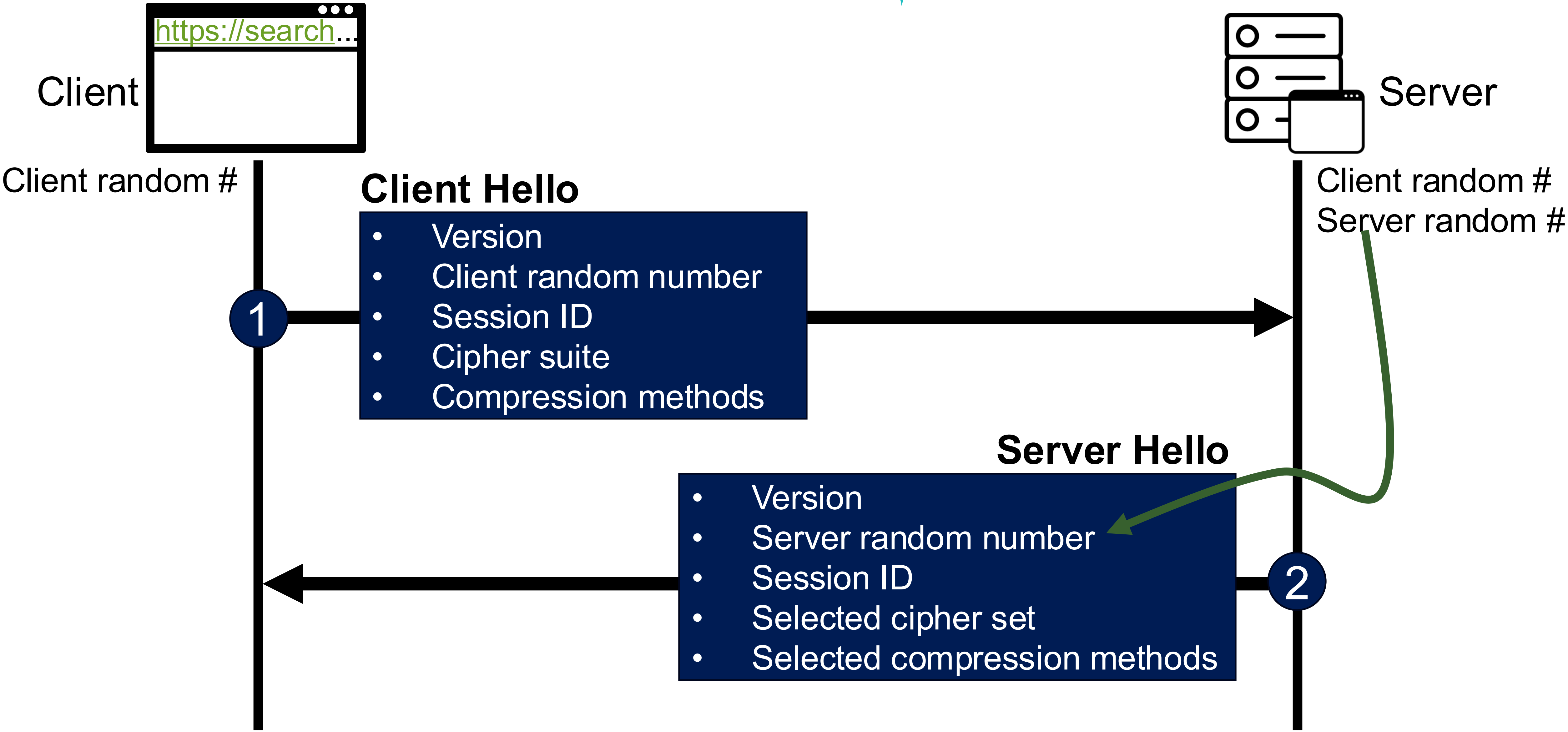


Client Hello

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

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

Phase 1: Establishing Security Capabilities



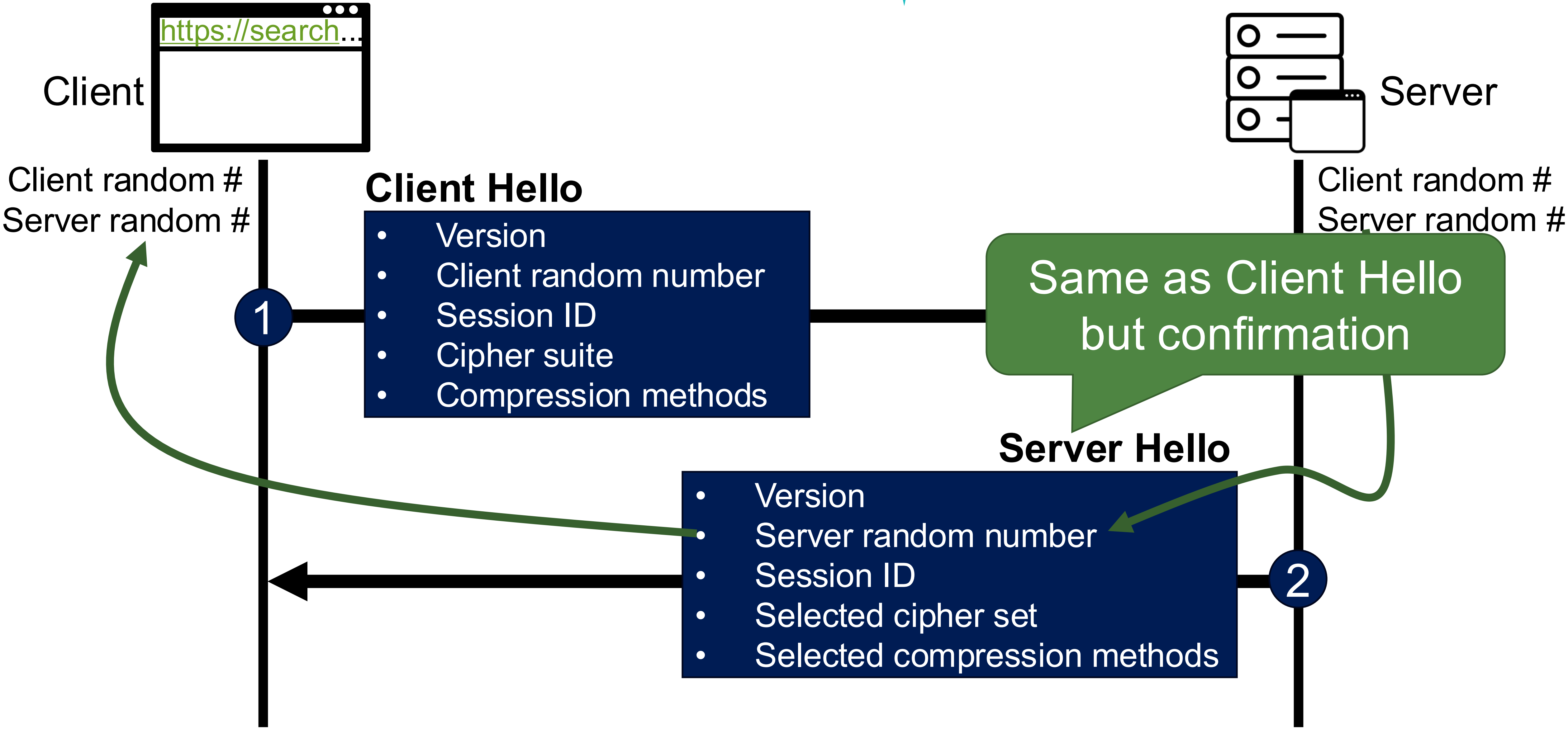
Client Hello

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

Server Hello

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

Phase 1: Establishing Security Capabilities



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

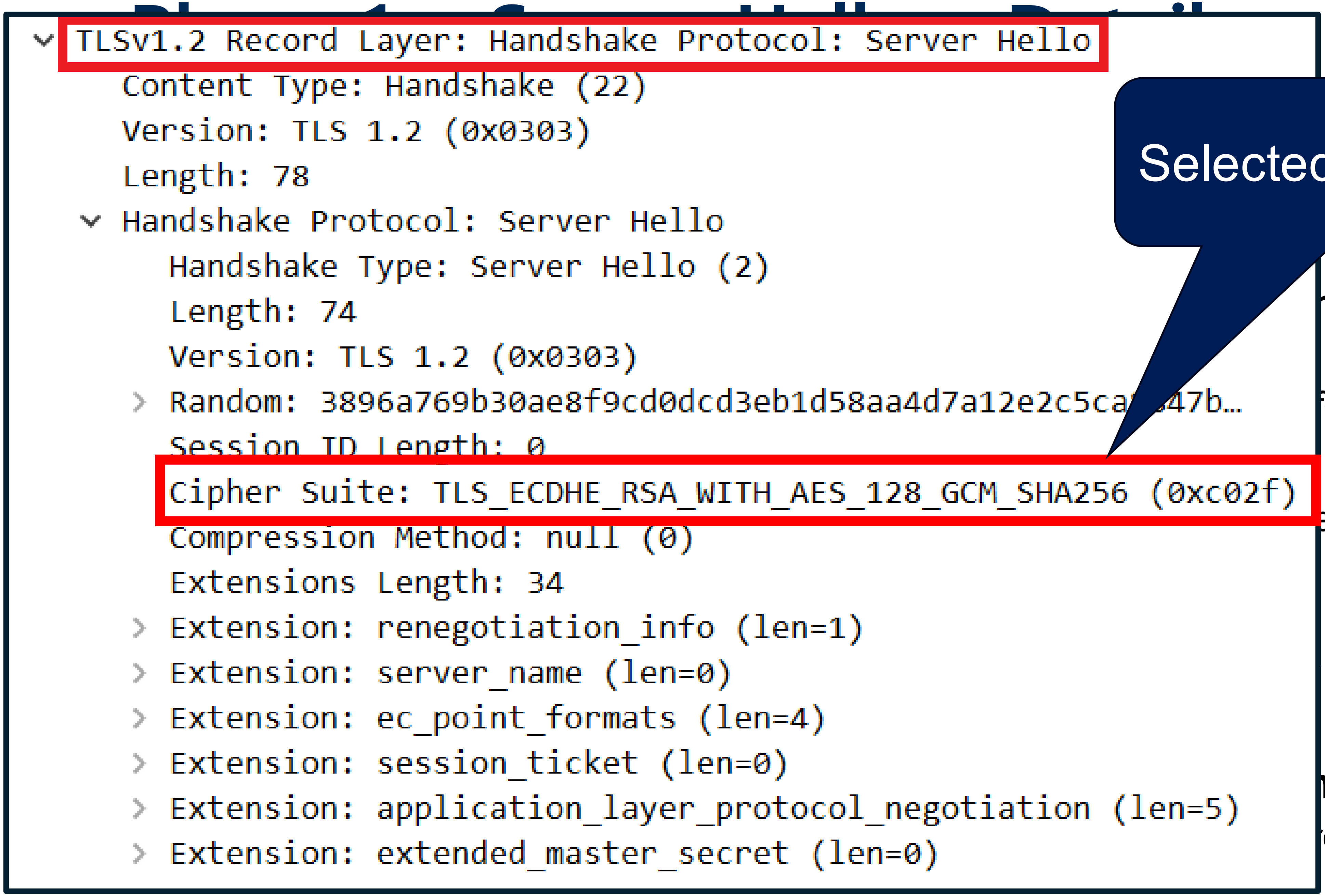
Phase 1 – Server Hello – Details

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

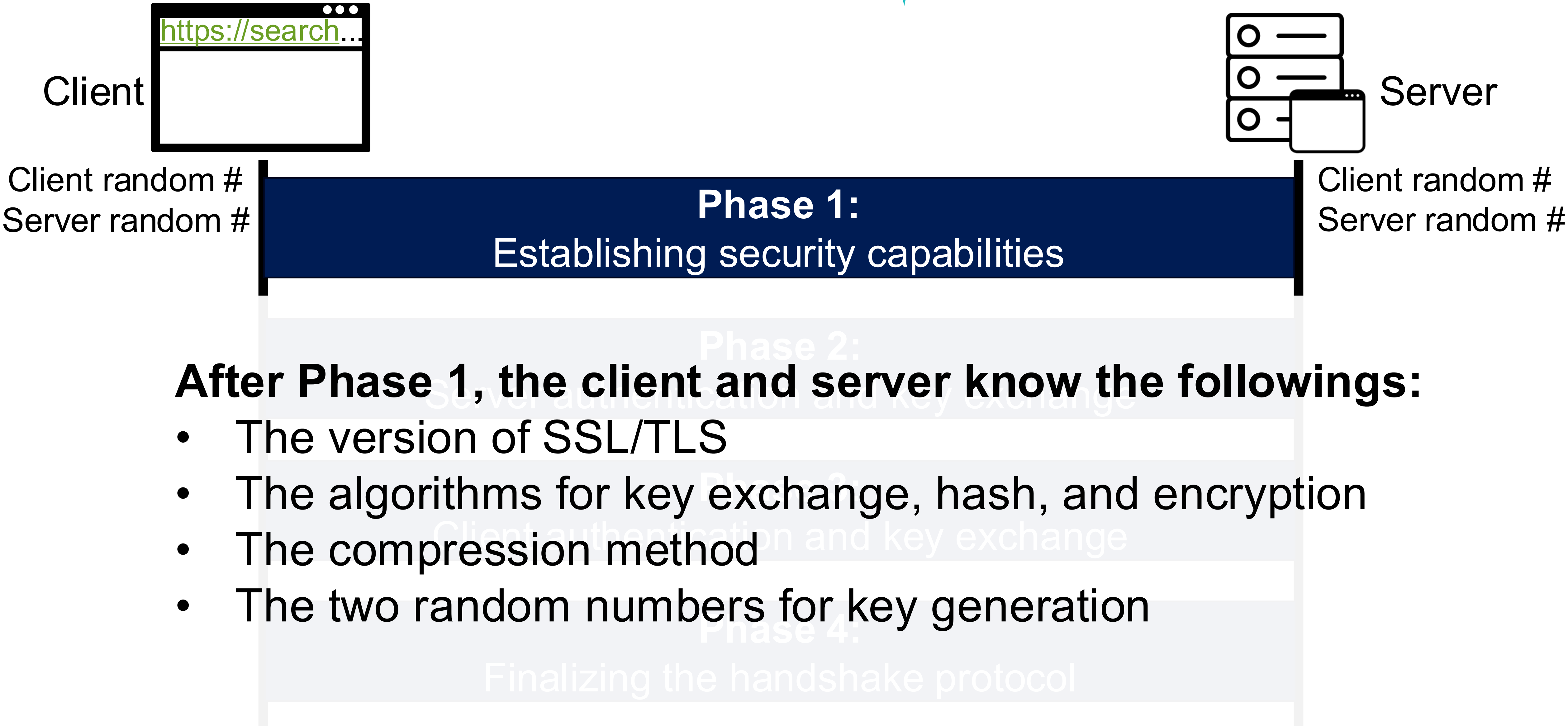
ero, old session ID

suite

hods

ession technique

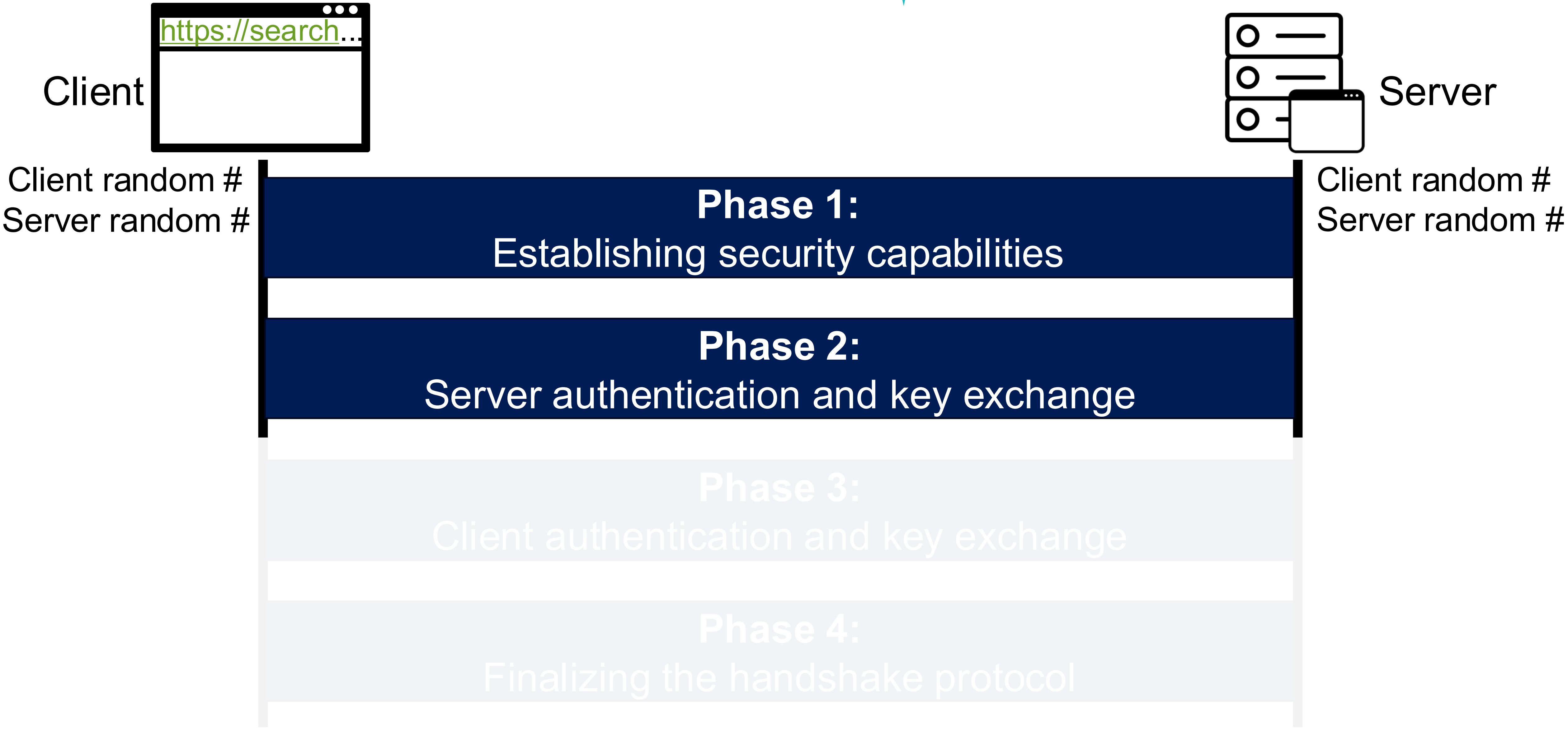
Phase 1: Establishing Security Capabilities


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, hash, and encryption
- The compression method
- The two random numbers for key generation

Phase 2: Server Auth. and Key Exchange



Client

Server

Client random #
Server random #

Client random #
Server random #

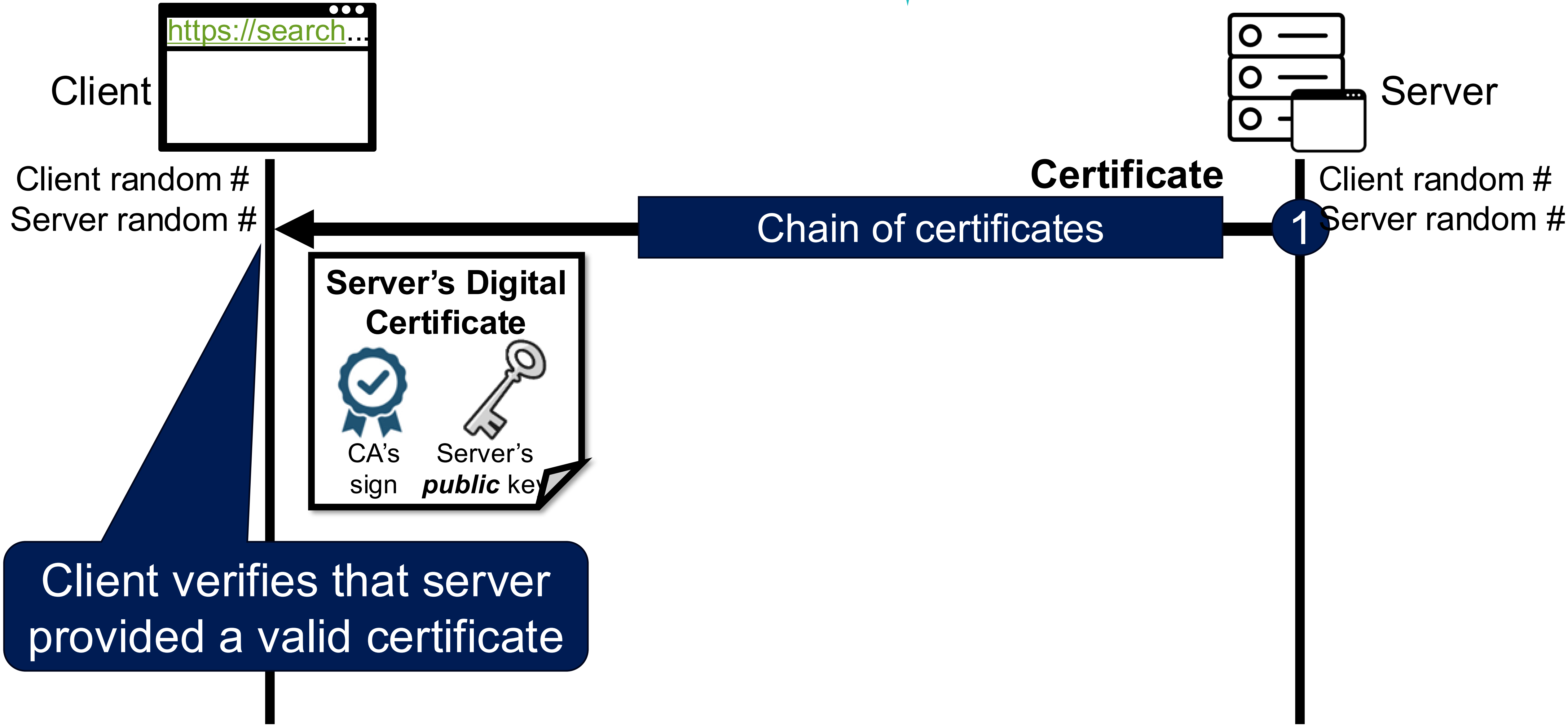
Phase 1:
Establishing security capabilities

Phase 2:
Server authentication and key exchange

Phase 3:
Client authentication and key exchange

Phase 4:
Finalizing the handshake protocol

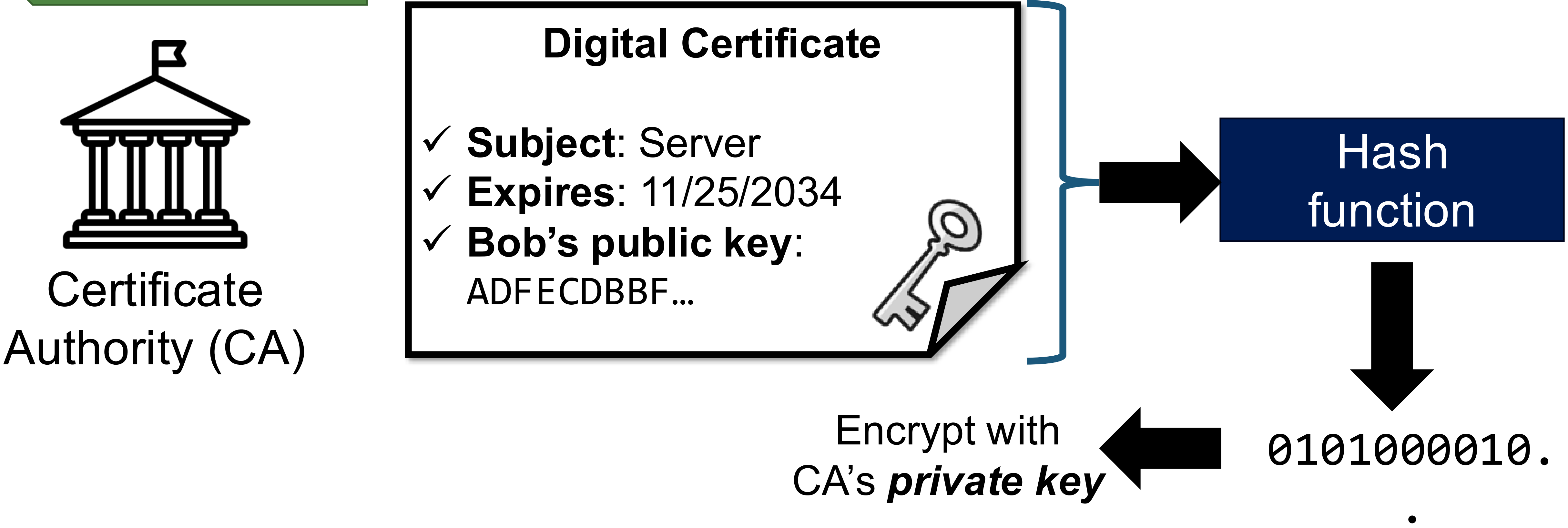
Phase 2: Server Auth. and Key Exchange



Client verifies that server provided a valid certificate

Recap: Hash-based Digital Signature in PKI

Signing



Certificate Authority (CA)

Digital Certificate

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

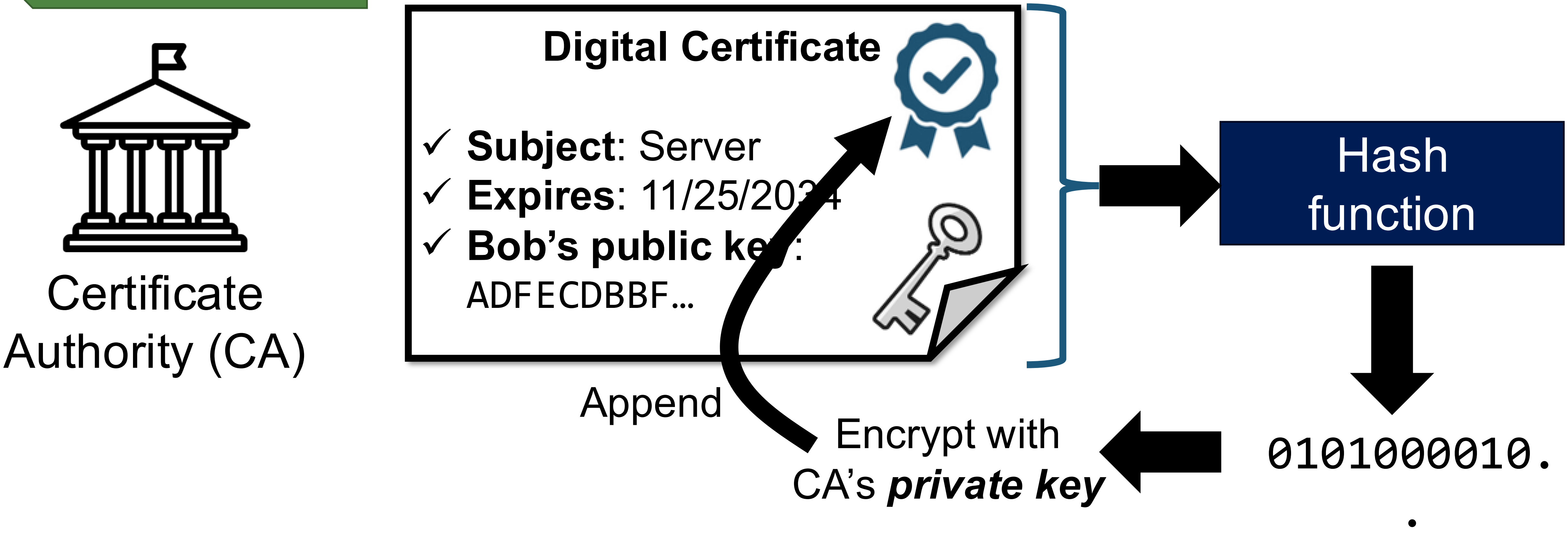
Hash function

0101000010. .

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

Recap: Hash-based Digital Signature in PKI

Signing



Certificate Authority (CA)

Digital Certificate

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

Hash function

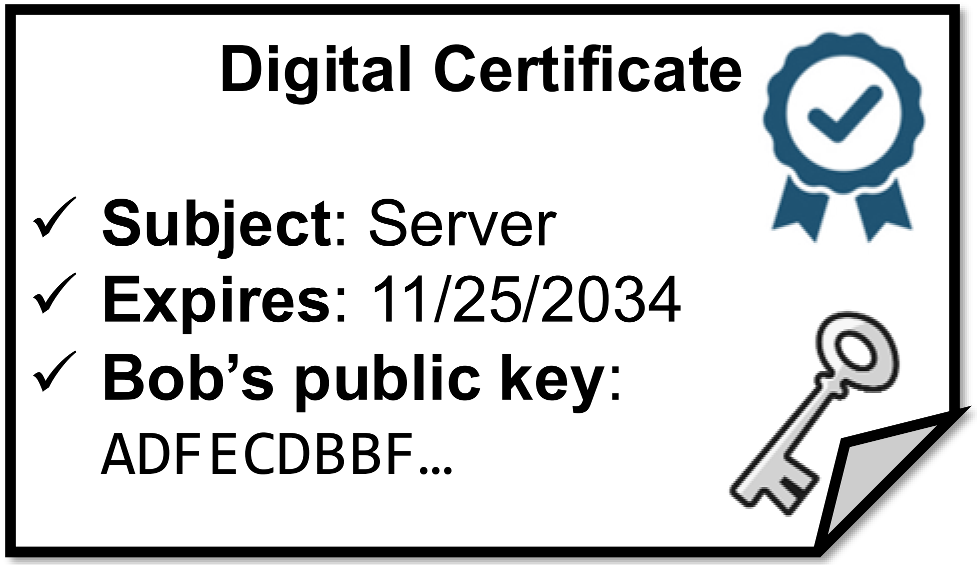
0101000010. .

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

Append

Recap: Hash-based Digital Signature in PKI

Verification

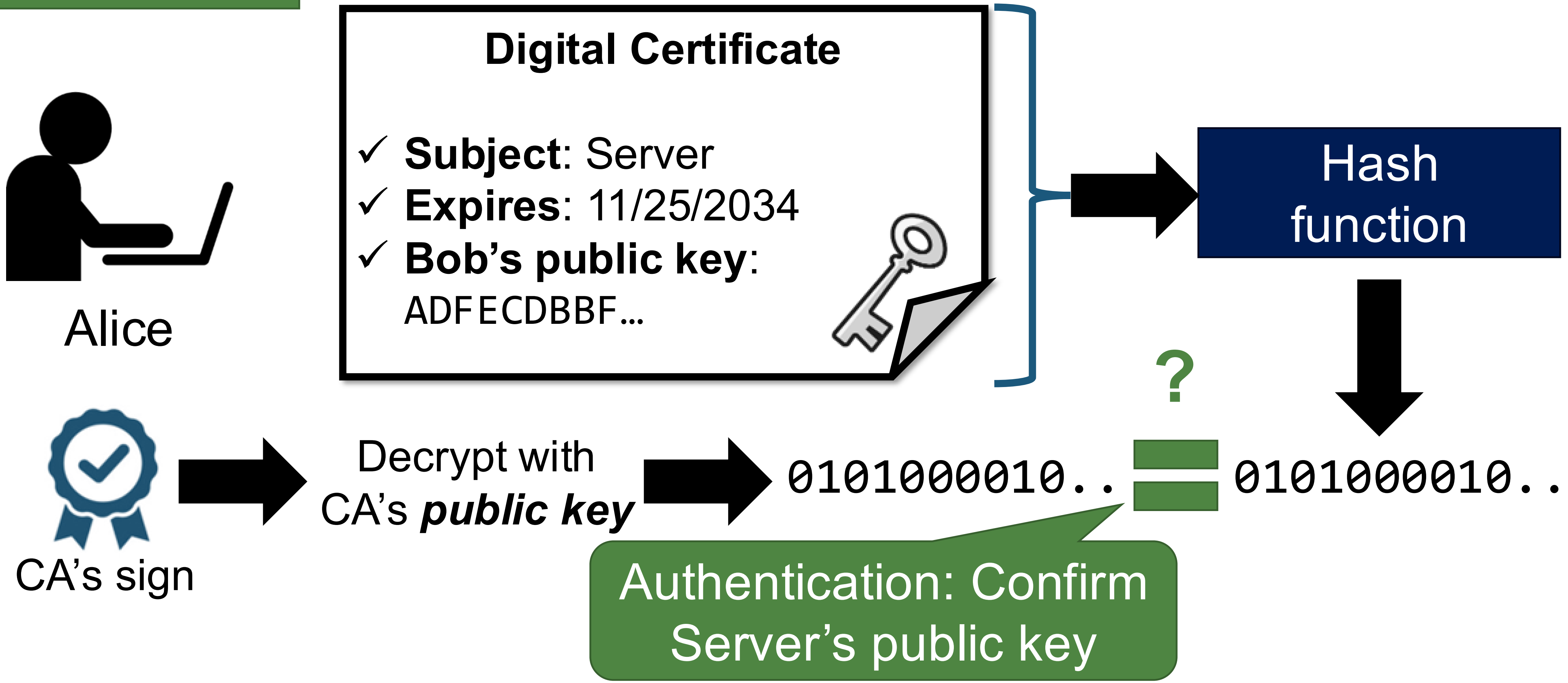


Digital Certificate

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

Recap: Hash-based Digital Signature in PKI

Verification

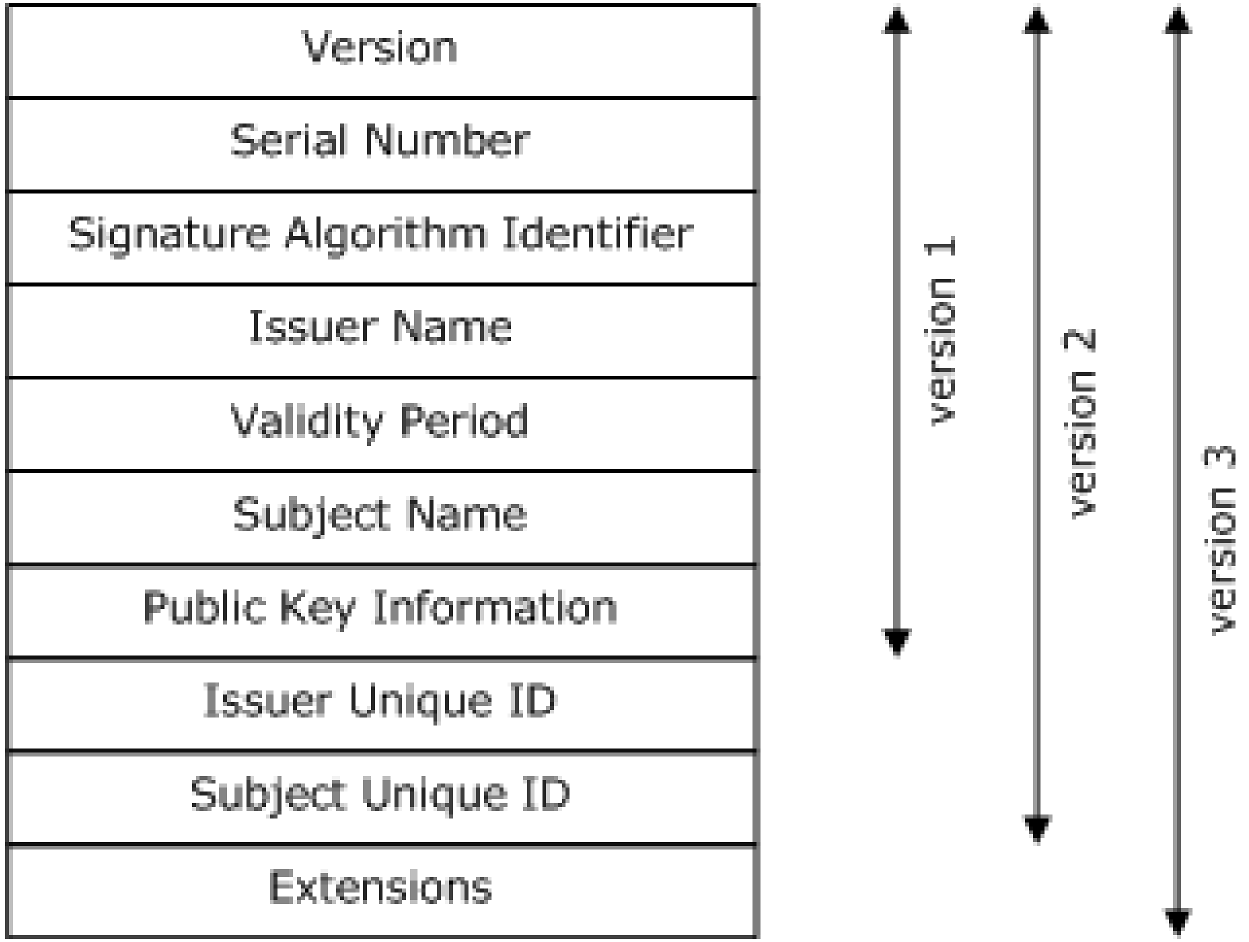


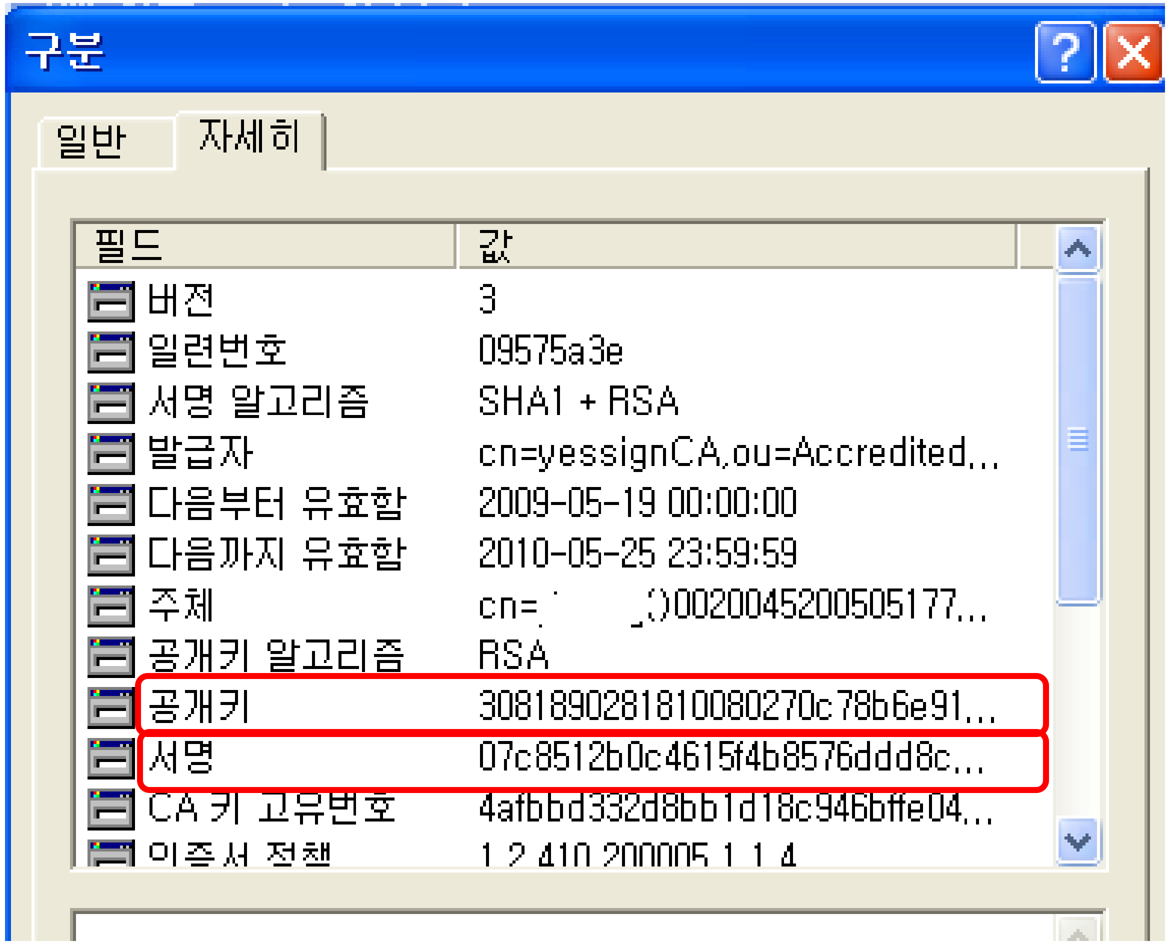
Digital Certificate

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

Authentication: Confirm Server's public key

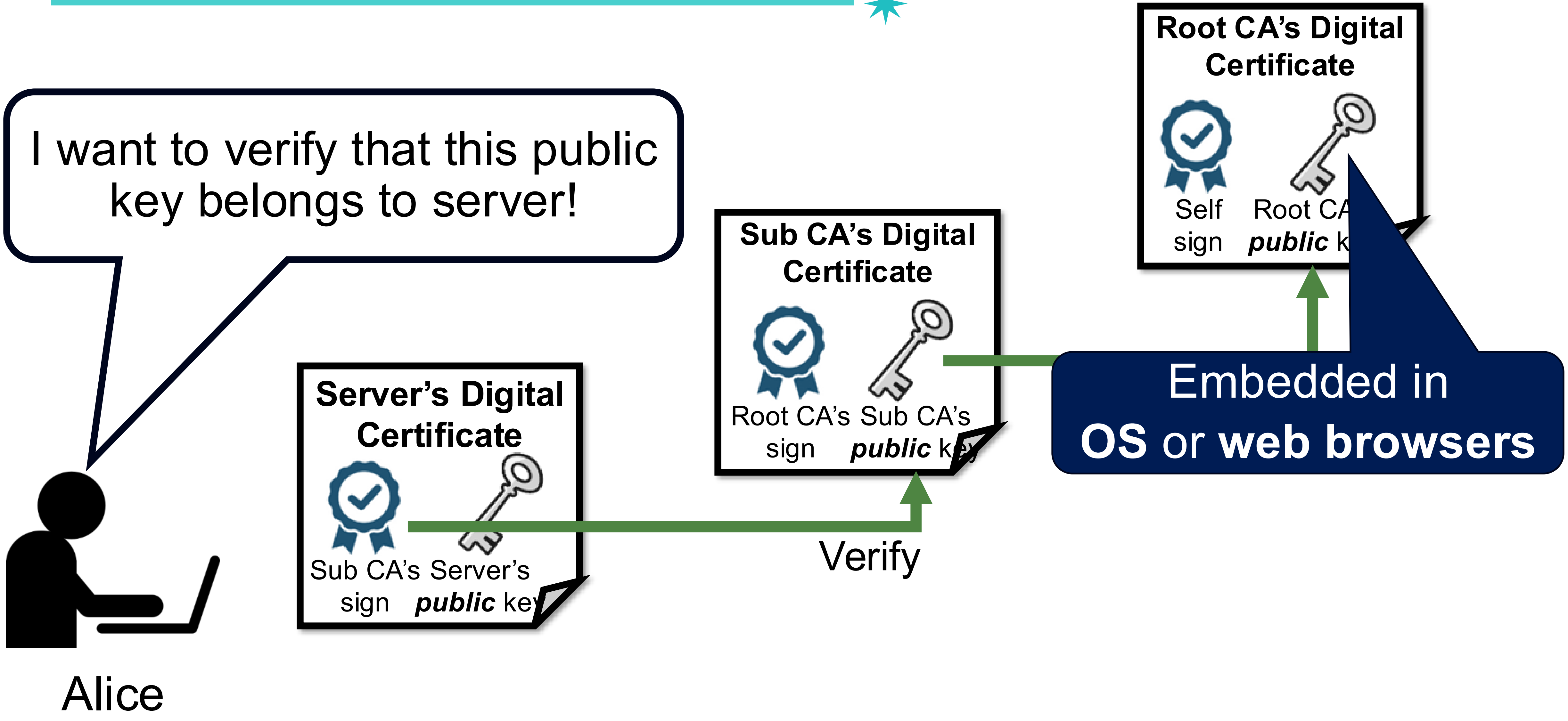
Recap: X.509 Certificate



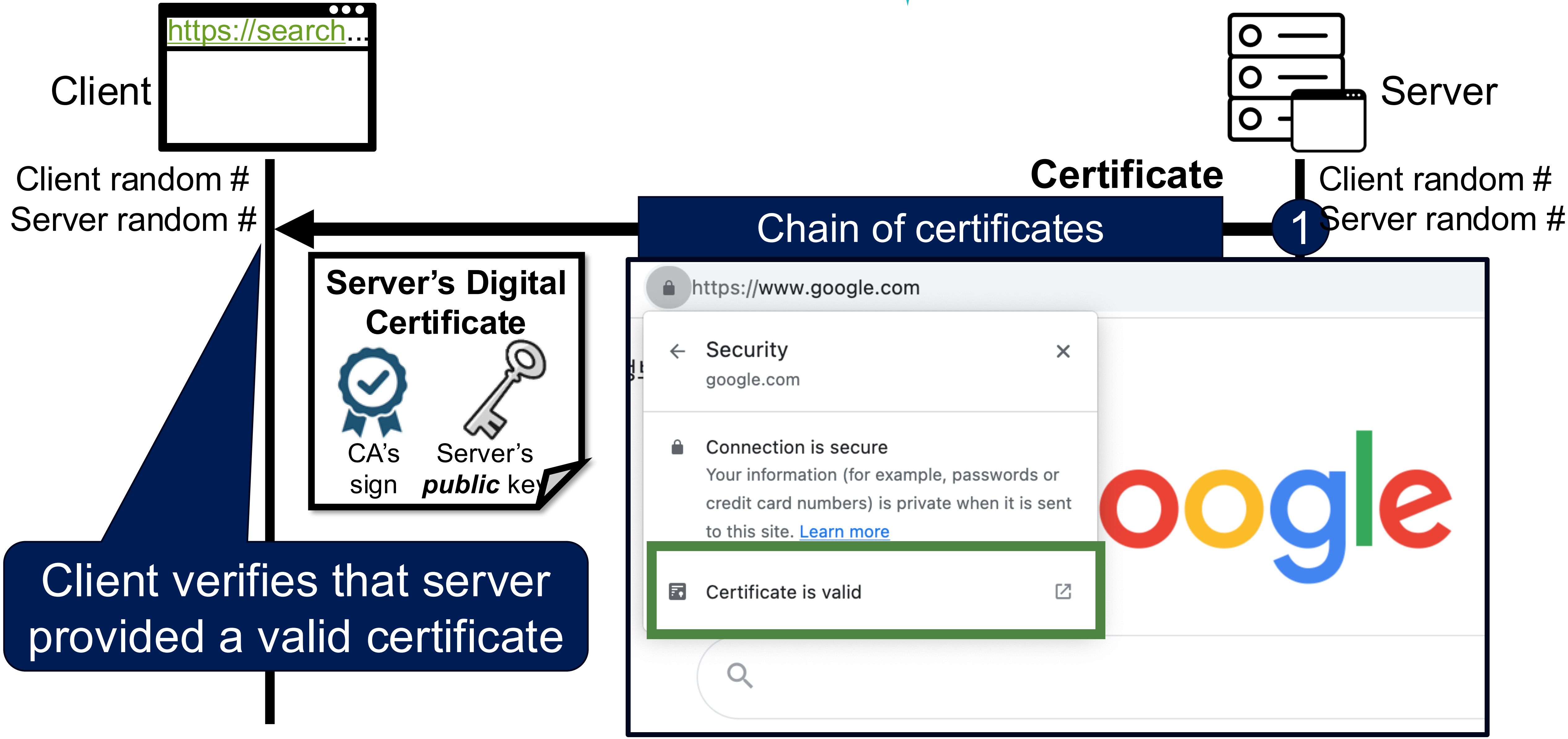


필드	값
버전	3
일련번호	09575a3e
서명 알고리즘	SHA1 + RSA
발급자	cn=yessignCA,ou=Accredited,...
다음부터 유효함	2009-05-19 00:00:00
다음까지 유효함	2010-05-25 23:59:59
주체	cn=[redacted]()0020045200505177,...
공개키 알고리즘	RSA
공개키	30818902818100802 70c78b6e91,...
서명	07c8512b0c4615f4b8576ddd8c,...
CA 키 고유번호	4afbbd332d8bb1d18c946bffe04,...
인증서 정책	1.2.410.200005.1.1.4

Recap: Chain of Trust


I want to verify that this public key belongs to server!
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
Alice

Phase 2: Server Auth. and Key Exchange



https://www.google.com

Security

google.com

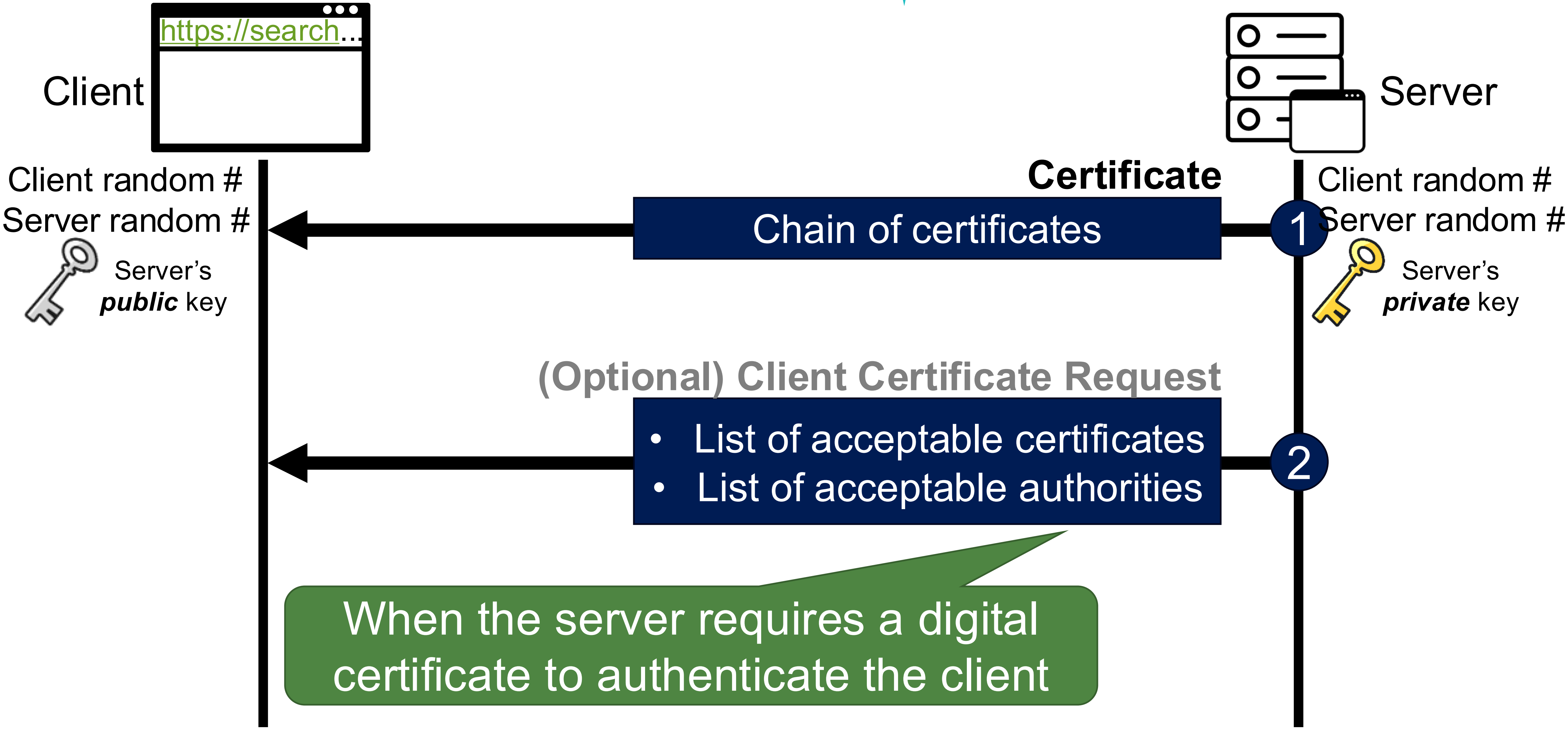
Connection is secure

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

Certificate is valid

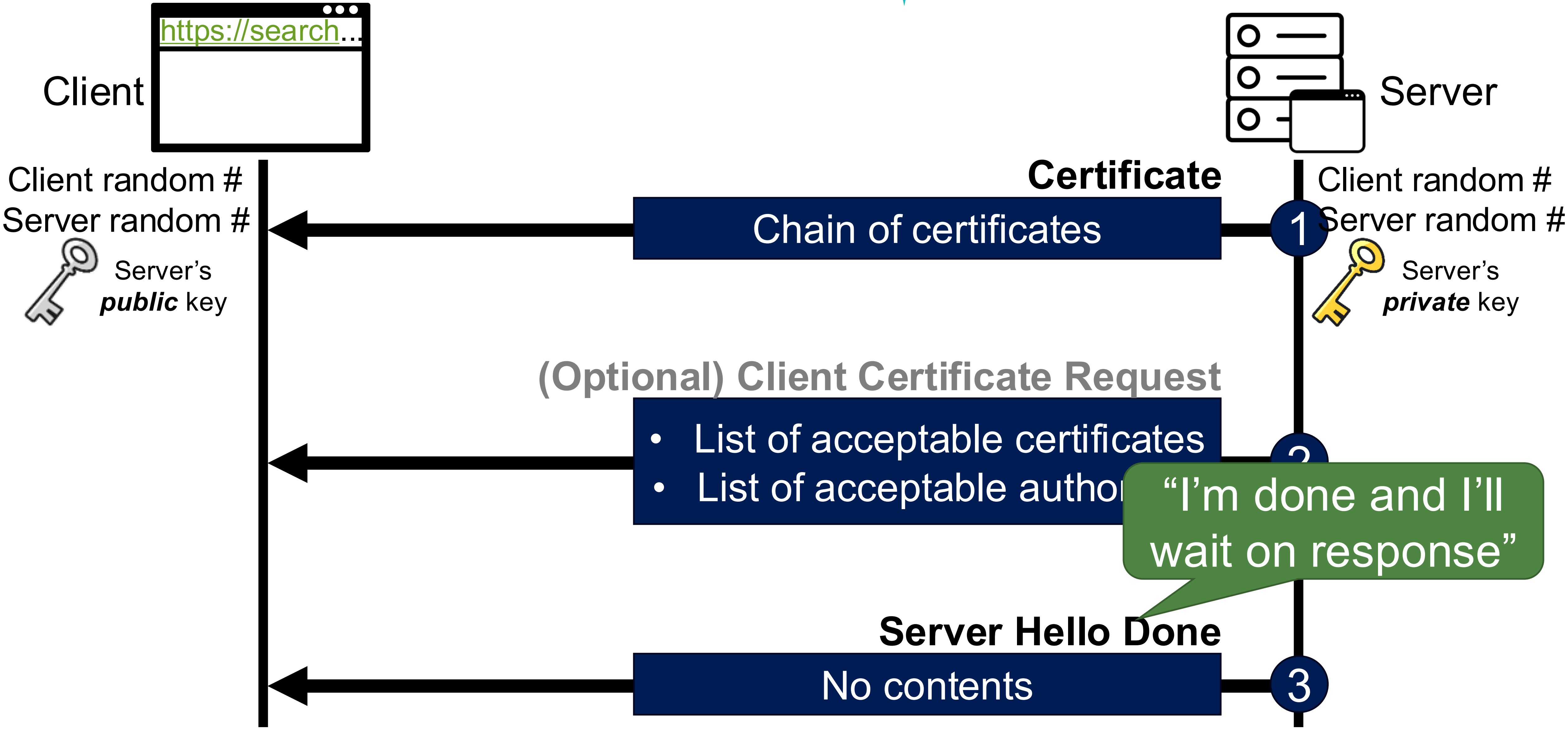
Client verifies that server provided a valid certificate

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



Certificate

Chain of certificates

(Optional) Client Certificate Request

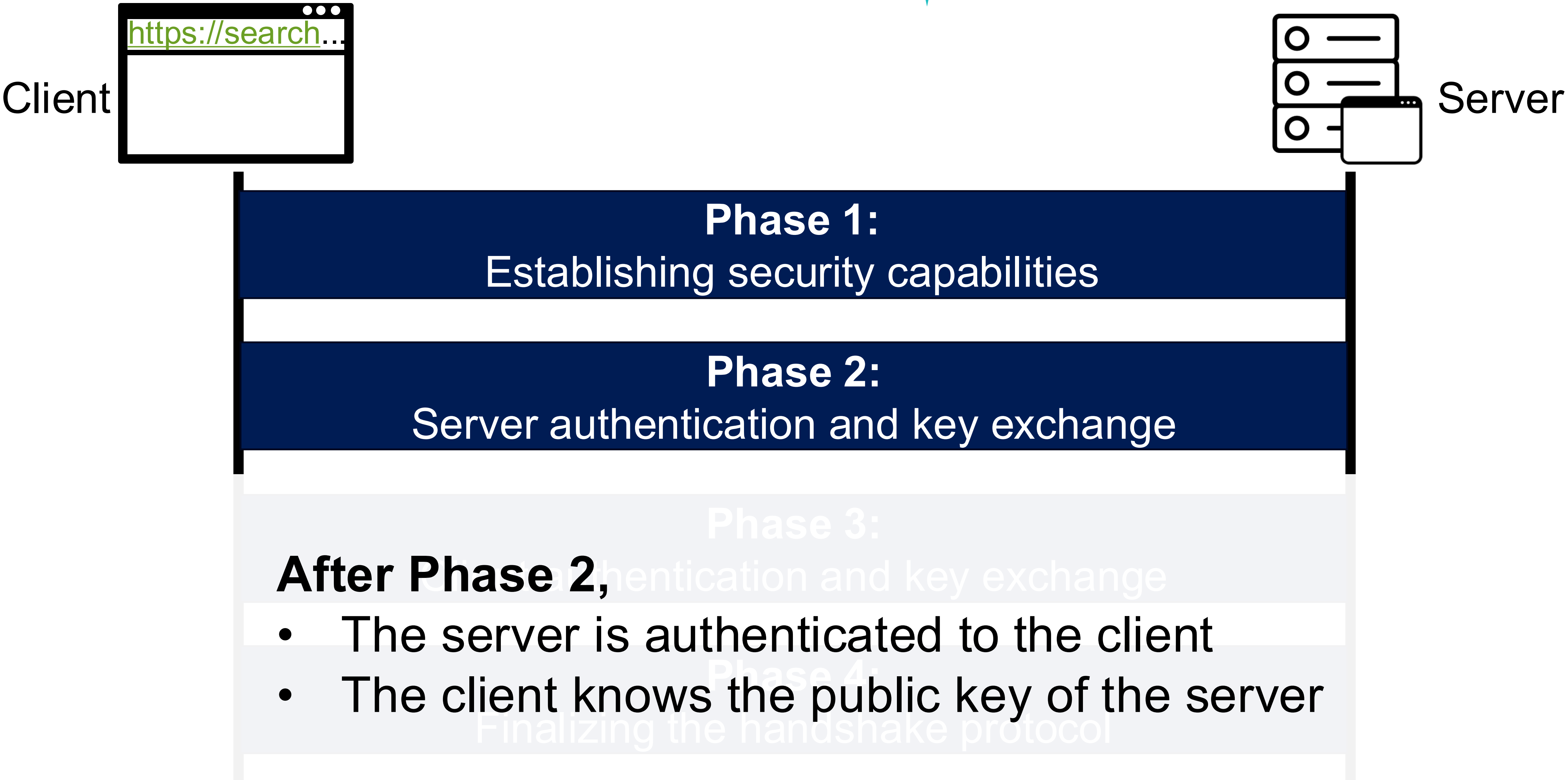
- List of acceptable certificates
- List of acceptable author

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

Server Hello Done

No contents

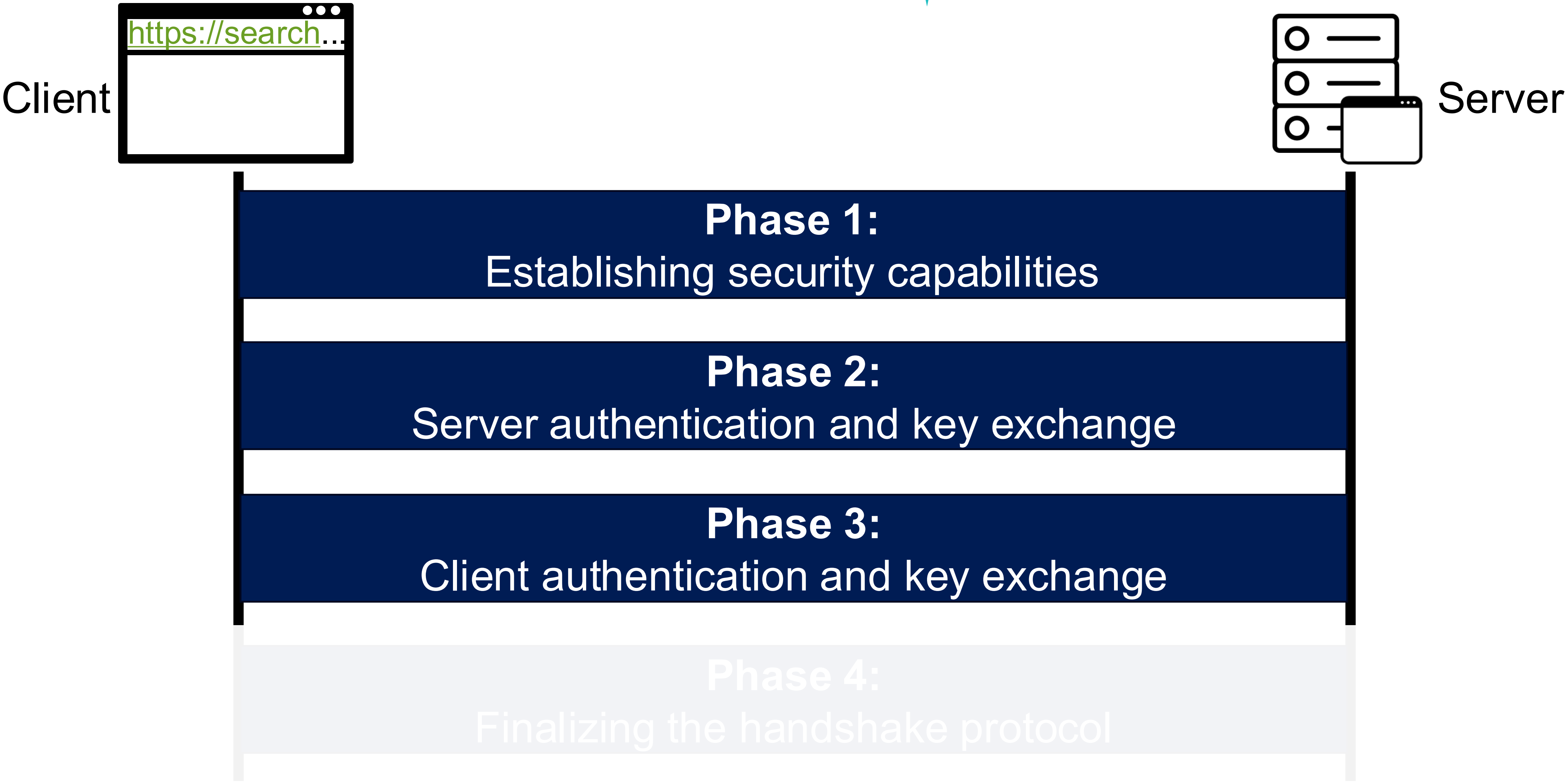
Phase 1: Establishing Security Capabilities


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

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



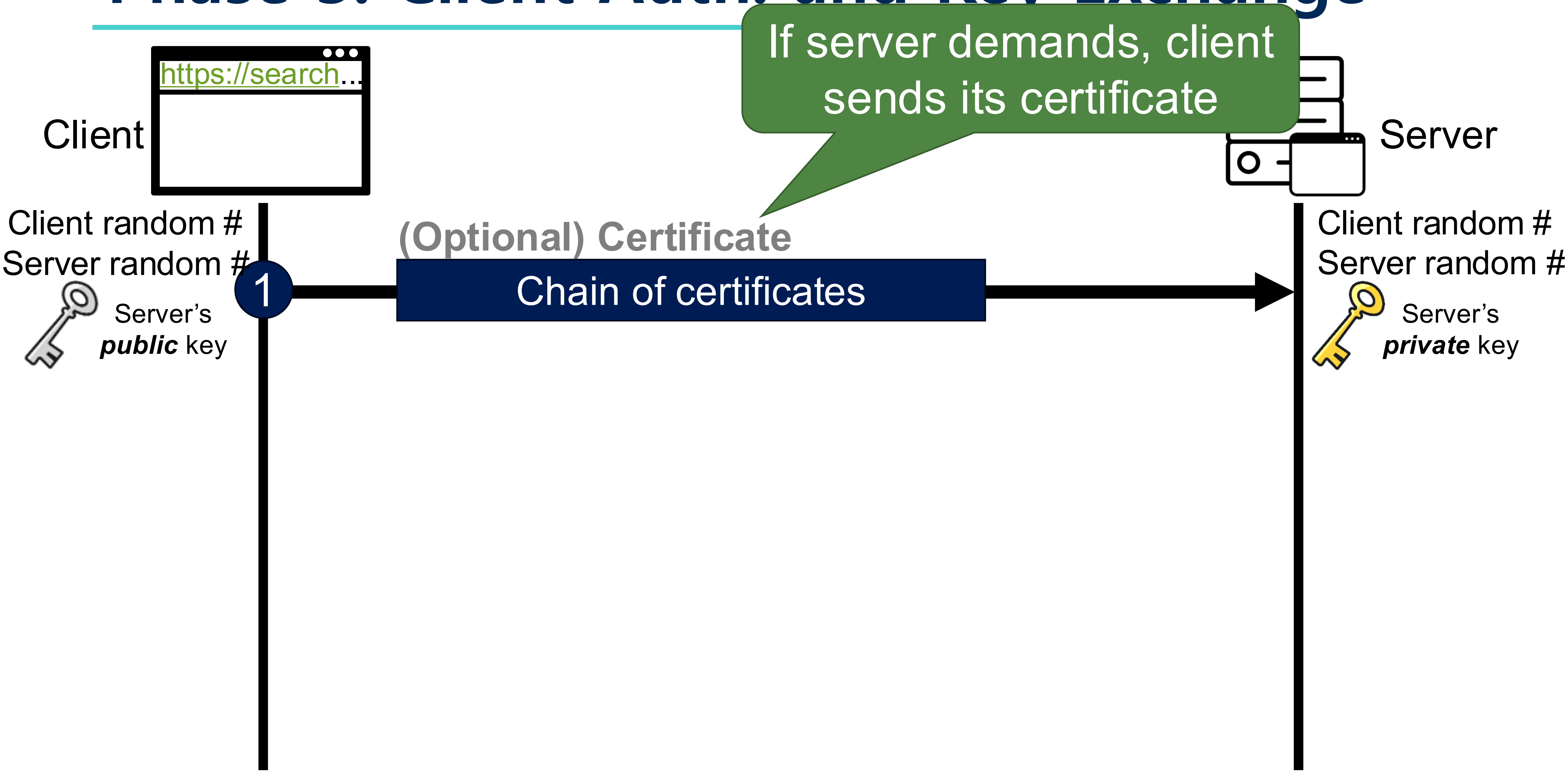
Phase 1:
Establishing security capabilities

Phase 2:
Server authentication and key exchange

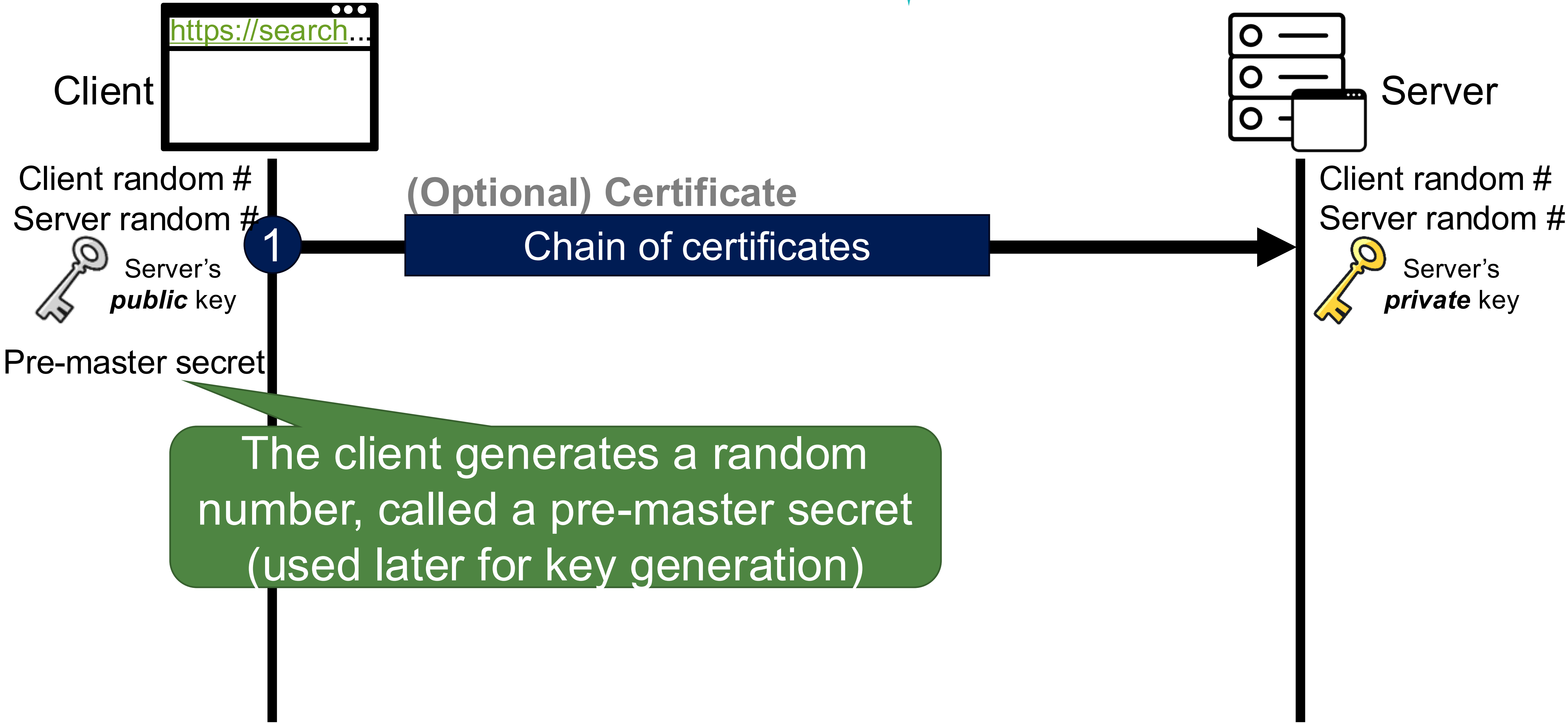
Phase 3:
Client authentication and key exchange

Phase 4:
Finalizing the handshake protocol

Phase 3: Client Auth. and Key Exchange

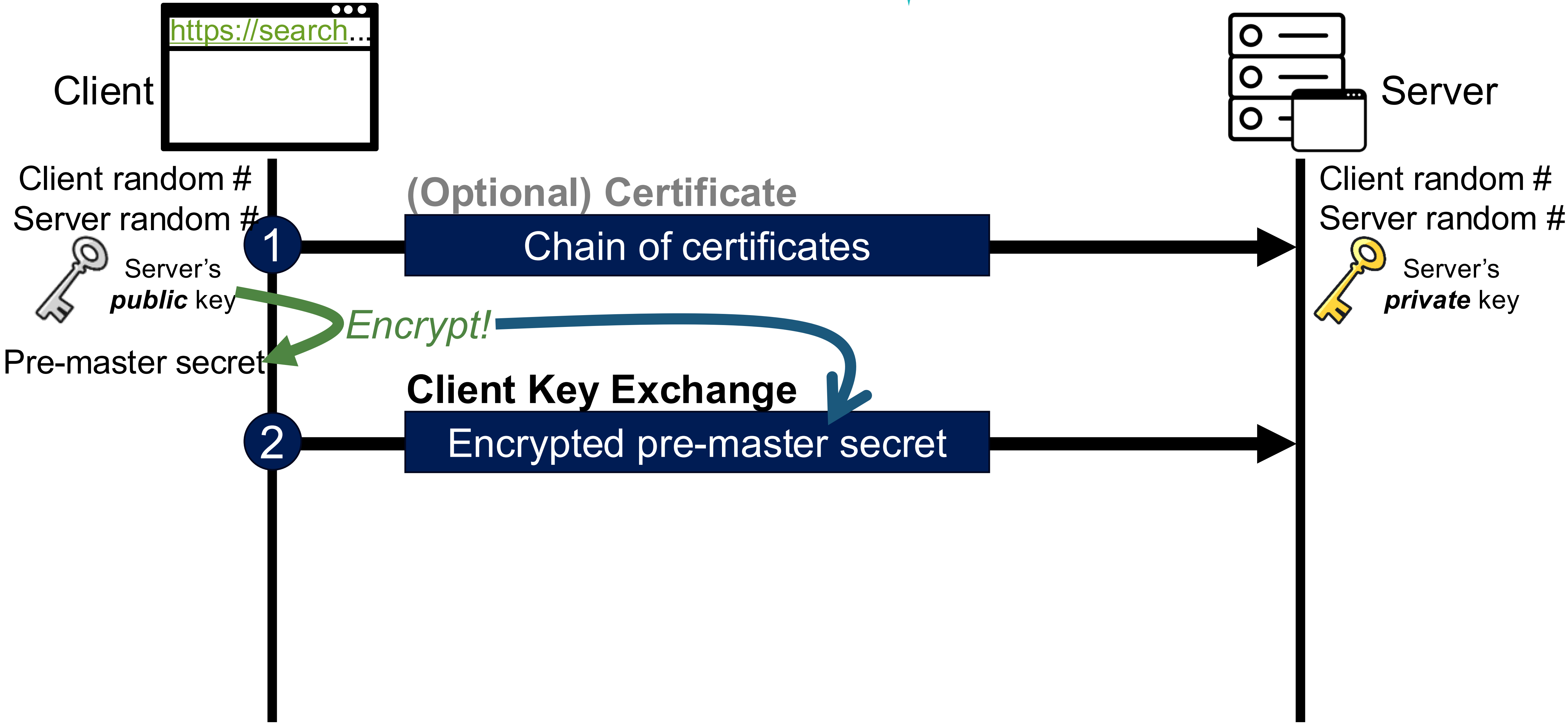


Phase 3: Client Auth. and Key Exchange

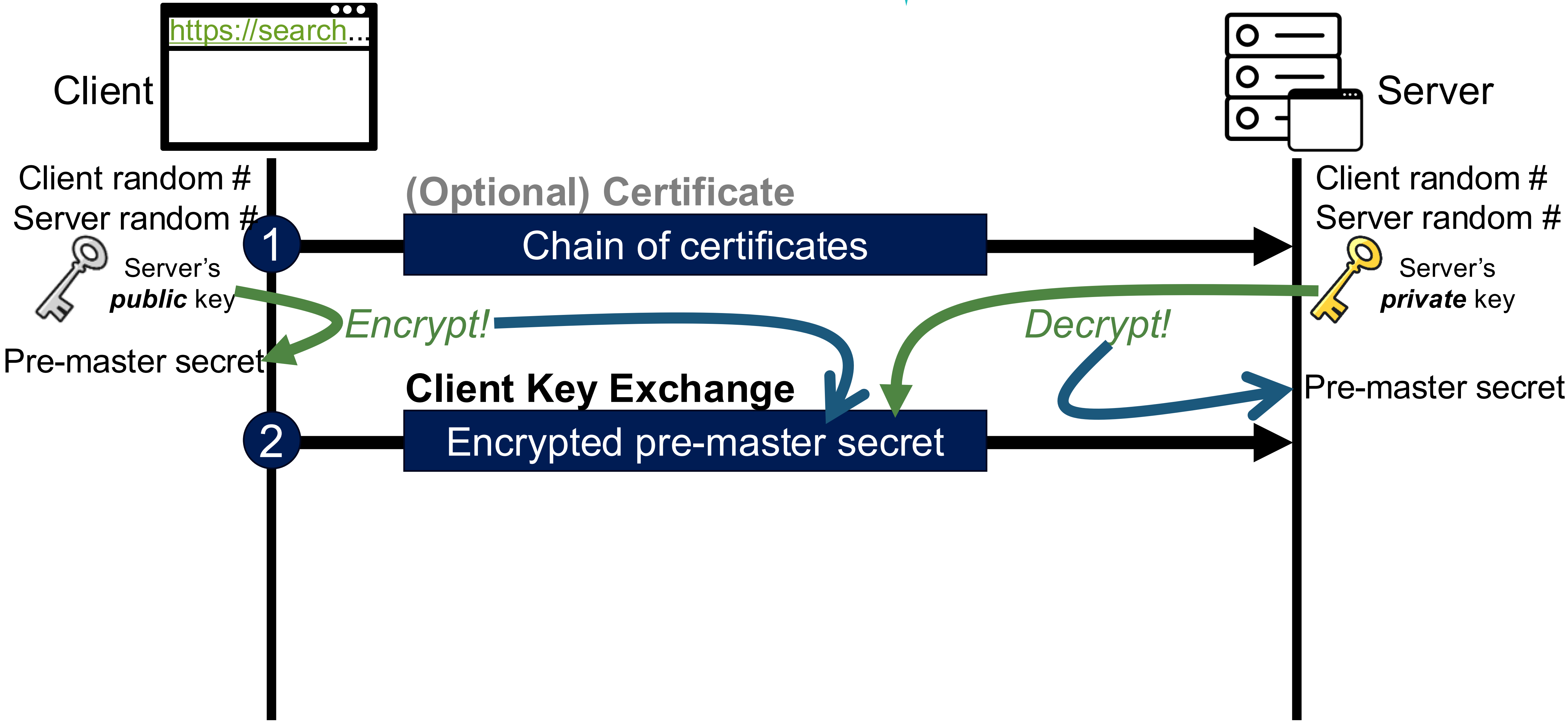


The client generates a random number, called a pre-master secret (used later for key generation)

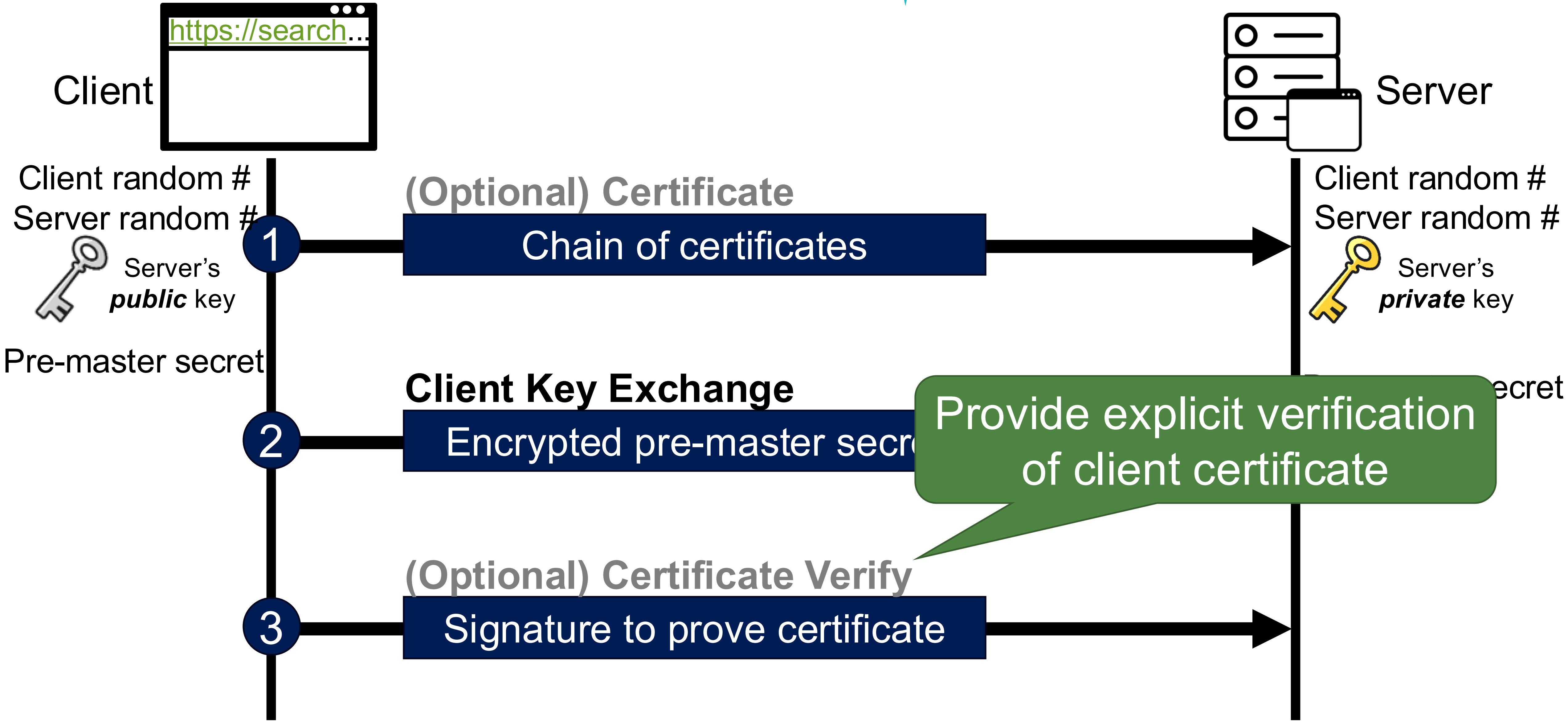
Phase 3: Client Auth. and Key Exchange



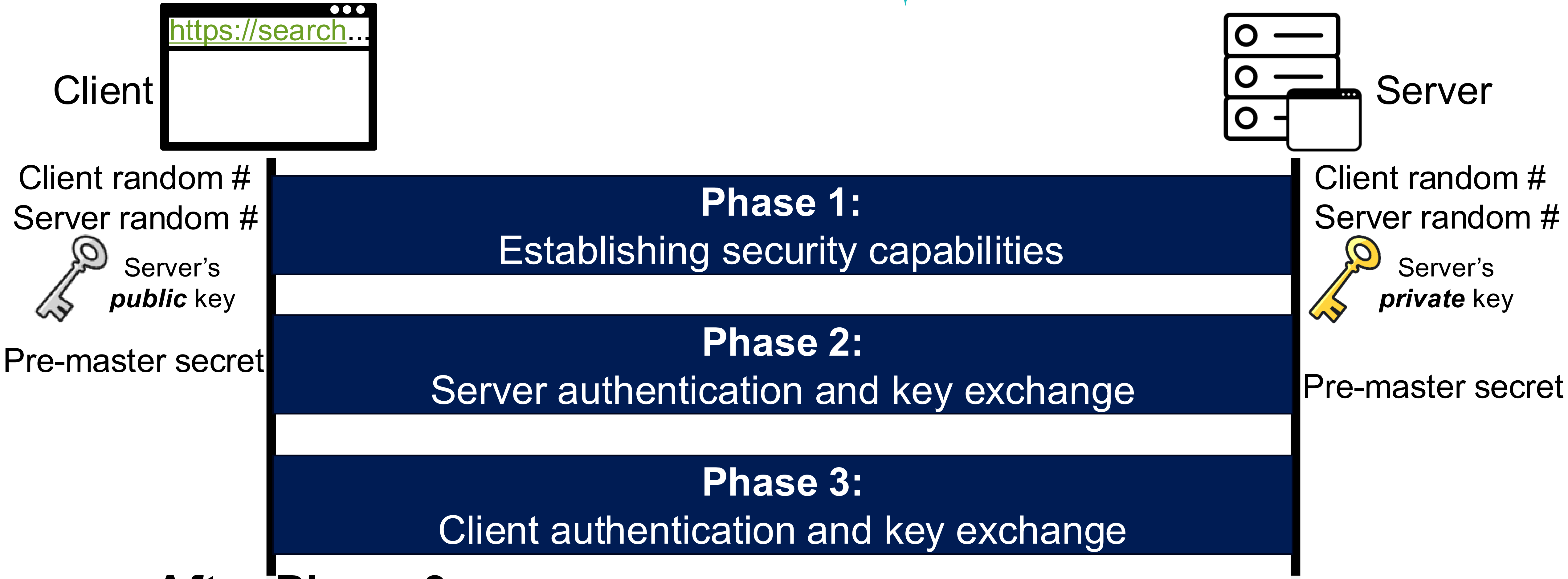
Phase 3: Client Auth. and Key Exchange



Phase 3: Client Auth. and Key Exchange



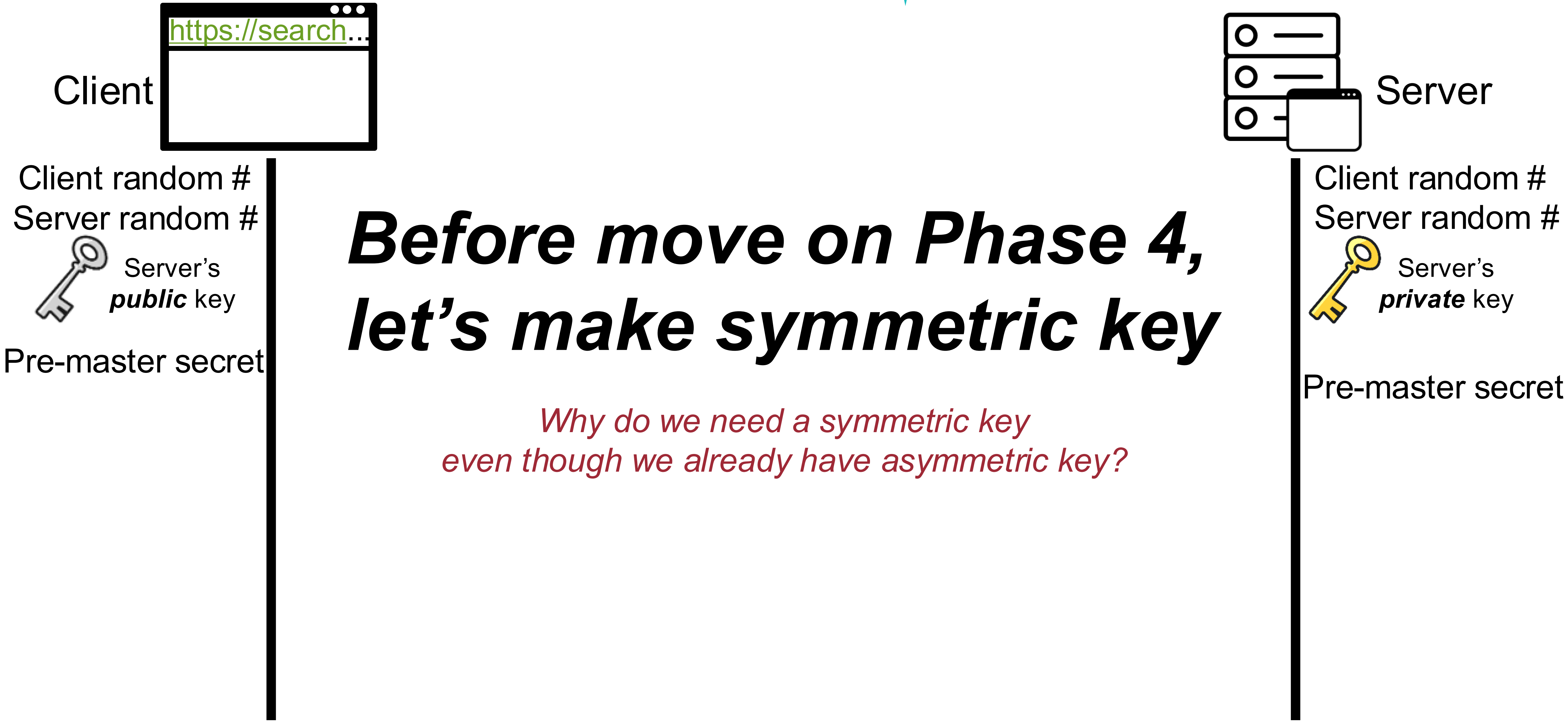
Phase 3: Client Auth. and Key Exchange



After Phase 3,

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

Phase 3: Client Auth. and Key Exchange



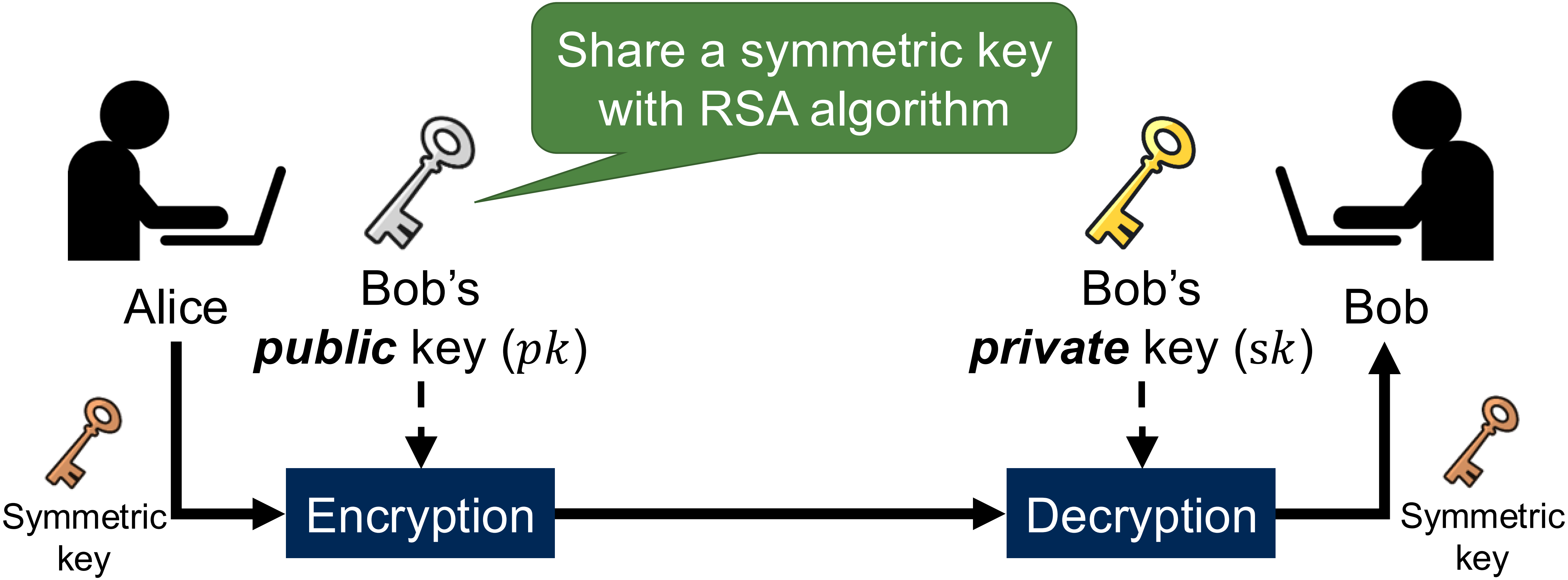
Before move on Phase 4, let's make symmetric key

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

Recap: Asymmetric-key Cryptography

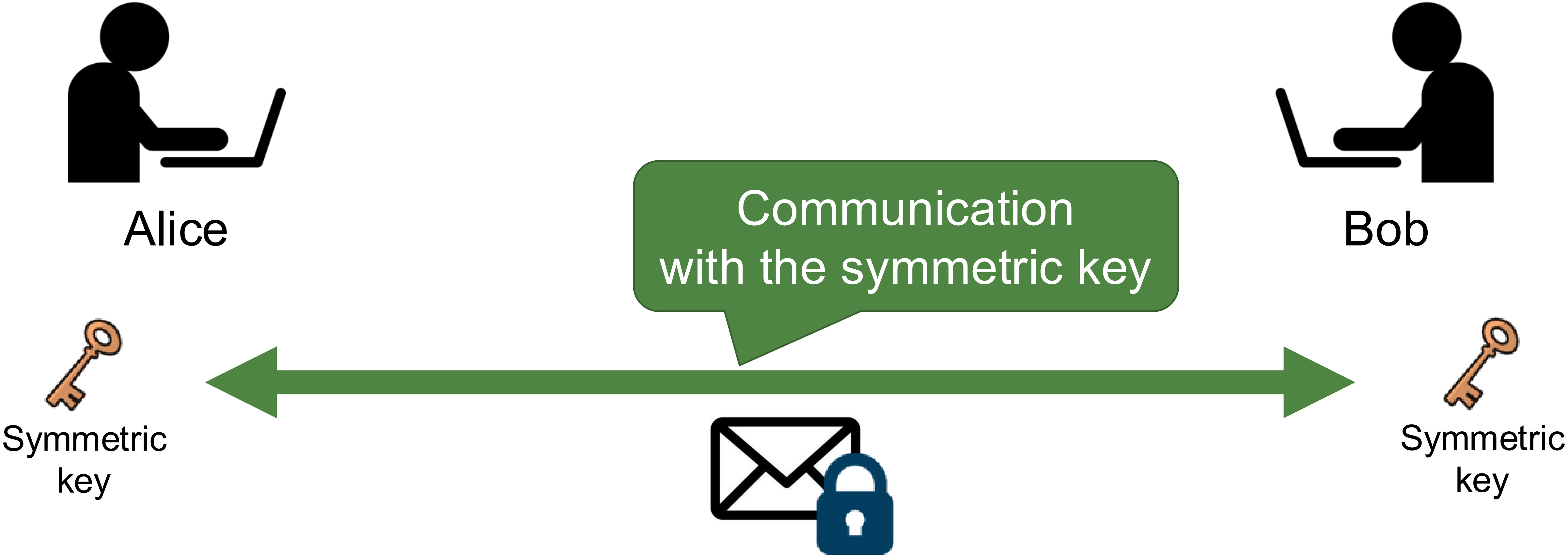
- Pros
  - No need to share a secret
  - Enable multiple senders to communicate privately with a single receiver
  - More applications: Digital sign
- Cons
  - **Slower in general**: due to the larger key
    - Roughly 2-3 orders of magnitude slower

Recap: Combination of Two Schemes



Share a symmetric key with RSA algorithm
Alice
Bob's *public* key (pk)
Bob's *private* key (sk)
Bob
Symmetric key
Encryption
Decryption
Symmetric key

Recap: Combination of Two Schemes



Alice

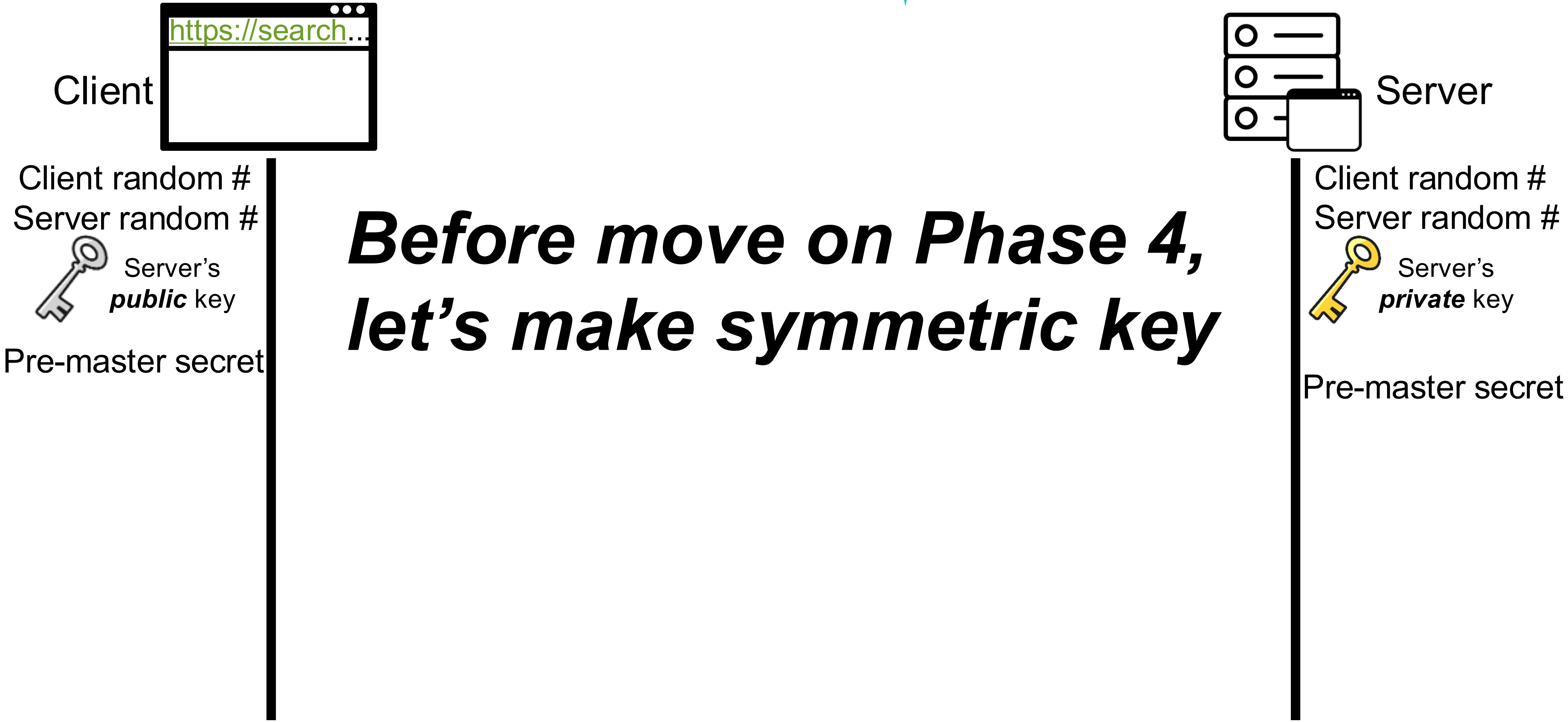
Communication with the symmetric key

Bob

Symmetric key

Symmetric key

Phase 3: Client Auth. and Key Exchange



Client

https://search...

Server

Client random #
Server random #

Server's
public key

Pre-master secret

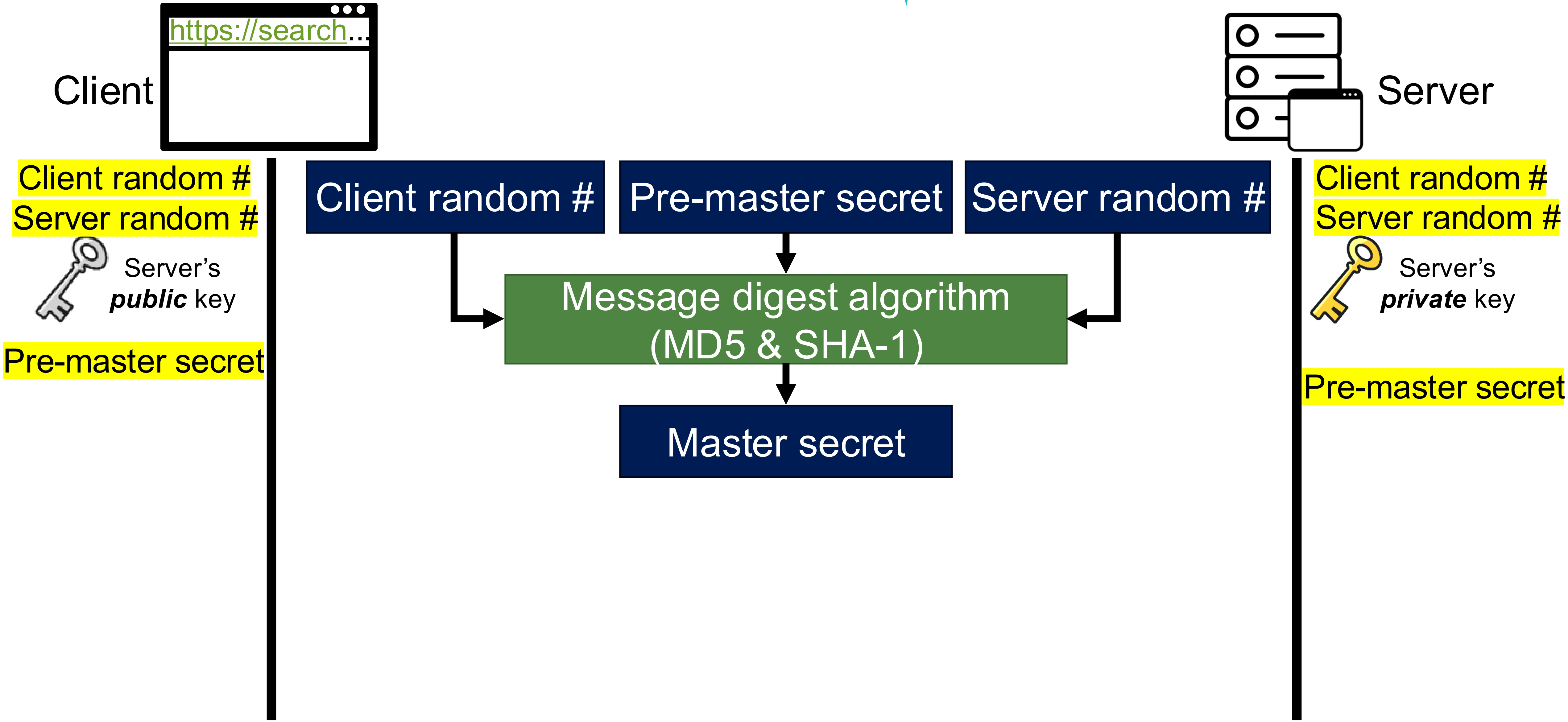
Before move on Phase 4, let's make symmetric key

Client random #
Server random #

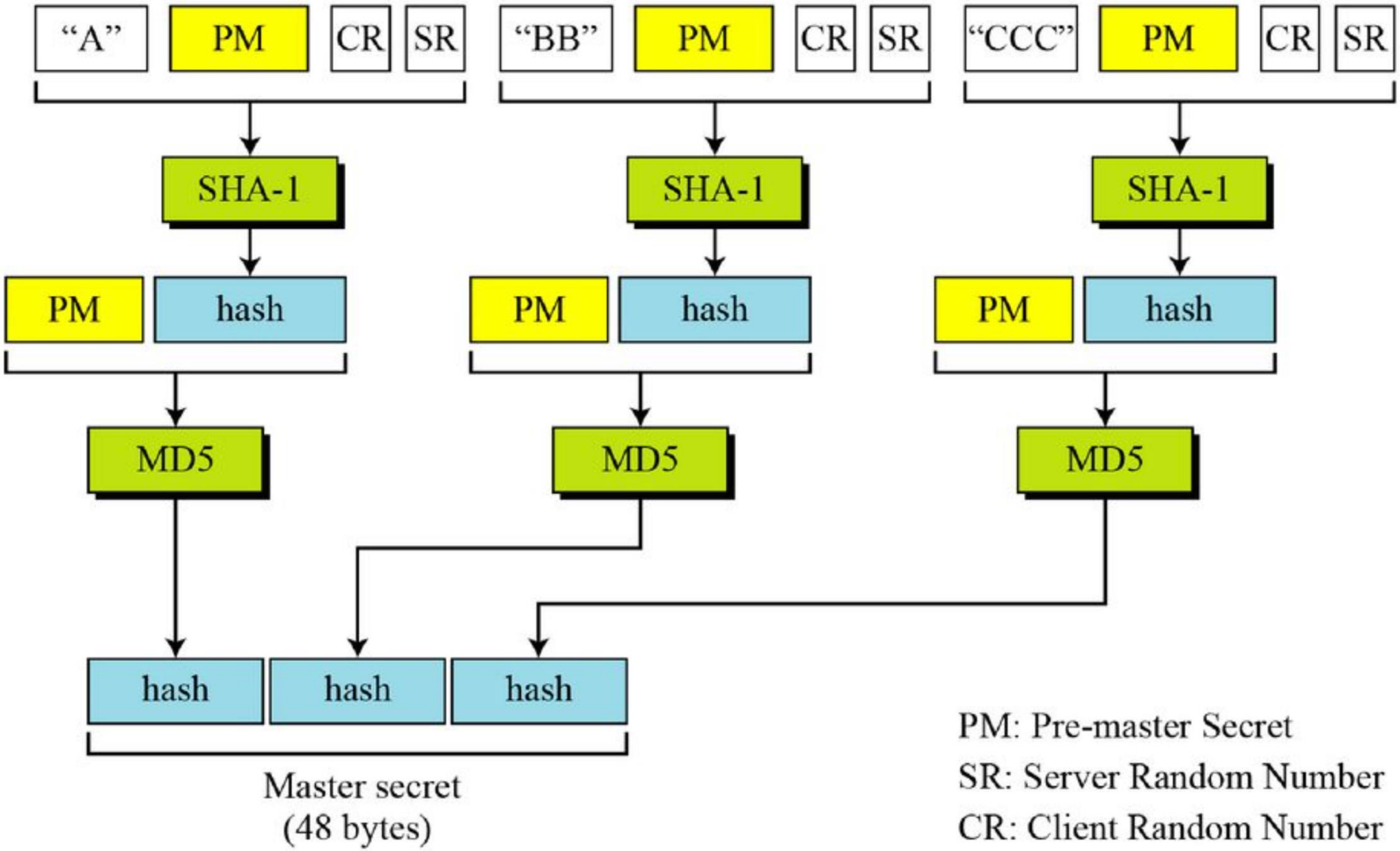
Server's
private key

Pre-master secret

Calculation of Master Secret



Calculation of Master Secret

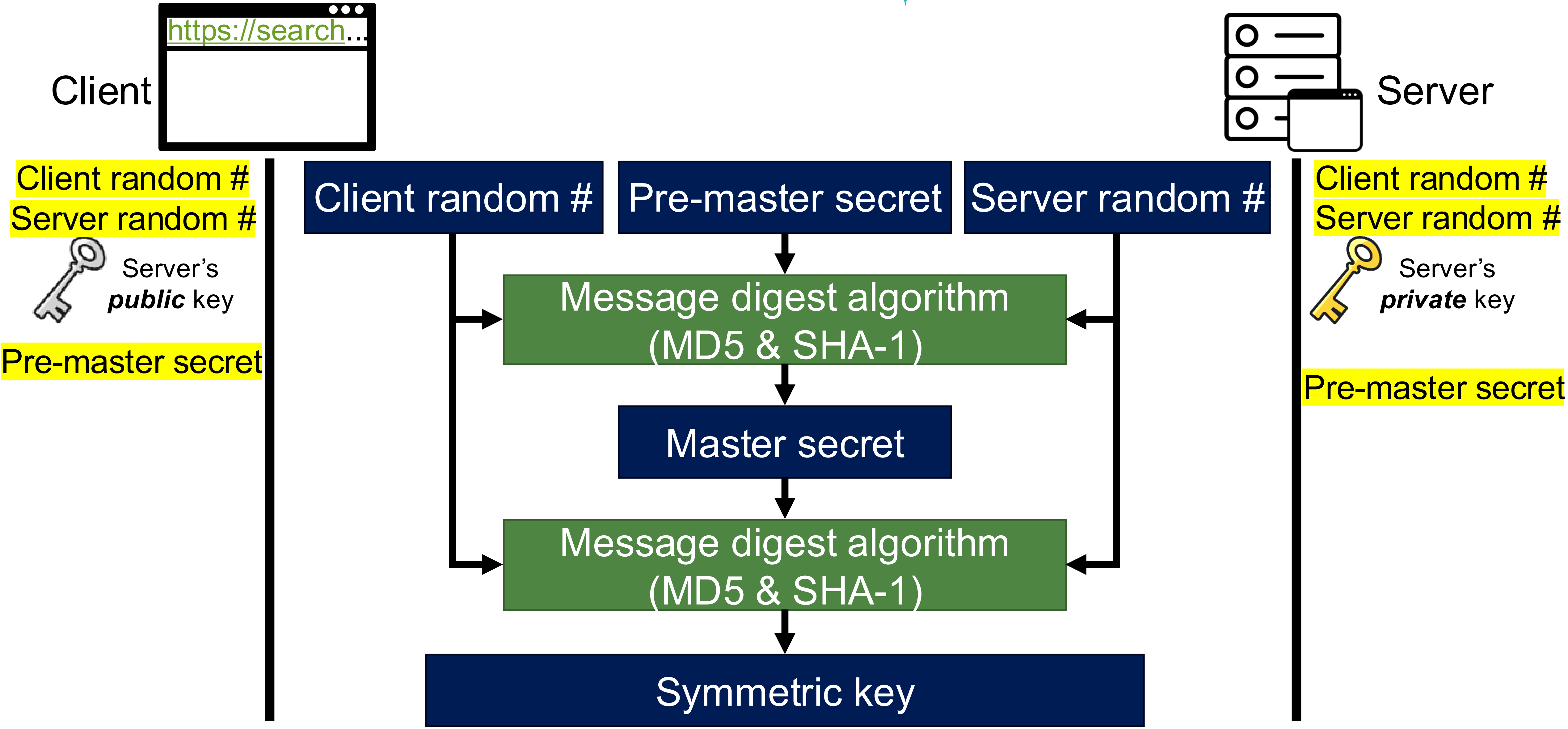


PM: Pre-master Secret

SR: Server Random Number

CR: Client Random Number

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

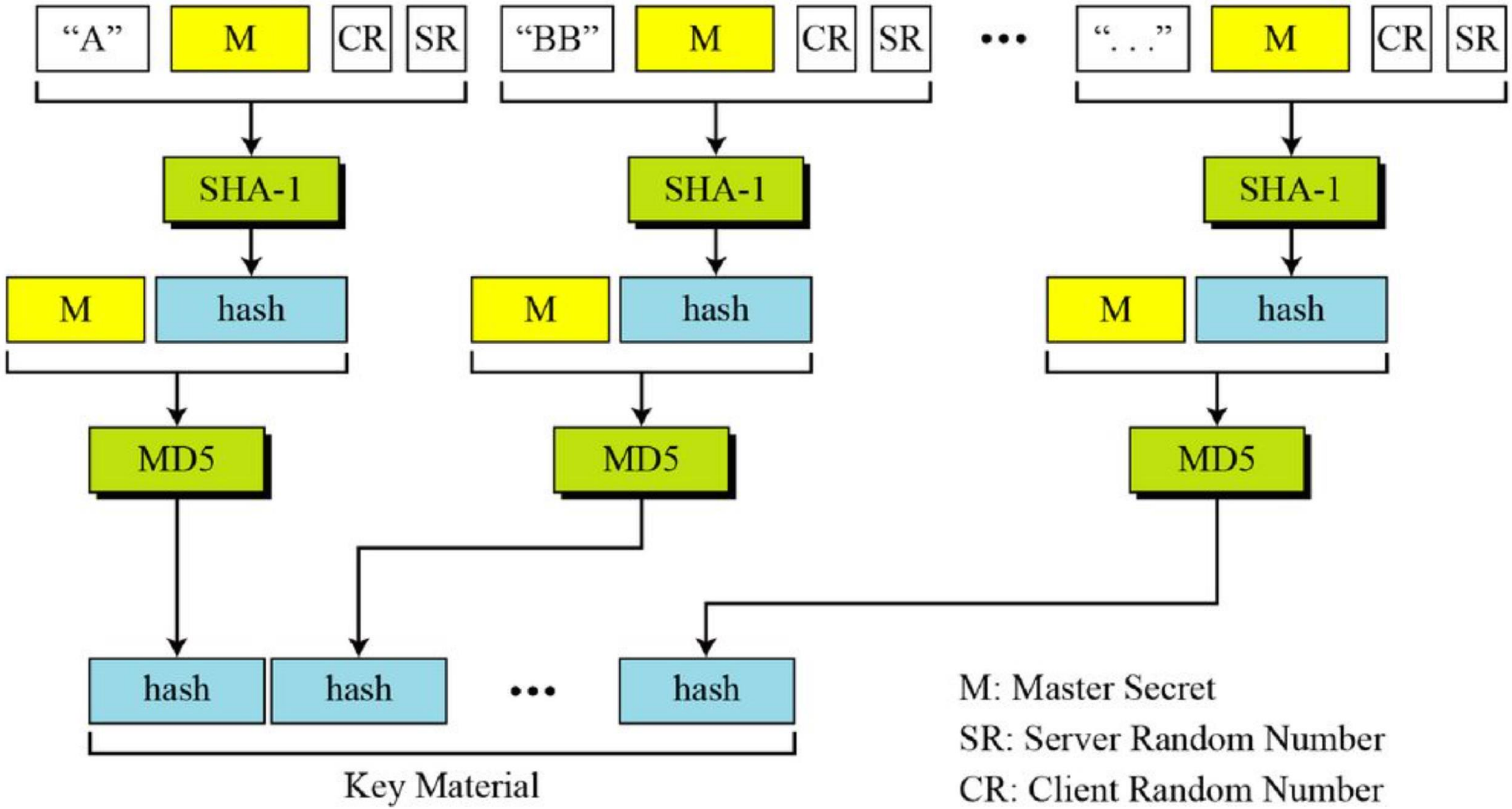
Message digest algorithm (MD5 & SHA-1)

Symmetric key

Server

Client random #
Server random #
Server's **private** key
Pre-master secret

Calculation of Symmetric Key

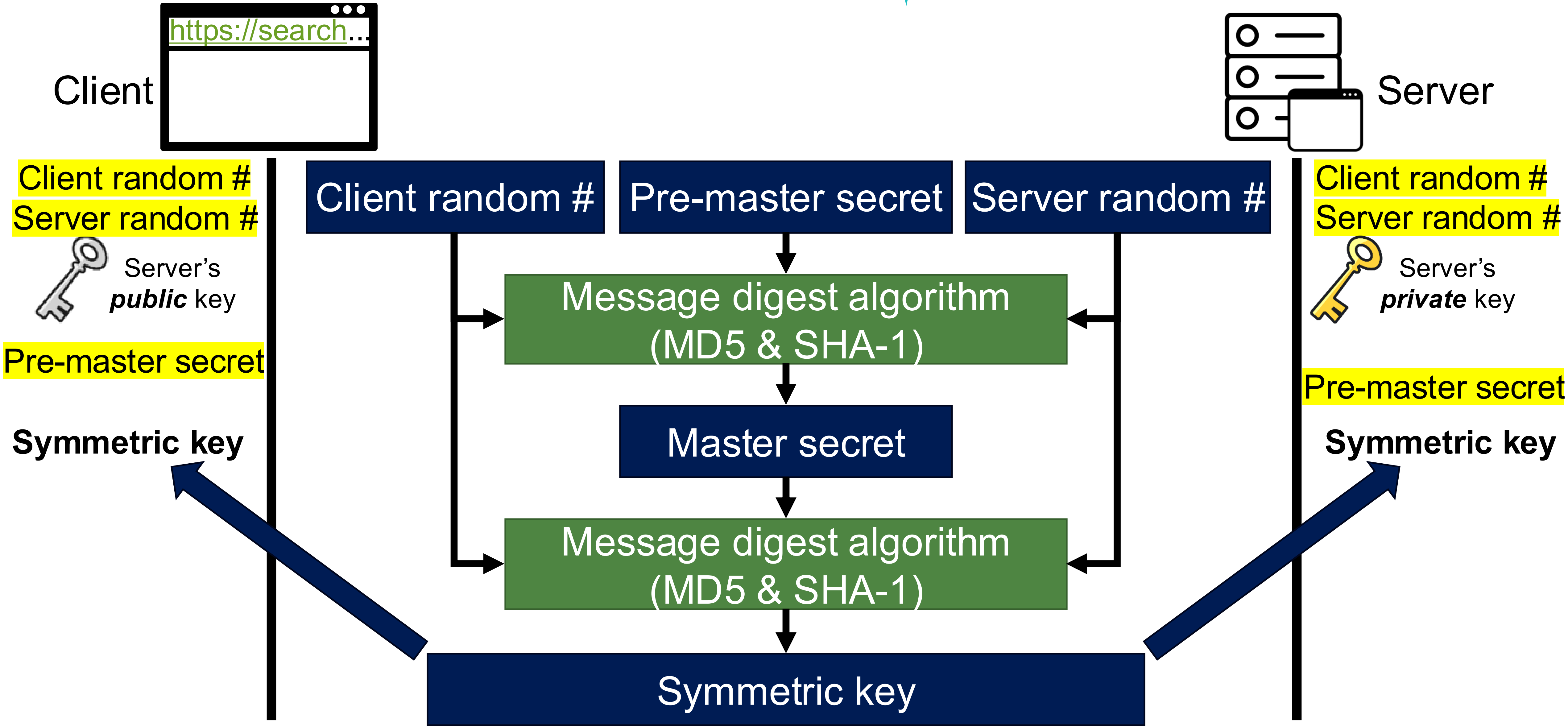


M: Master Secret

SR: Server Random Number

CR: Client Random Number

Calculation of Symmetric Key



Client

https://search...

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

Server

Client random #

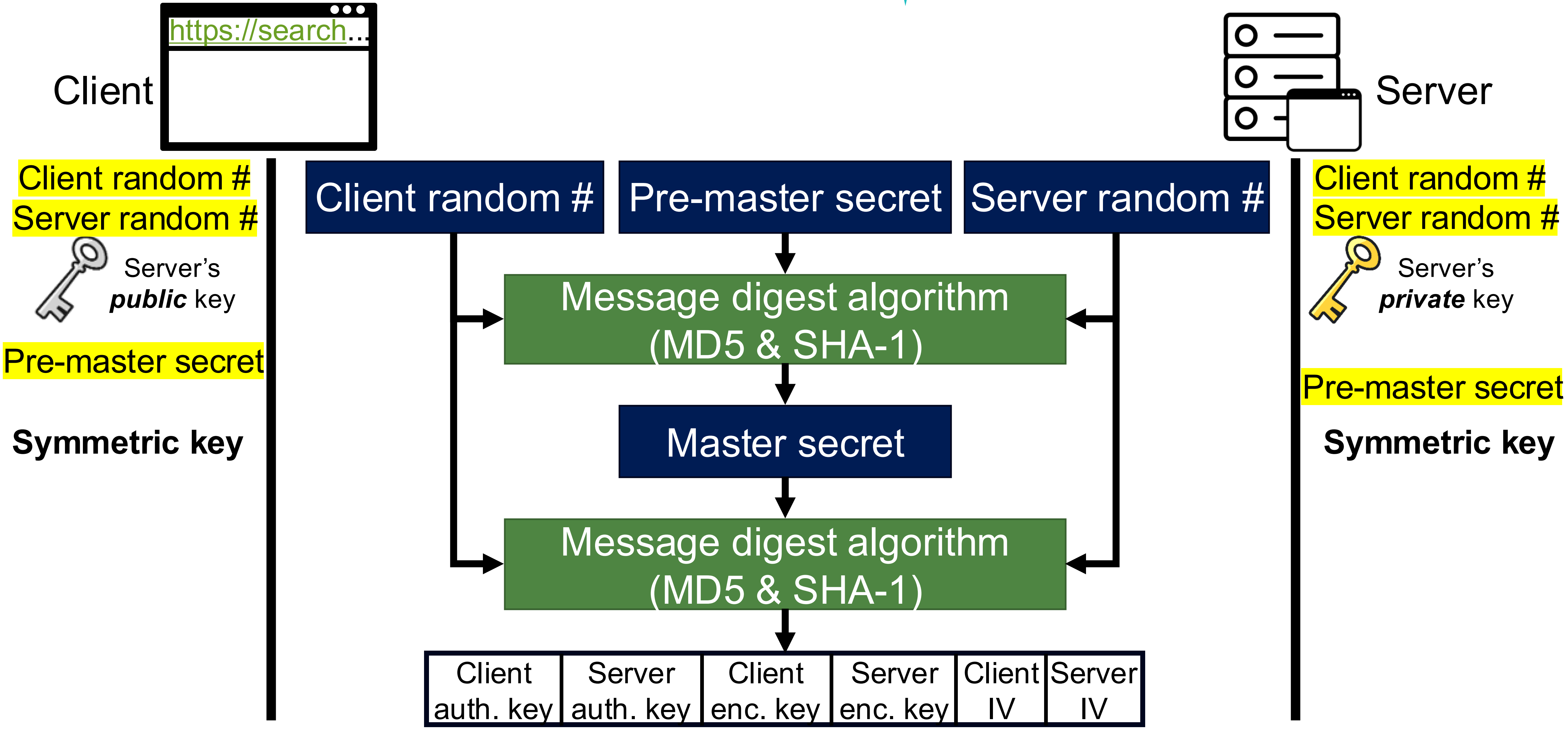
Server random #

Server's *private* key

Pre-master secret

Symmetric key

Calculation of Symmetric Key



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)

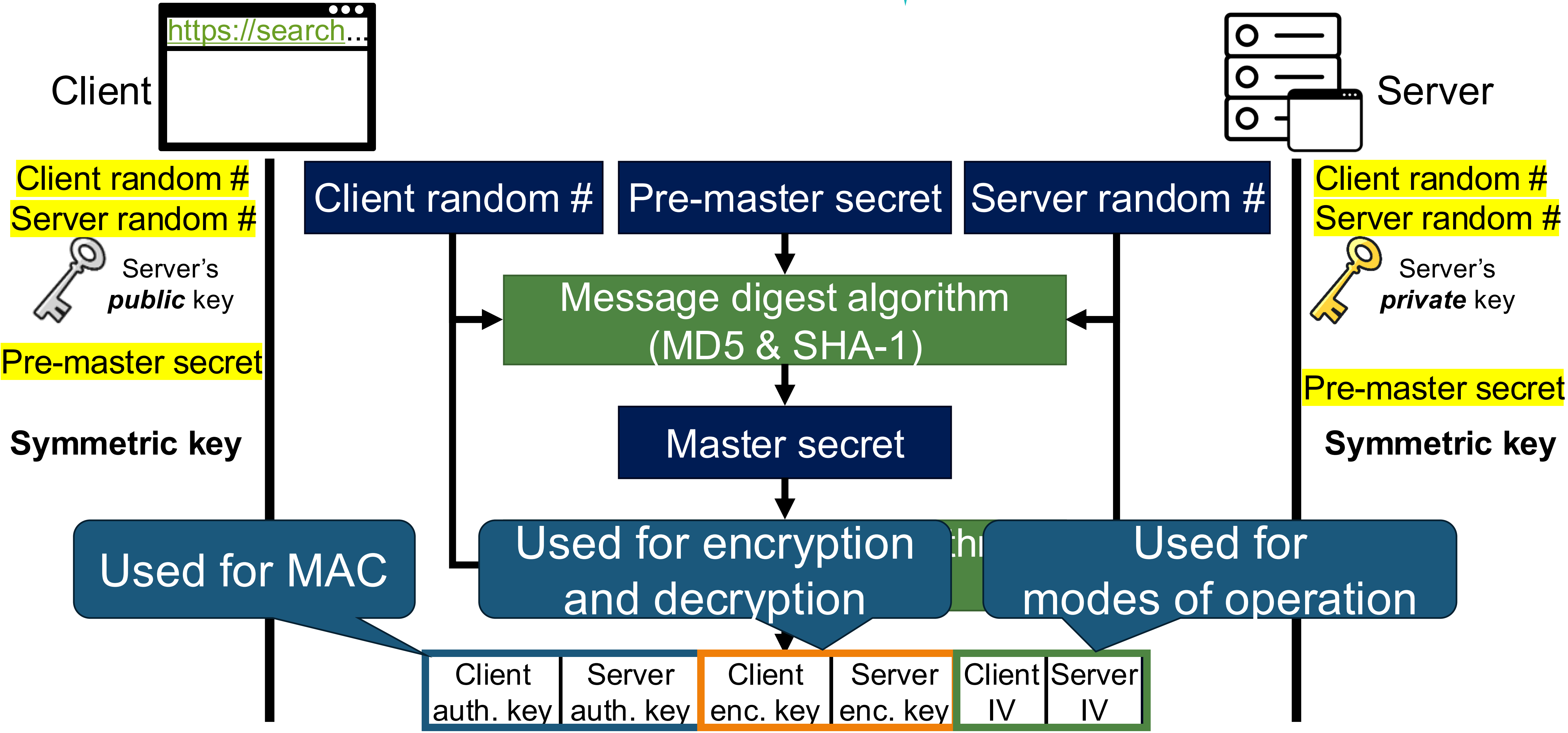
Client auth. key	Server auth. key	Client enc. key	Server enc. key	Client IV	Server IV

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

Pre-master secret

Symmetric key

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

thr

Client random #

Server random #

Server's *private* key

Pre-master secret

Symmetric key

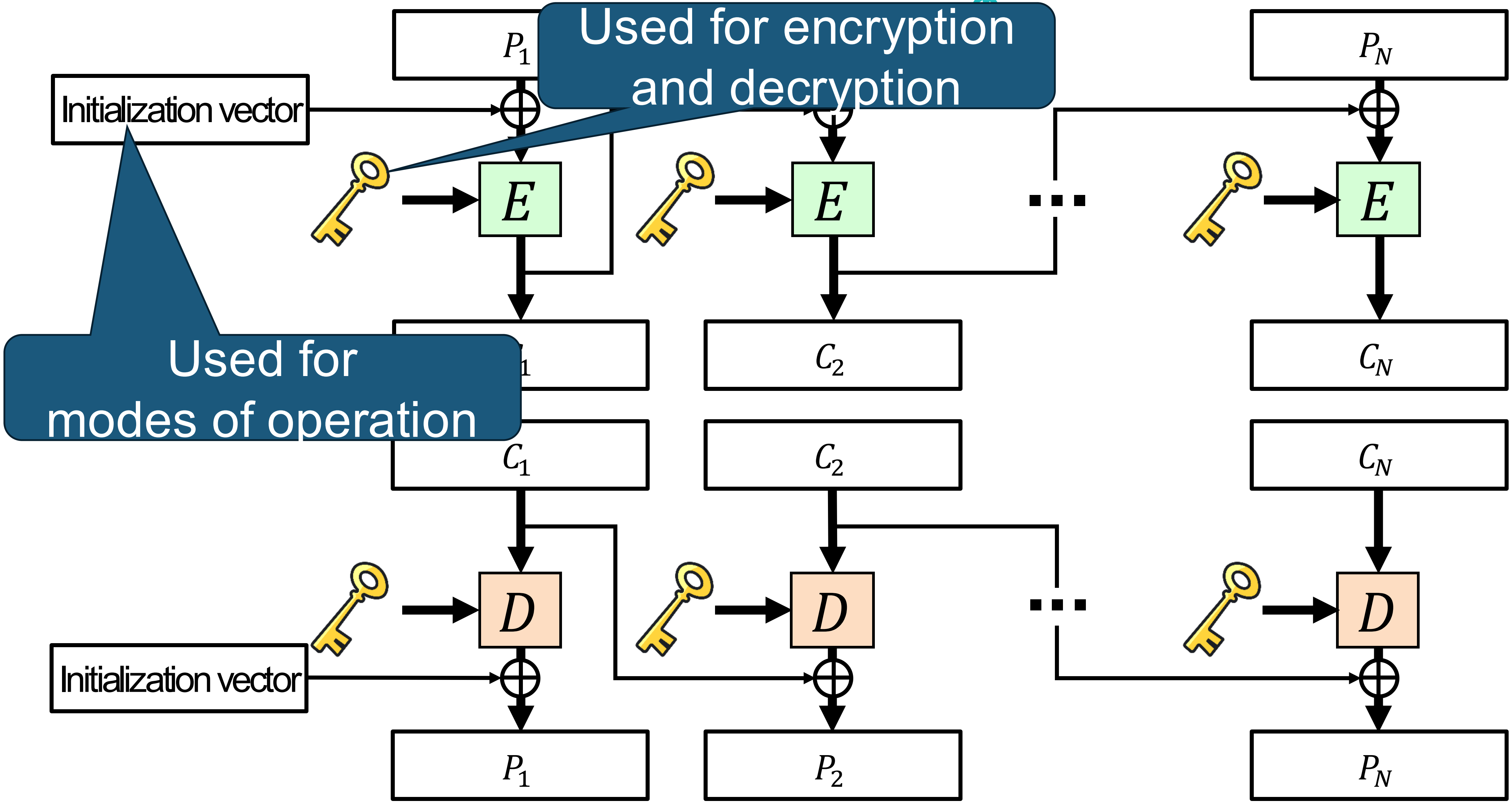
Used for MAC

Used for encryption and decryption

Used for modes of operation

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

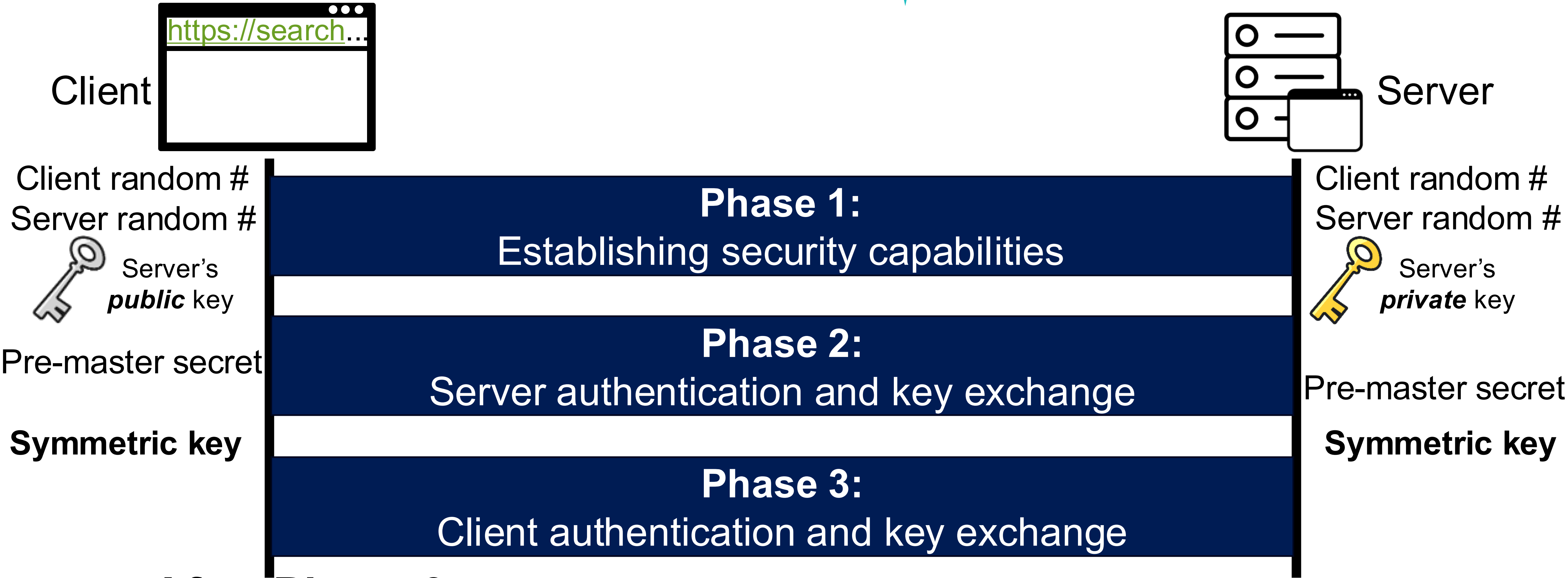
Recap: Cipher Block Chaining (CBC)



Used for encryption and decryption

Used for modes of operation

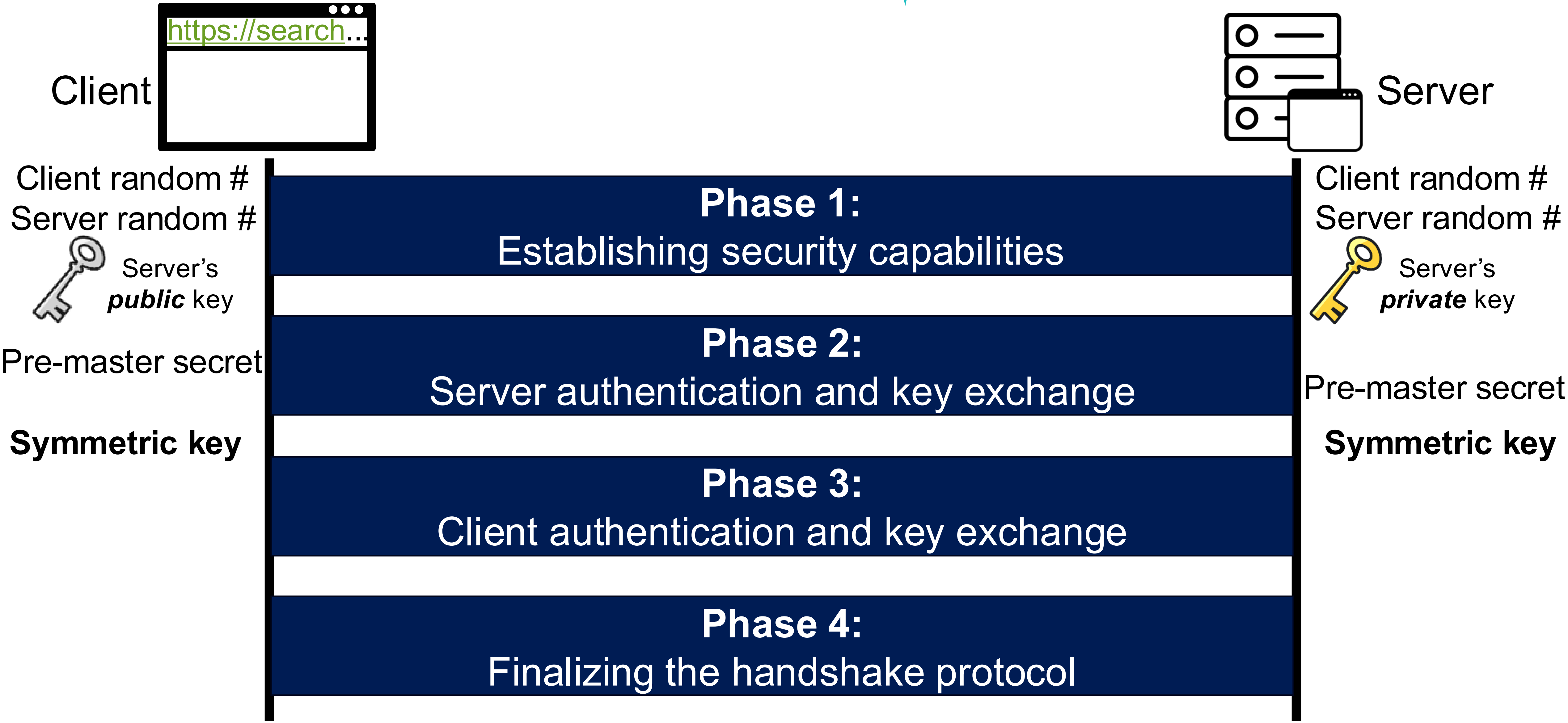
Phase 3: Client Auth. and Key Exchange



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



Client

https://search...

Server

Client random #
Server random #

Server's
public key

Pre-master secret

Symmetric key

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

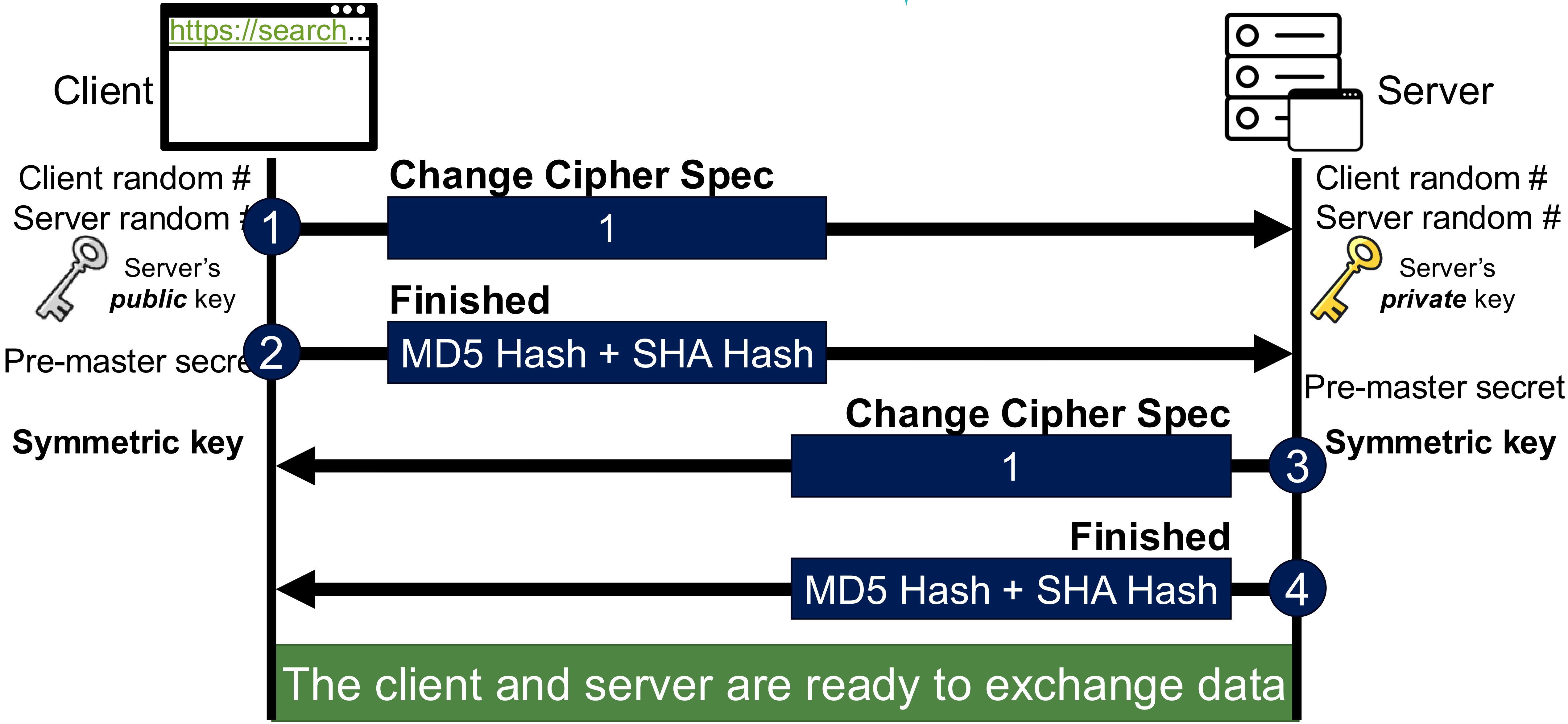
Client random #
Server random #

Server's
private key

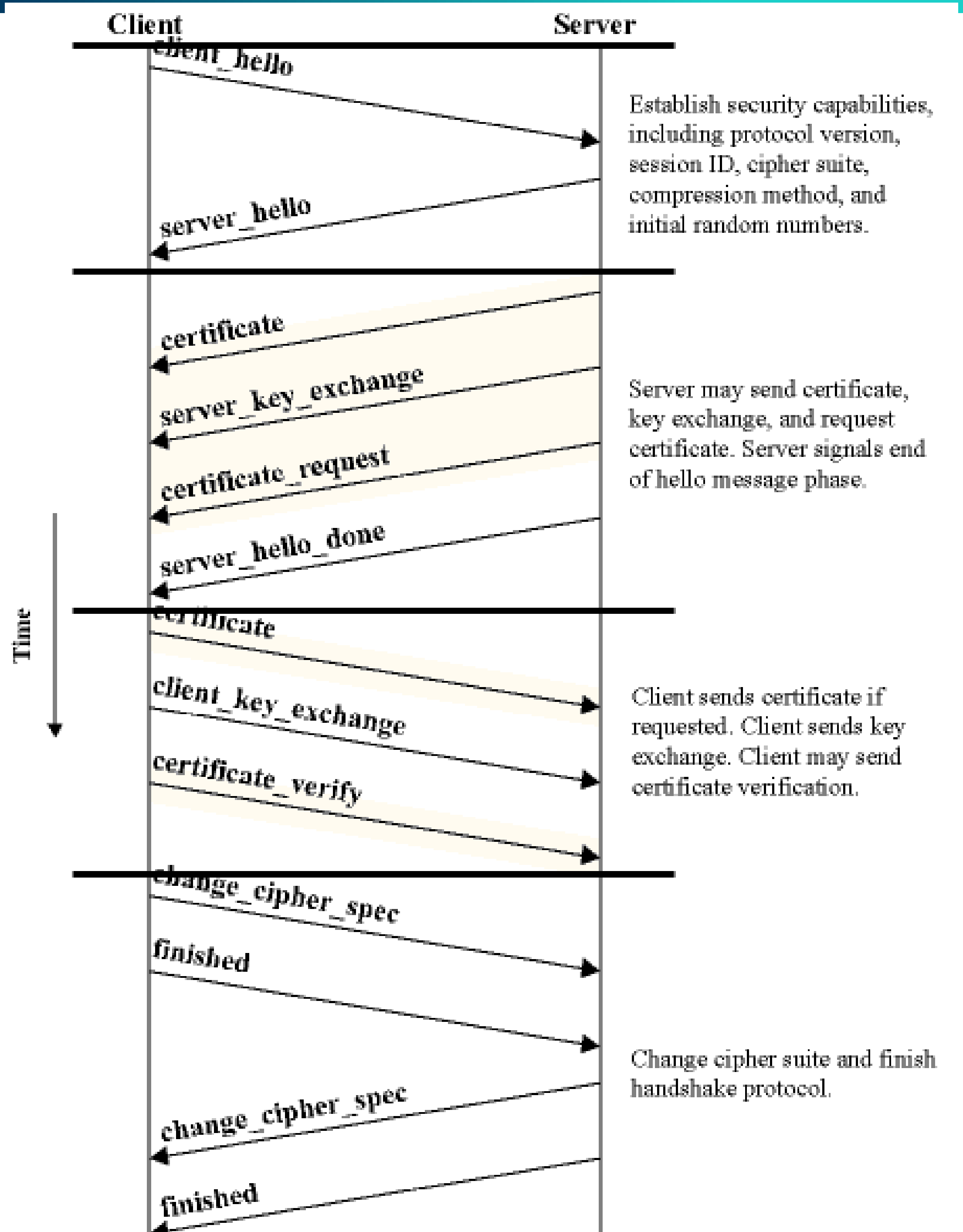
Pre-master secret

Symmetric key

Phase 4: Finalizing the Handshake Protocol

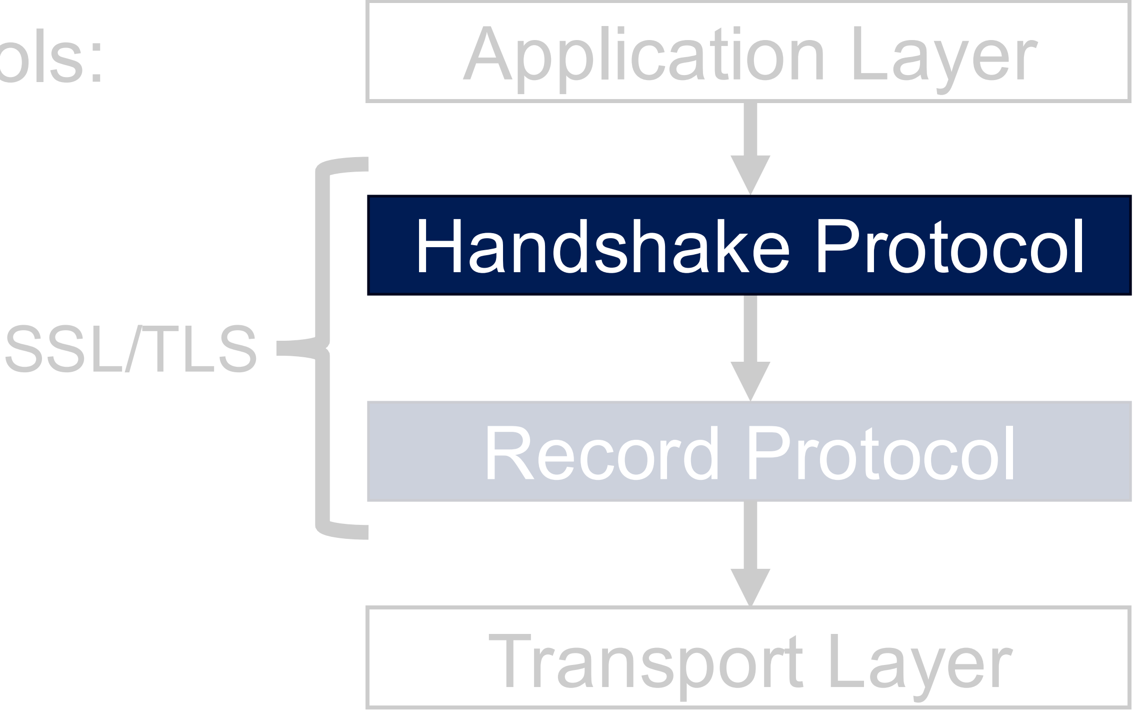


Handshake Protocol Summary


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.

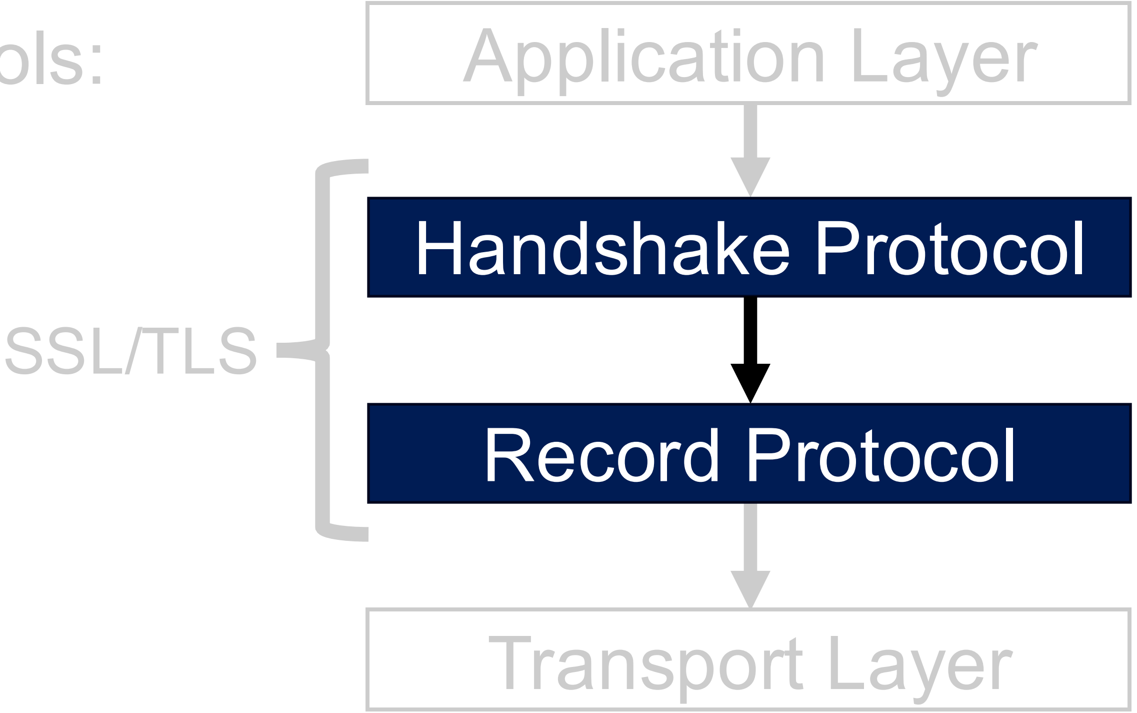
SSL/TLS Basics

- 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/TLS Record Protocol

- Uses the symmetric keys established in the handshake protocol to protect **confidentiality**, **integrity**, and **authenticity** of data exchange

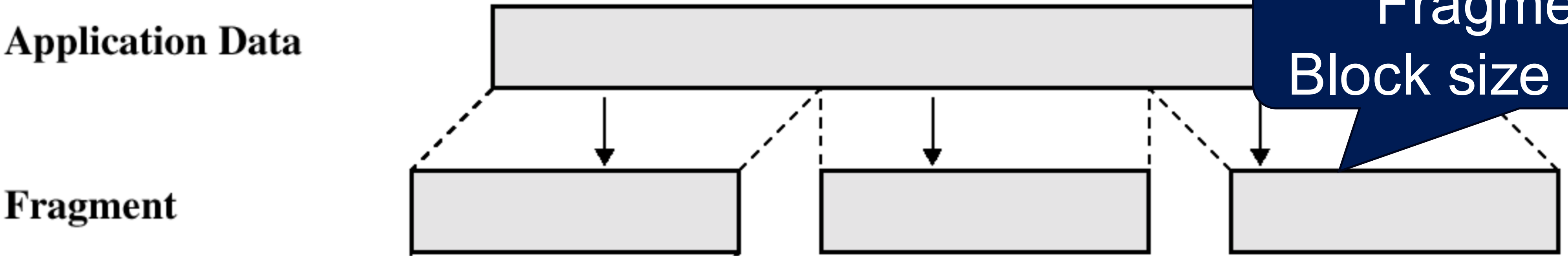
- **Confidentiality**
  - Using symmetric encryption

- **Integrity (+ Authenticity)**
  - Using a MAC with shared secret key

SSL Record Protocol Operation

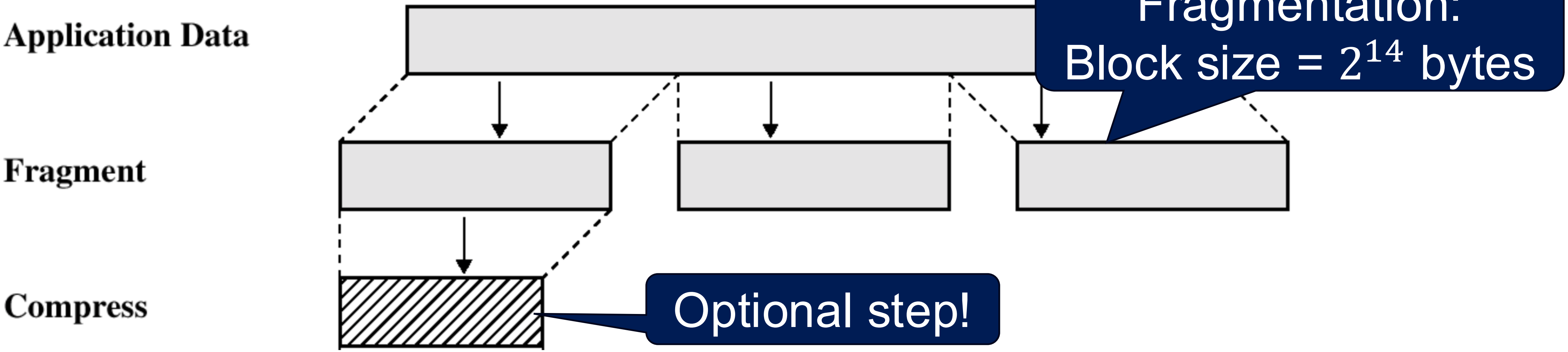
Application Data

SSL Record Protocol Operation



Fragmentation:
Block size = 2^{14} bytes

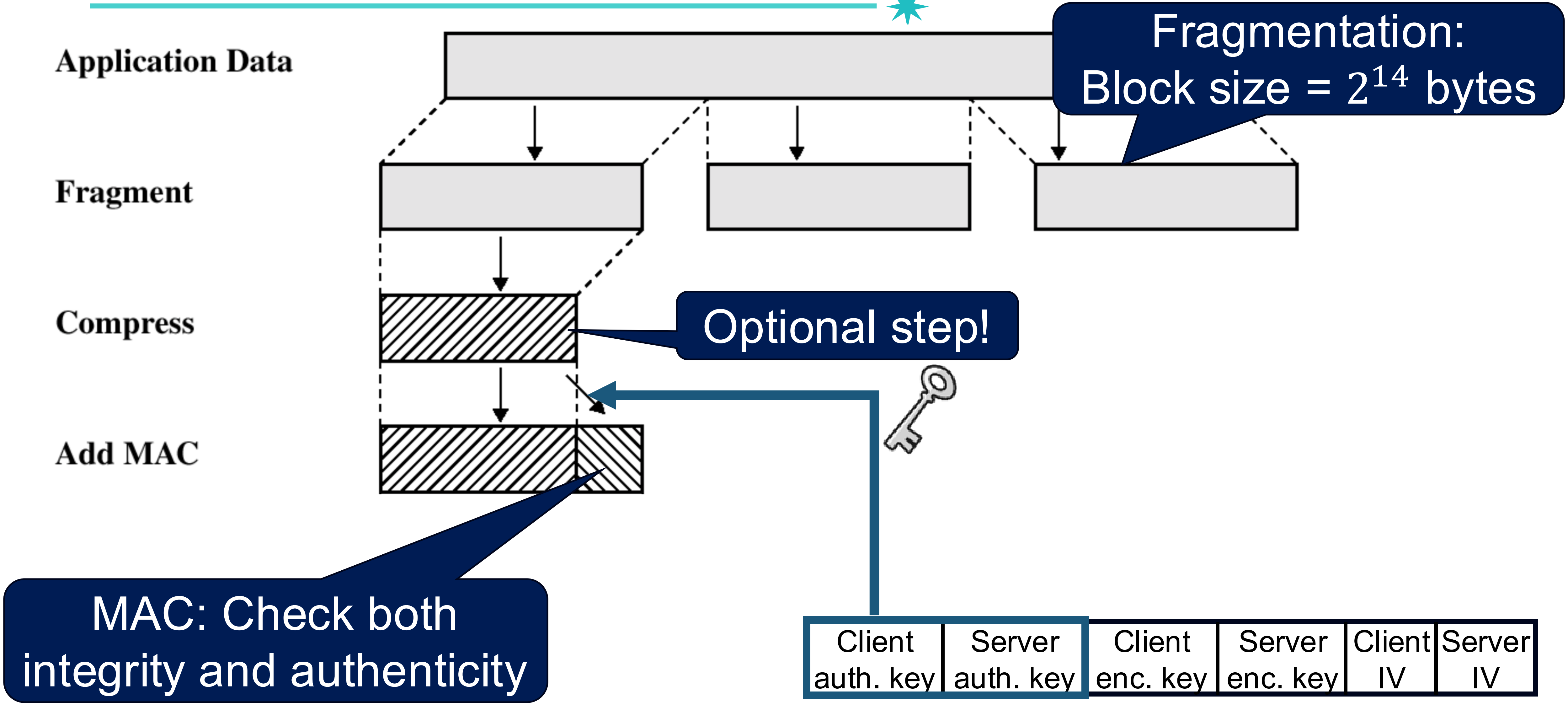
SSL Record Protocol Operation



Fragmentation:
Block size = 2^{14} bytes

Optional step!

SSL Record Protocol Operation



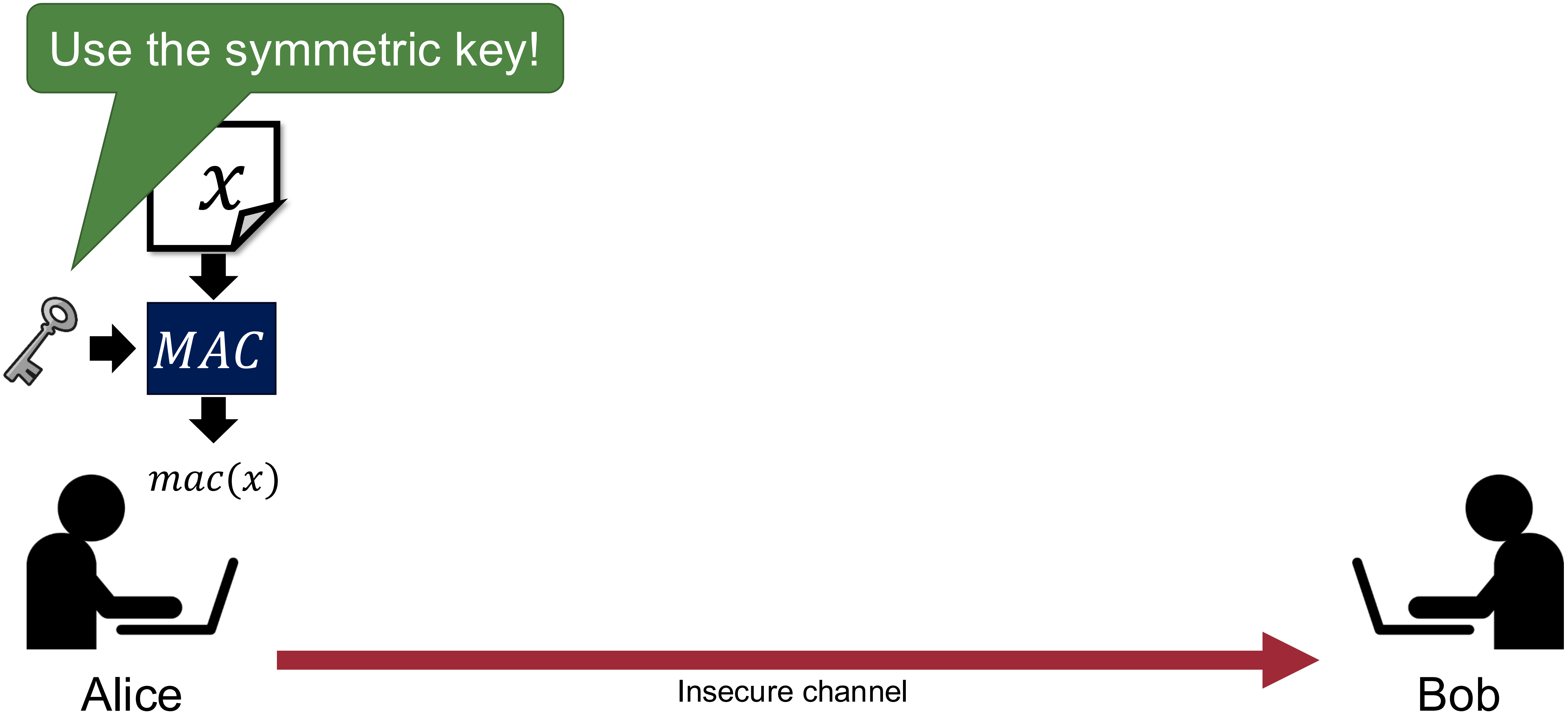
Fragmentation: Block size = 2^{14} bytes

Optional step!

MAC: Check both integrity and authenticity

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

Recap: Message Authentication Codes (MAC)



Use the symmetric key!

x

MAC

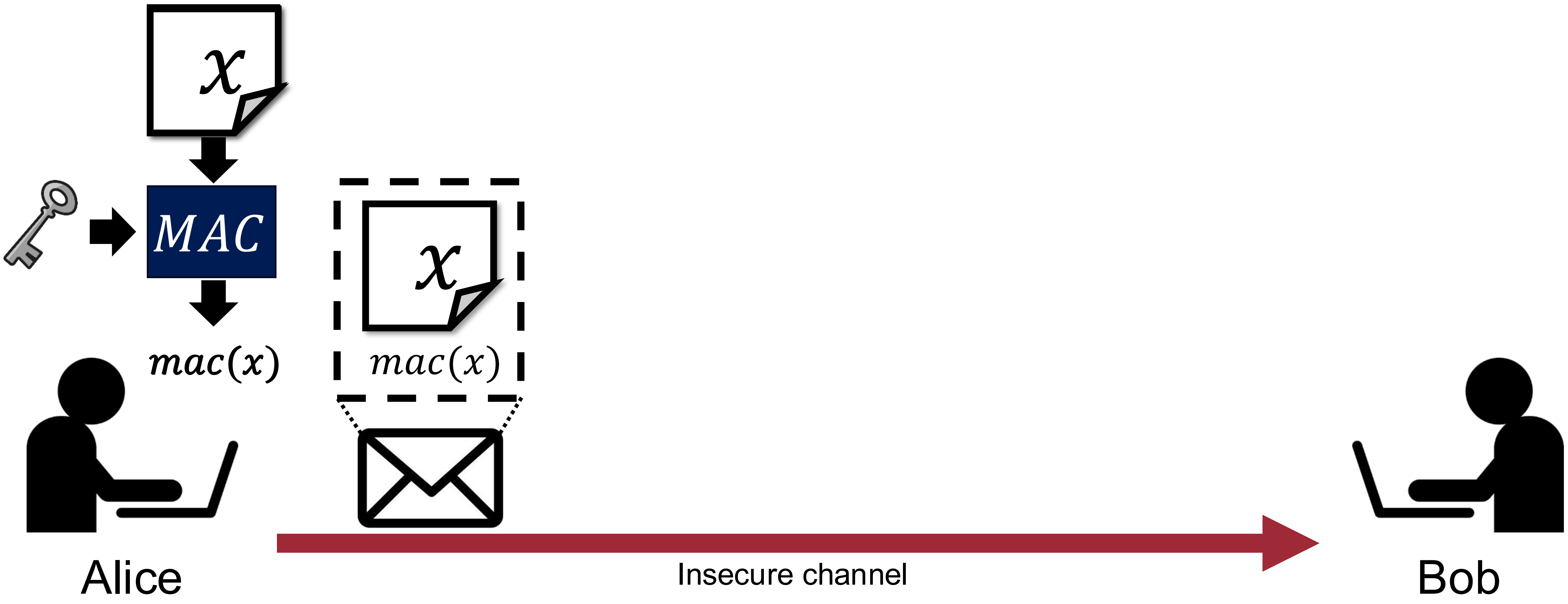
$mac(x)$

Alice

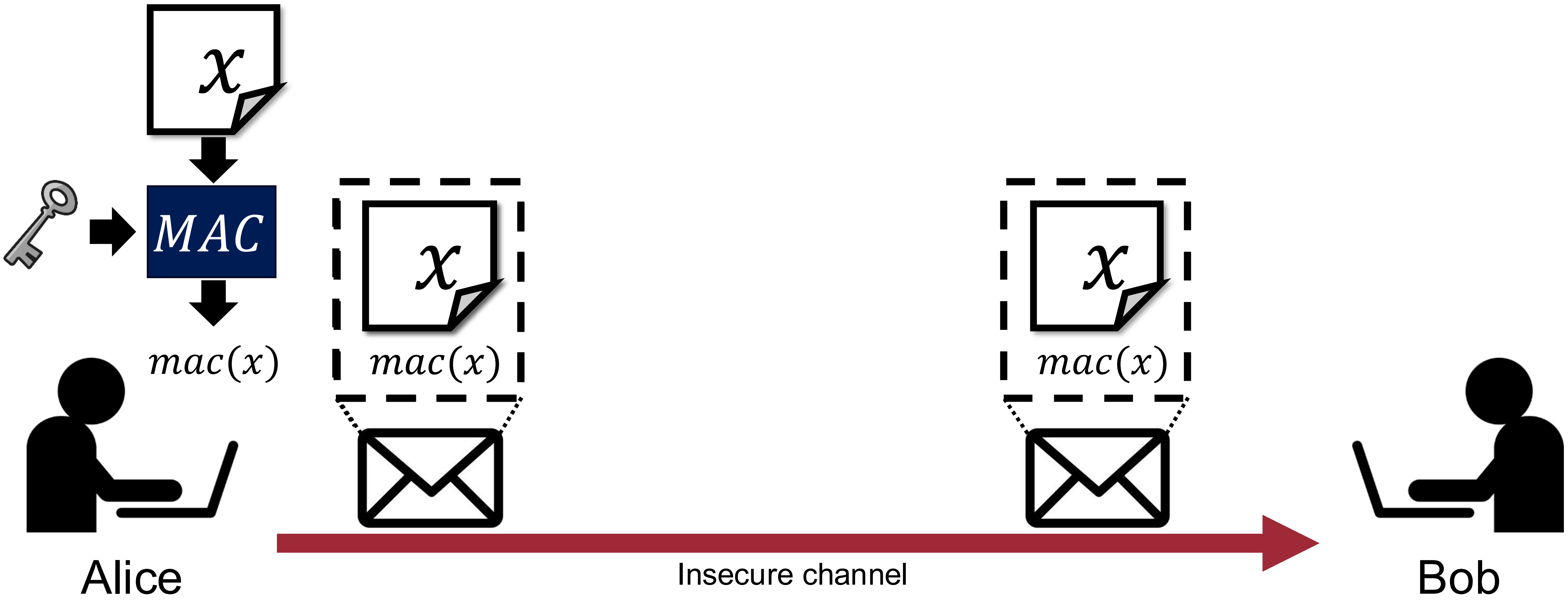
Insecure channel

Bob

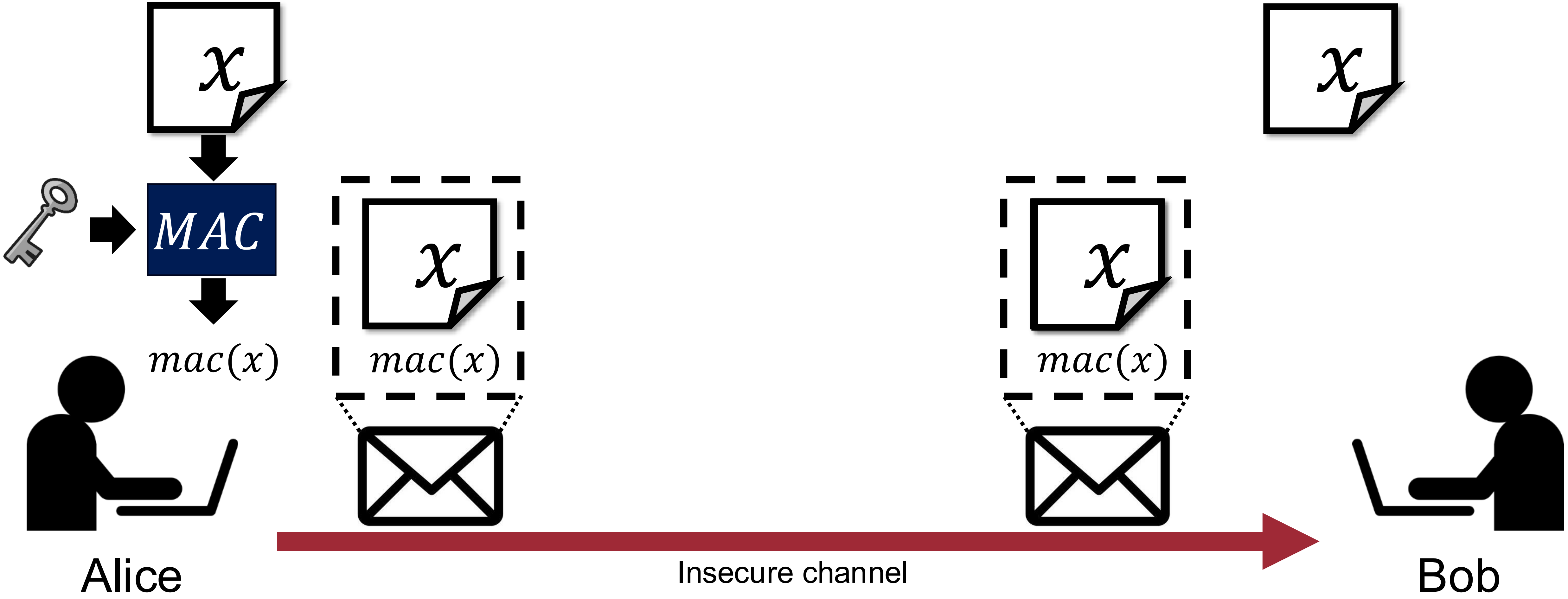
Recap: Message Authentication Codes (MAC)



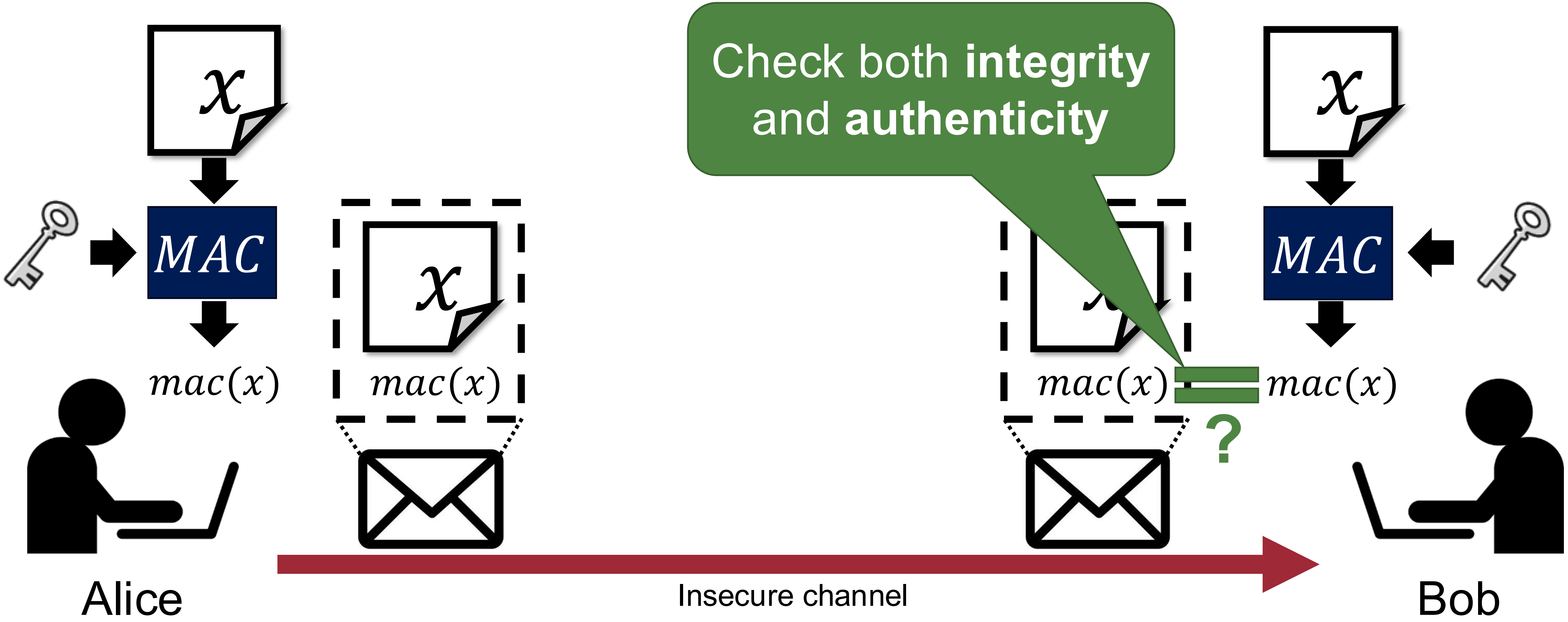
Recap: Message Authentication Codes (MAC)



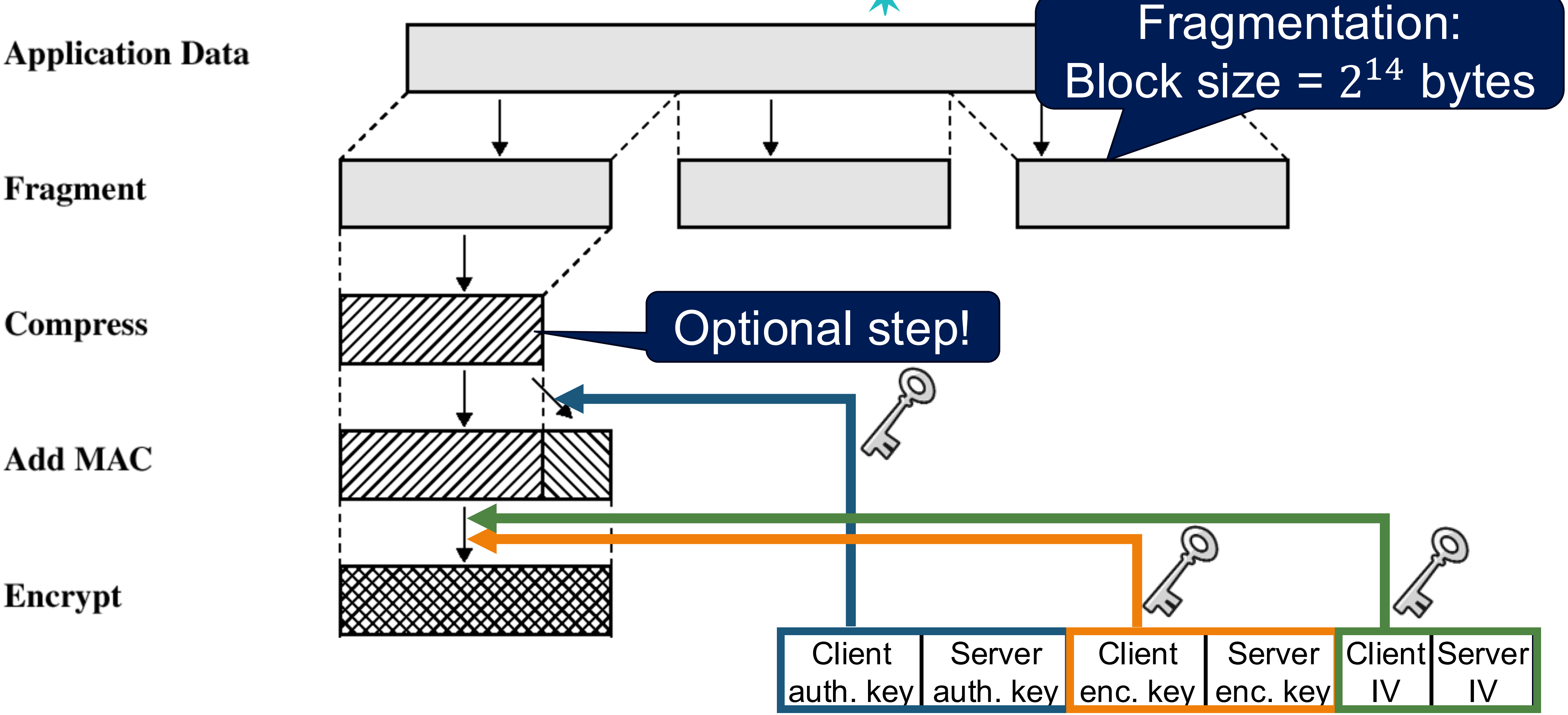
Recap: Message Authentication Codes (MAC)



Recap: Message Authentication Codes (MAC)



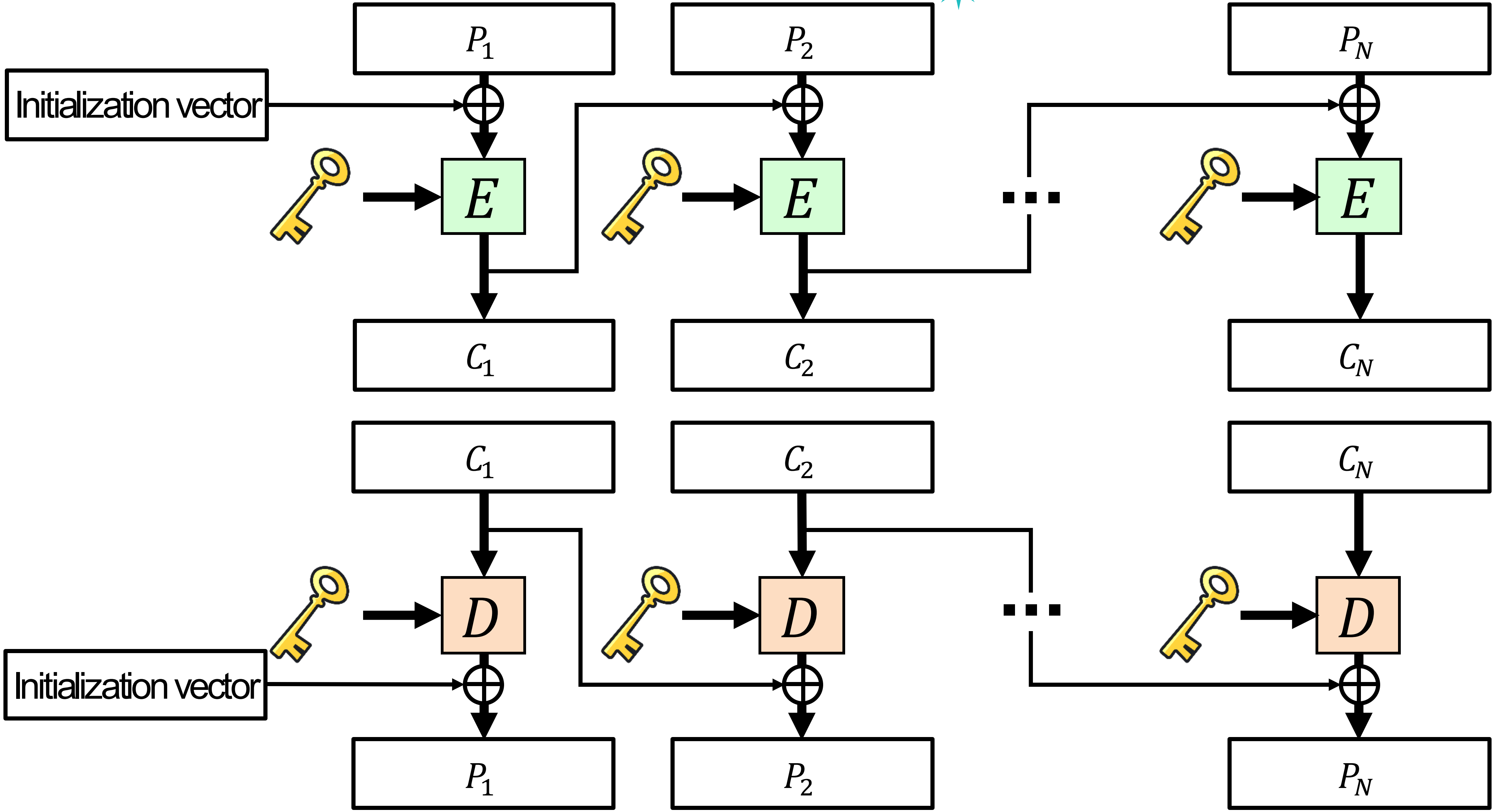
SSL Record Protocol Operation



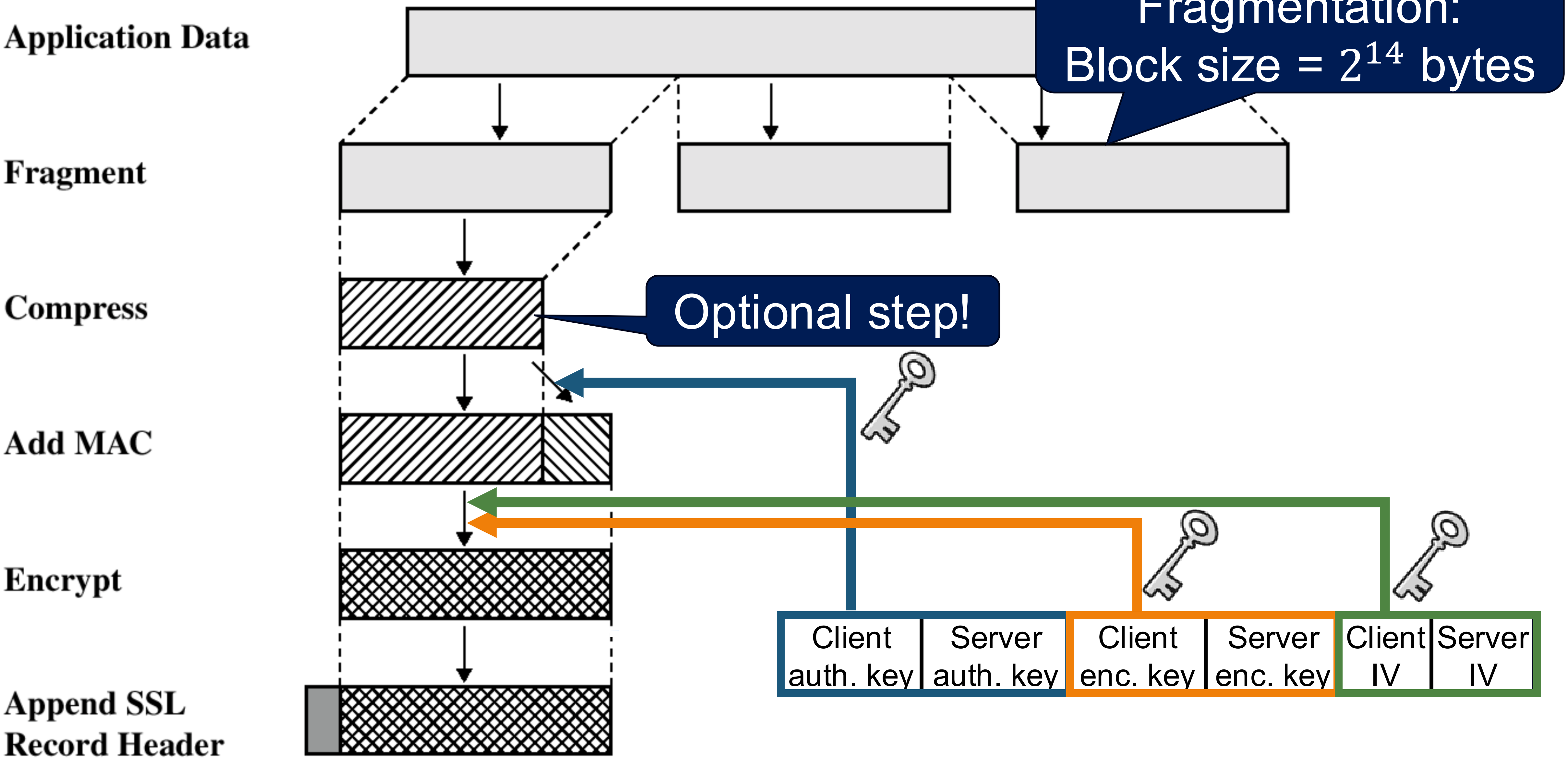
Fragmentation: Block size = 2^{14} bytes

Optional step!

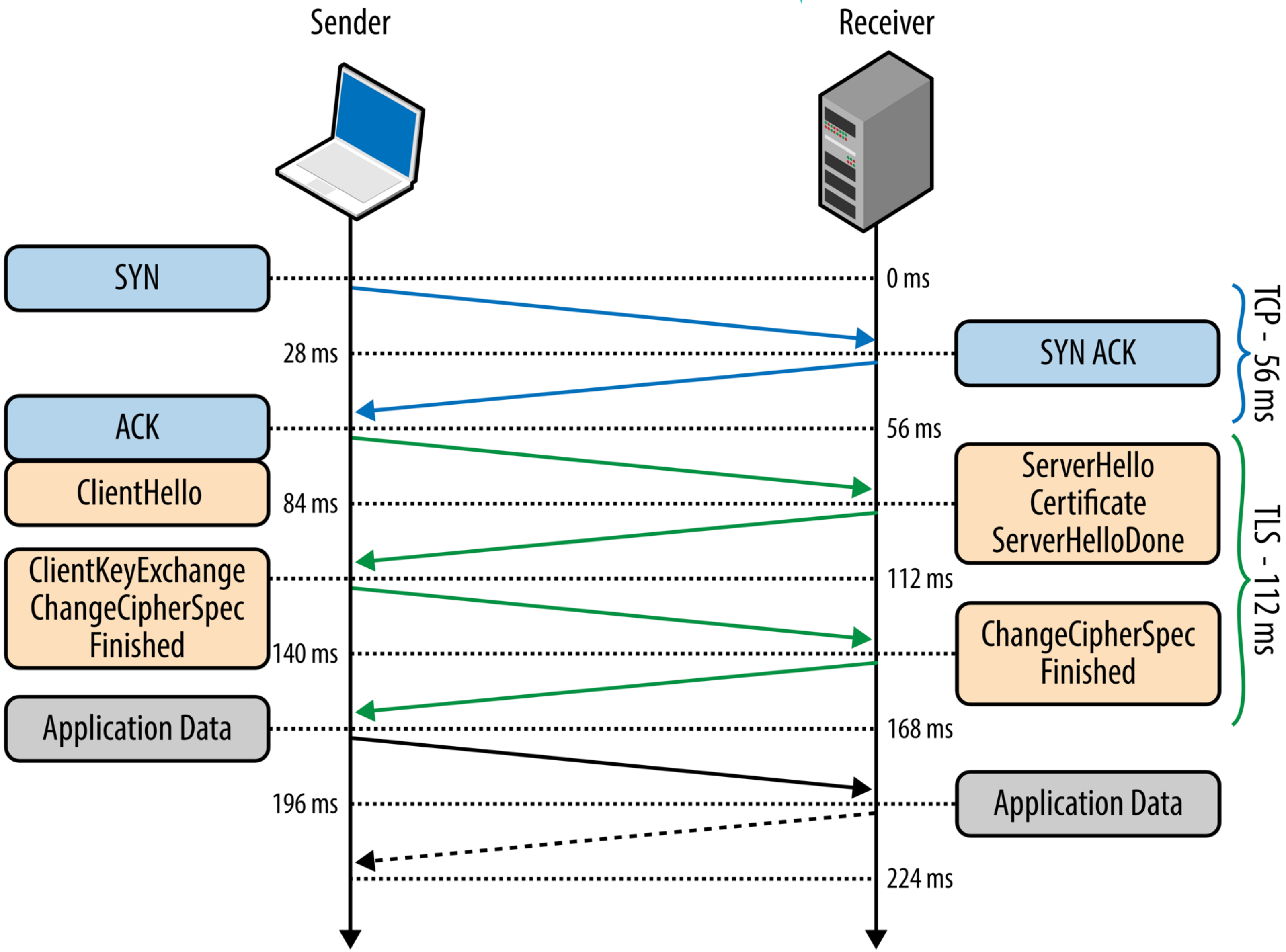
Recap: Cipher Block Chaining (CBC)



SSL Record Protocol Operation



SSL/TLS Final Overview



How SSL/TLS Provides Security Properties?

- Security goals: achieving…
  - **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 (and MAC)

How SSL/TLS Provides Security Properties?

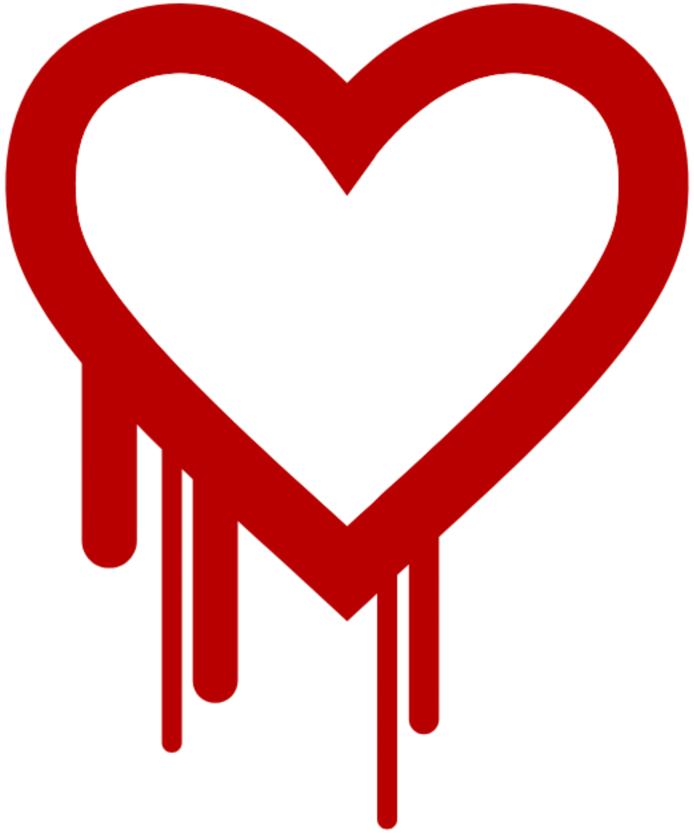
- Security goals: achieving…
  - **Confidentiality**
    - Asymmetric-key algorithm for key exchange (pre-master key)
    - Symmetric-key algorithm for data exchange

Are we safe now?

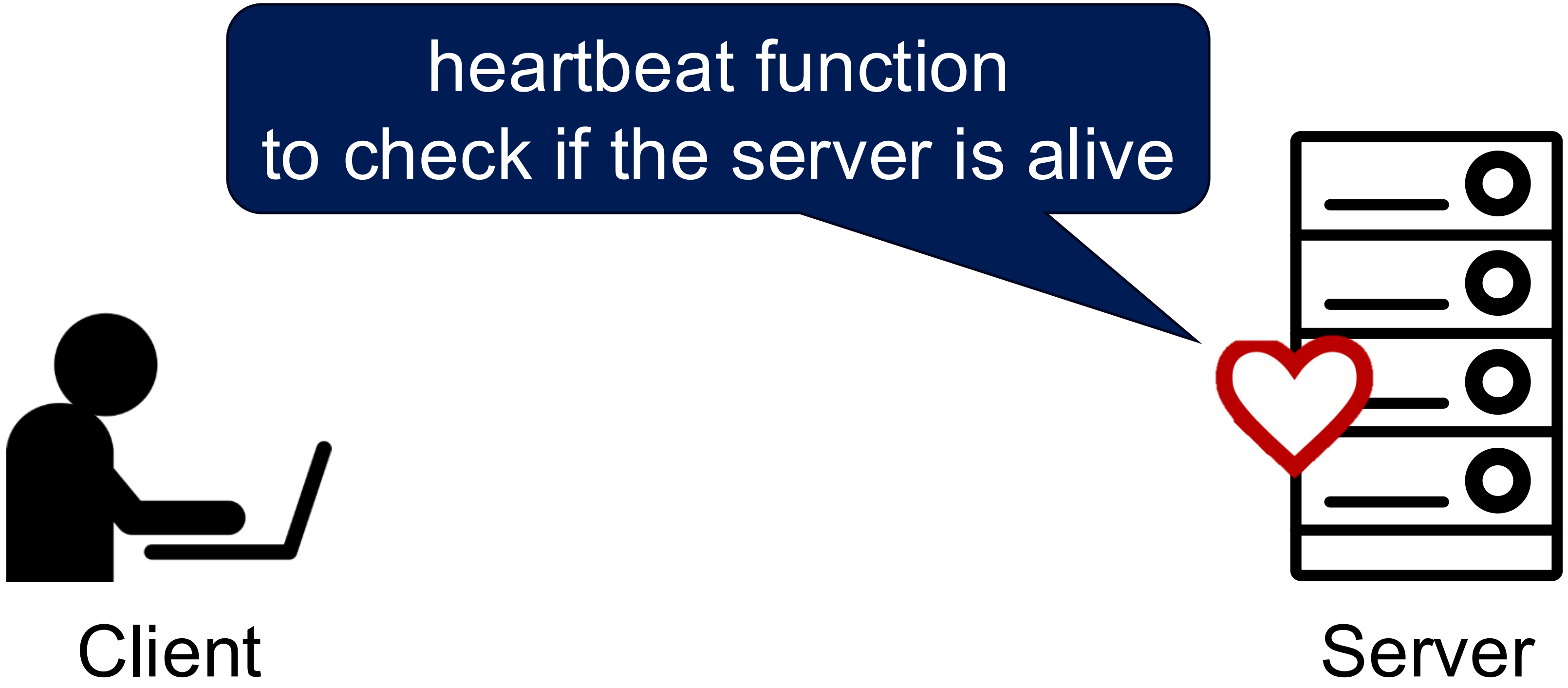
    - MAC (with hash algorithm)
    - If an attacker modifies the message, the recipient can detect the modification
  - **Authentication**
    - Authenticate the identity of the server using the server's certificate (and MAC)

Recap: Heartbleed Bug (in 2014)

- Famous bug in OpenSSL (in TLS ***heartbeat***)
- An attacker can steal private keys

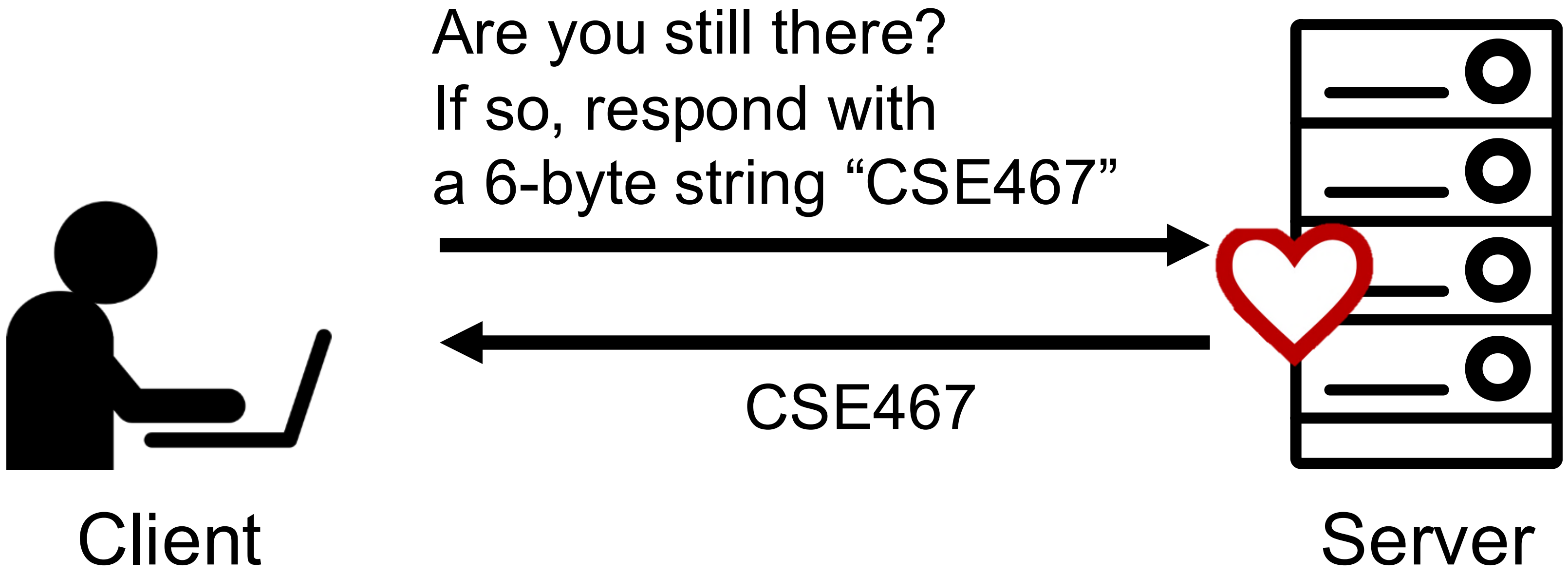


Heartbleed Bug: High-level Workflow

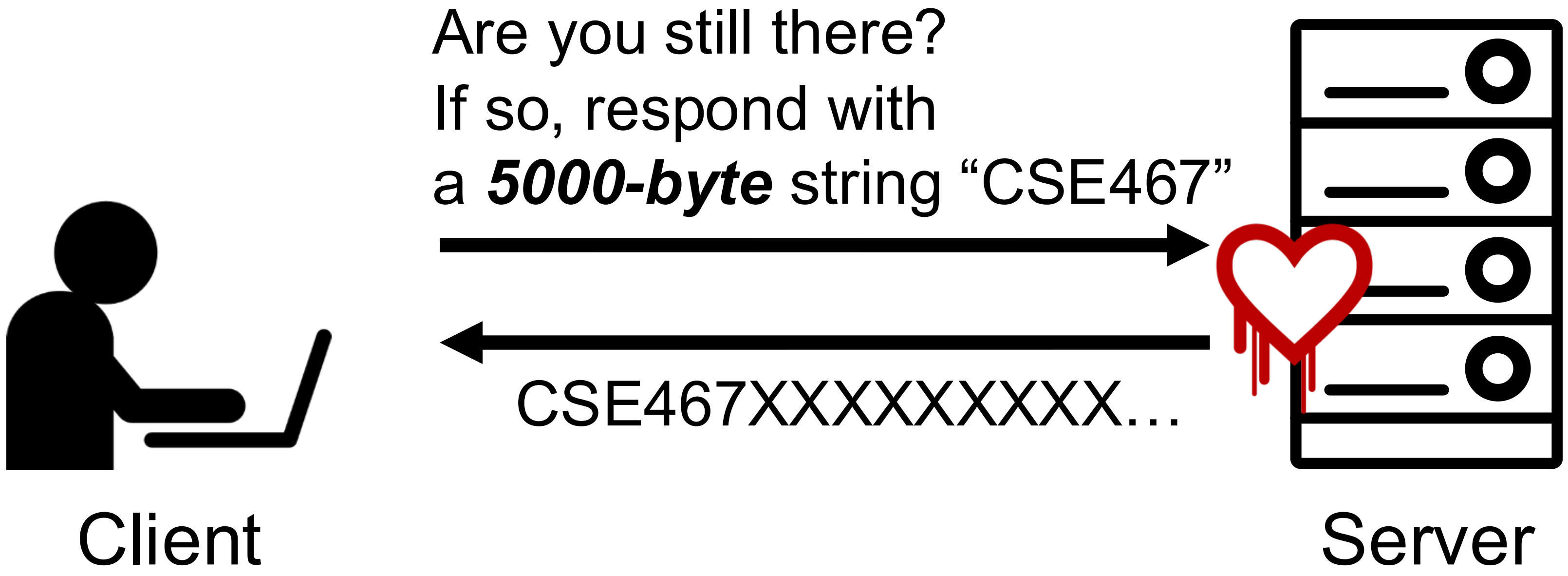


heartbeat function
to check if the server is alive

Heartbleed Bug: High-level Workflow



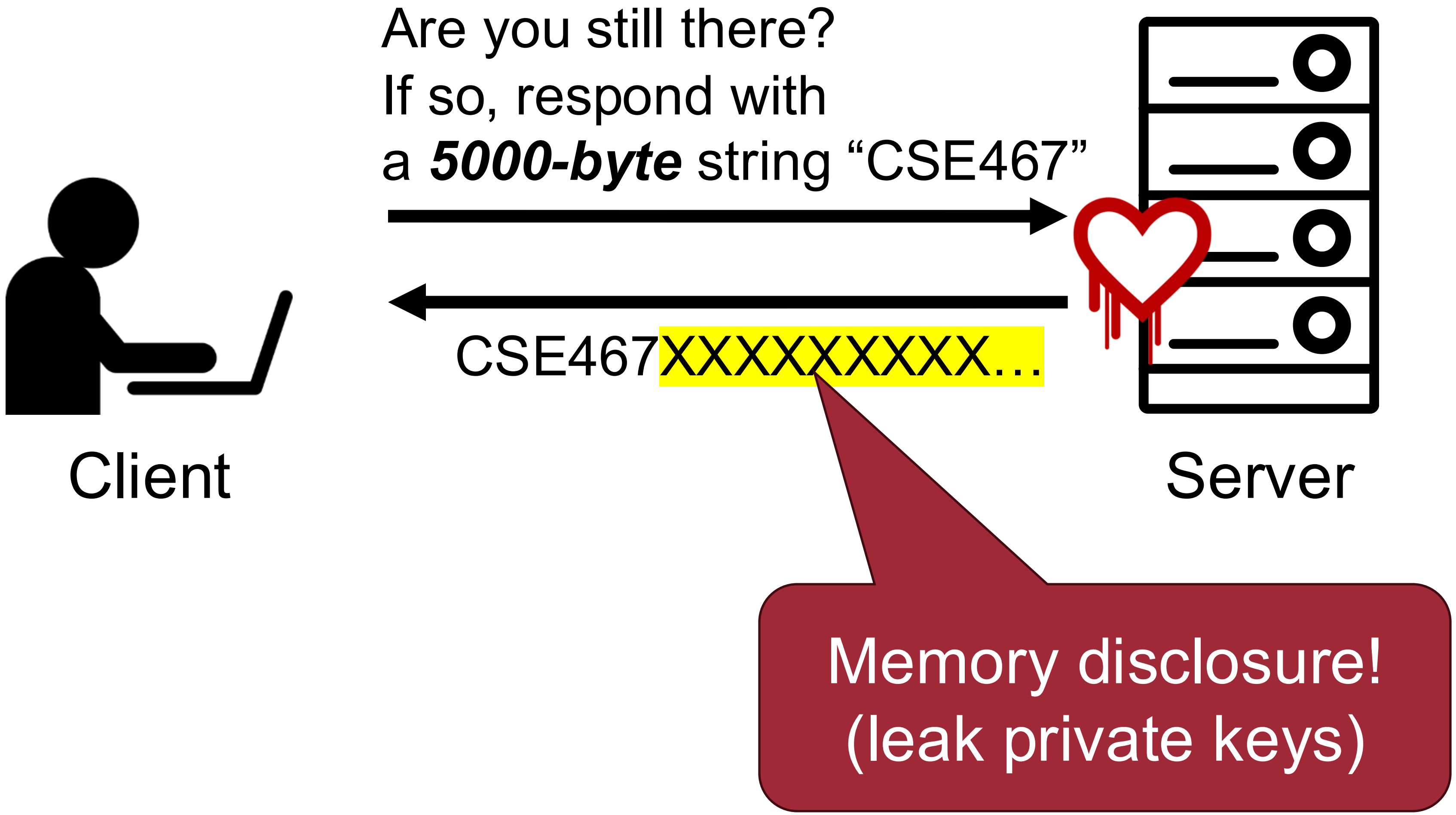
Heartbleed Bug: High-level Workflow



Are you still there?
If so, respond with
a ***5000-byte*** string “CSE467”

CSE467XXXXXXXXXX…

Heartbleed Bug: High-level Workflow



Are you still there?
If so, respond with
a ***5000-byte*** string “CSE467”

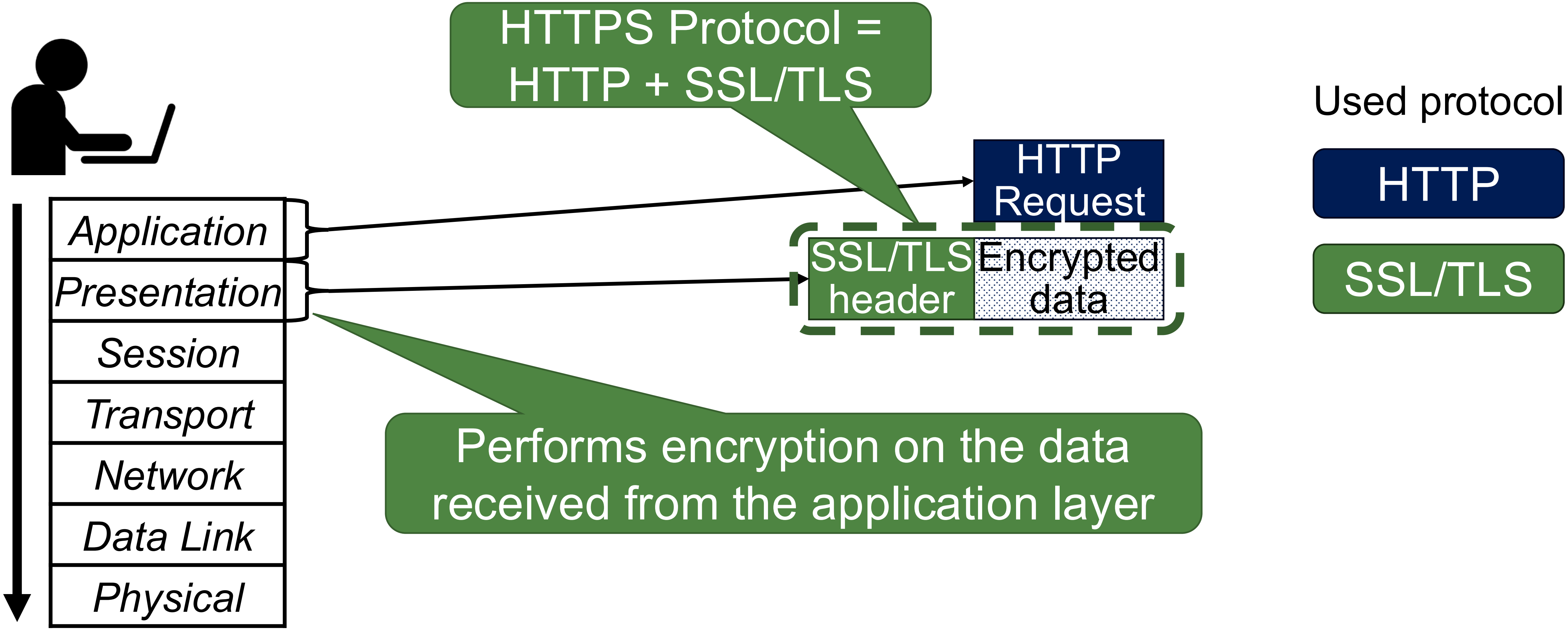
CSE467XXXXXXXXXX…

Memory disclosure!
(leak private keys)

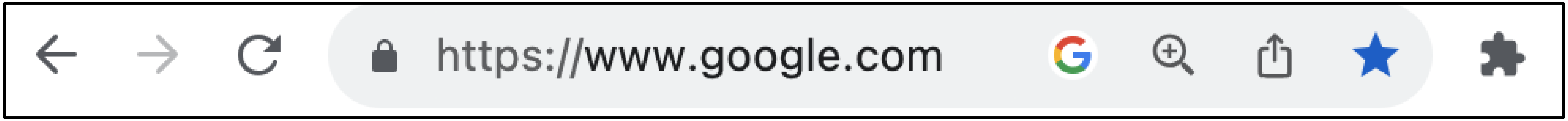
HTTPS

HTTPS

- Adding a protocol layer for secure communication!



HTTPS – The Lock Icon

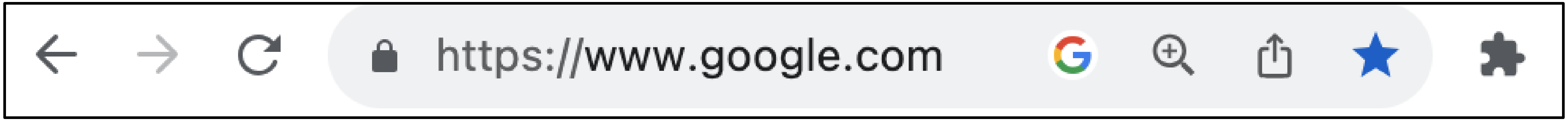


https://www.google.com

- 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



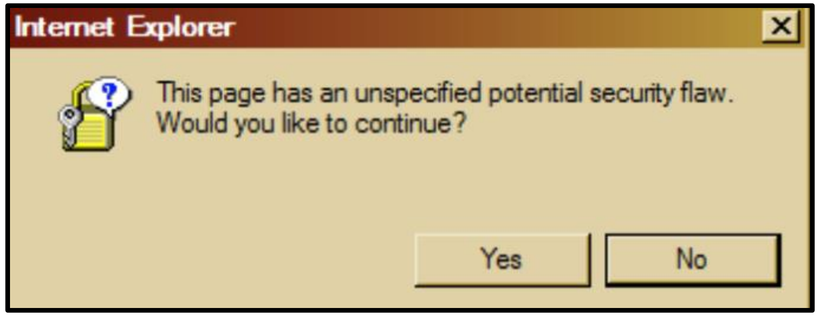
- Goal: the client (Human) can identify s
  - SSL/TLS is being used to protect against

- 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!”

What happens if page served over HTTPS but contains HTTP?

Mixed Content: Combining HTTPS and HTTP

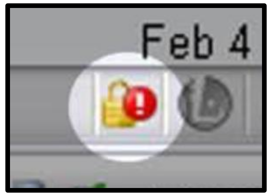
- Page served over HTTPS but contains HTTP
  - IE 7: no lock, warning



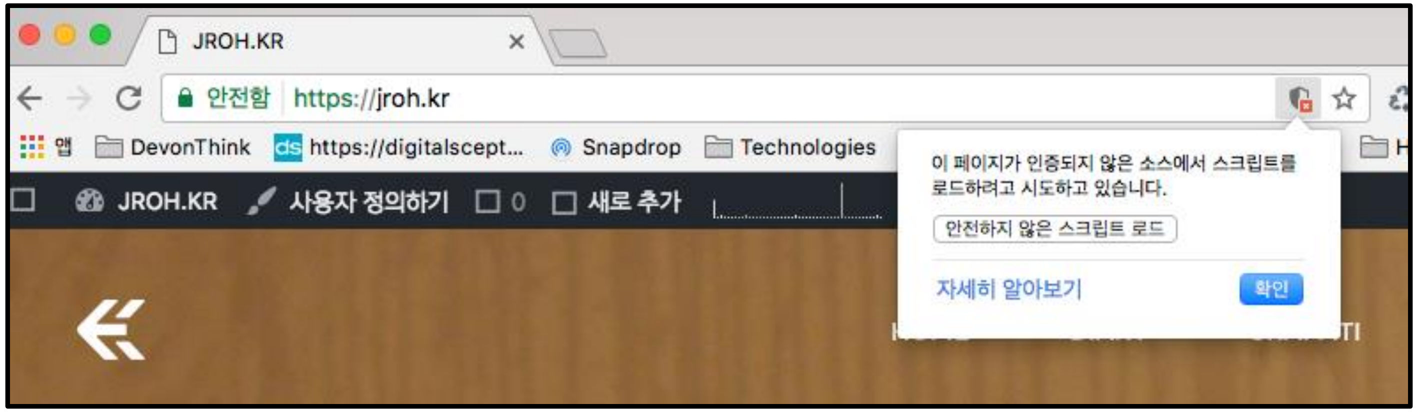
Internet Explorer

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

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

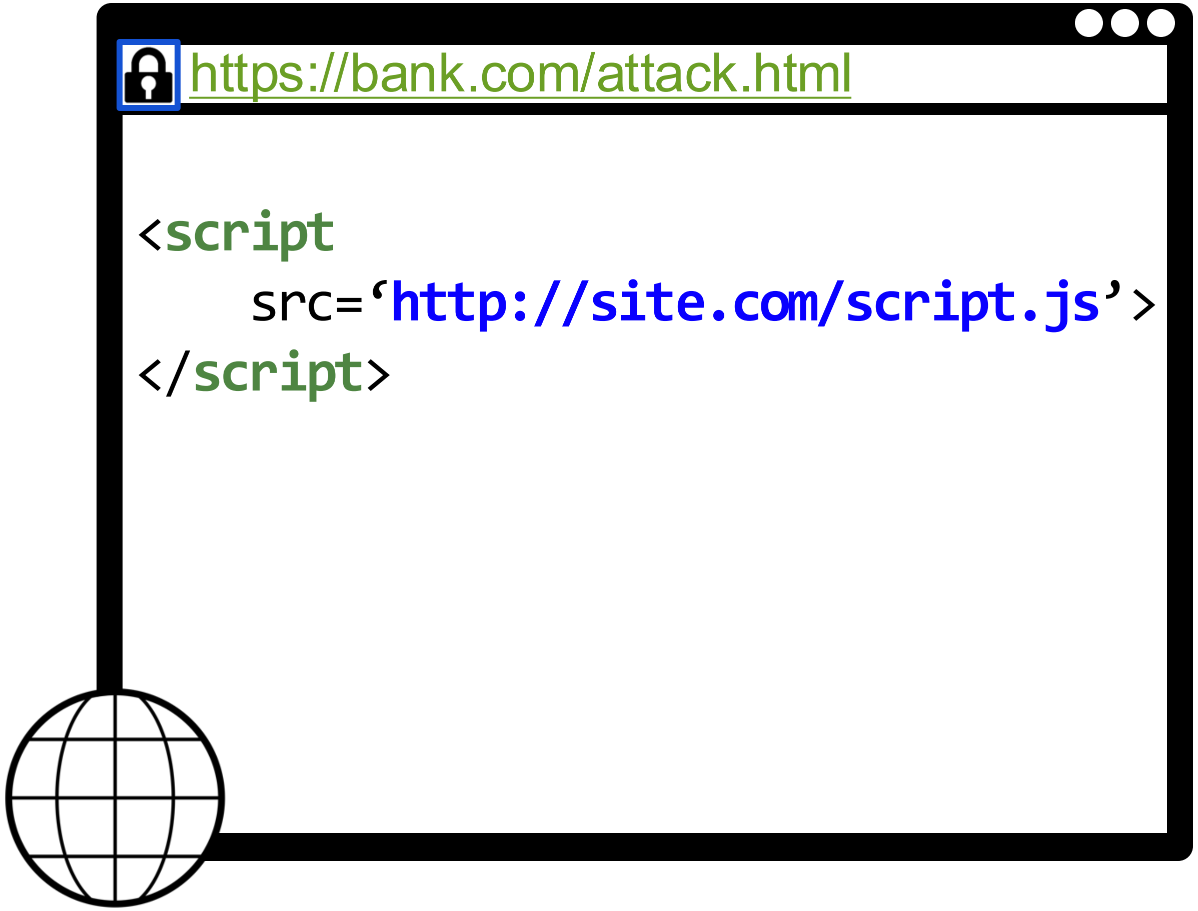


  - Safari: does not detect mixed content
  - Chrome: lock icon, warning



이 페이지가 인증되지 않은 소스에서 스크립트를 로드하려고 시도하고 있습니다.

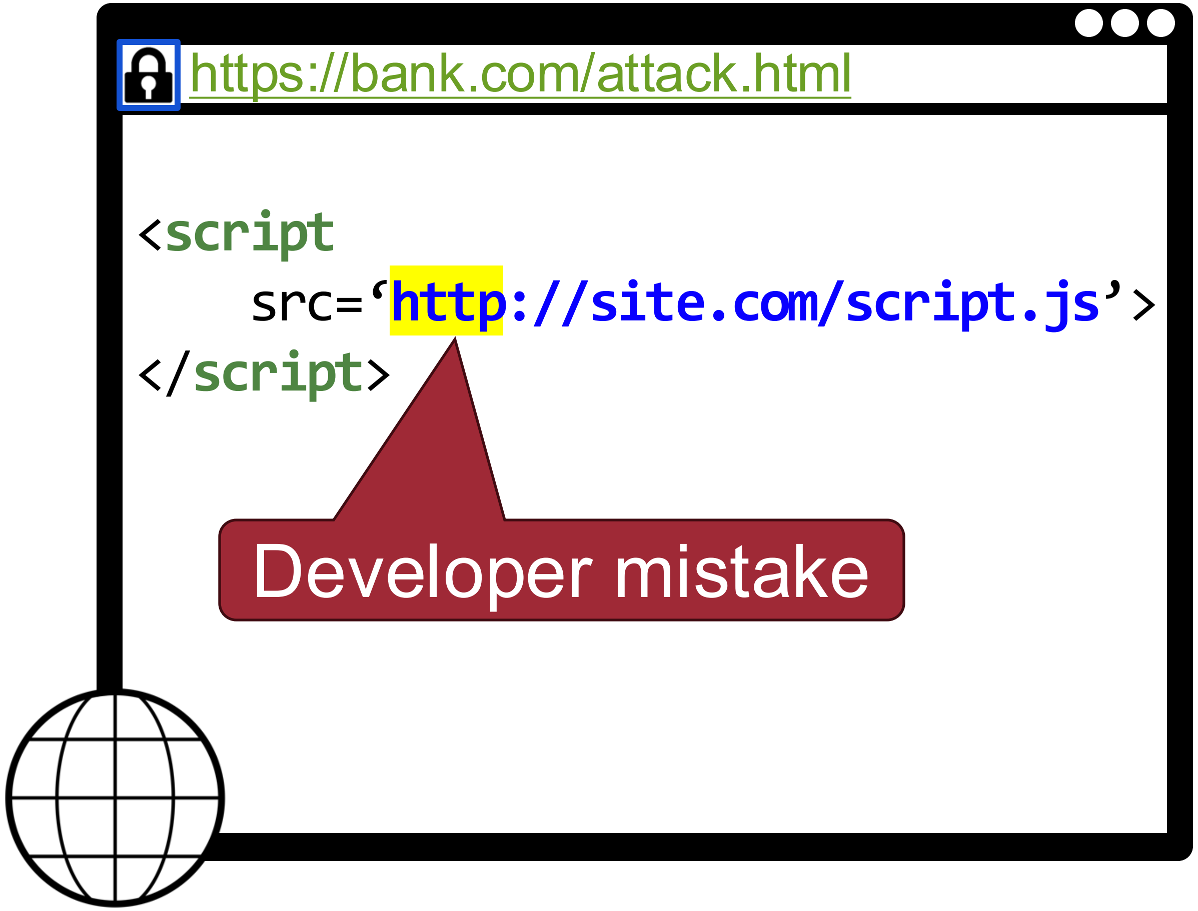
Mixed Content and Network Attacks



https://bank.com/attack.html

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

Mixed Content and Network Attacks

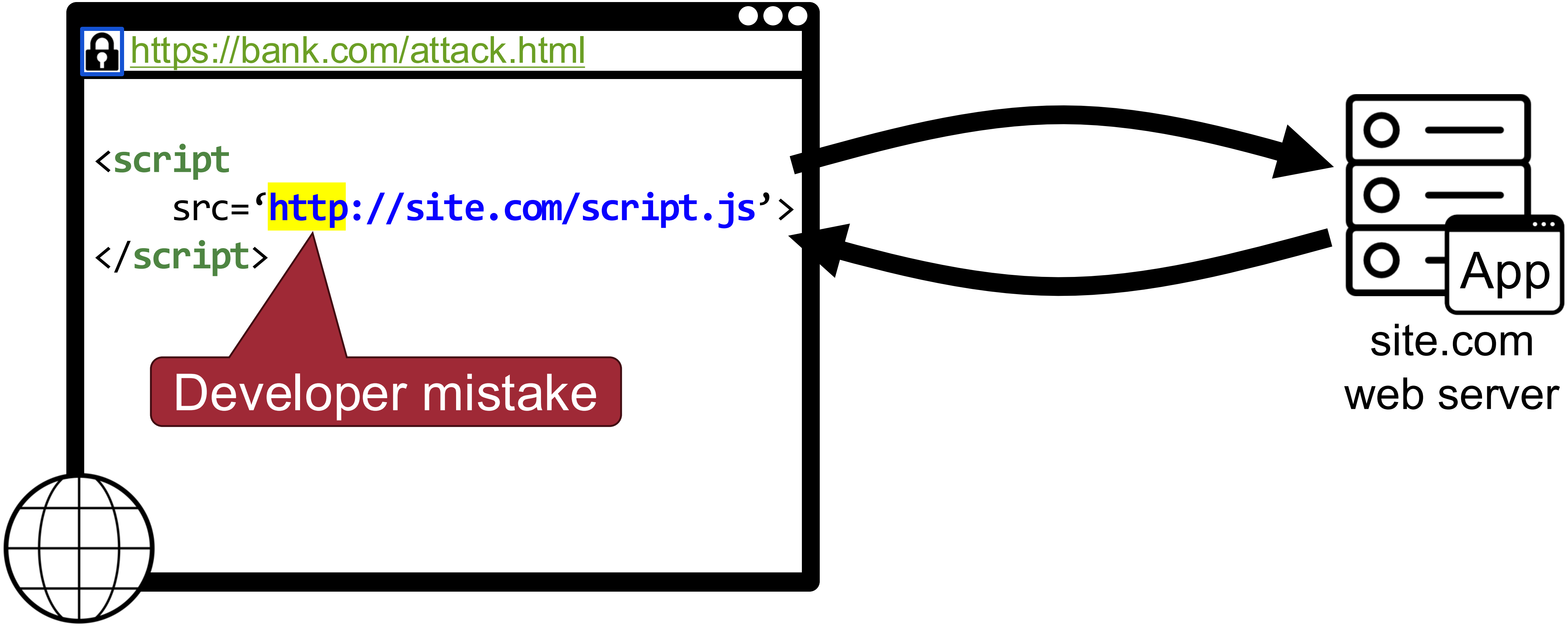


https://bank.com/attack.html

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

Developer mistake

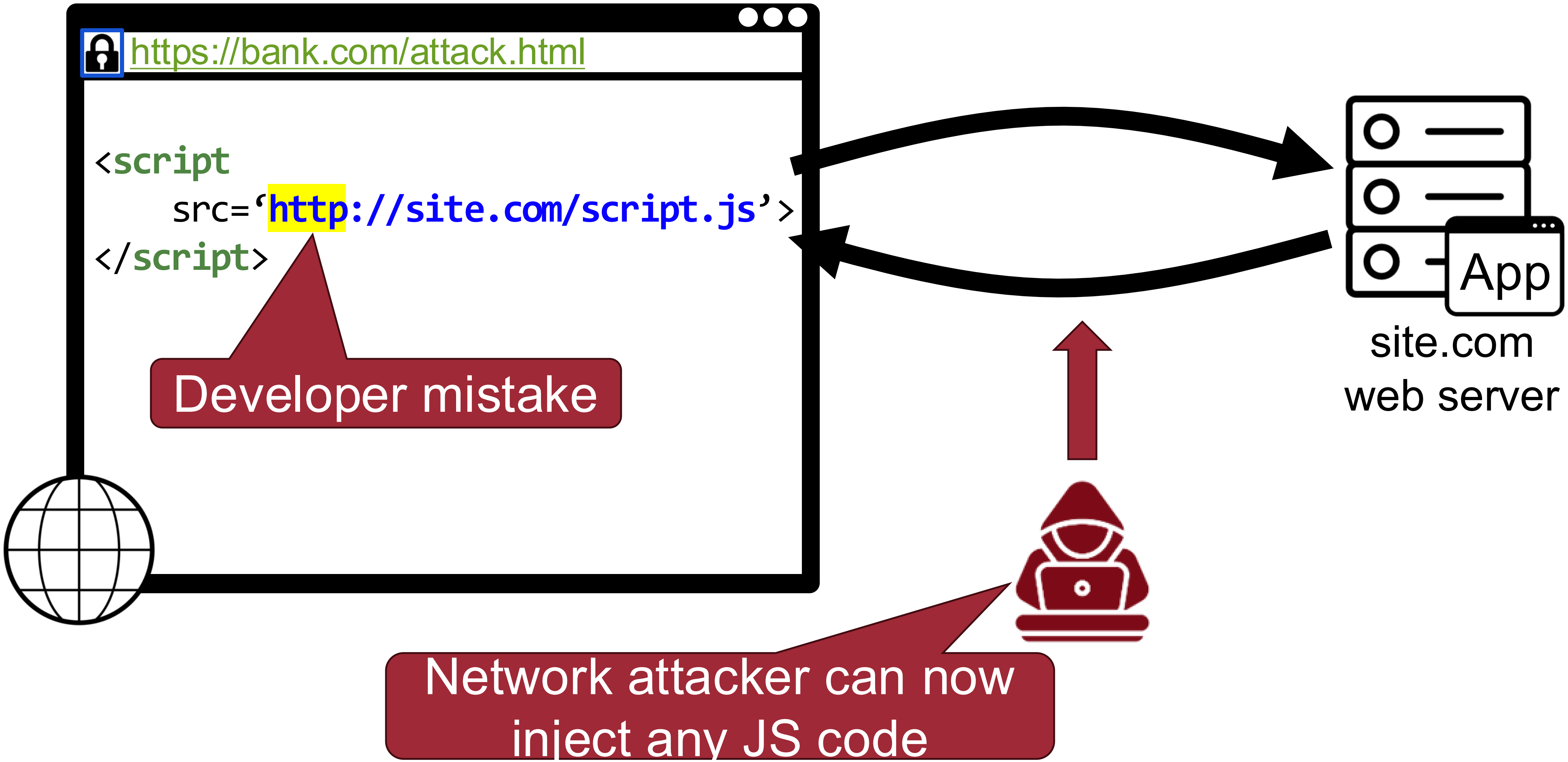
Mixed Content and Network Attacks



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

Developer mistake

Mixed Content and Network Attacks

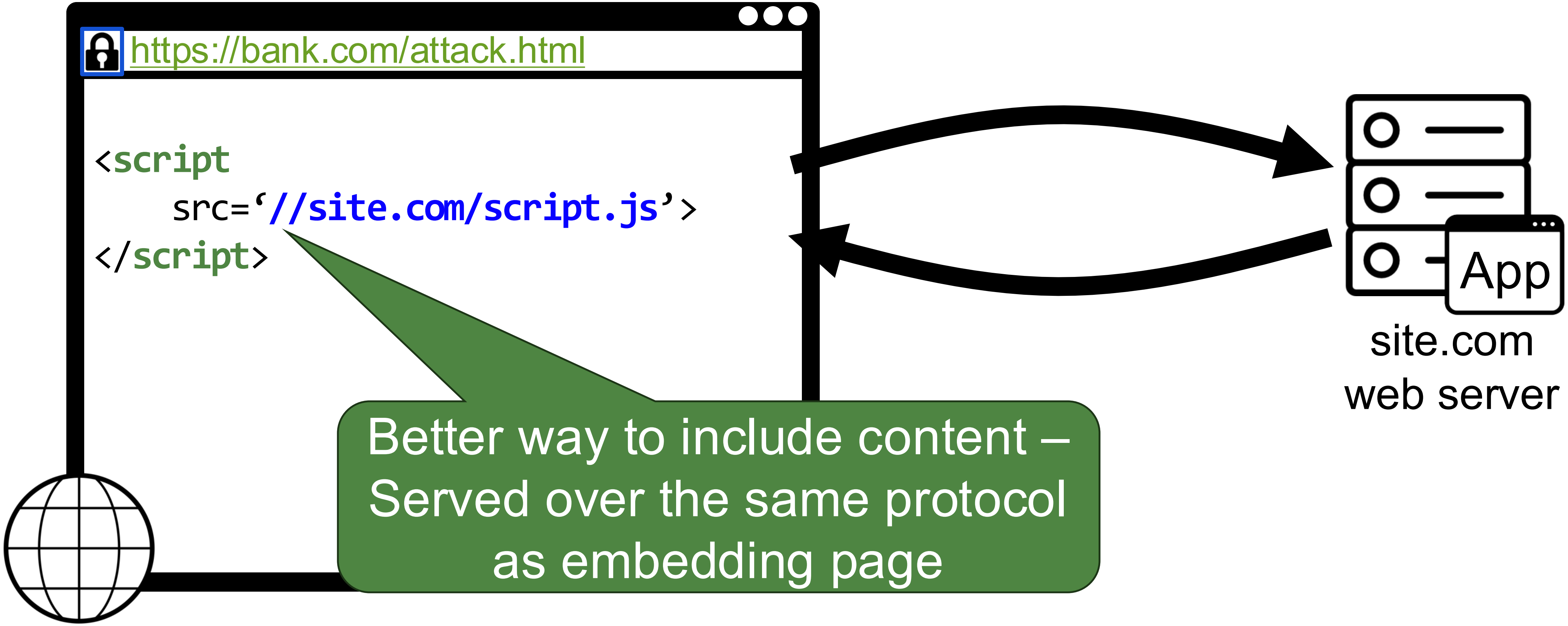


```
<script
     src='http://site.com/script.js'>
</script>
```

Developer mistake

Network attacker can now inject any JS code

Mixed Content and Network Attacks

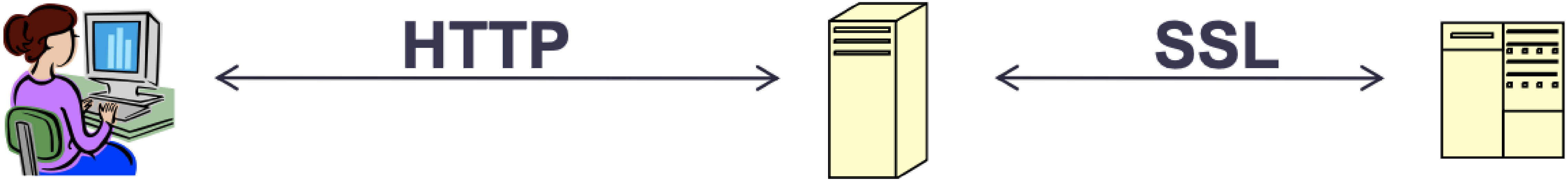


```
<script
     src='//site.com/script.js'>
</script>
```

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>
```

CSP for TLS Enforcement

- `block-all-mixed-content`
  - Instruct browsers to block all mixed content
- upgrade-insecure-requests
  - Automatically rewrite all HTTP URLs to HTTPS upon page loading

Summary

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

Question?