CSE610: Web Programming & Security

10. SSL/TLS & HTTPS

Seongil Wi

Department of Computer Science and Engineering

Midterm Exam

- April. 18 (Thursday)
- Class Time (1h 15m)

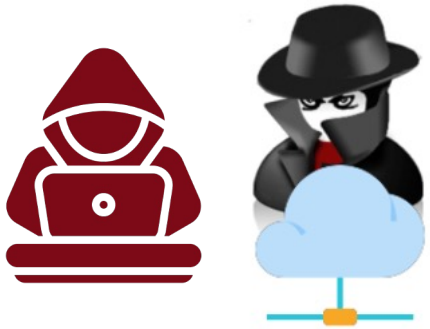
- Descriptive type questions
- Closed book

HW1 Due Date

- Due date (writeup report): 4/9 (Tue), 11:59PM

Today's Topic

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

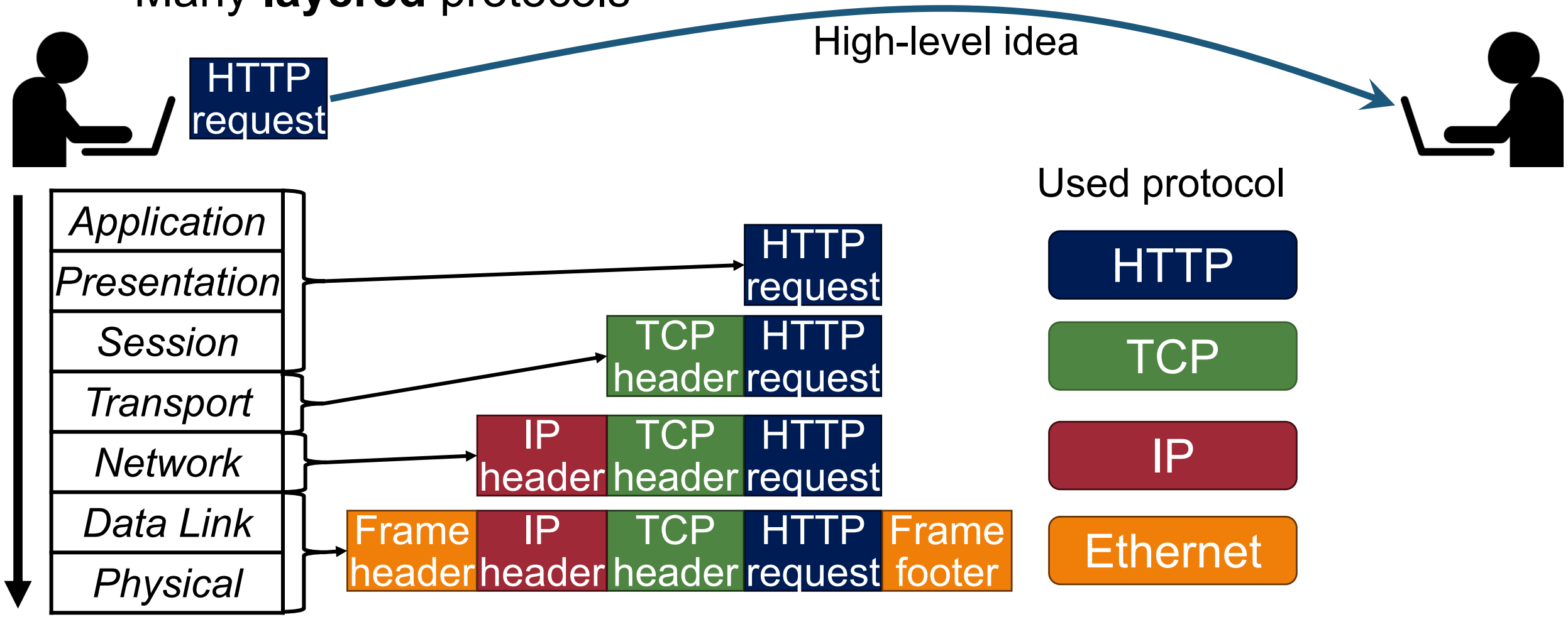


- **Remote attacker:** can connect to remote system via the network
  - Mostly targets the server

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

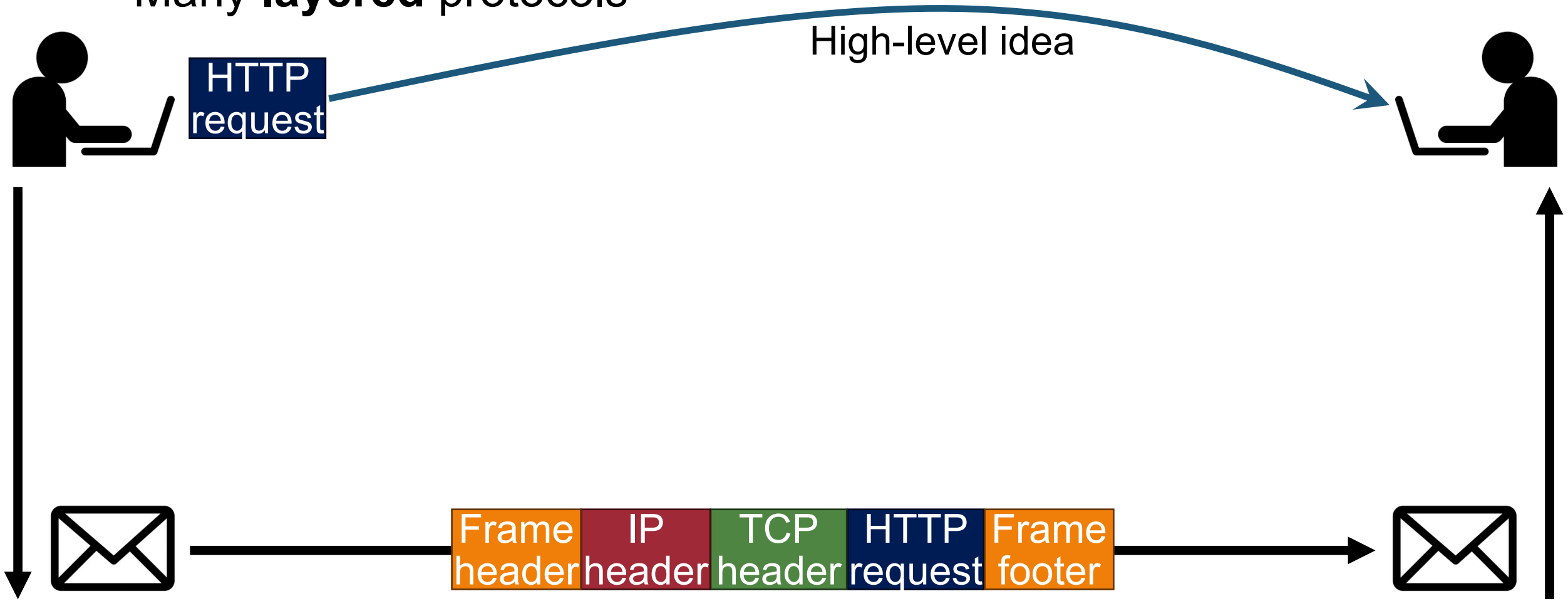
Protocol

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



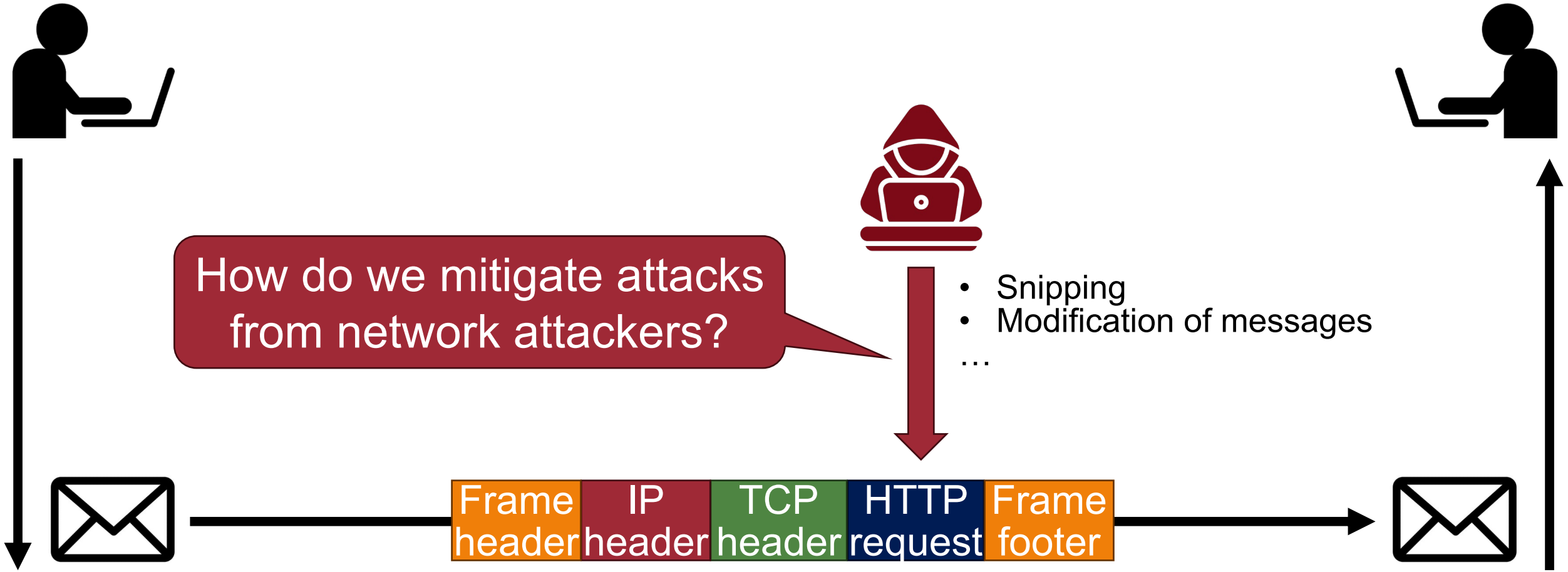
Protocol

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



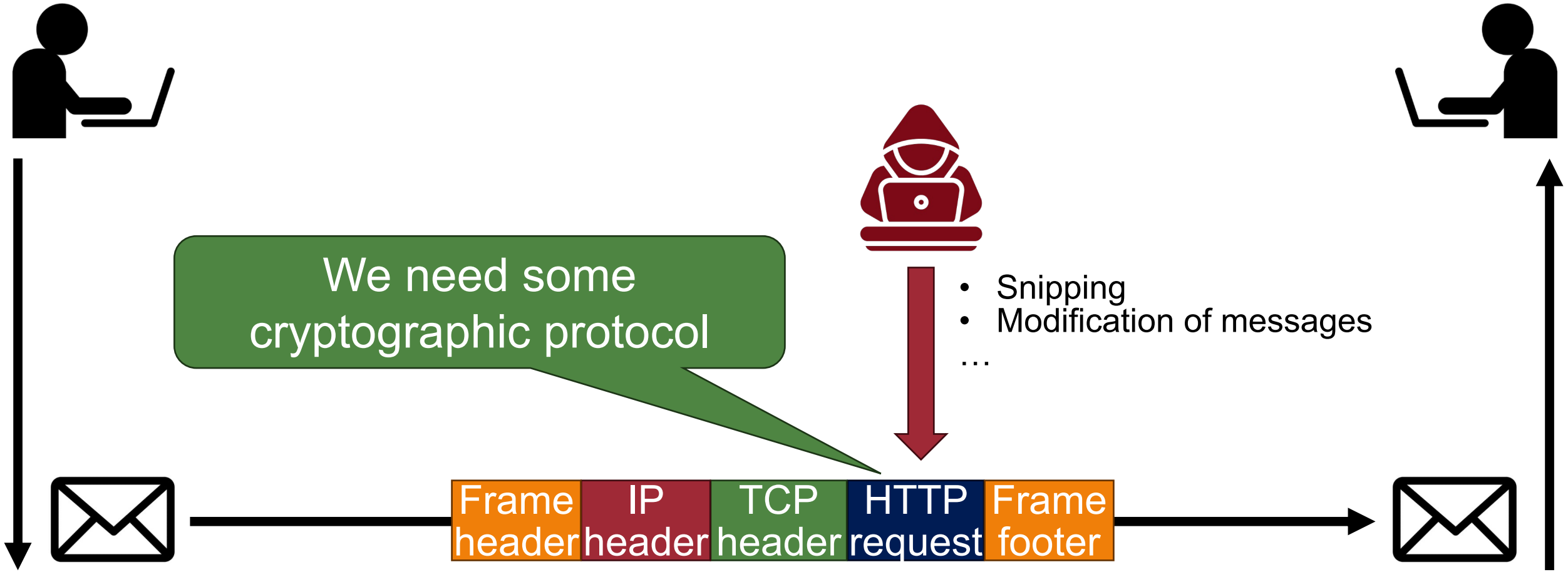
Network Attackers

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



Motivation: Cryptographical Protocol

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



SSL/TLS

What is SSL/TLS?

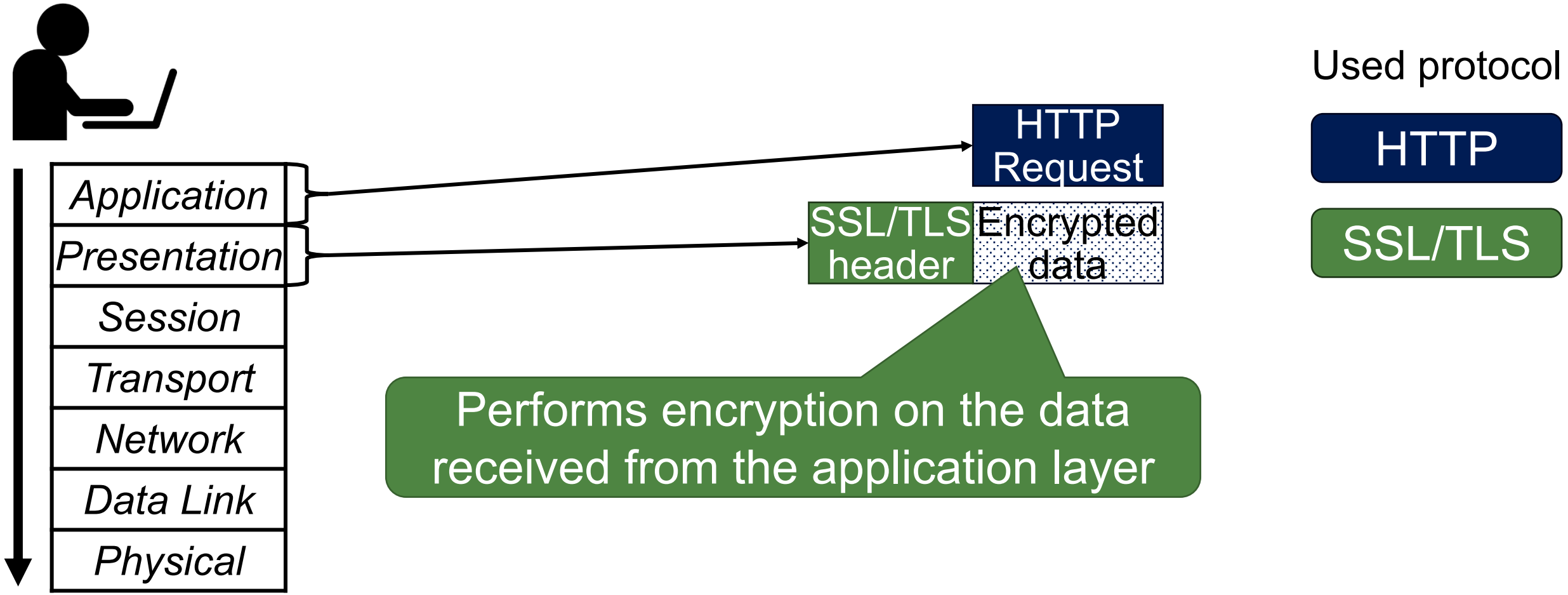
- **Secure Sockets Layer (SSL)** and **Transport Layer Security (TLS)** protocols
  - Same protocol design, different crypto algorithms
  - (Reserved) port number: 443

- Security goals: achieving…
  - Confidentiality
  - Integrity
  - Authentication

- ***De facto* standard for Internet security**

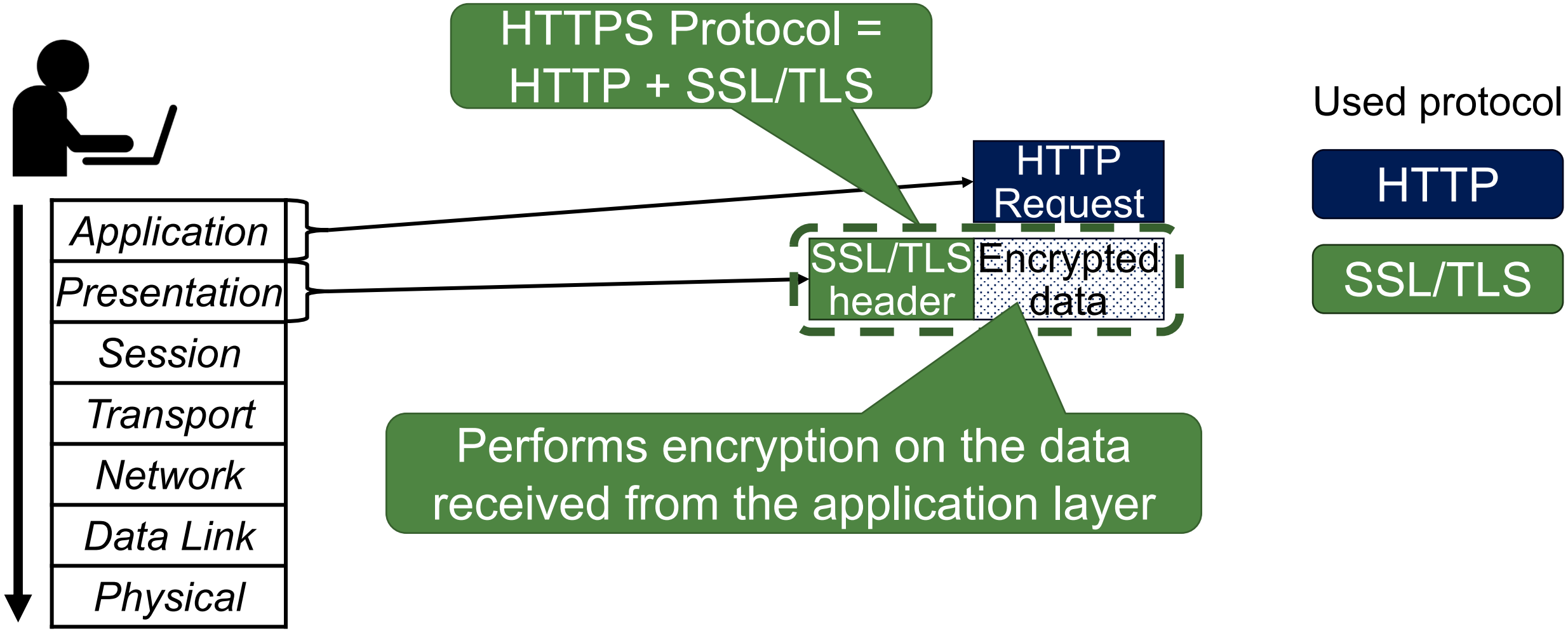
SSL/TLS Basic Idea

- Adding a protocol layer for secure communication!



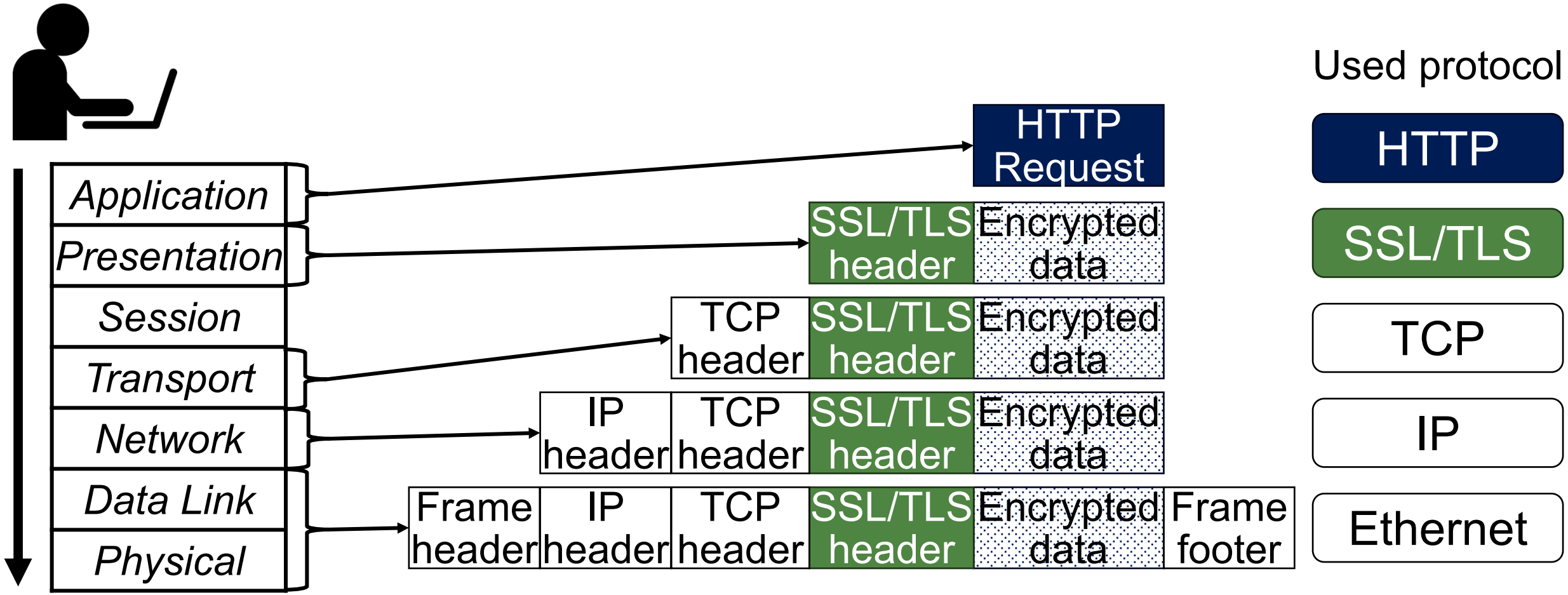
SSL/TLS Basic Idea

- Adding a protocol layer for secure communication!



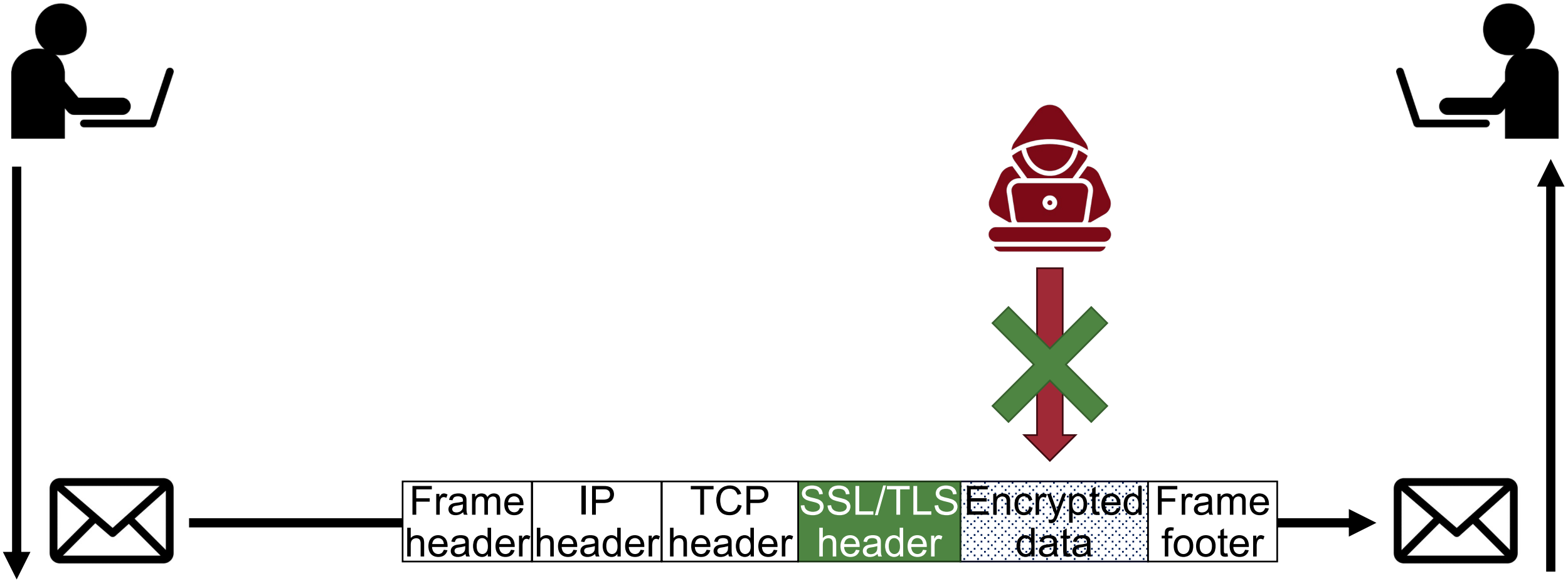
SSL/TLS Basic Idea

- Adding a protocol layer for secure communication!



SSL/TLS Basic Idea

- Adding a protocol layer for secure communication!



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

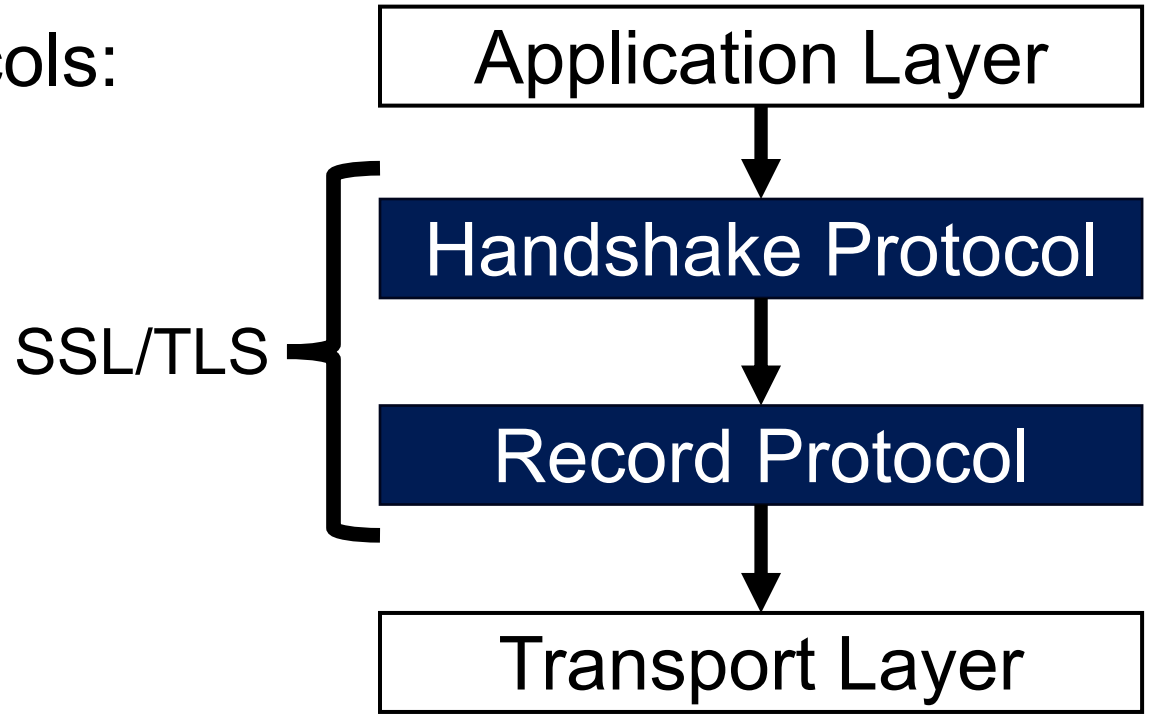
Aug. 2018

TLS 1.3

SSL/TLS Basics

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

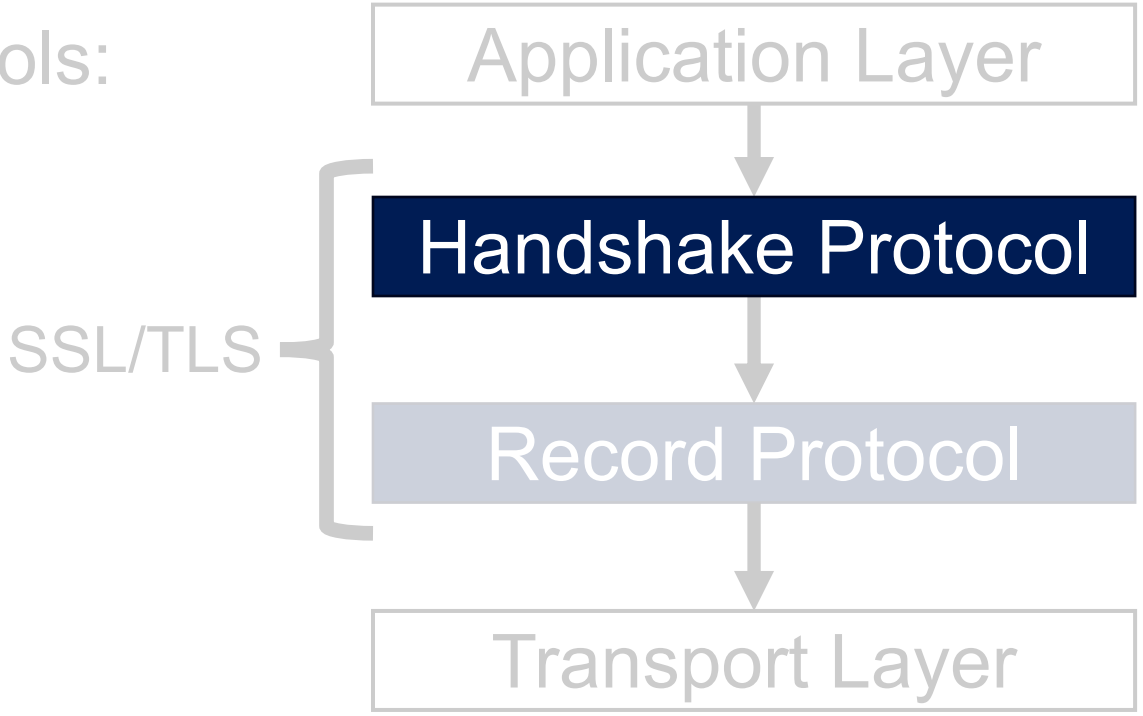
- Composed of two layers of protocols:
  1. Handshake protocol
  2. Record protocol



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

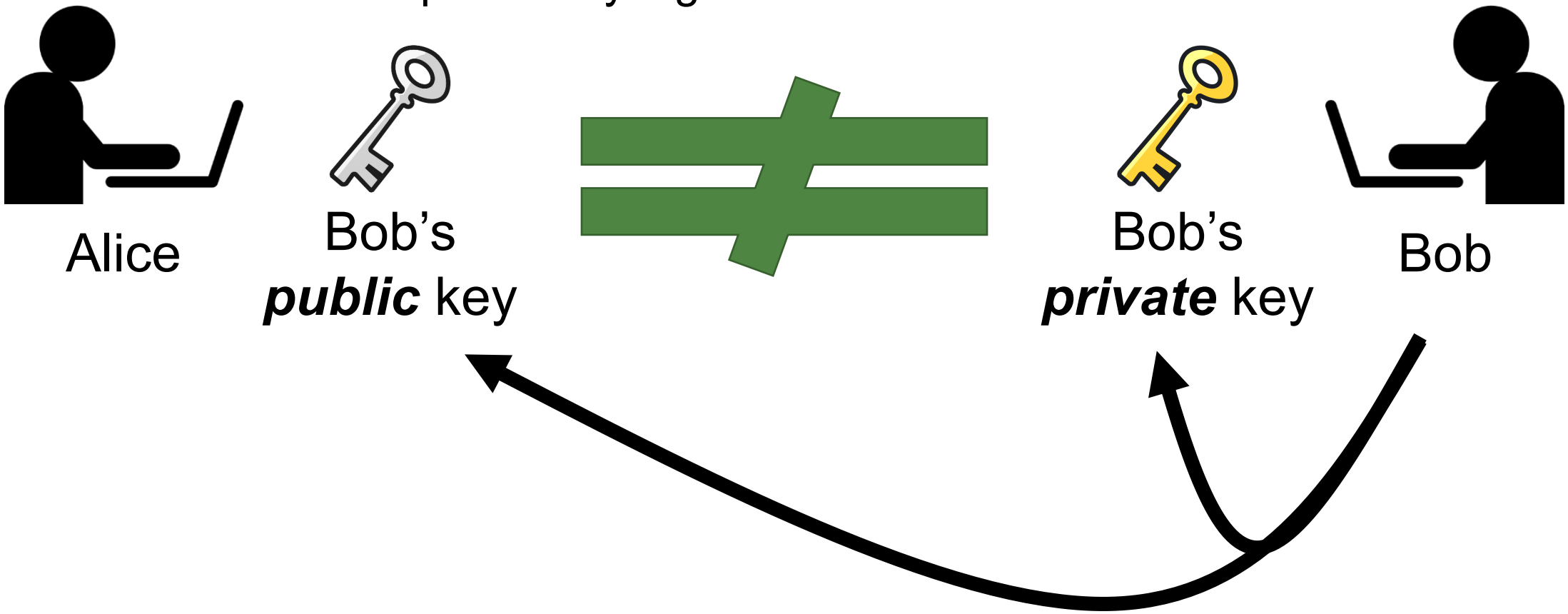


SSL/TLS Handshake Protocol

- 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

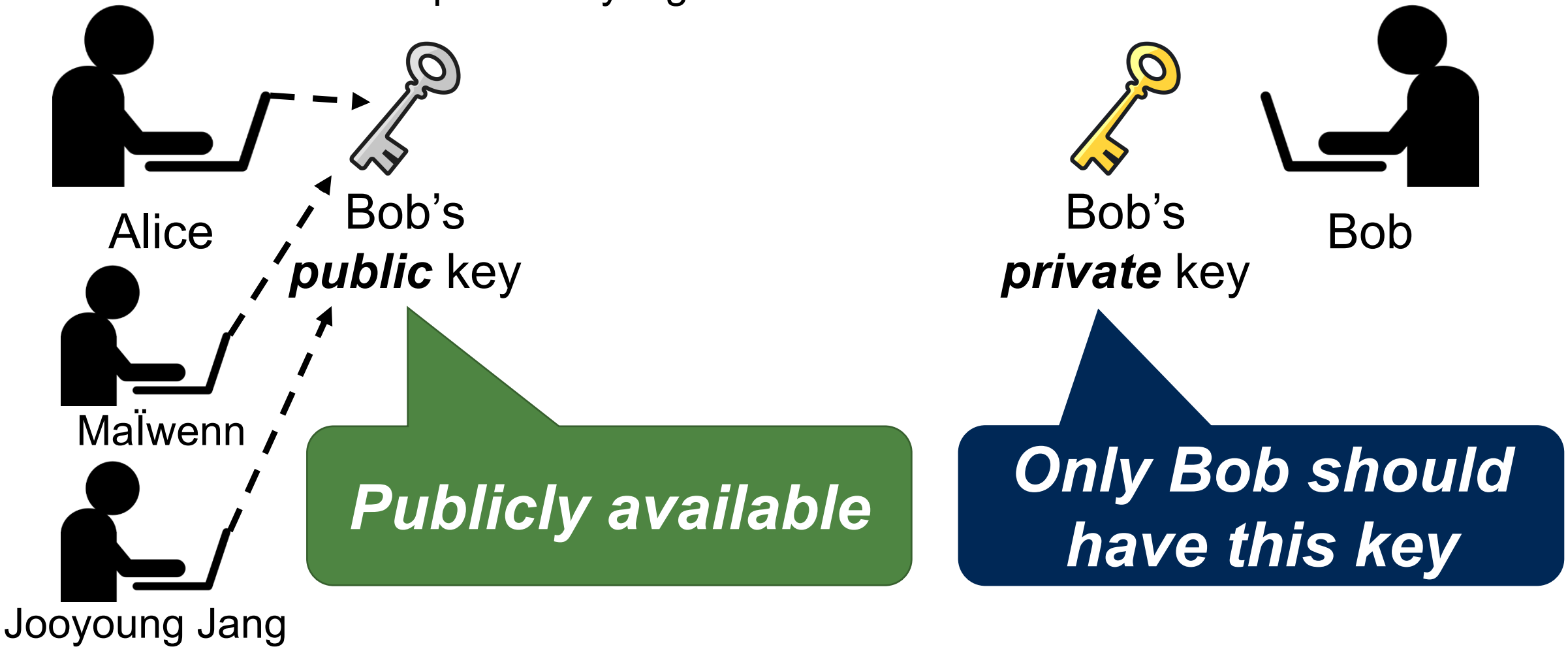


Ref: Asymmetric Key Cryptography

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

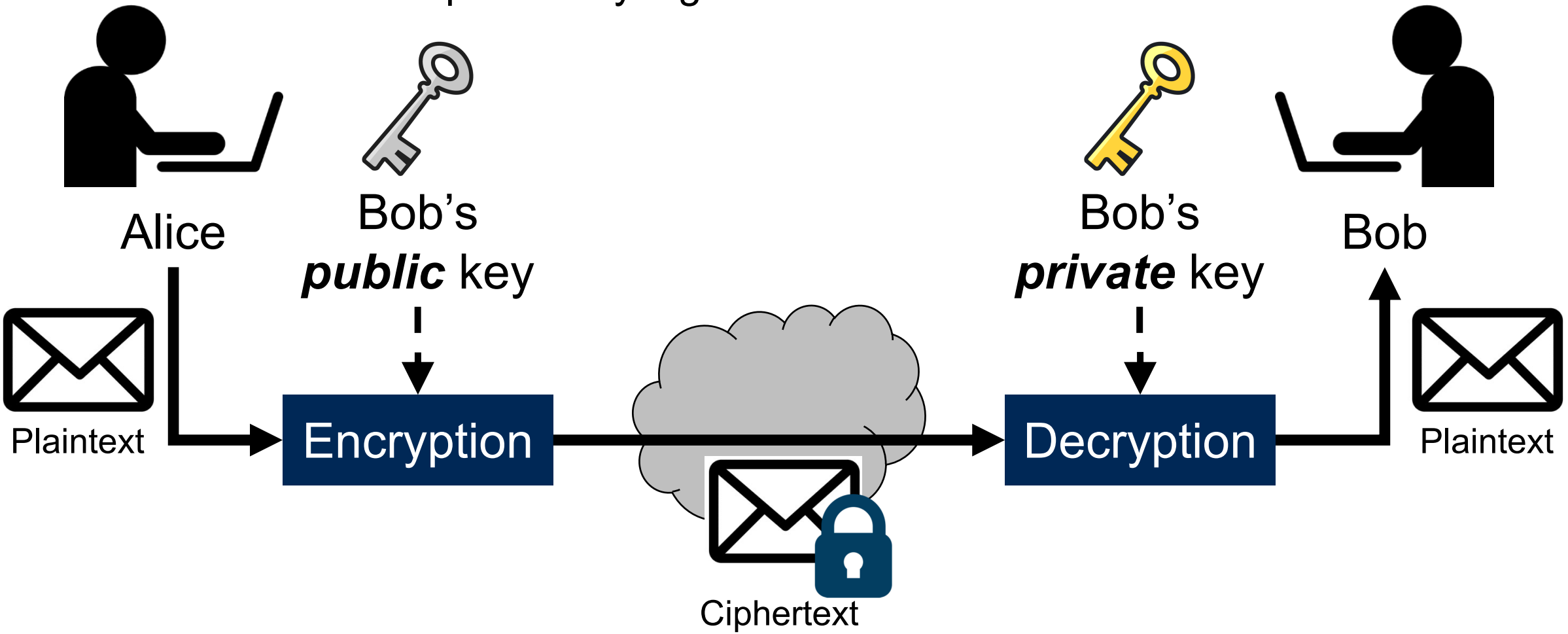
Ref: Asymmetric Key Cryptography

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

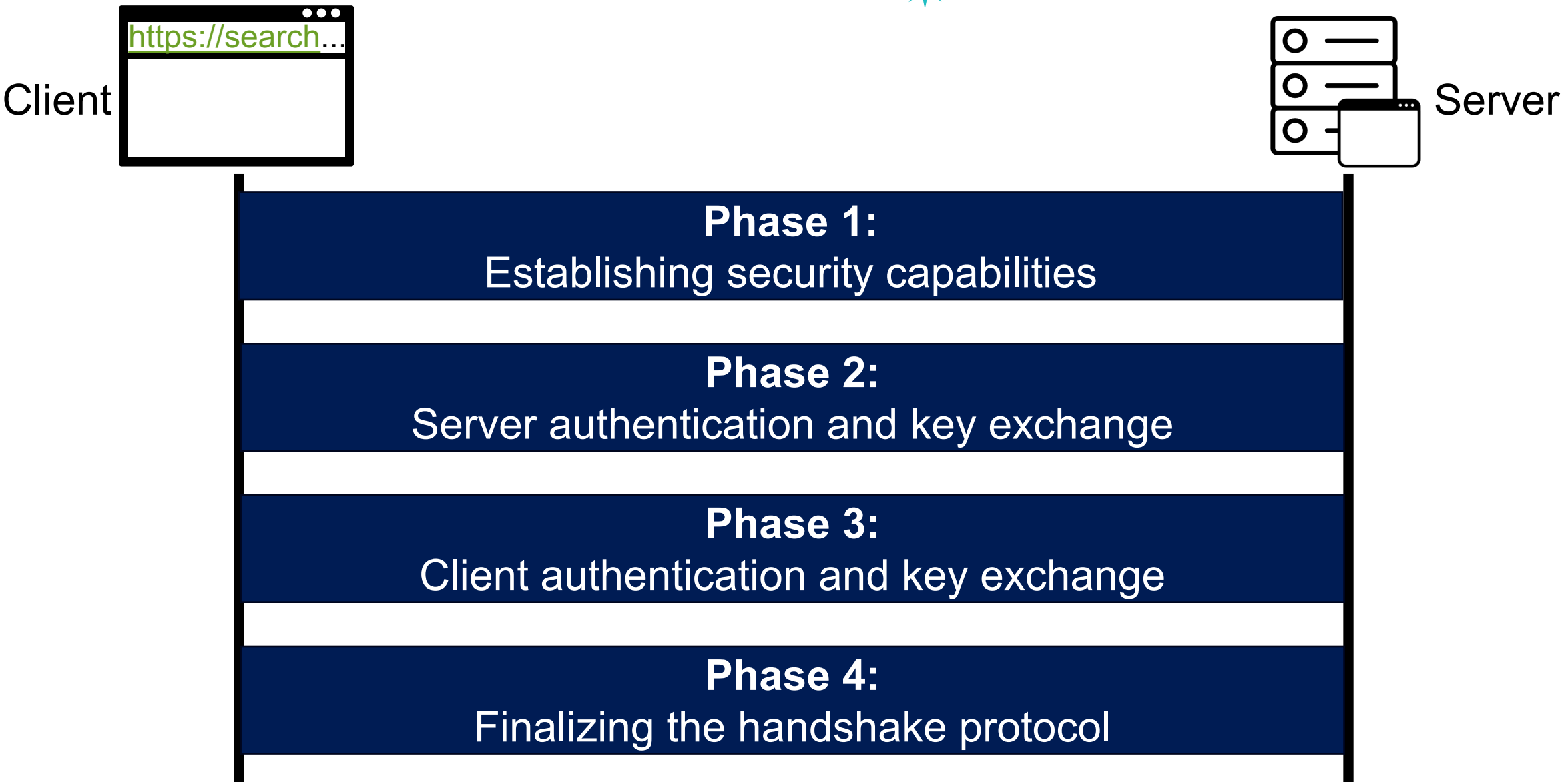


Ref: Asymmetric Key Cryptography

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

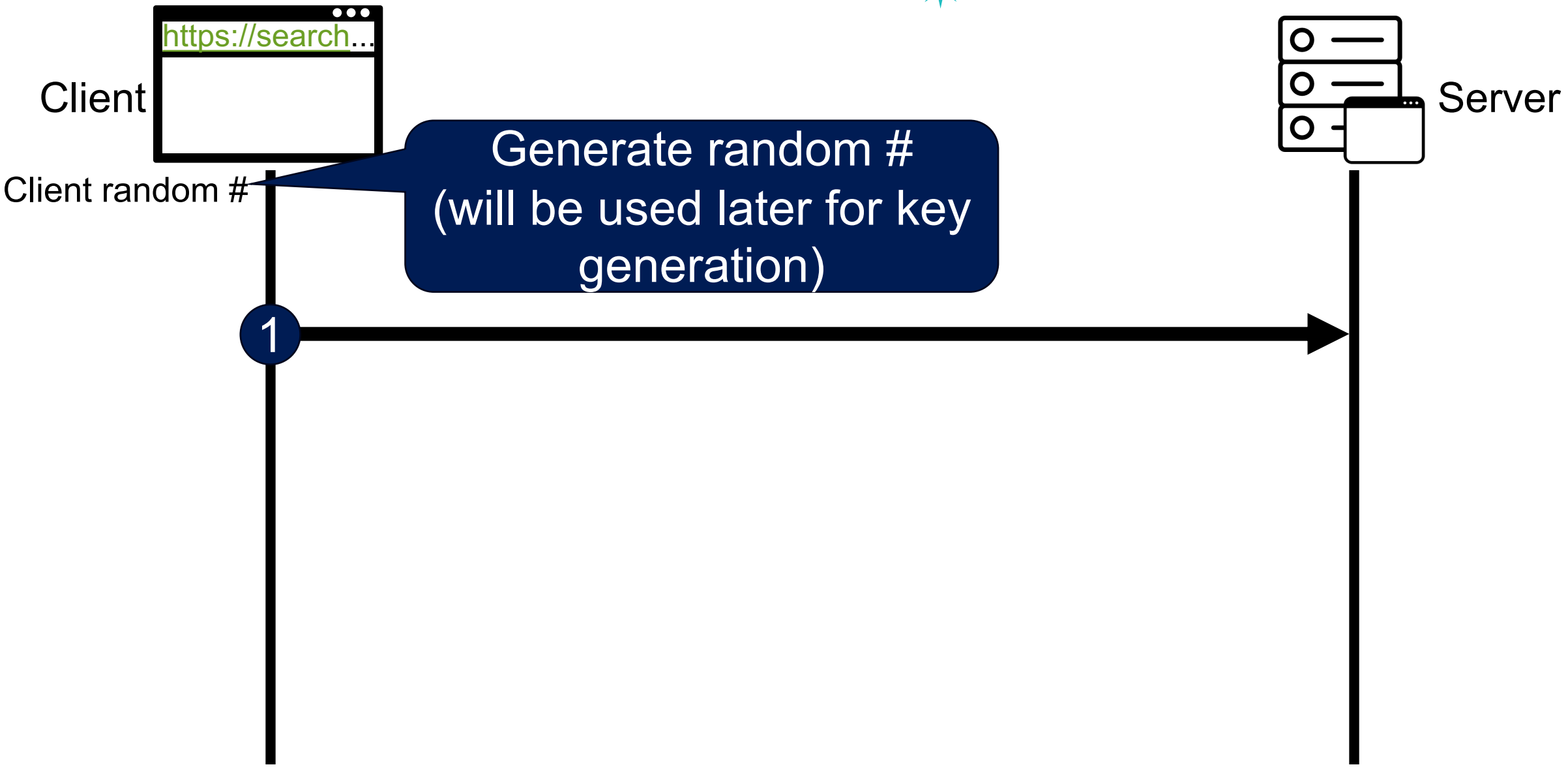


Four Phases of Handshake Protocol

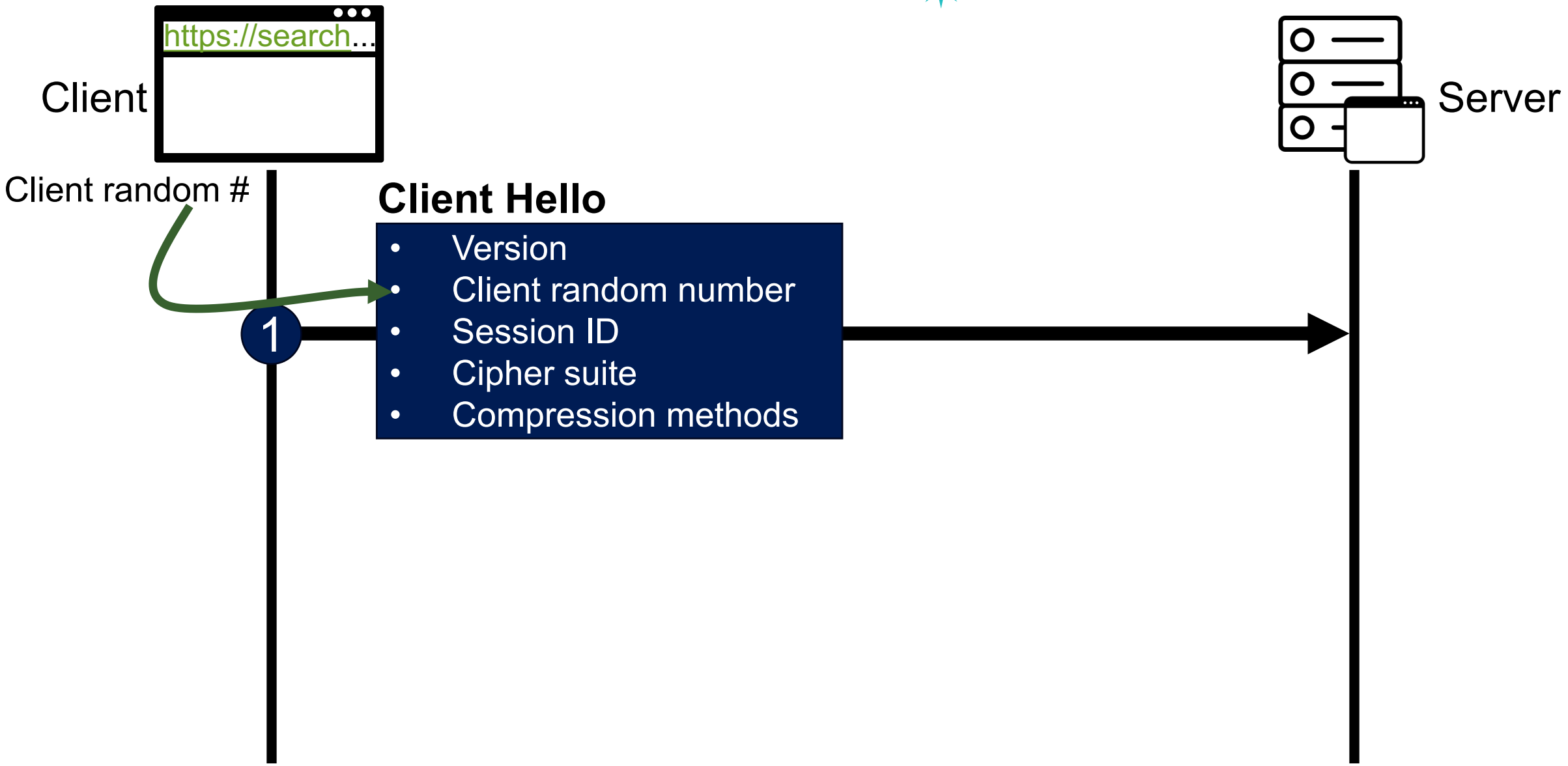


Phase 1: Establishing Security Capabilities

Phase 1: Establishing Security Capabilities



Phase 1: Establishing Security Capabilities



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:

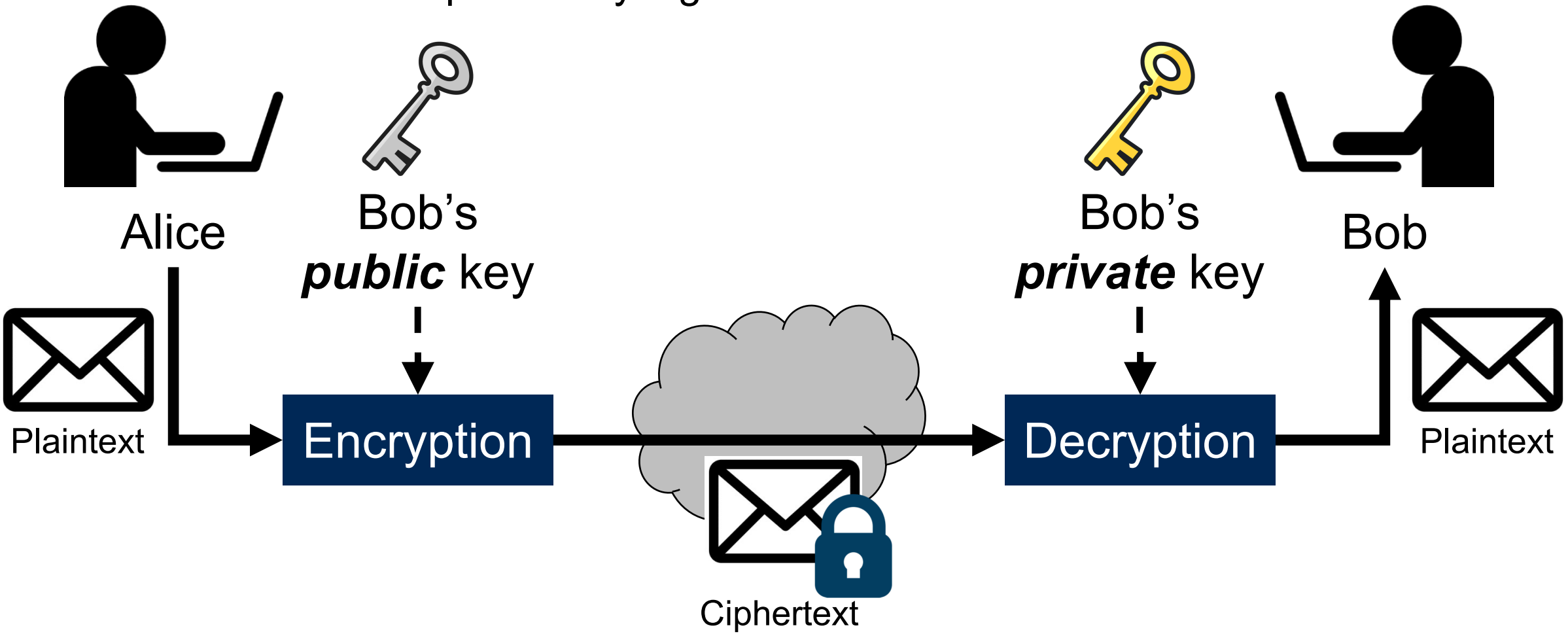
`TLS_RSA_WITH_AES_128_CBC_SHA`

Protocol

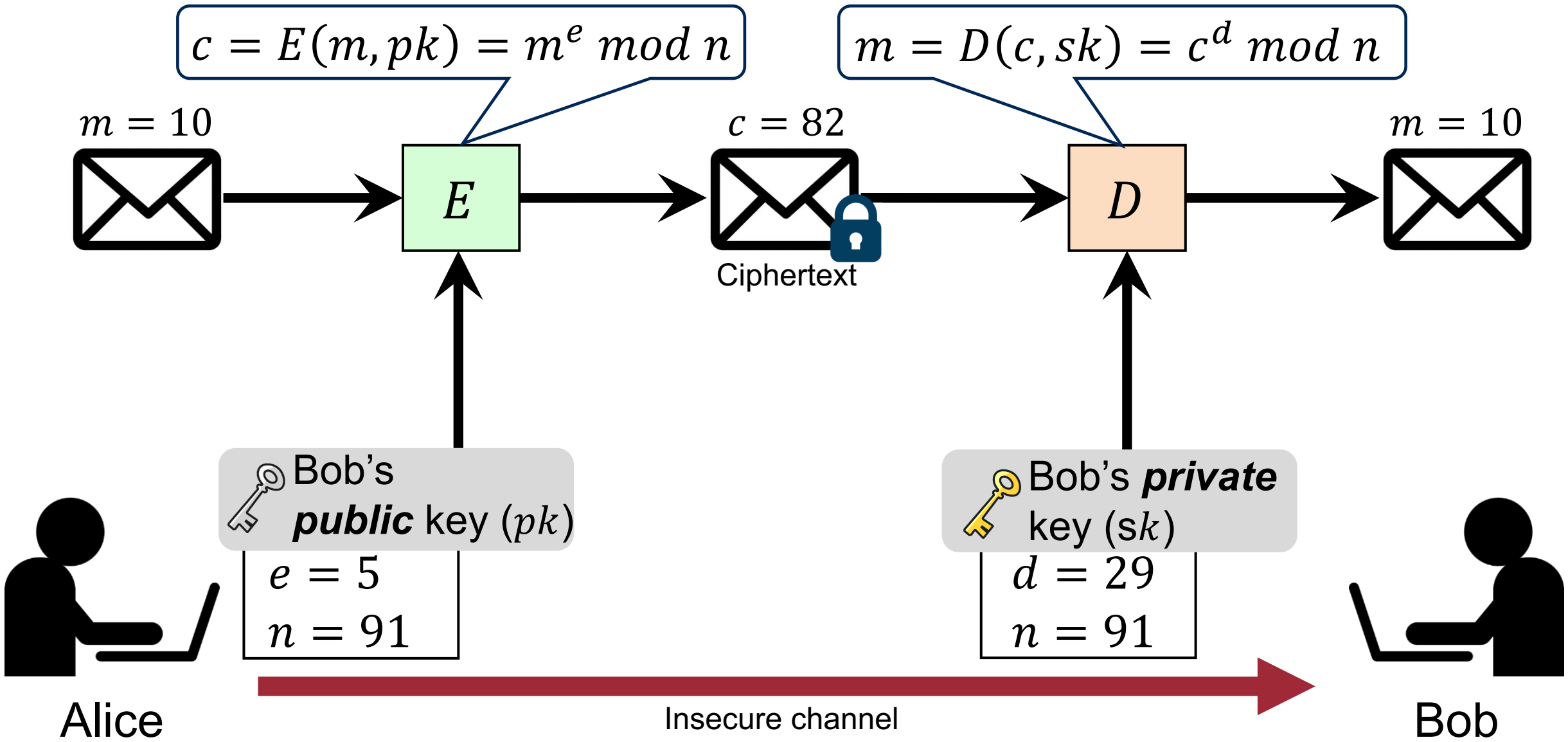
(Asymmetric) Encryption/decryption algorithm (for handshake protocol)

Recap: Asymmetric Key Cryptography

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



Ref: RSA Algorithm



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`

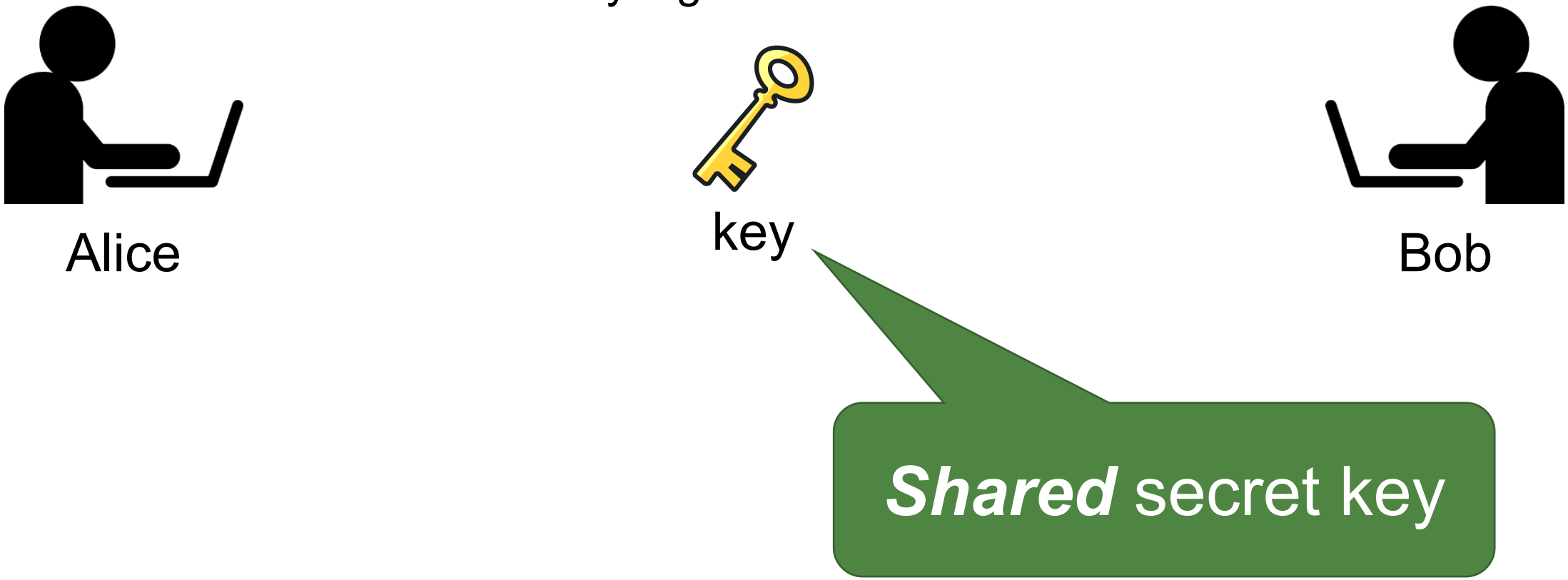
Protocol

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

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

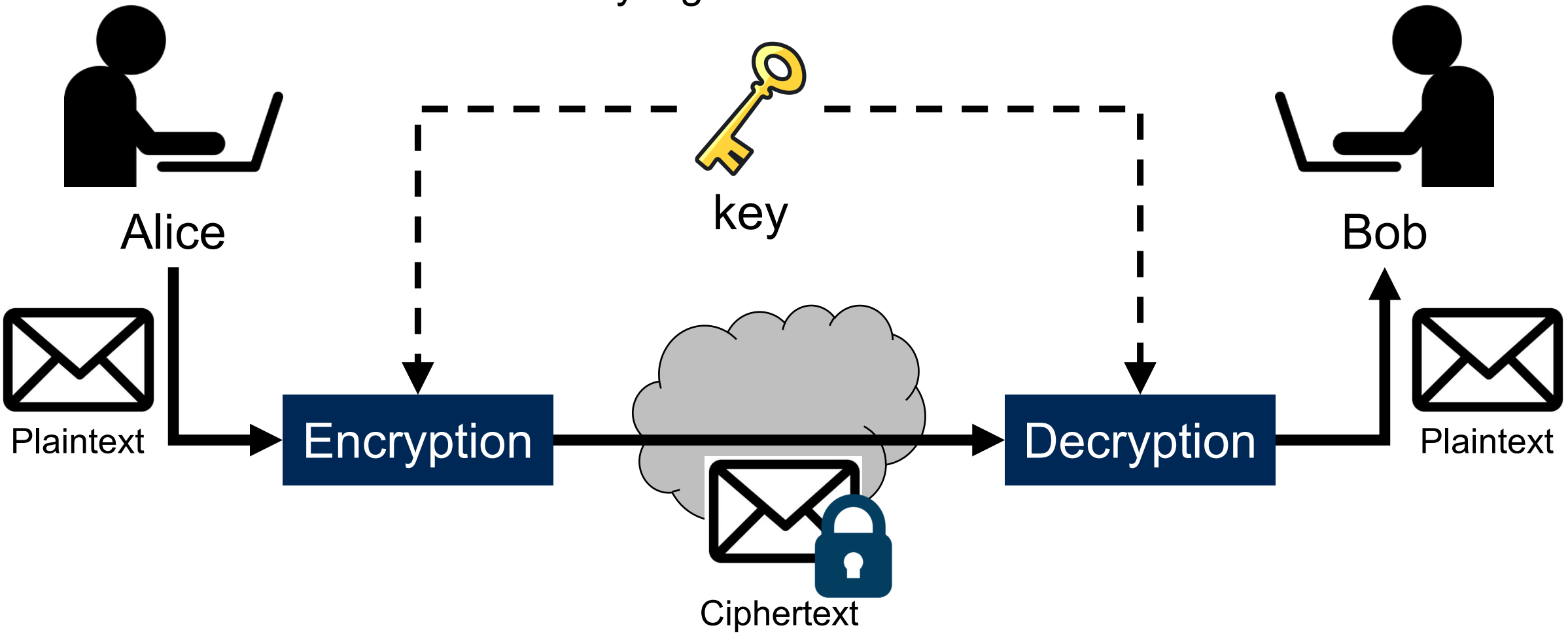
Ref: Symmetric Key Cryptography

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

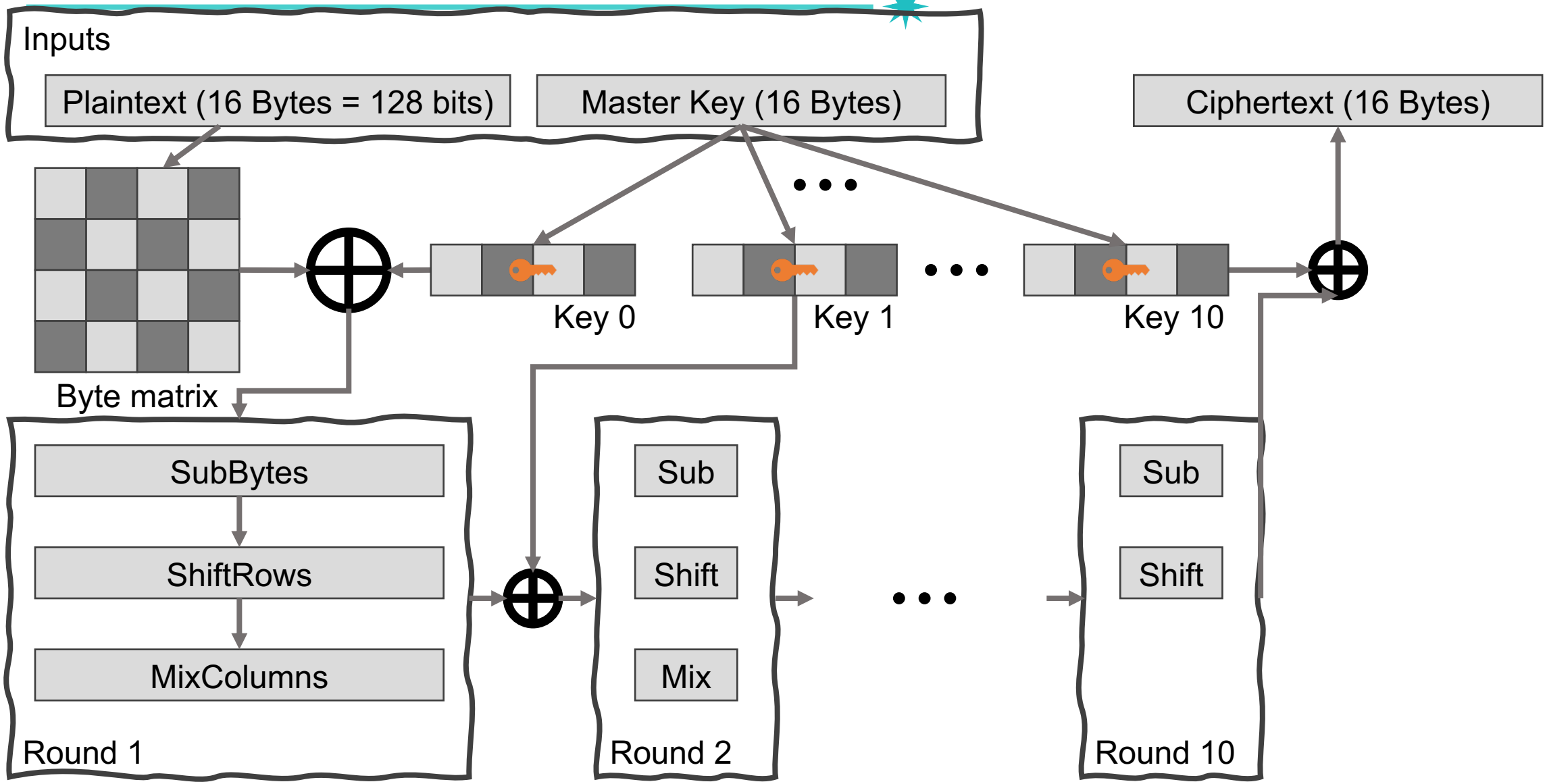


Ref: Symmetric Key Cryptography

- 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

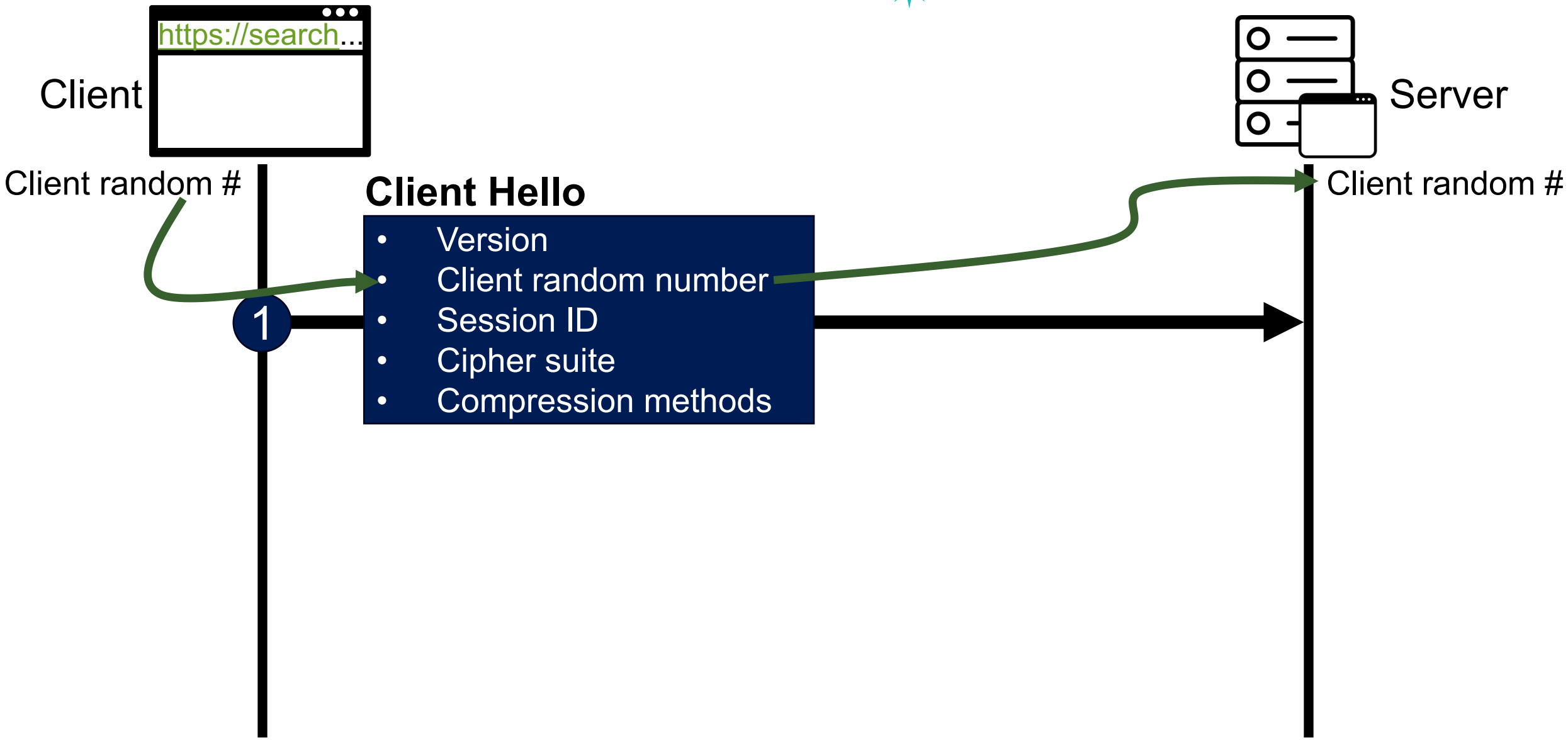
Client Hello –

- **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

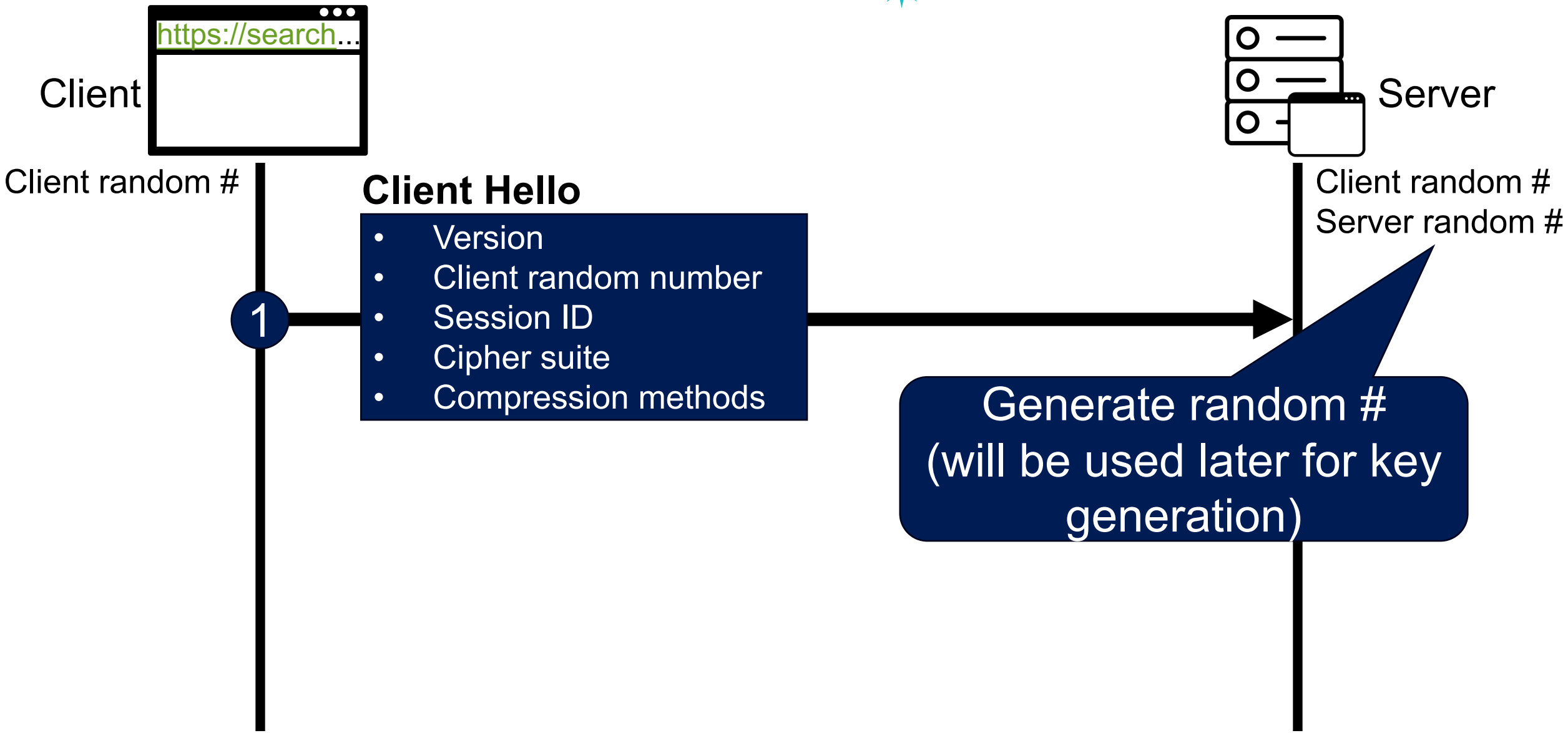
In decreasing order of preference

```
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.0 (0x0301)
    Length: 512
    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
      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)
```

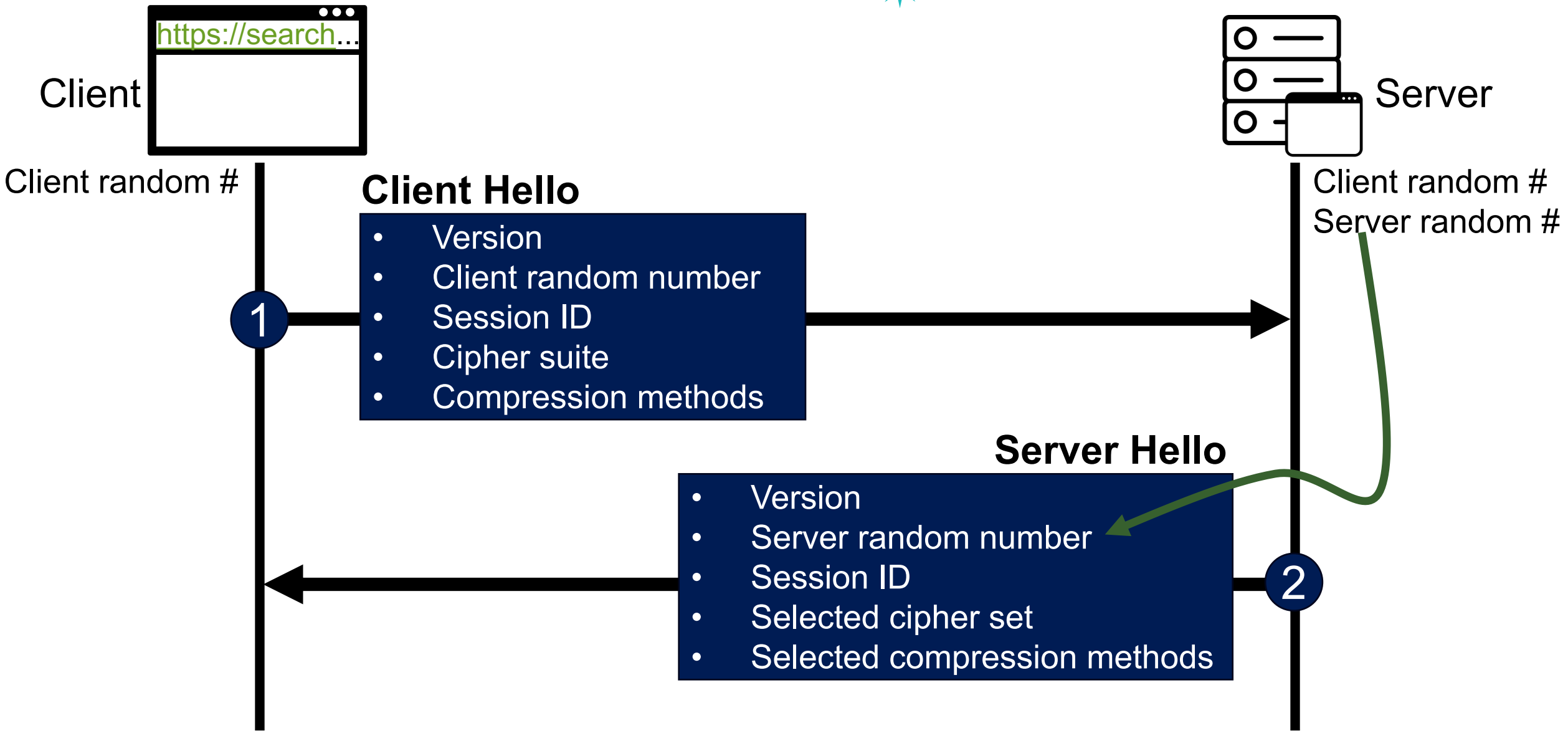
Phase 1: Establishing Security Capabilities



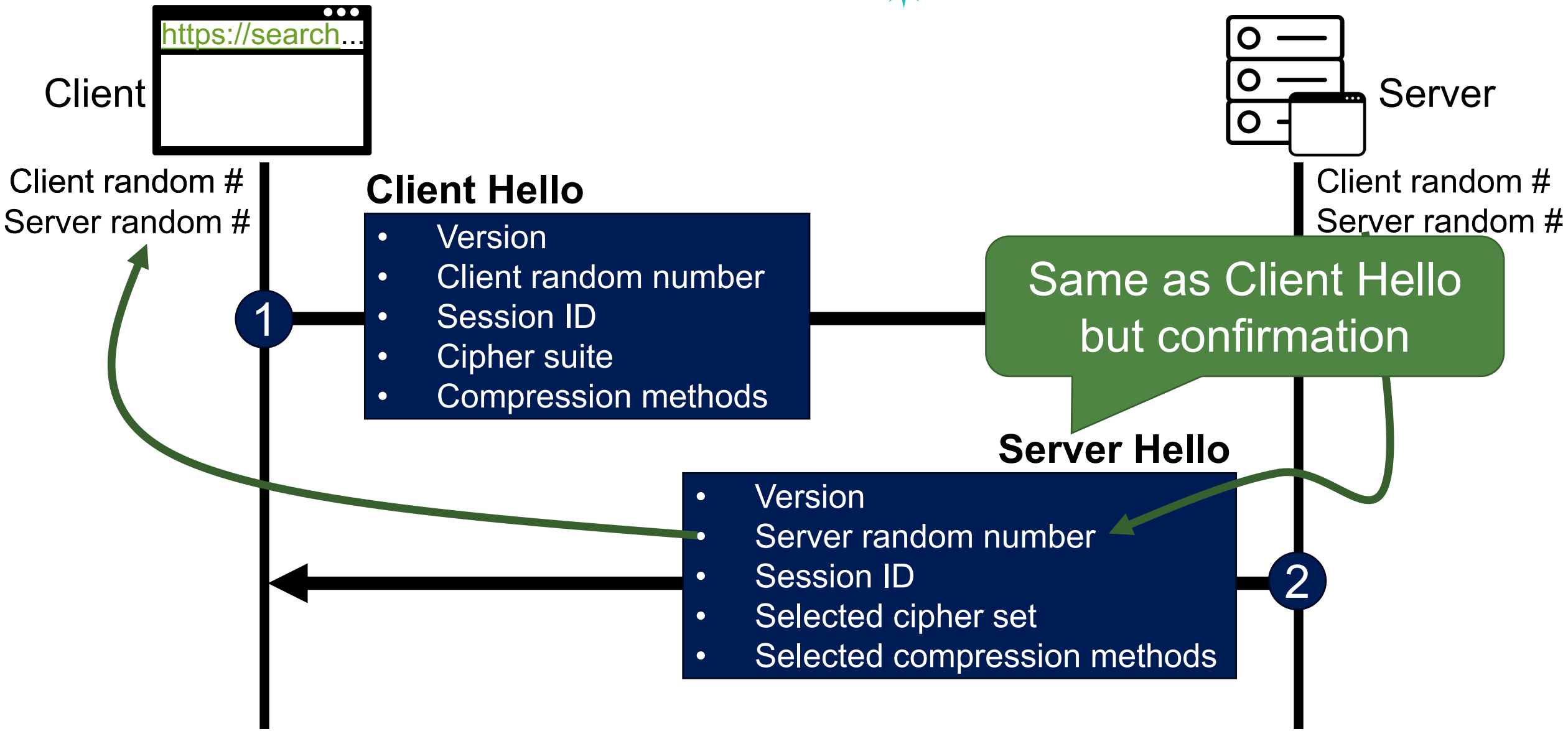
Phase 1: Establishing Security Capabilities



Phase 1: Establishing Security Capabilities



Phase 1: Establishing Security Capabilities



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

mber

stamp + 28 random bytes

for key generation

ero, old session ID

suite

nods

ession technique

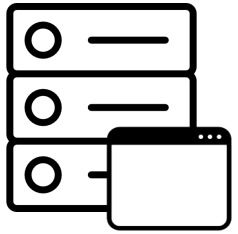
Phase 1: Establishing Security Capabilities

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

Client



Server

Client random #
Server random #

Client random #
Server random #

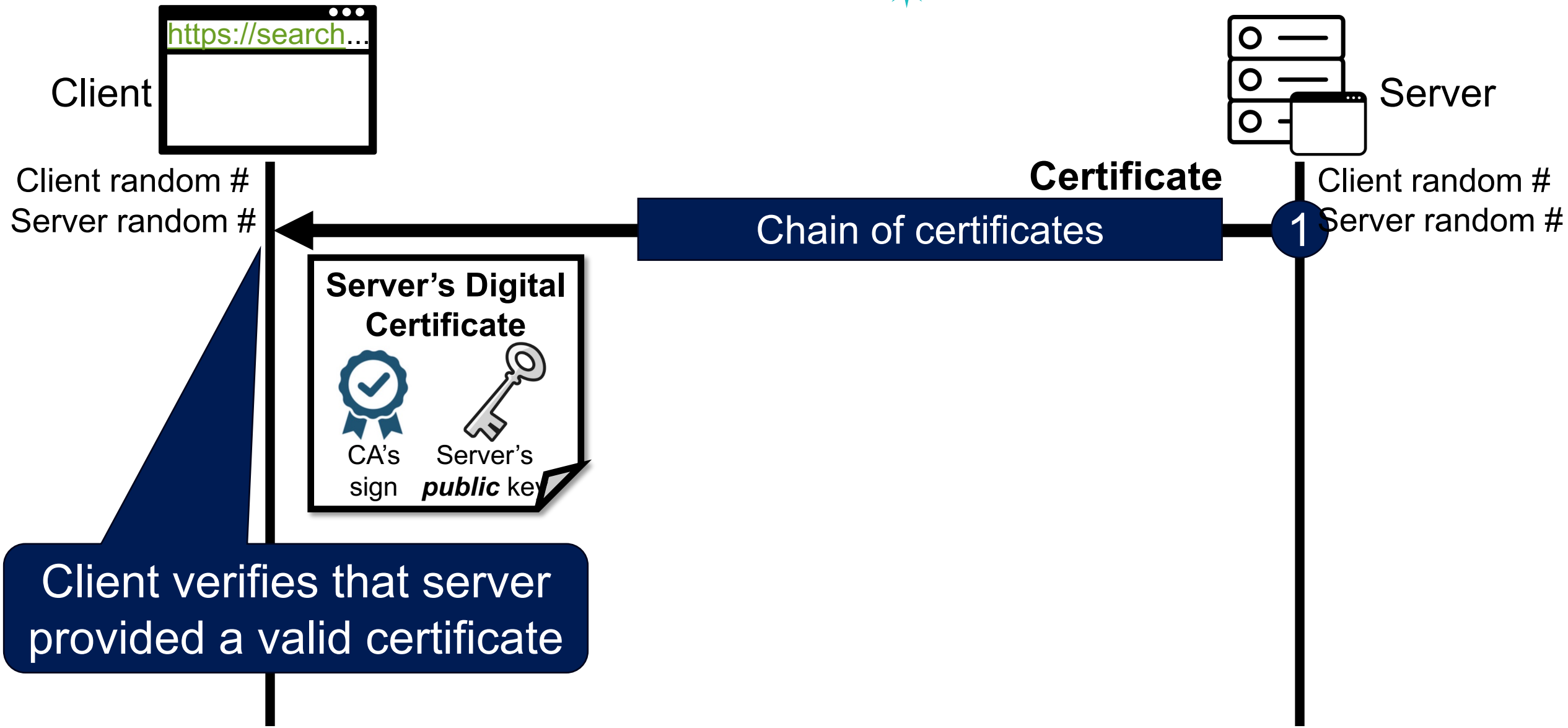
Phase 1:
Establishing security capabilities

Phase 2:
Server authentication and key exchange

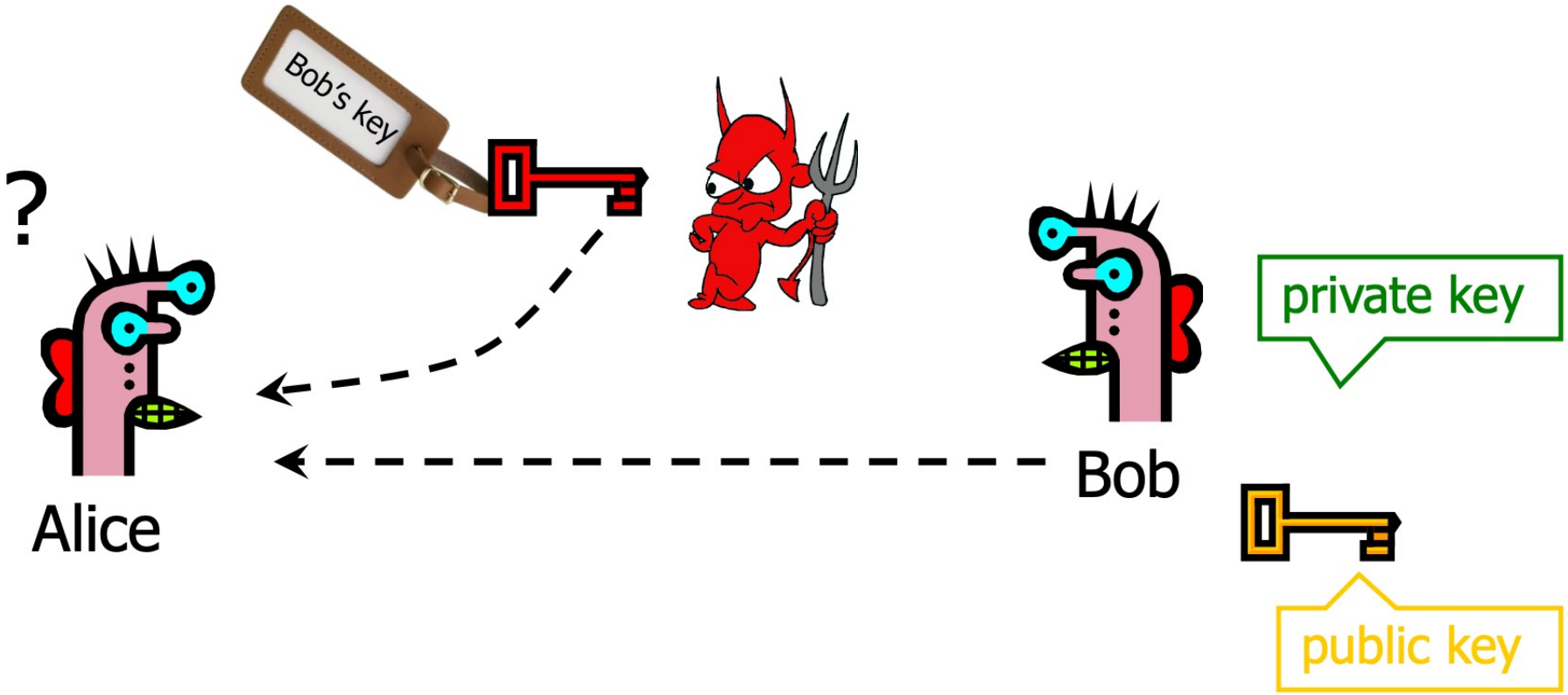
Phase 3:
Client authentication and key exchange

Phase 4:
Finalizing the handshake protocol

Phase 2: Server Auth. and Key Exchange



Motivational Question

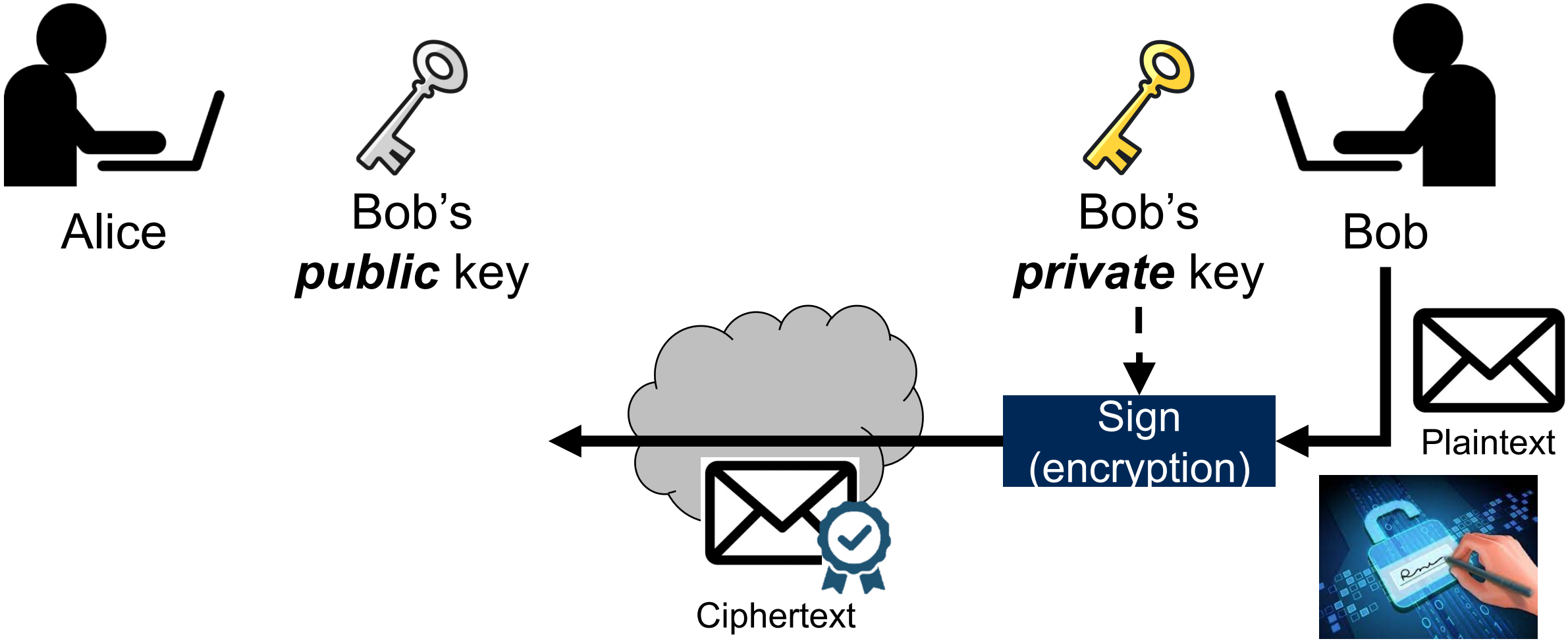


How does Alice know that the public key she received is **really Bob's public key**?

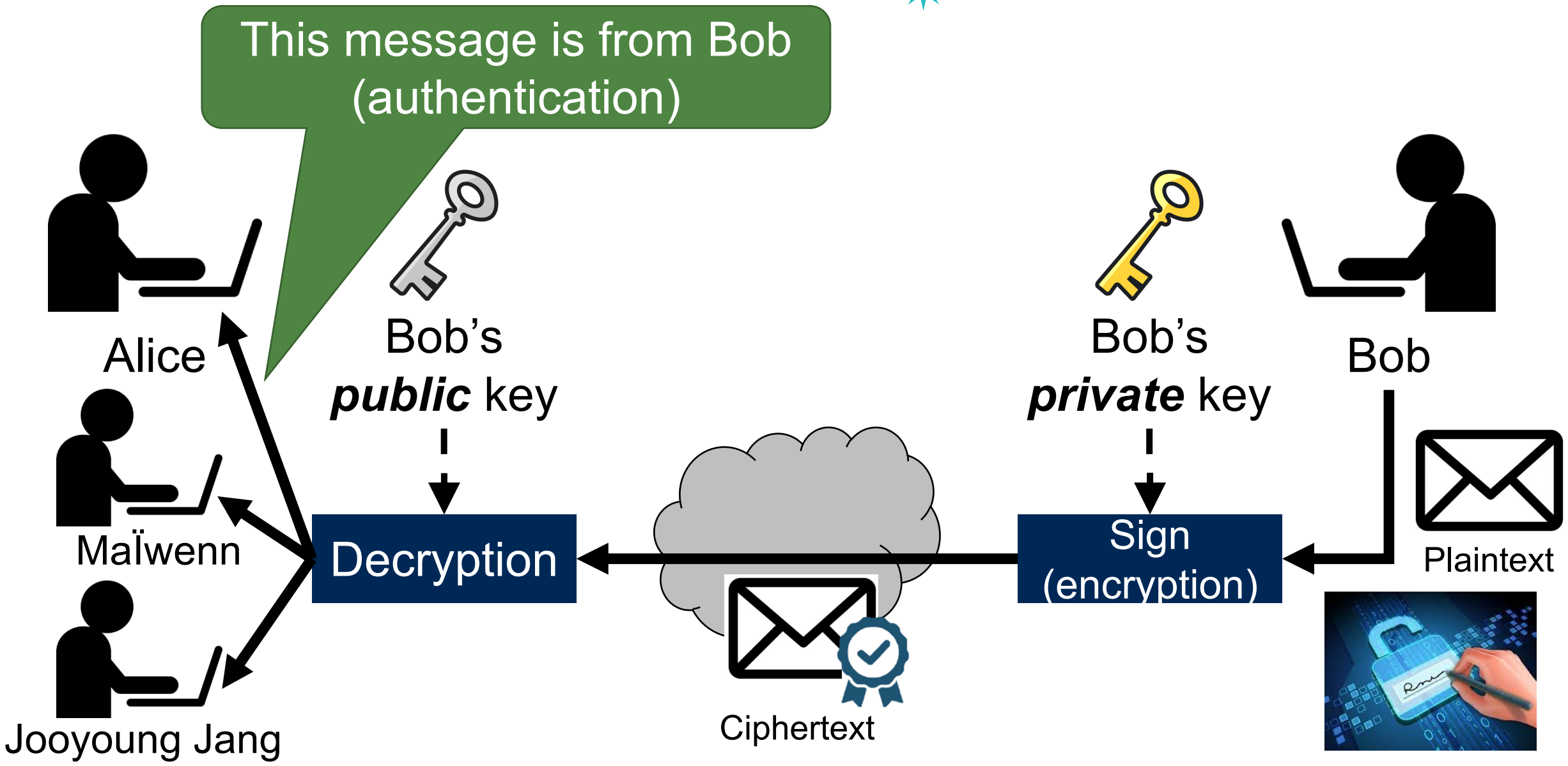
Digital Certificate

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

Digital Signature

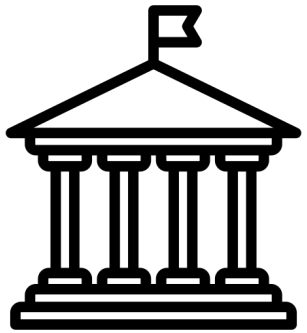


Digital Signature



Digital Certificate

Signing



Certificate Authority (CA)

Digital Certificate

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

Hash function

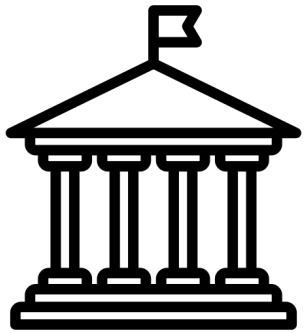
0101000010..

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

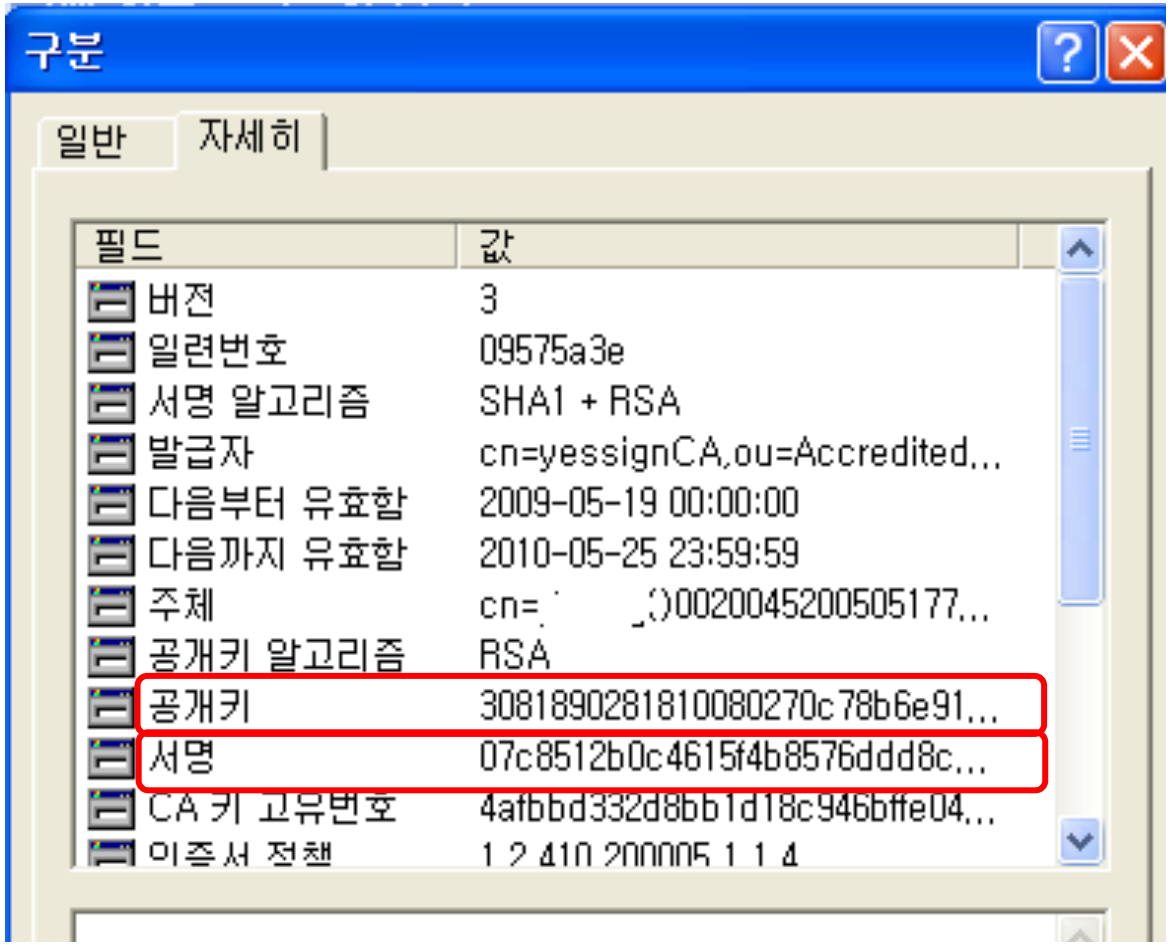
X.509 Certificate

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

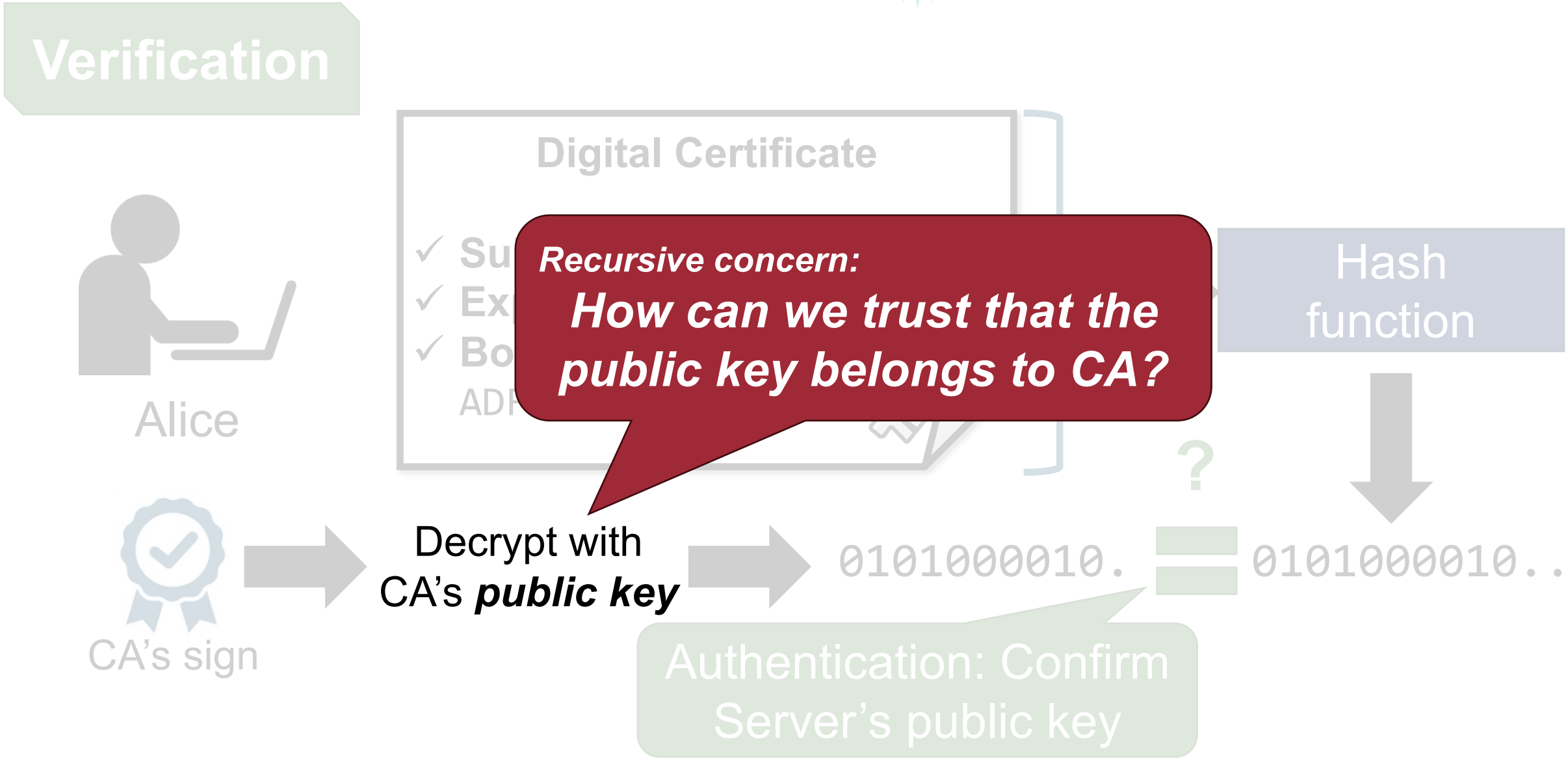
version 1

version 2

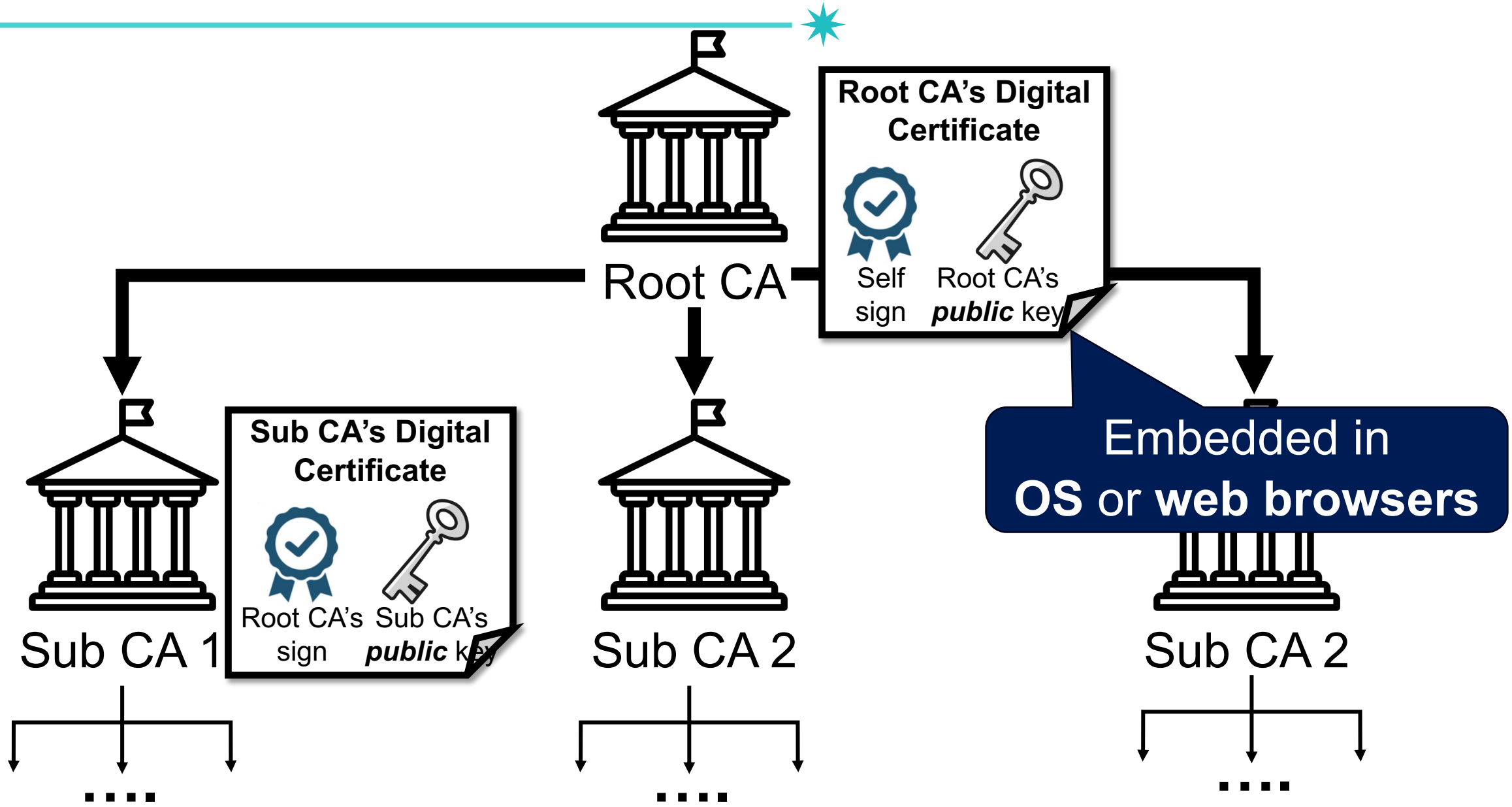
version 3



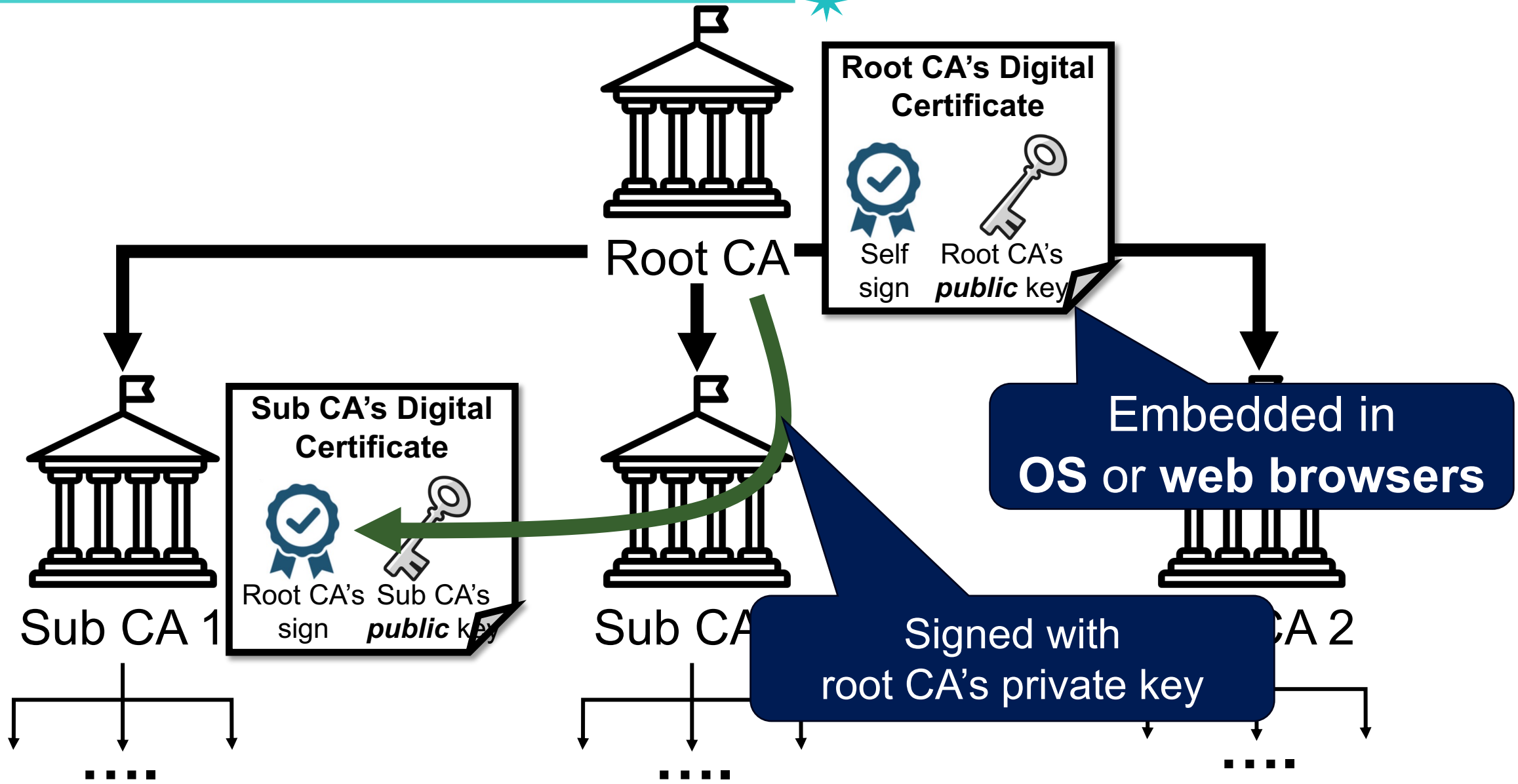
Recursive Concern



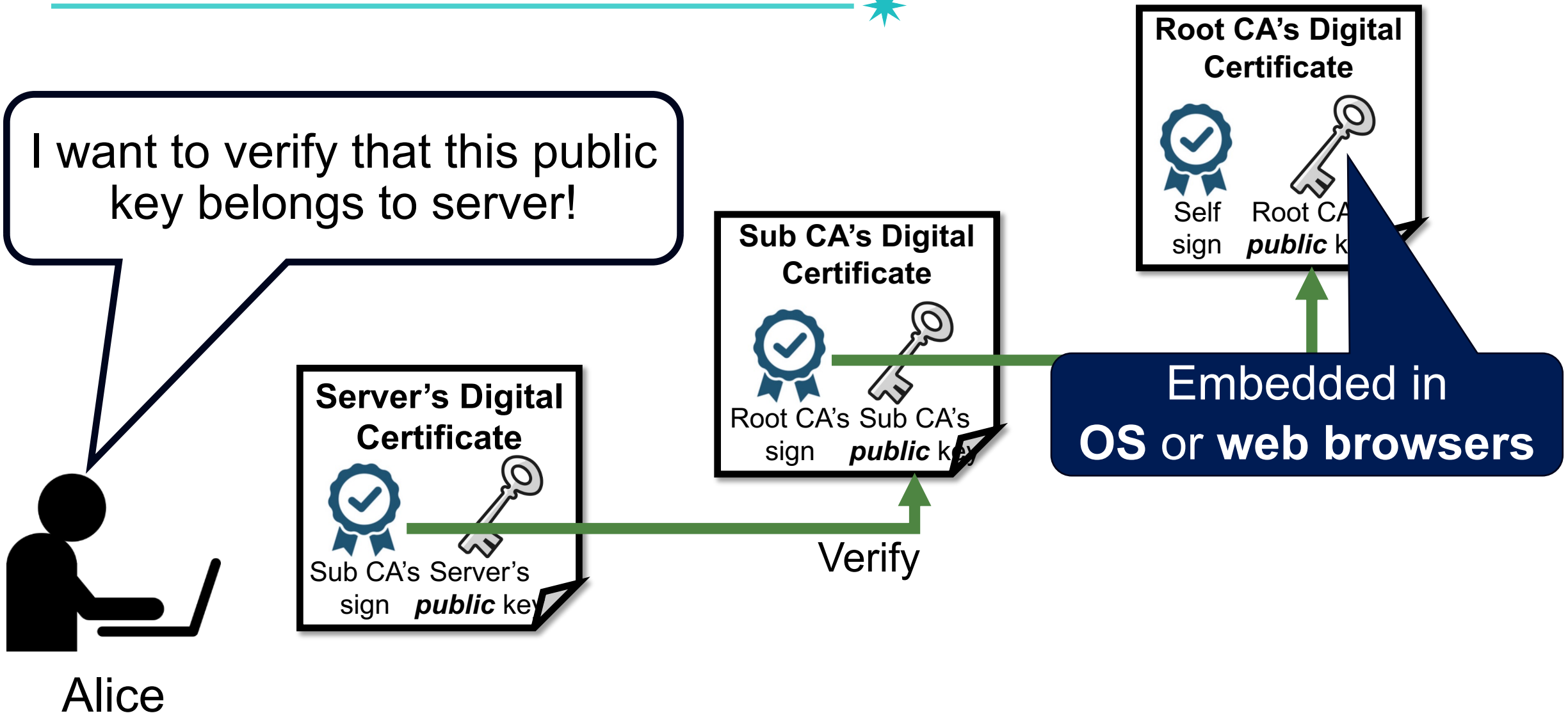
Chain of Trust



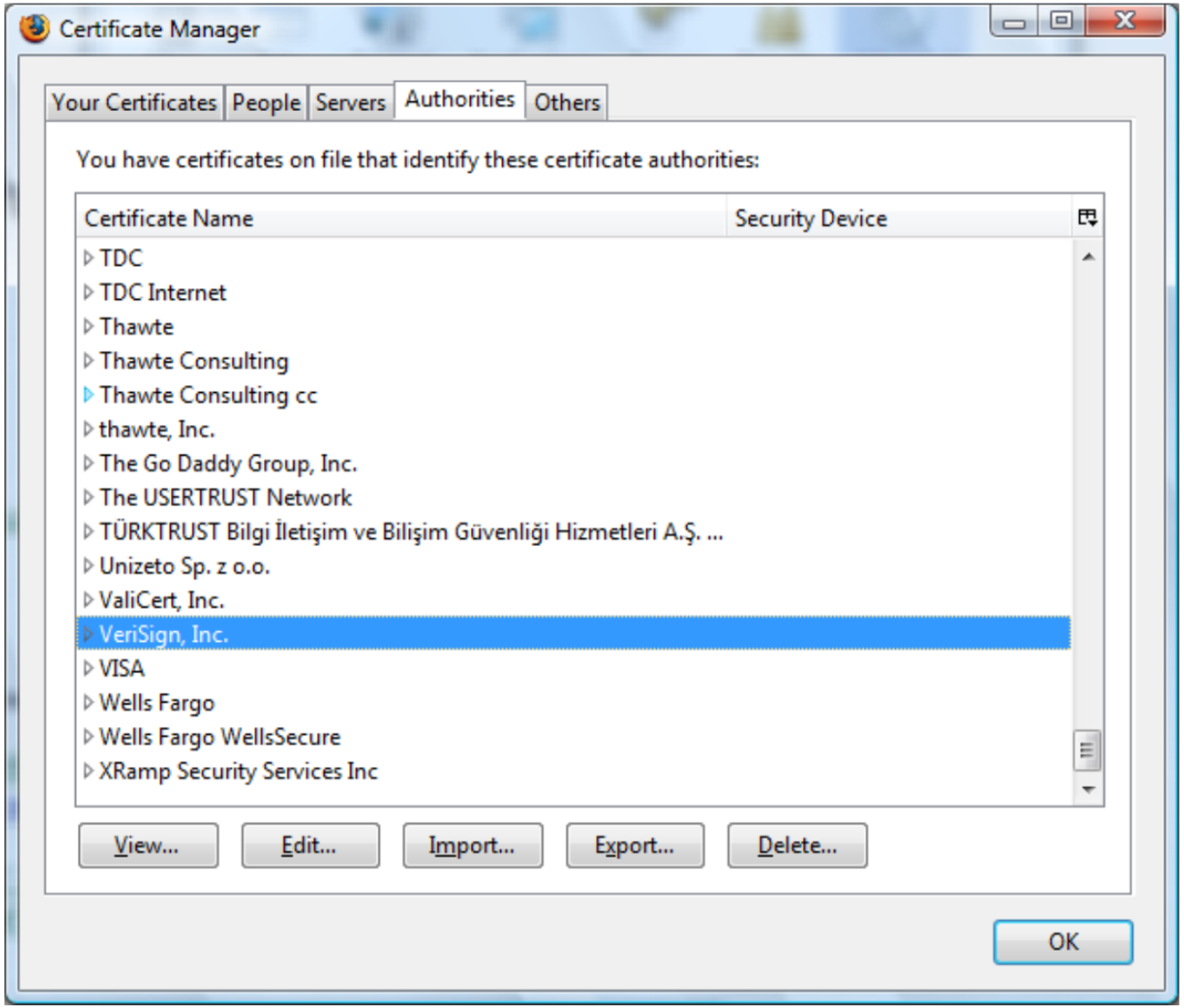
Chain of Trust



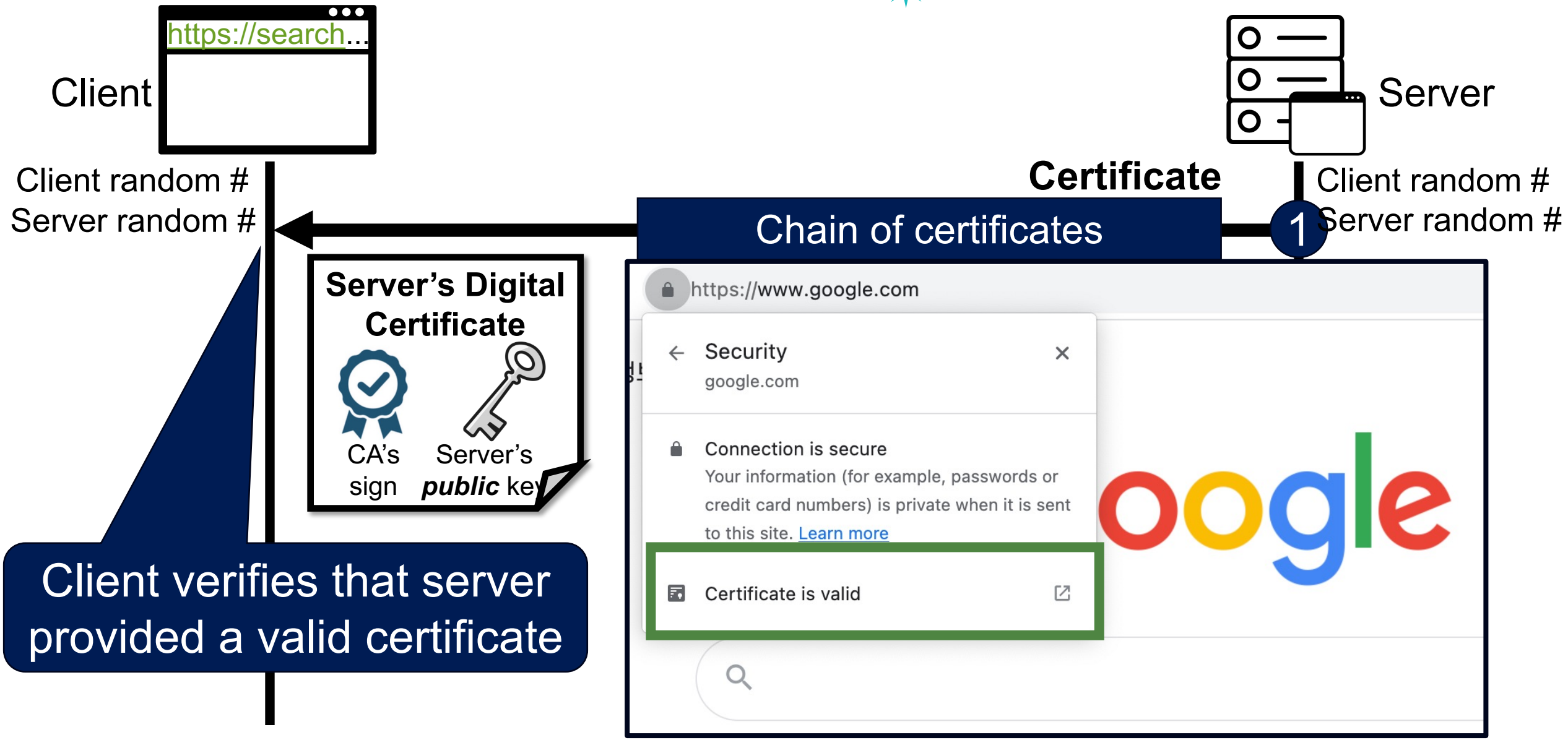
Chain of Trust



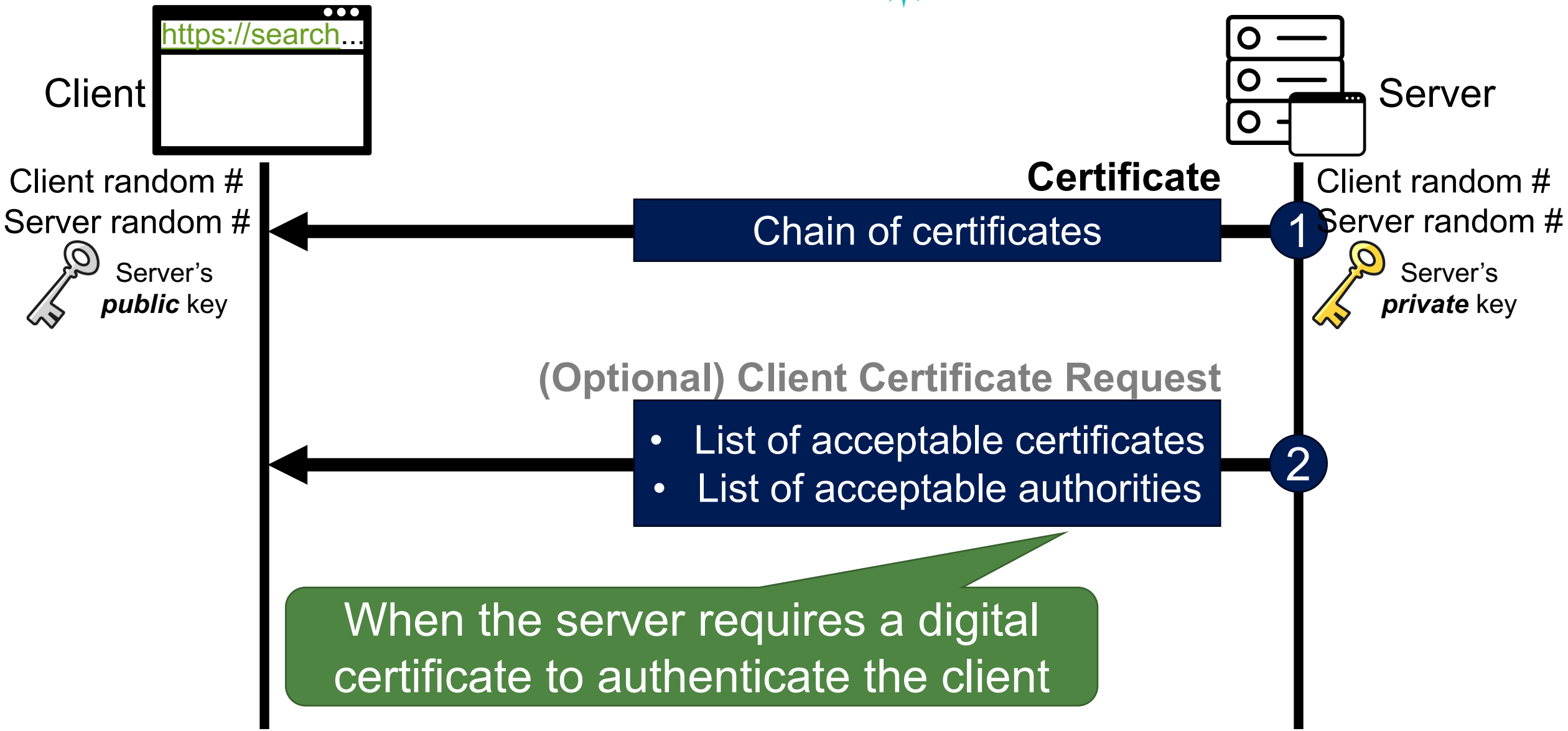
Browsers are Pre-configured with 100+ Trusted CAs



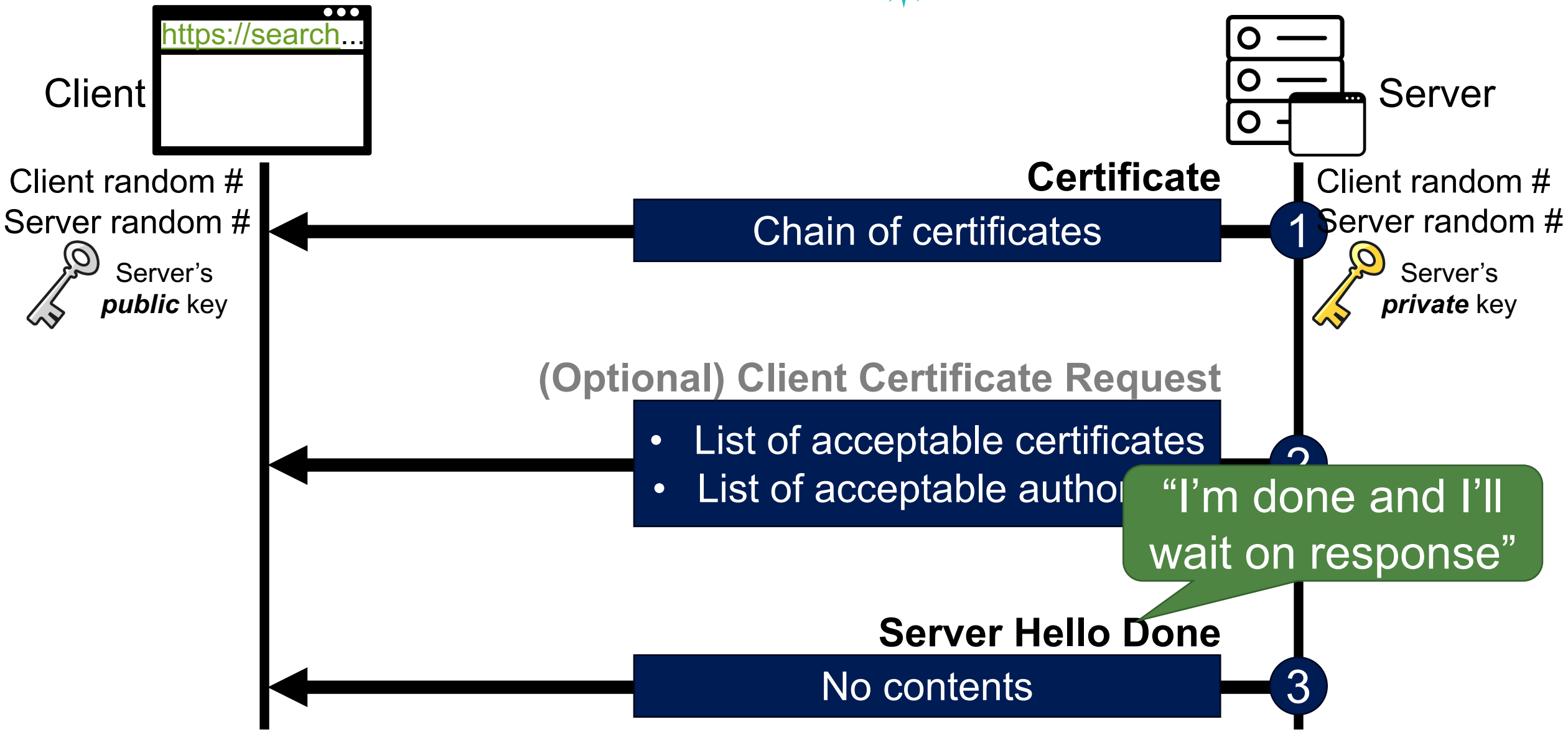
Phase 2: Server Auth. and Key Exchange



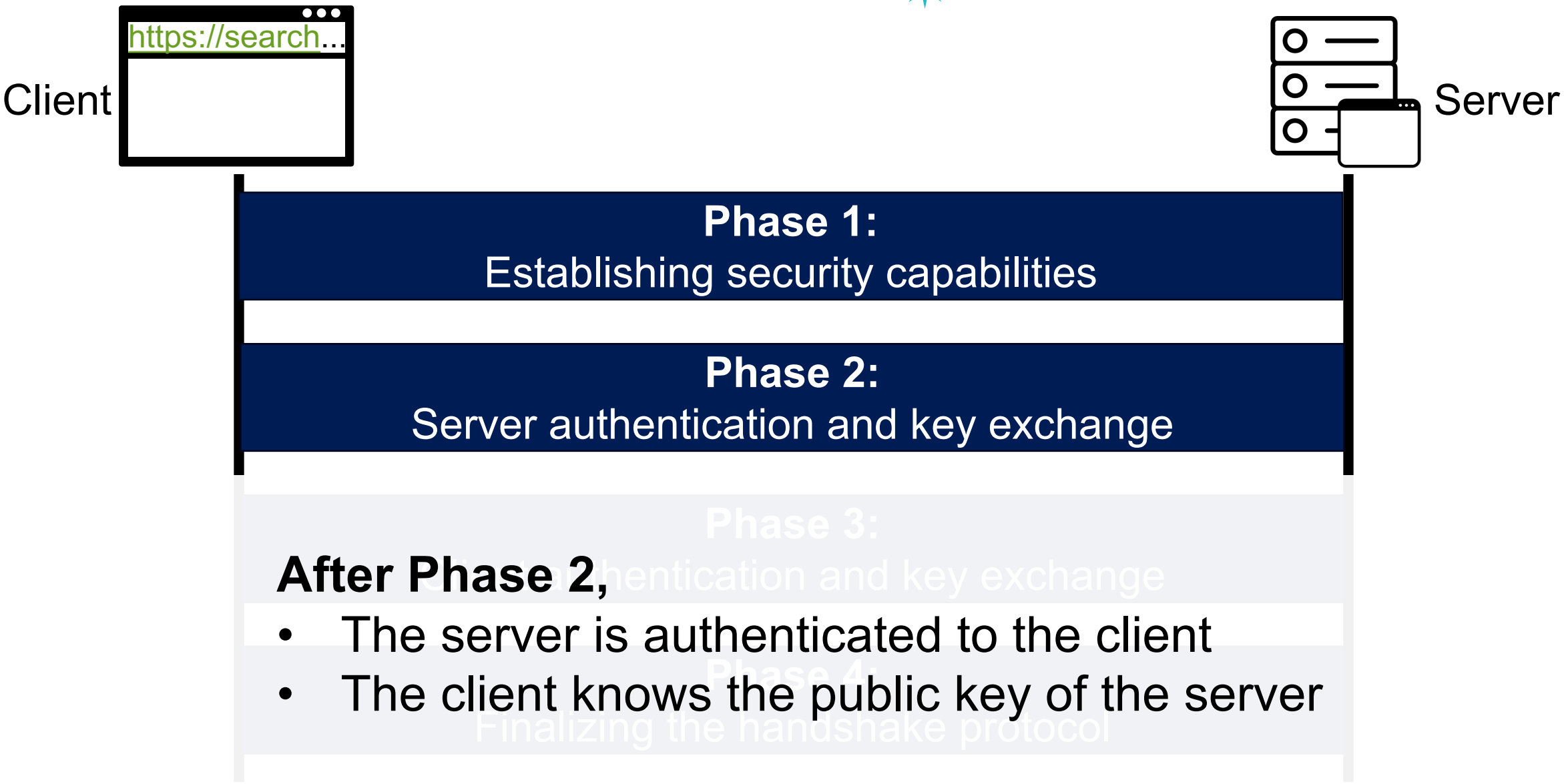
Phase 2: Server Auth. and Key Exchange



Phase 2: Server Auth. and Key Exchange



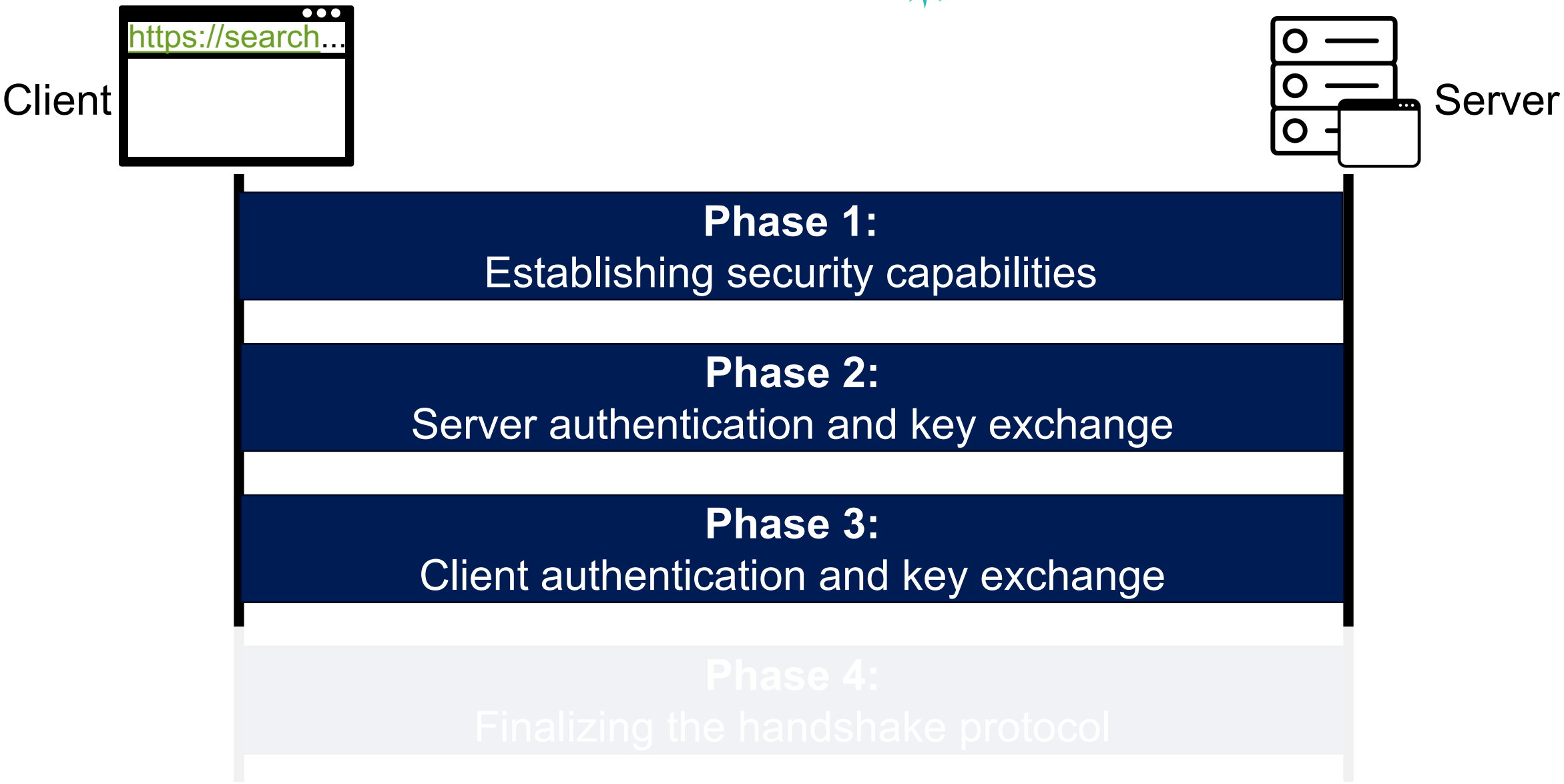
Phase 1: Establishing Security Capabilities



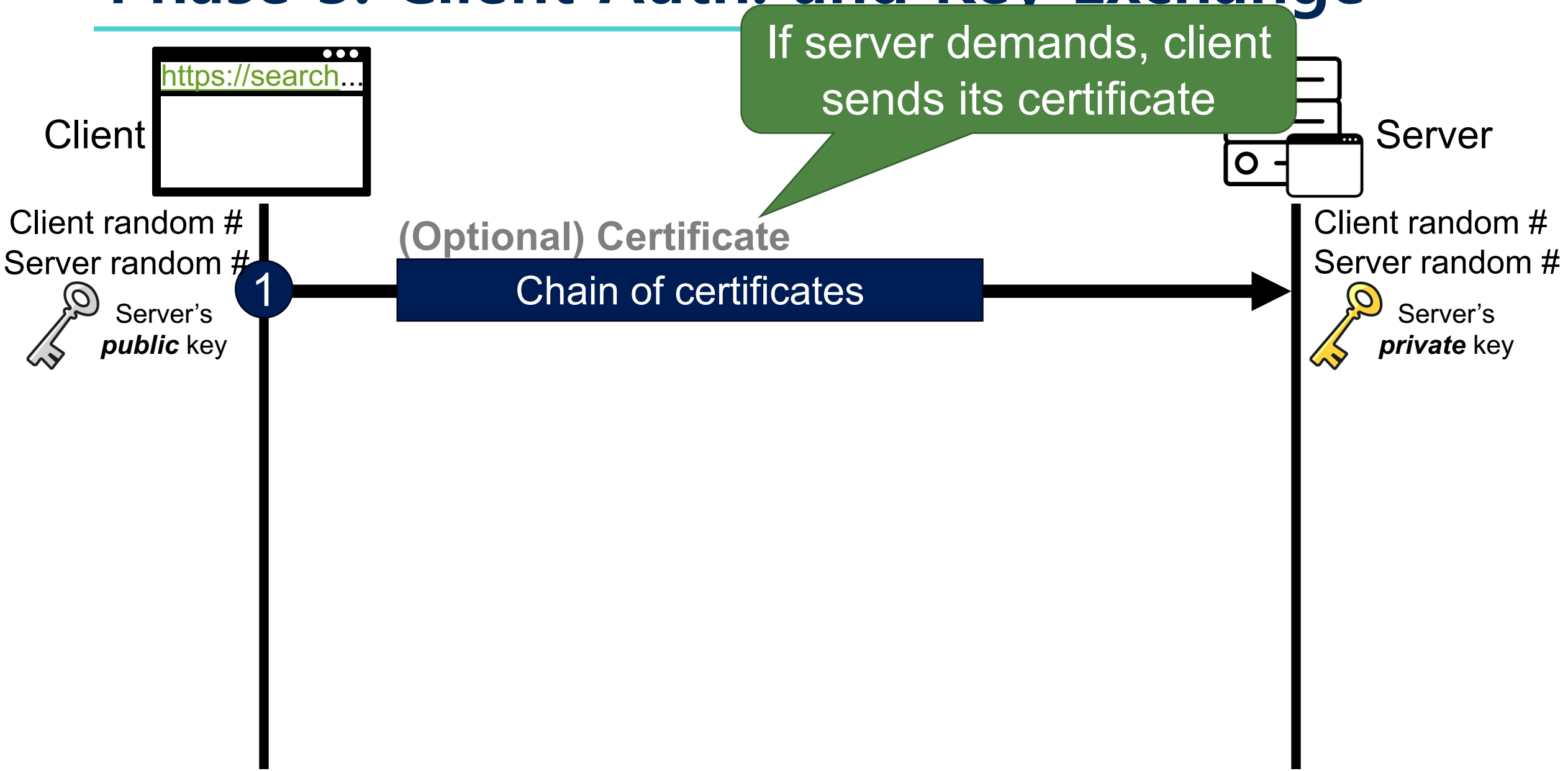
After Phase 2,

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

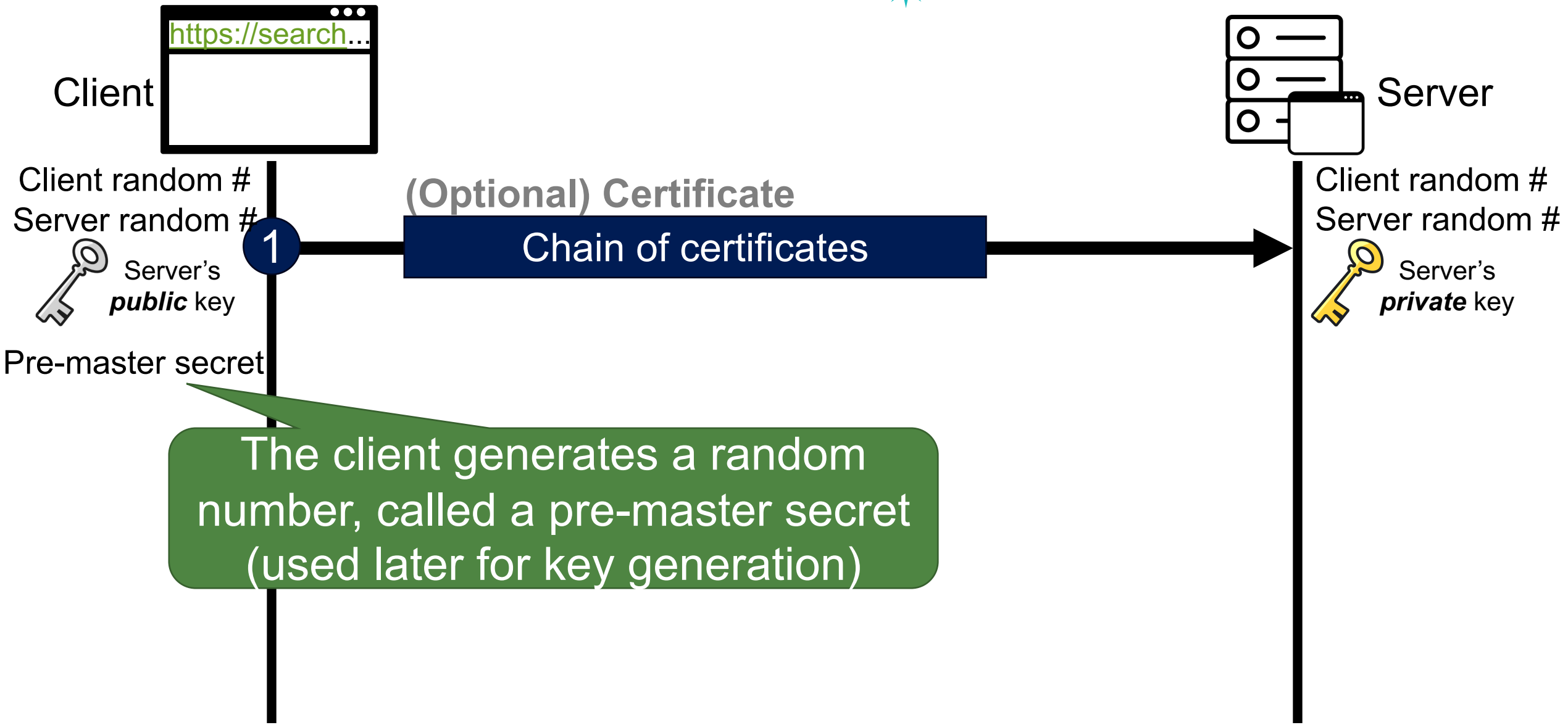
Phase 3: Client Auth. and Key Exchange



Phase 3: Client Auth. and Key Exchange

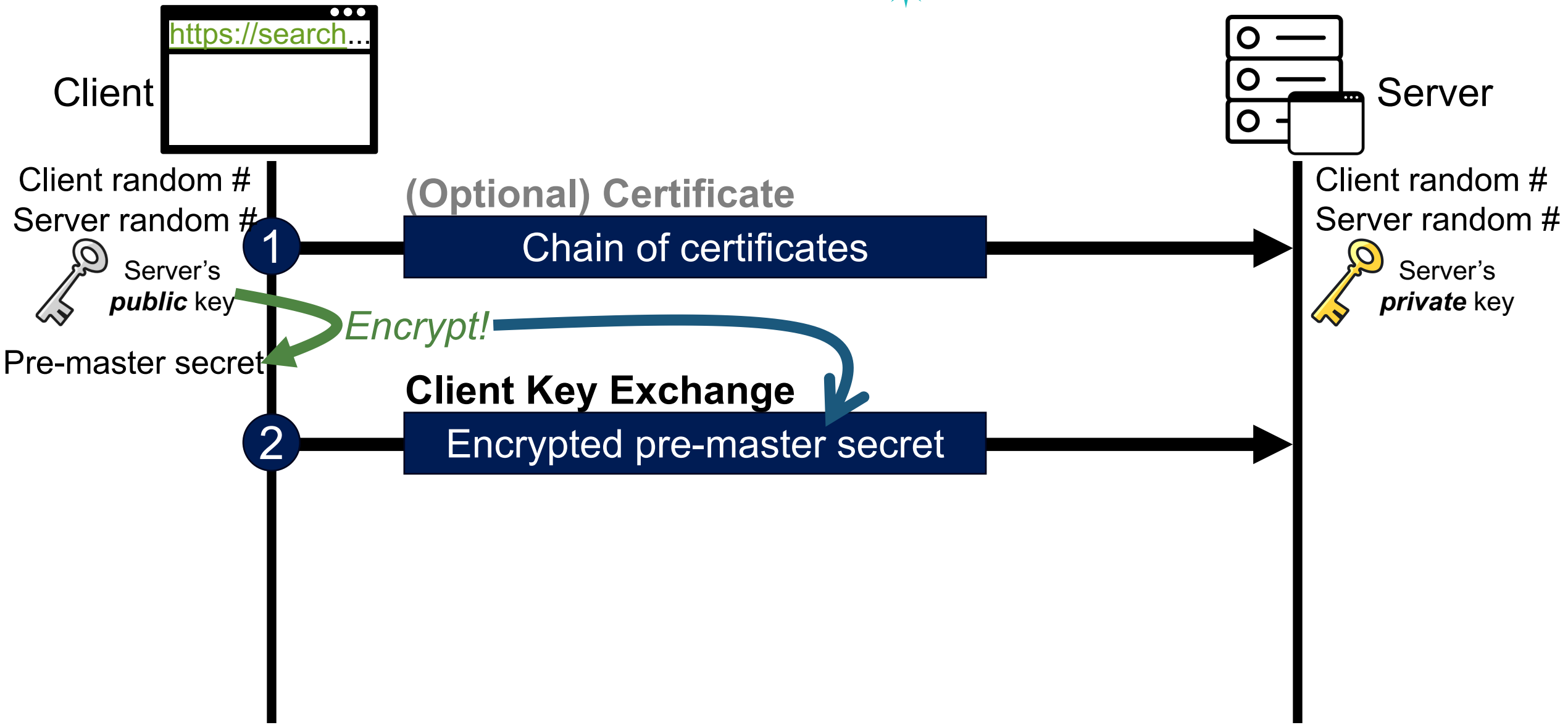


Phase 3: Client Auth. and Key Exchange

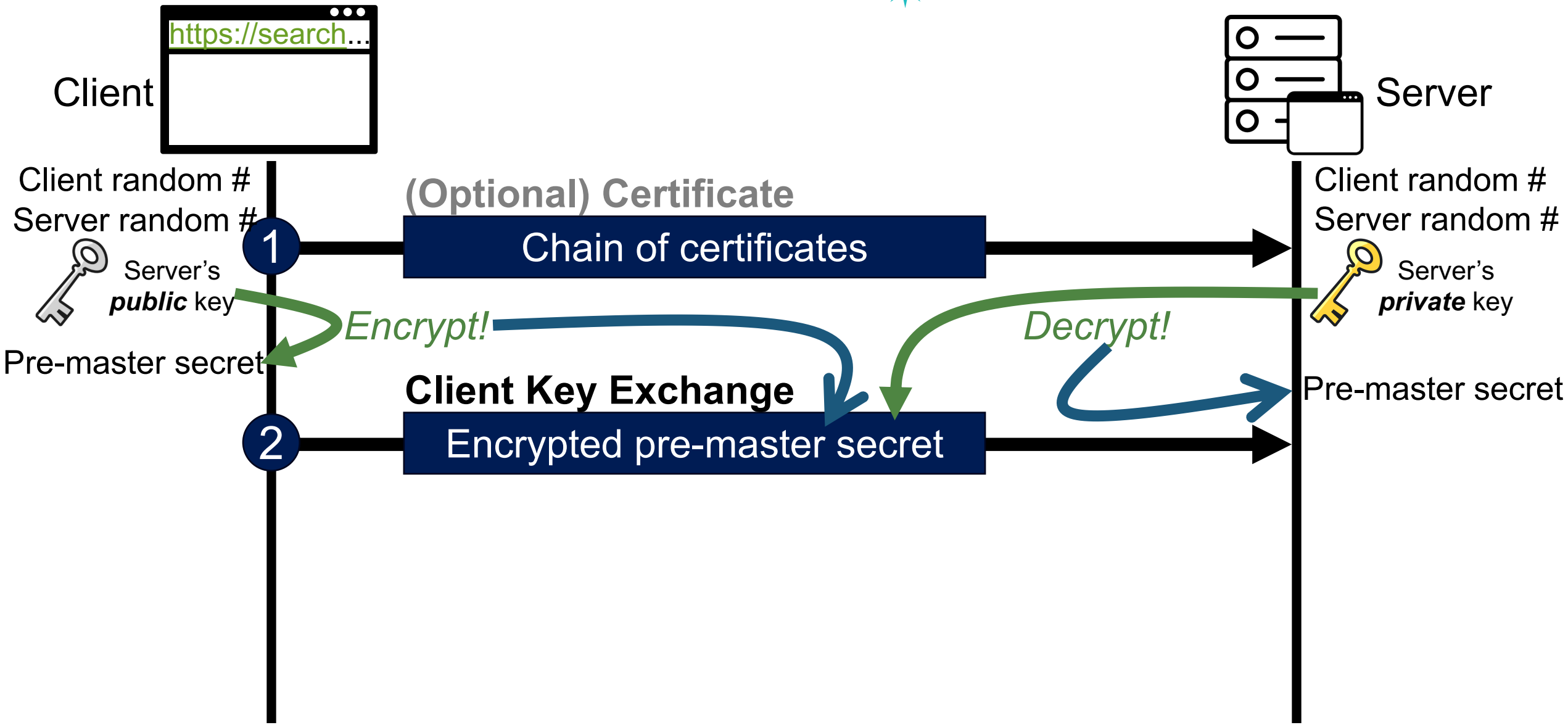


The client generates a random number, called a pre-master secret (used later for key generation)

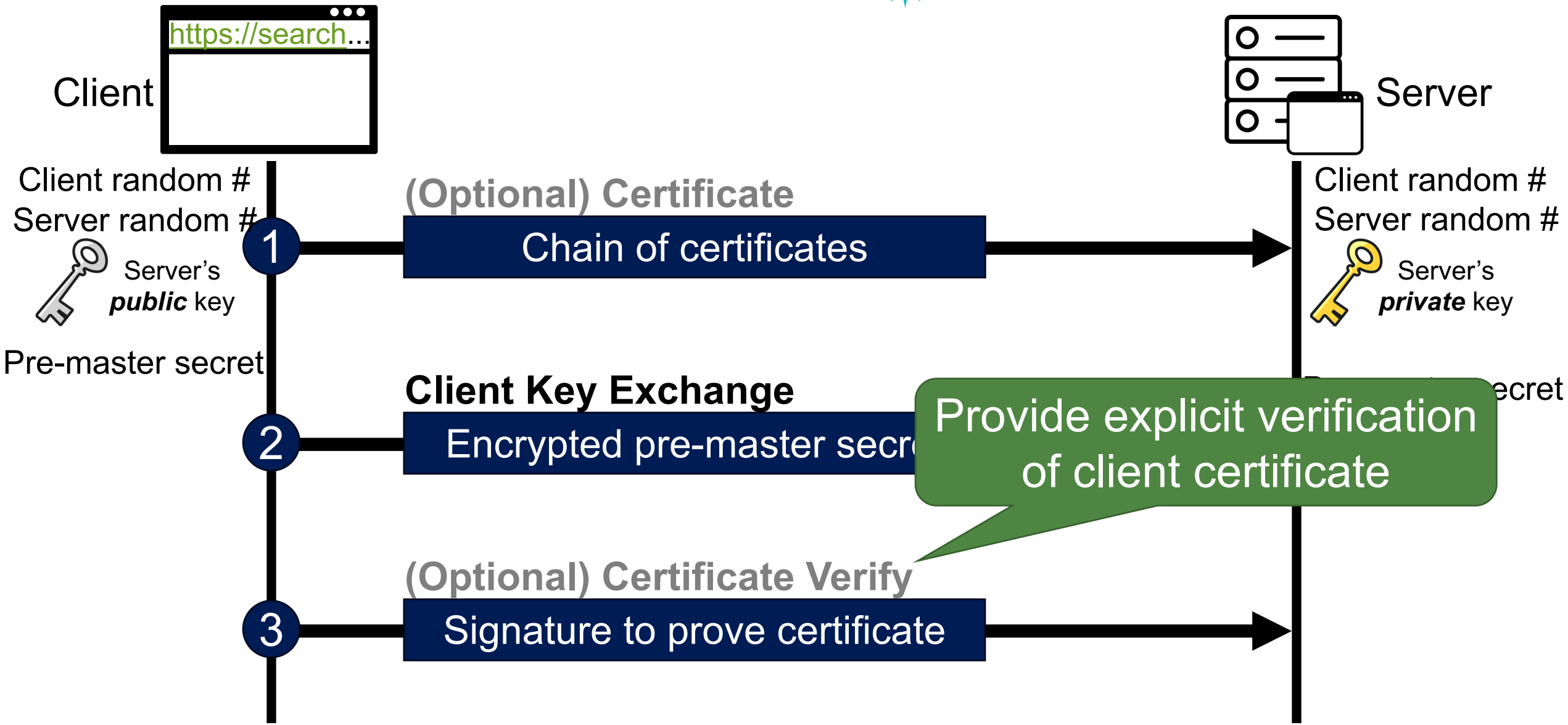
Phase 3: Client Auth. and Key Exchange



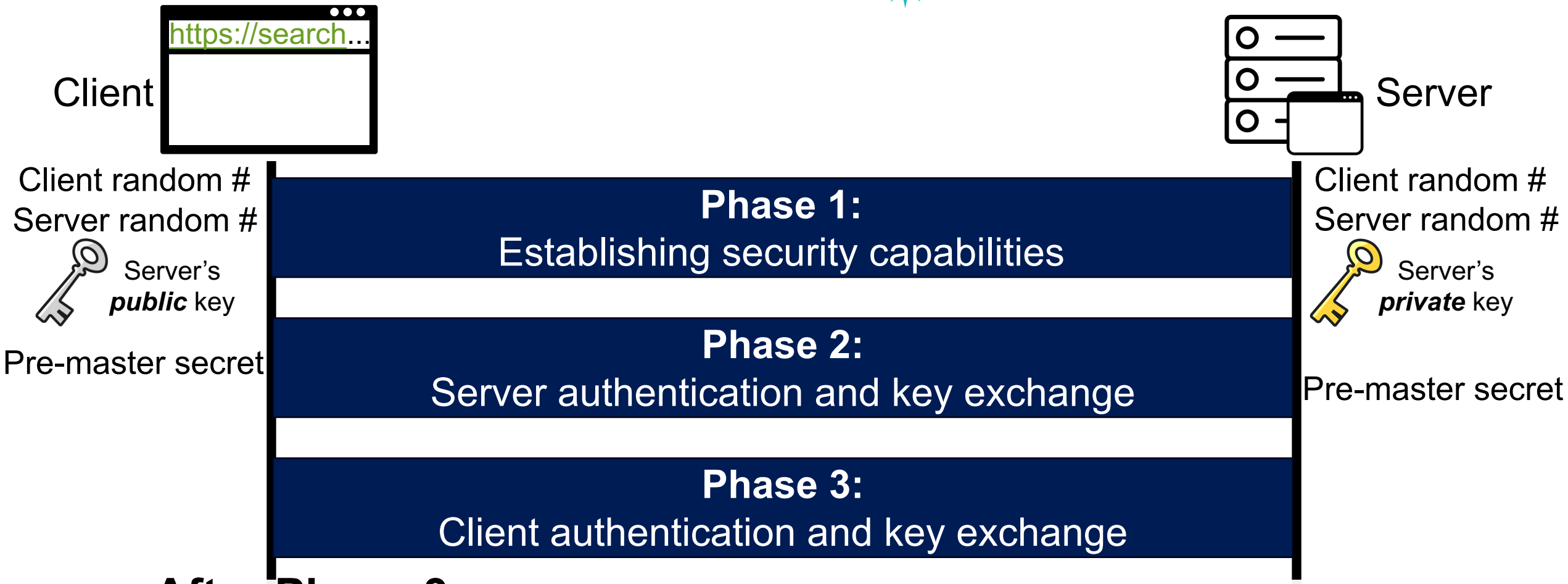
Phase 3: Client Auth. and Key Exchange



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

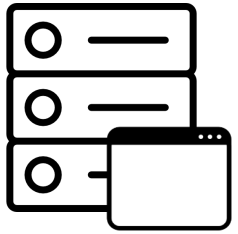
Phase 3: Client Auth. and Key Exchange

Client

Client random #
Server random #

Server's ***public*** key

Pre-master secret



Server

Client random #
Server random #

Server's ***private*** key

Pre-master secret

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

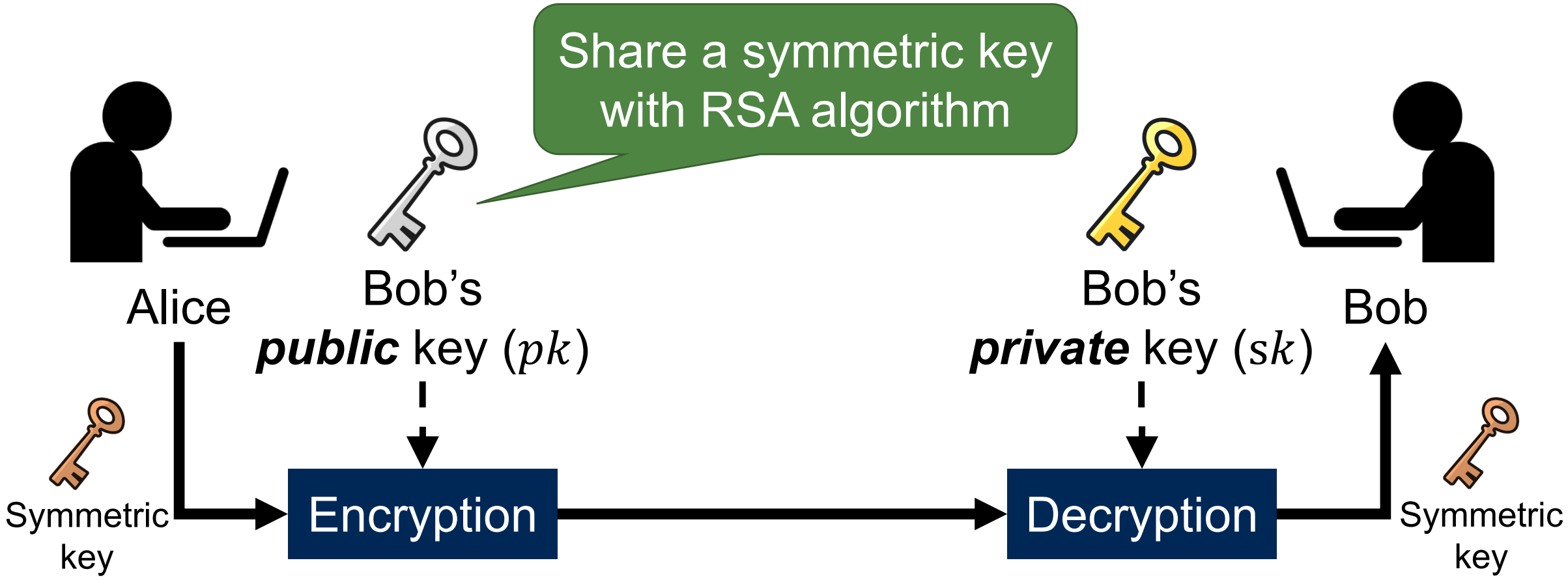
Why do we need a symmetric key
even though we already have asymmetric key?

Pros & Cons: Asymmetric-key Cryptography

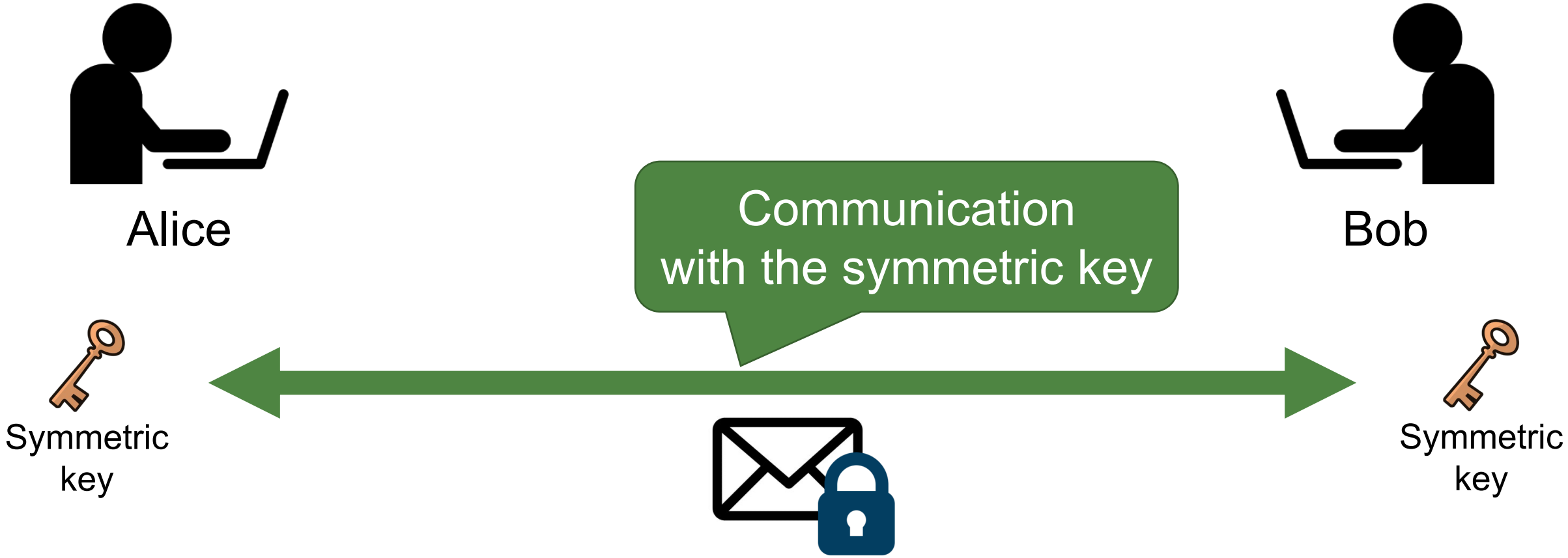
- Pros
  - No need to share a secret
  - Enable multiple senders to communicate privately with a single receiver
  - More applications: Digital sign

- Cons
  - **Slower in general**: due to the larger key
    - Roughly 2-3 orders of magnitude slower

Solution: Combination of Two Schemes



Solution: Combination of Two Schemes



Phase 3: Client Auth. and Key Exchange

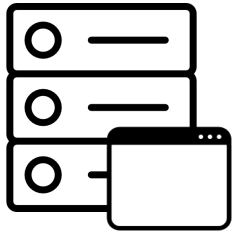
Client

Client random #
Server random #

Server's ***public*** key

Pre-master secret

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



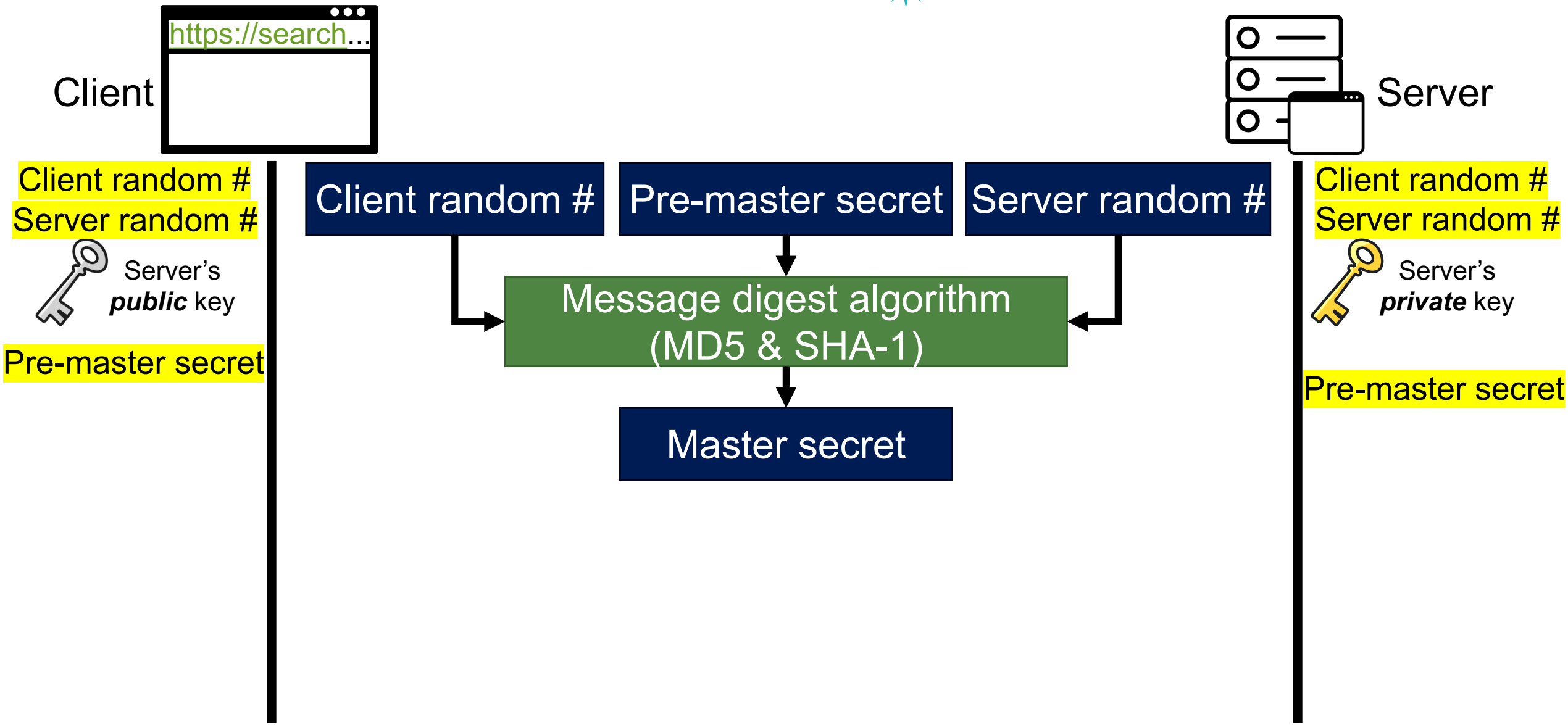
Server

Client random #
Server random #

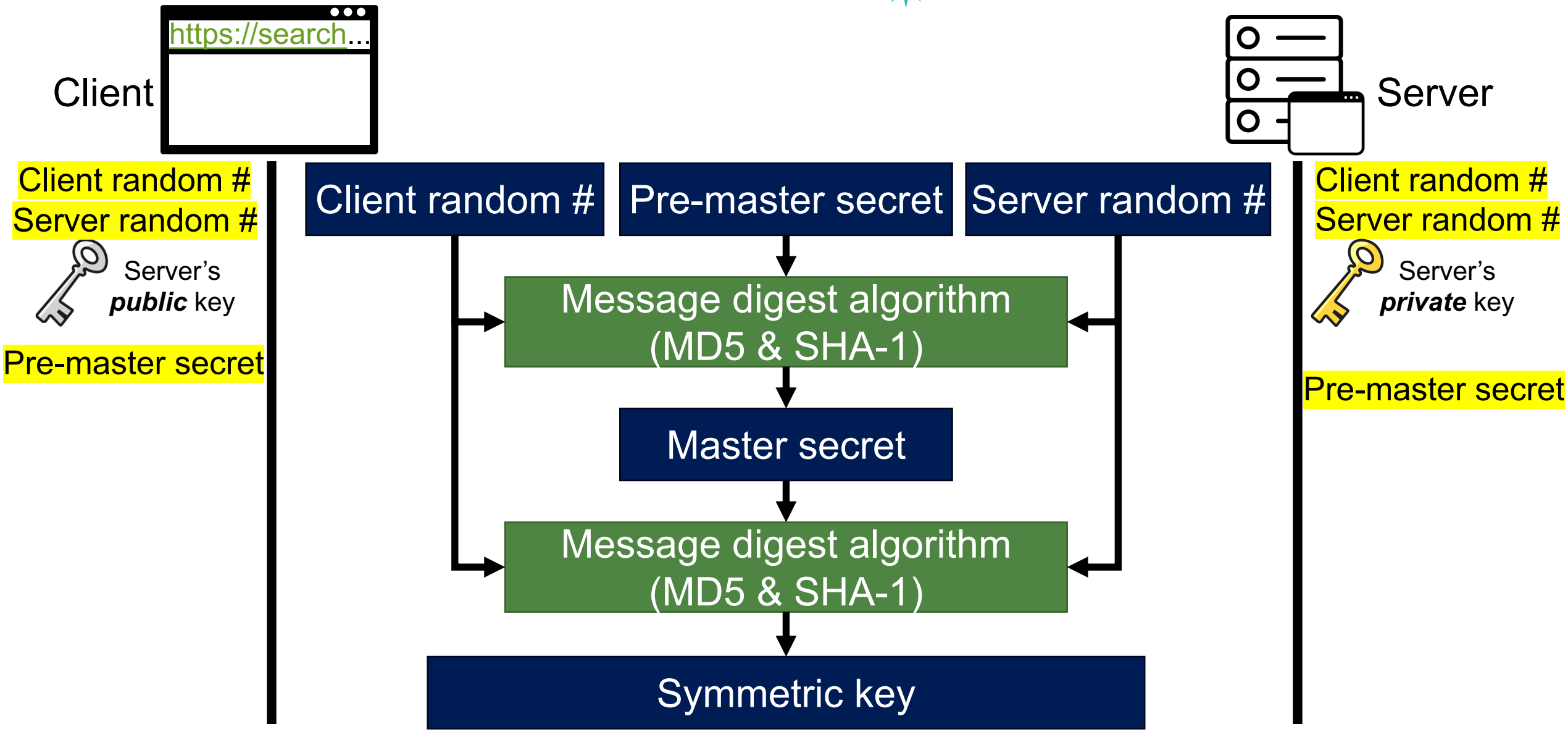
Server's ***private*** key

Pre-master secret

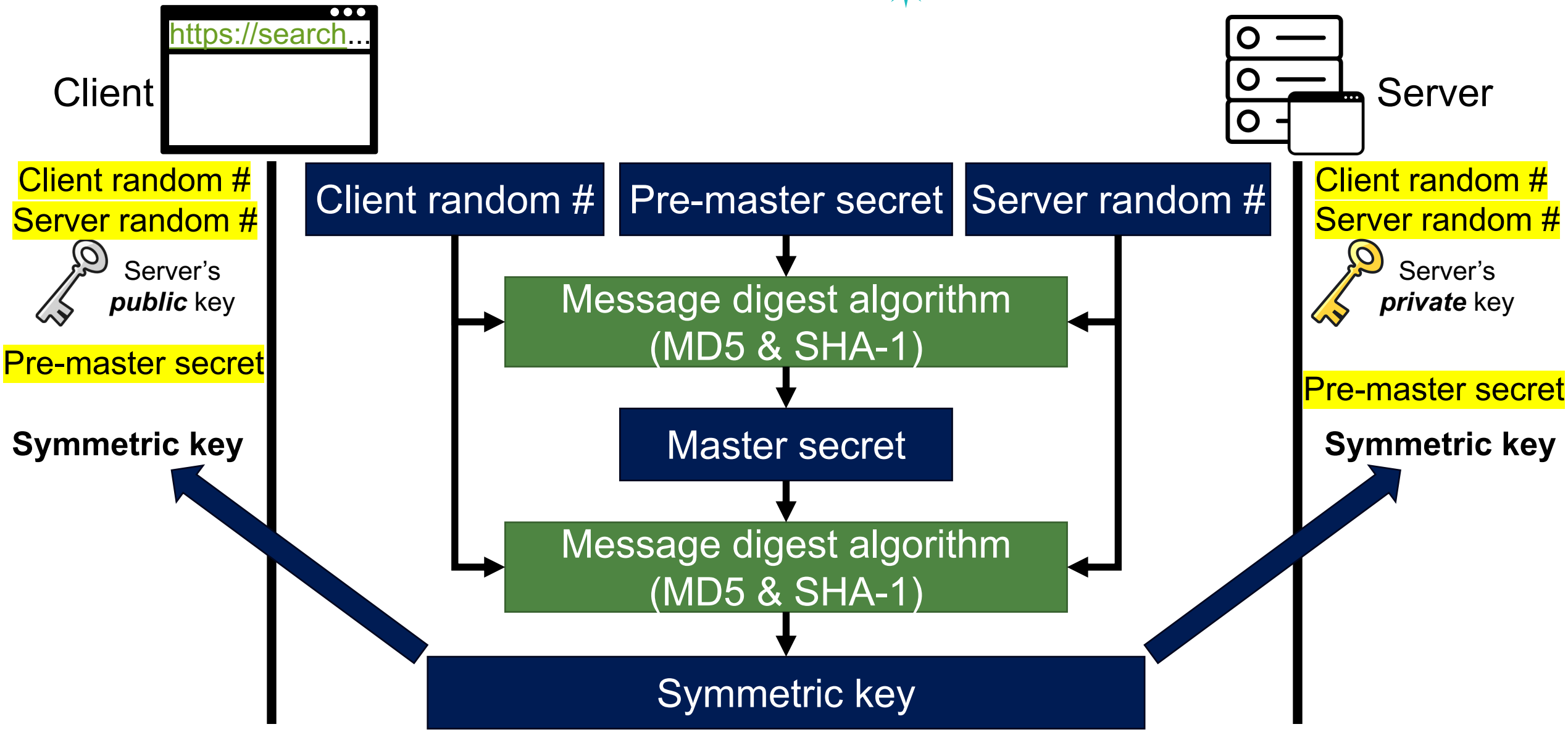
Calculation of Master Secret



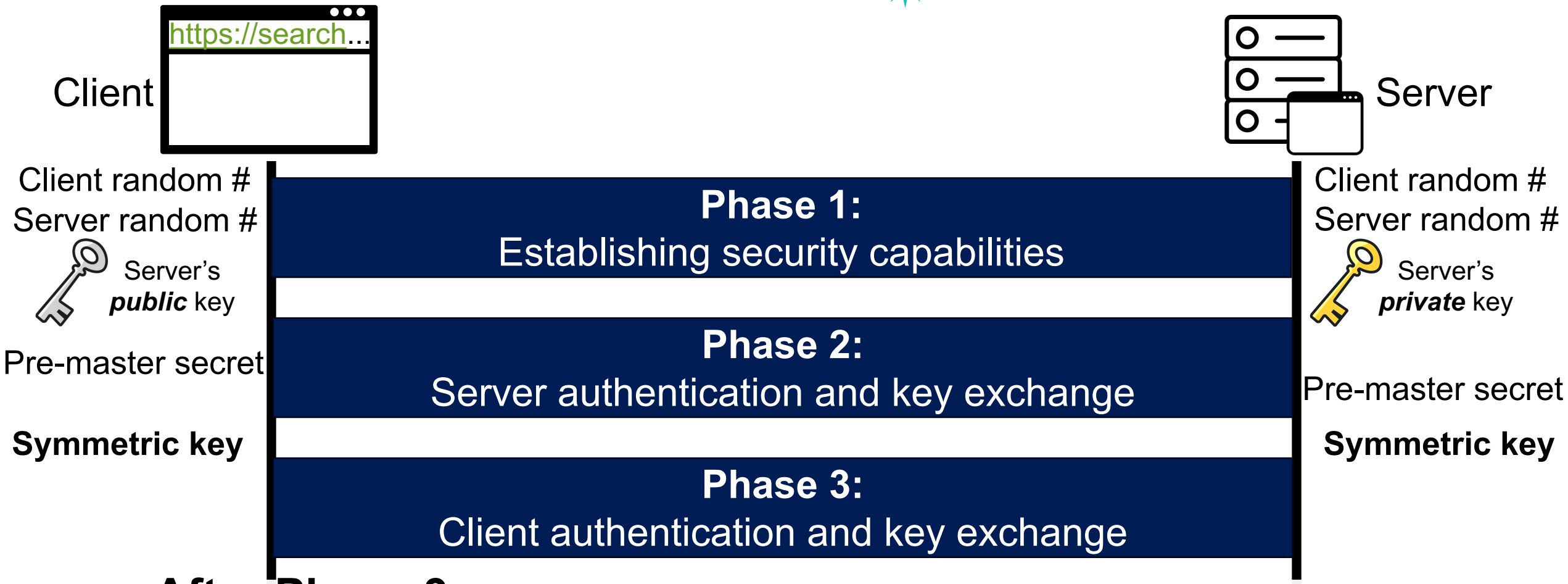
Calculation of Symmetric Key



Calculation of Symmetric Key



Phase 3: Client Auth. and Key Exchange

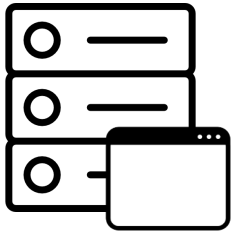


After Phase 3,

- (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



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

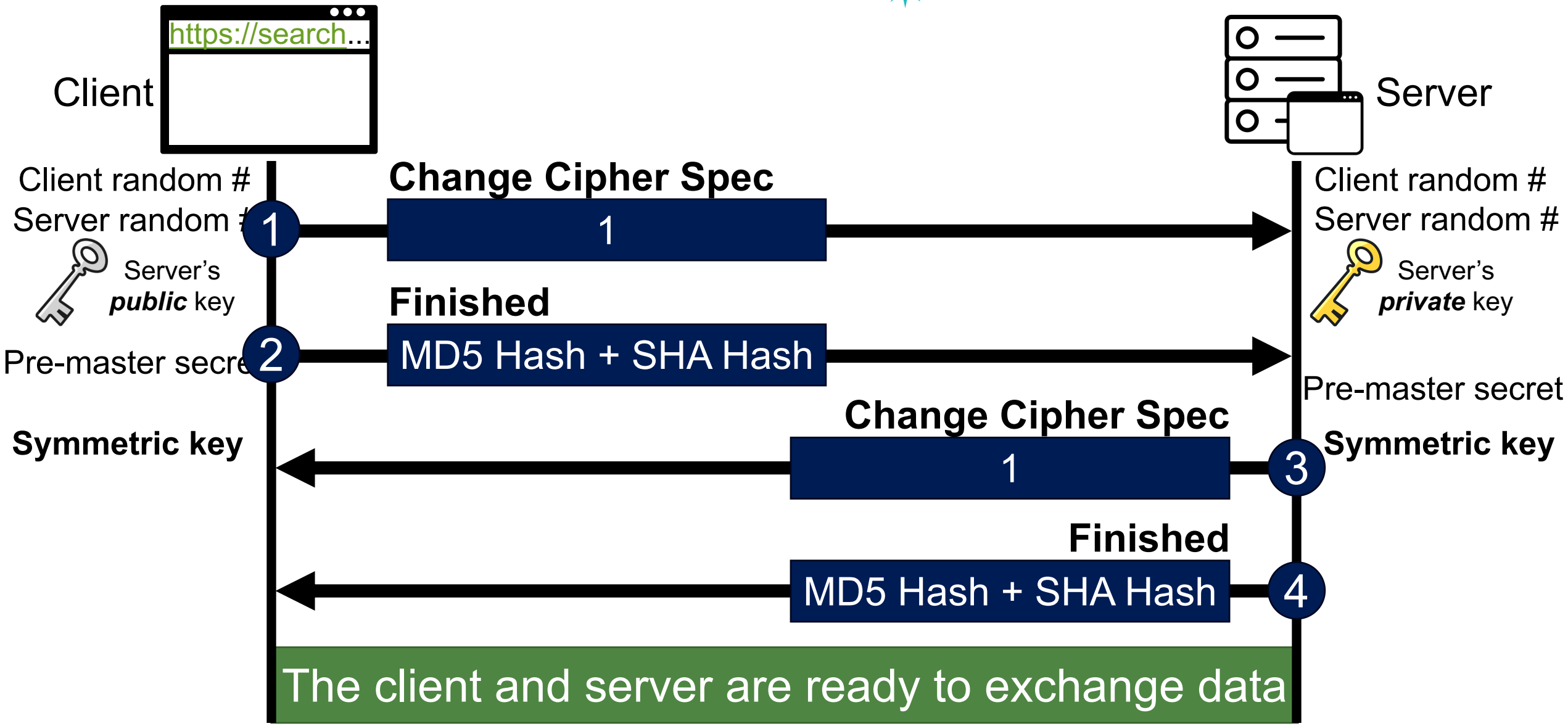
Client random #
Server random #

Server's ***private*** key

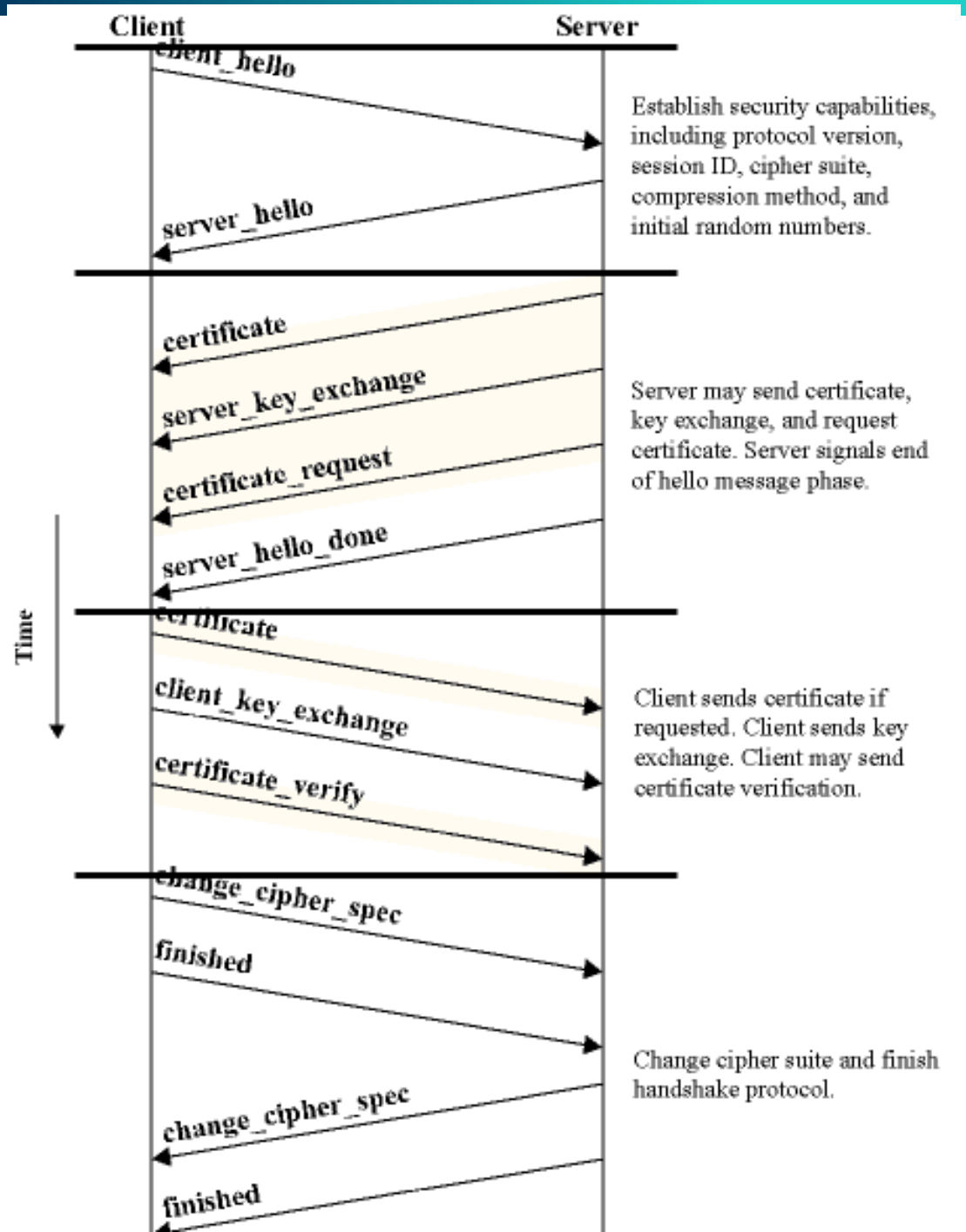
Pre-master secret

Symmetric key

Phase 4: Finalizing the Handshake Protocol



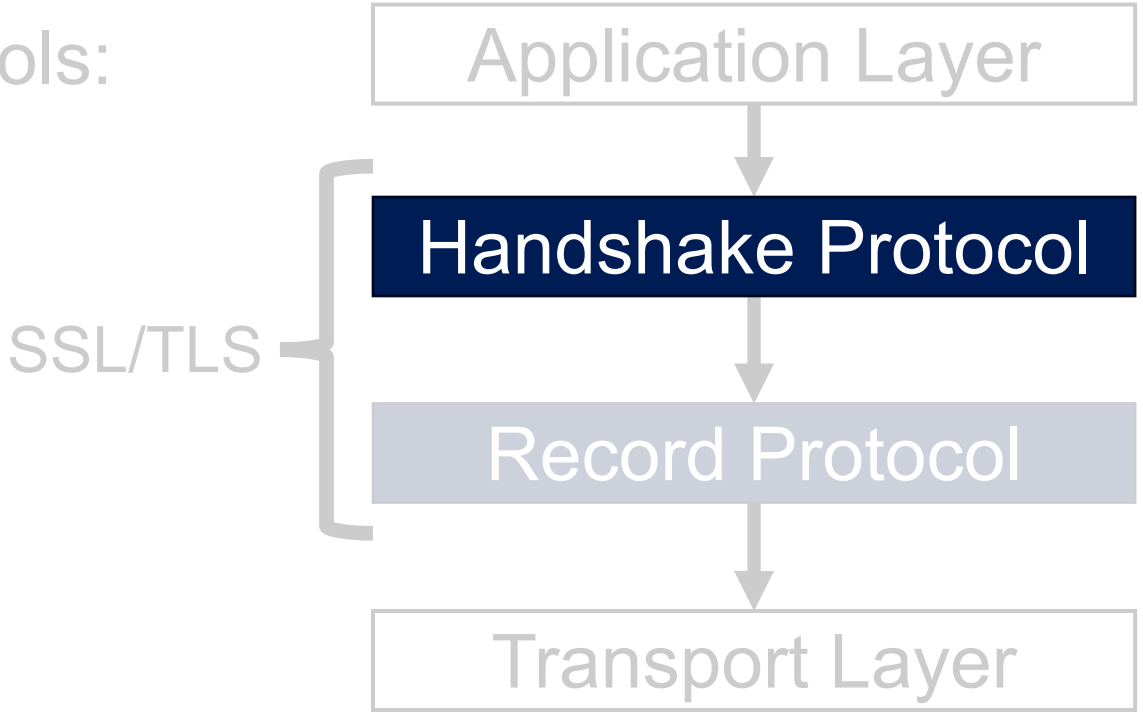
Handshake Protocol Summary



SSL/TLS Basics

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

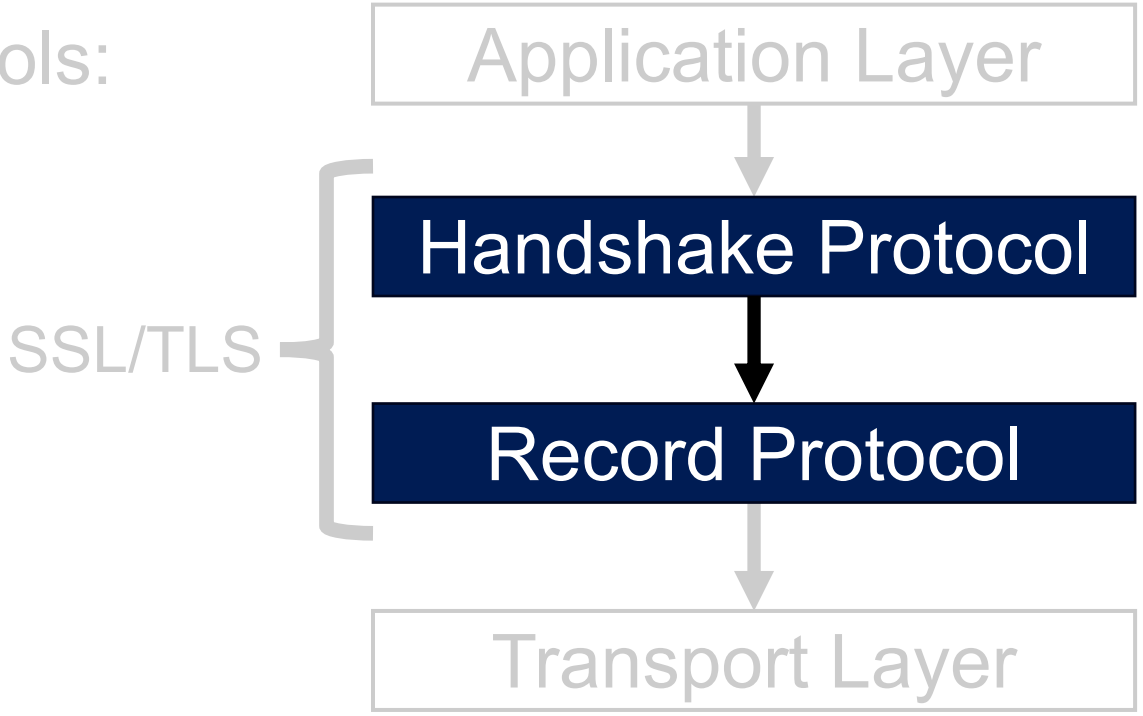
- Composed of two layers of protocols:
  1. Handshake protocol
  2. Record protocol



SSL/TLS Basics

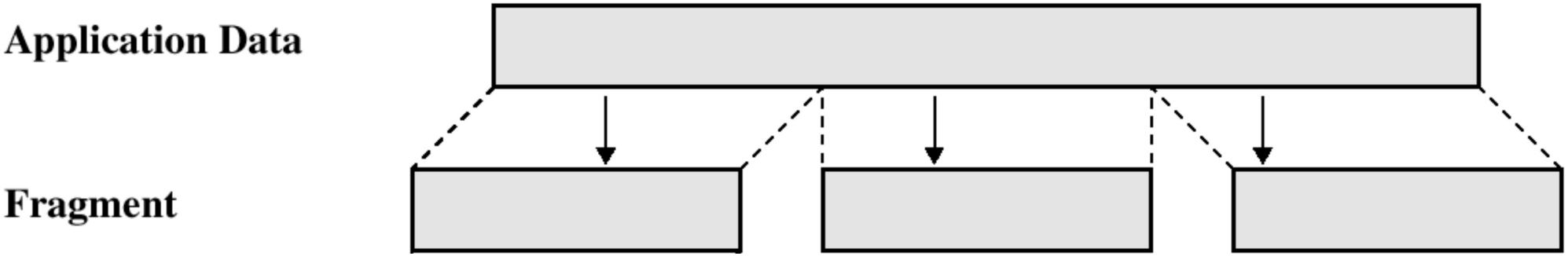
- Runs in the presentation layer
- Uses symmetric crypto, asymmetric crypto, and digital signatures

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

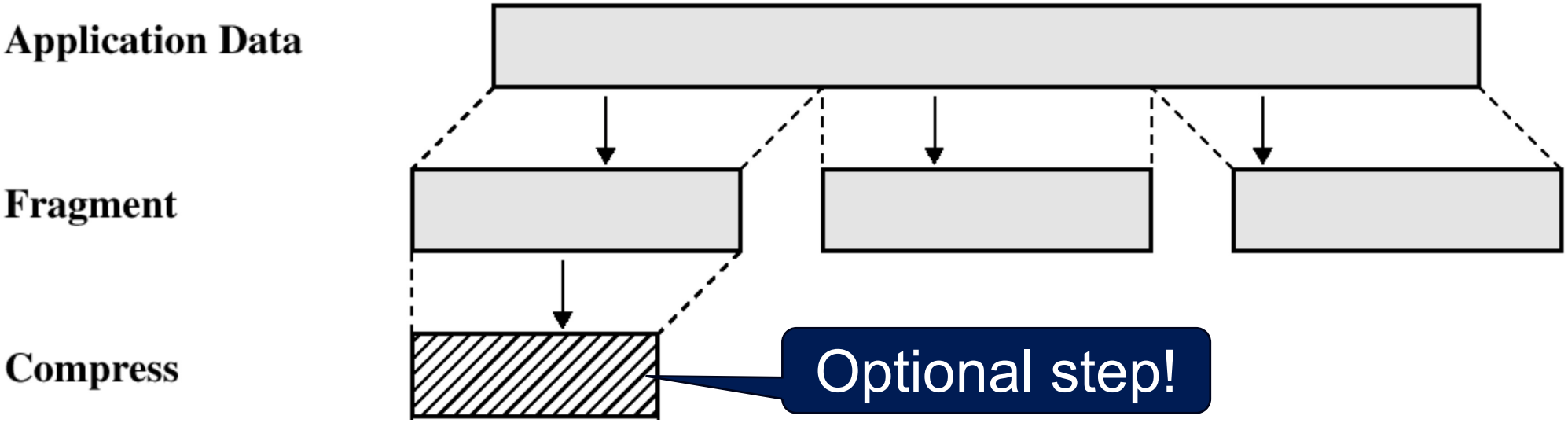


SSL Record Protocol Operation

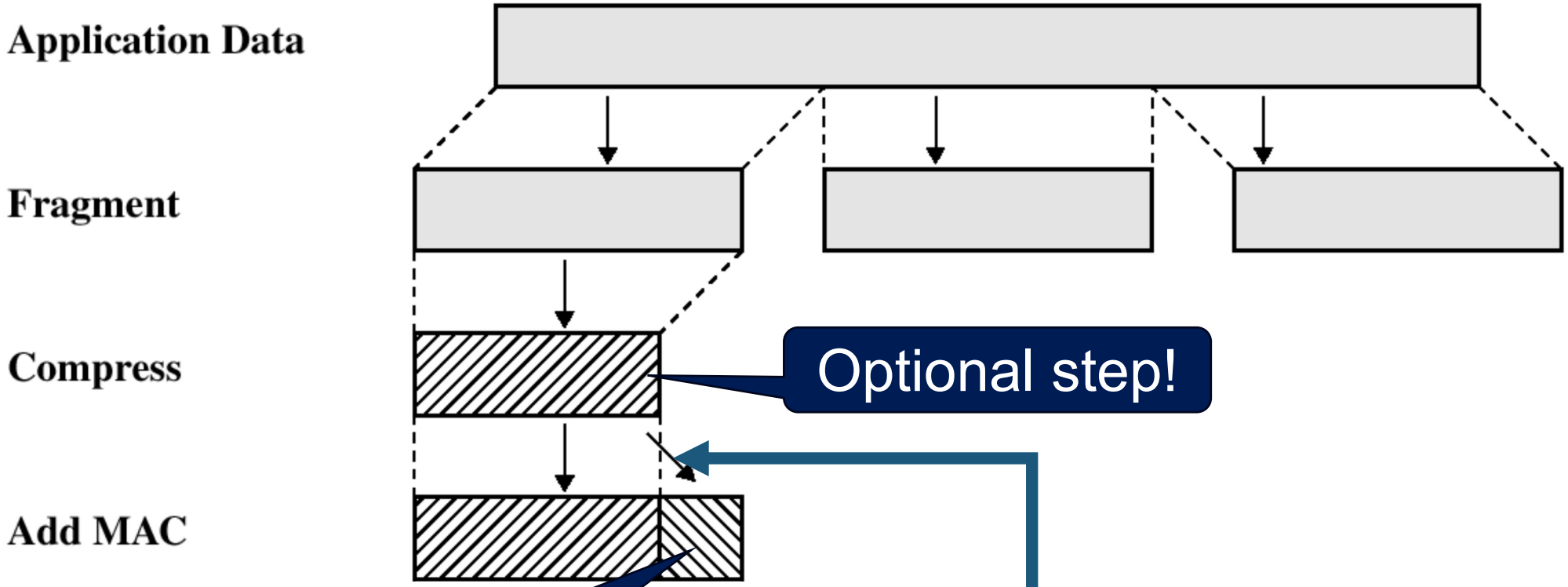
SSL Record Protocol Operation



SSL Record Protocol Operation



SSL Record Protocol Operation



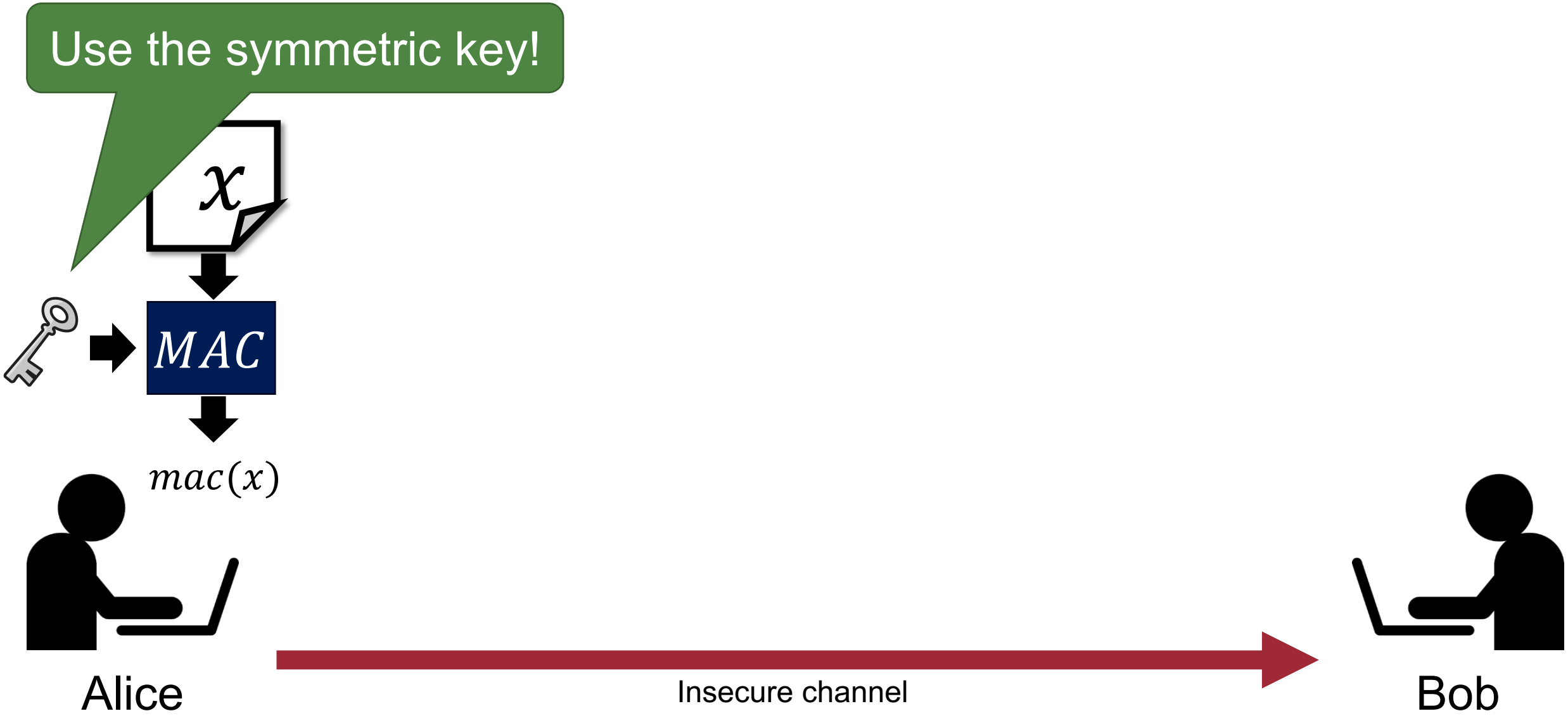
Optional step!

MAC: Check both integrity and authenticity

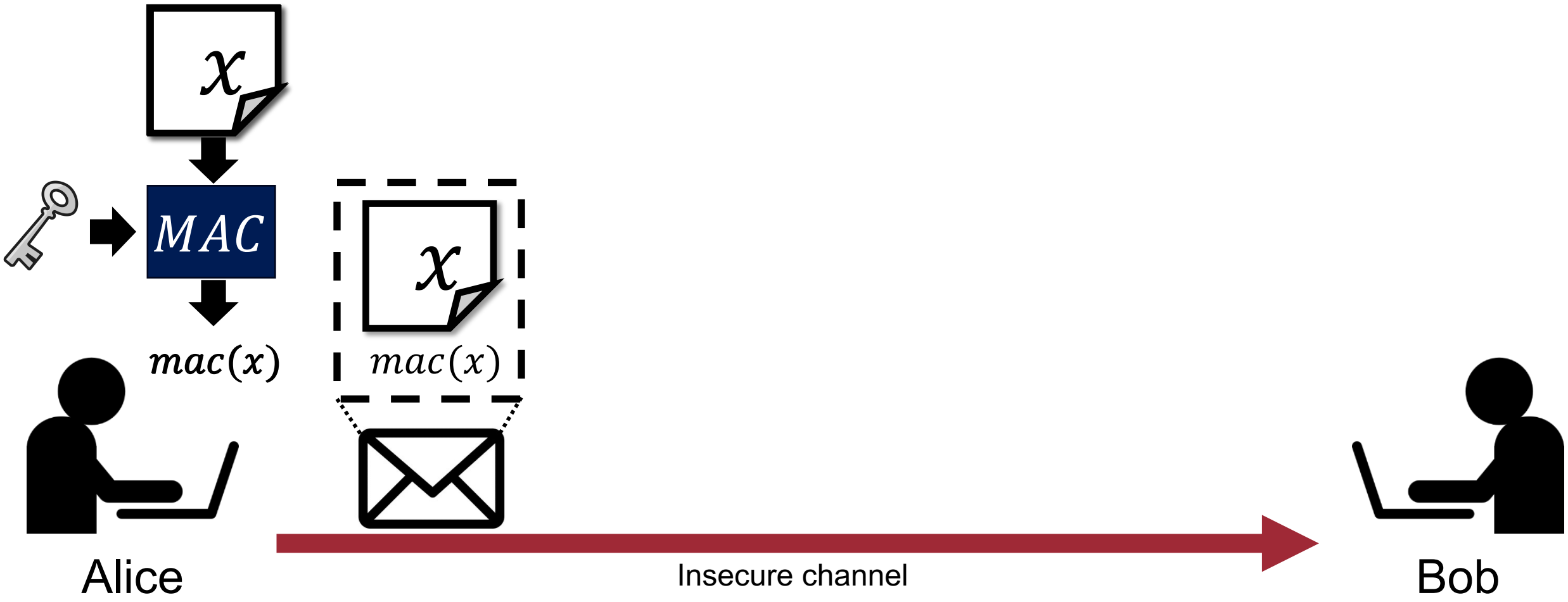
Client auth. key	Server auth. key	Client enc. key	Server enc. key	Client IV	Server IV

Symmetric Key

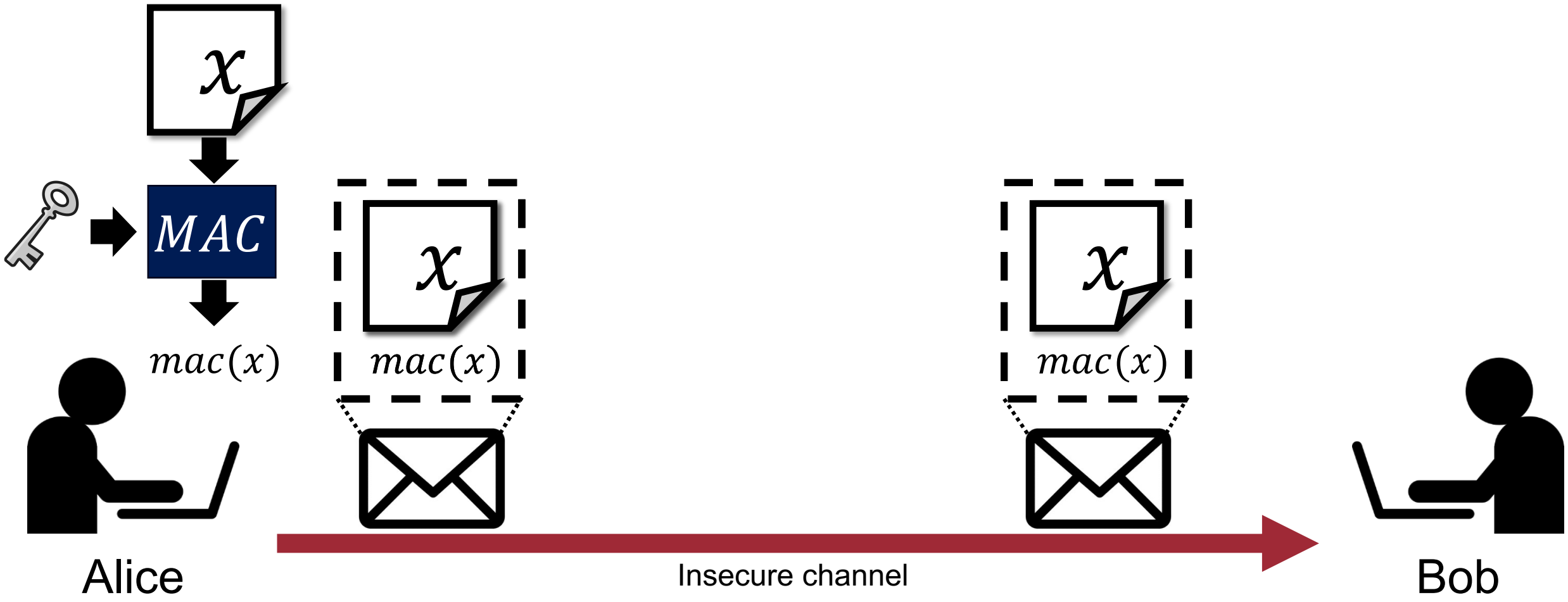
Ref: Message Authentication Codes (MAC)



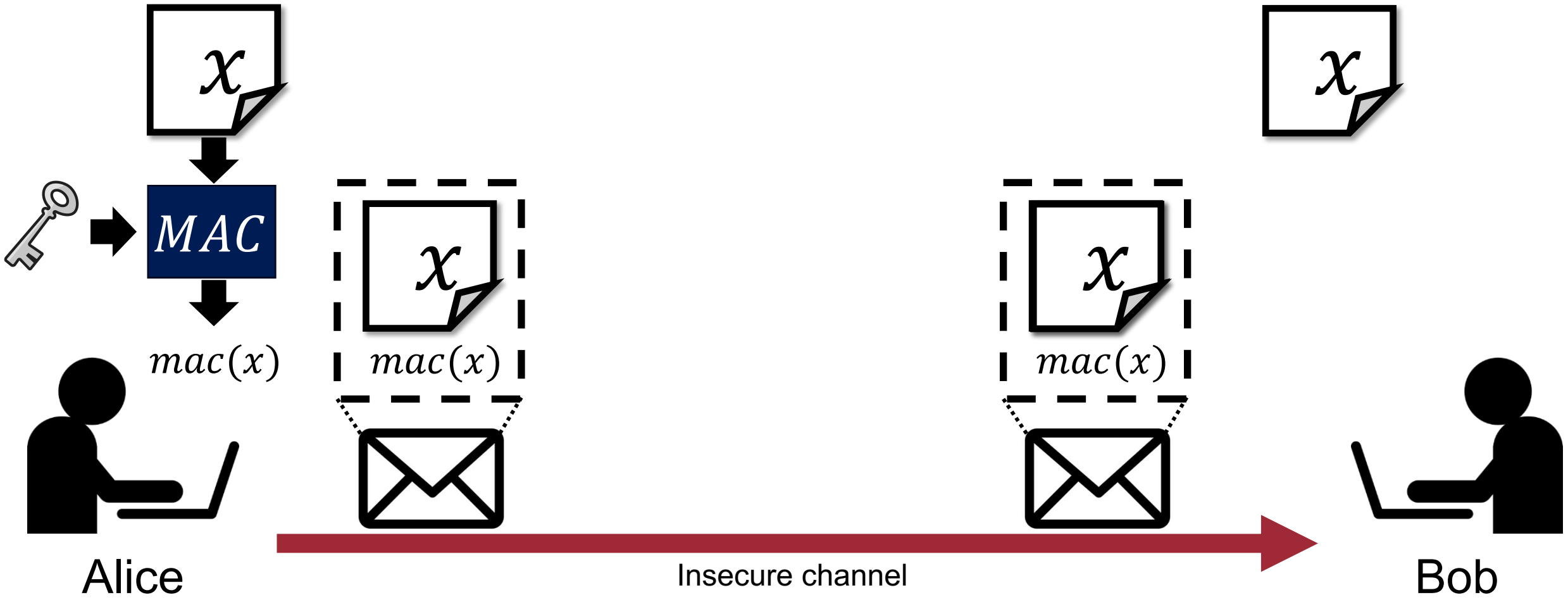
Ref: Message Authentication Codes (MAC)



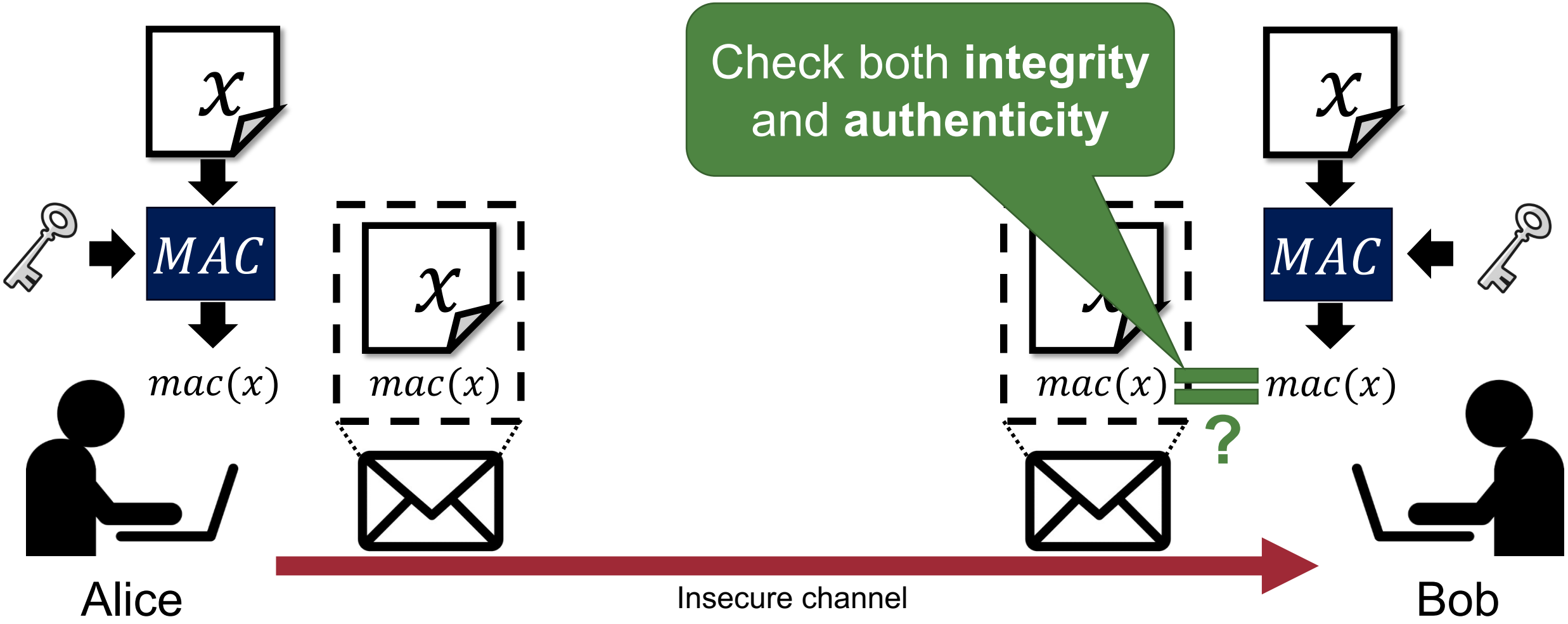
Ref: Message Authentication Codes (MAC)



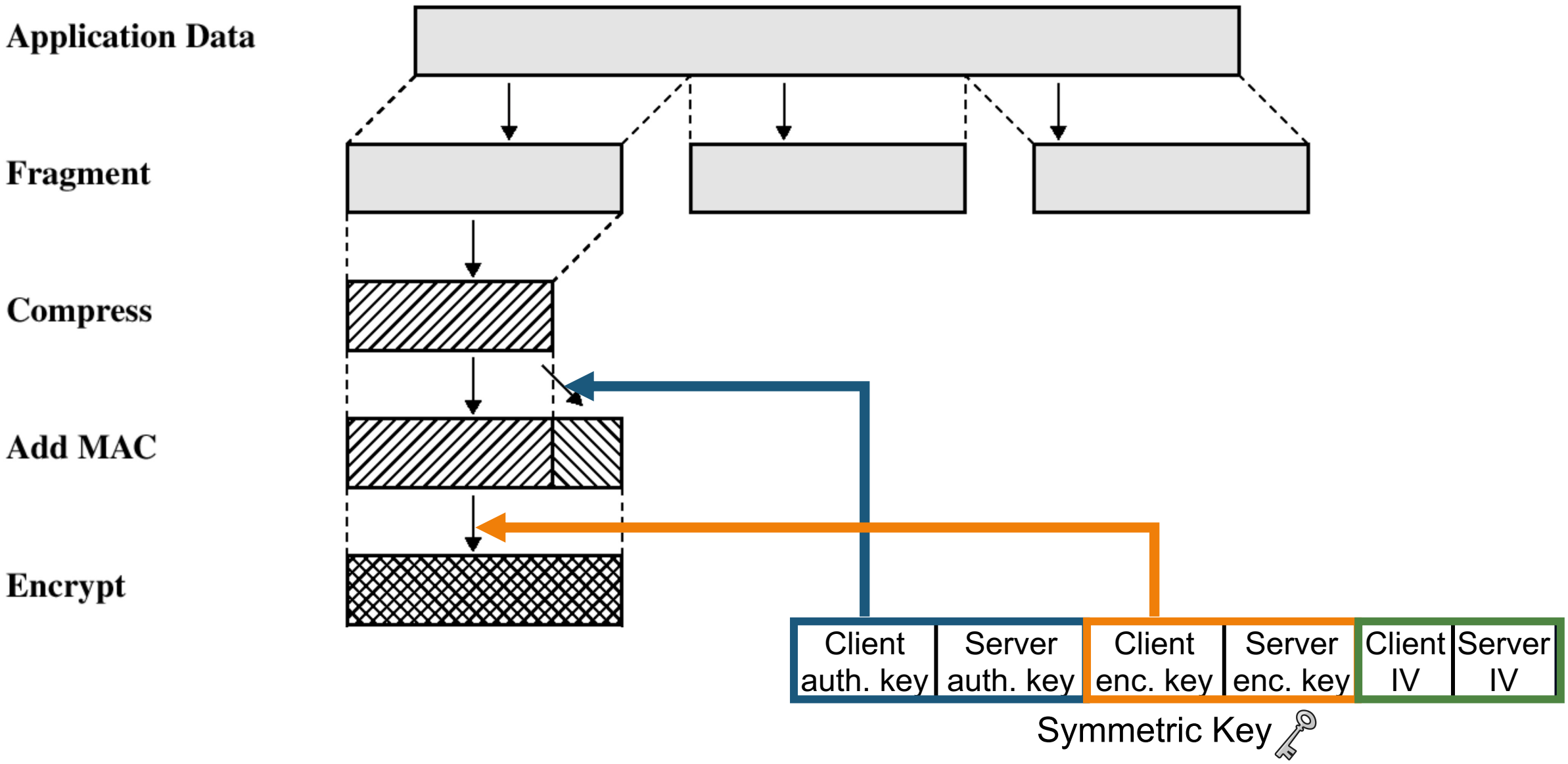
Ref: Message Authentication Codes (MAC)



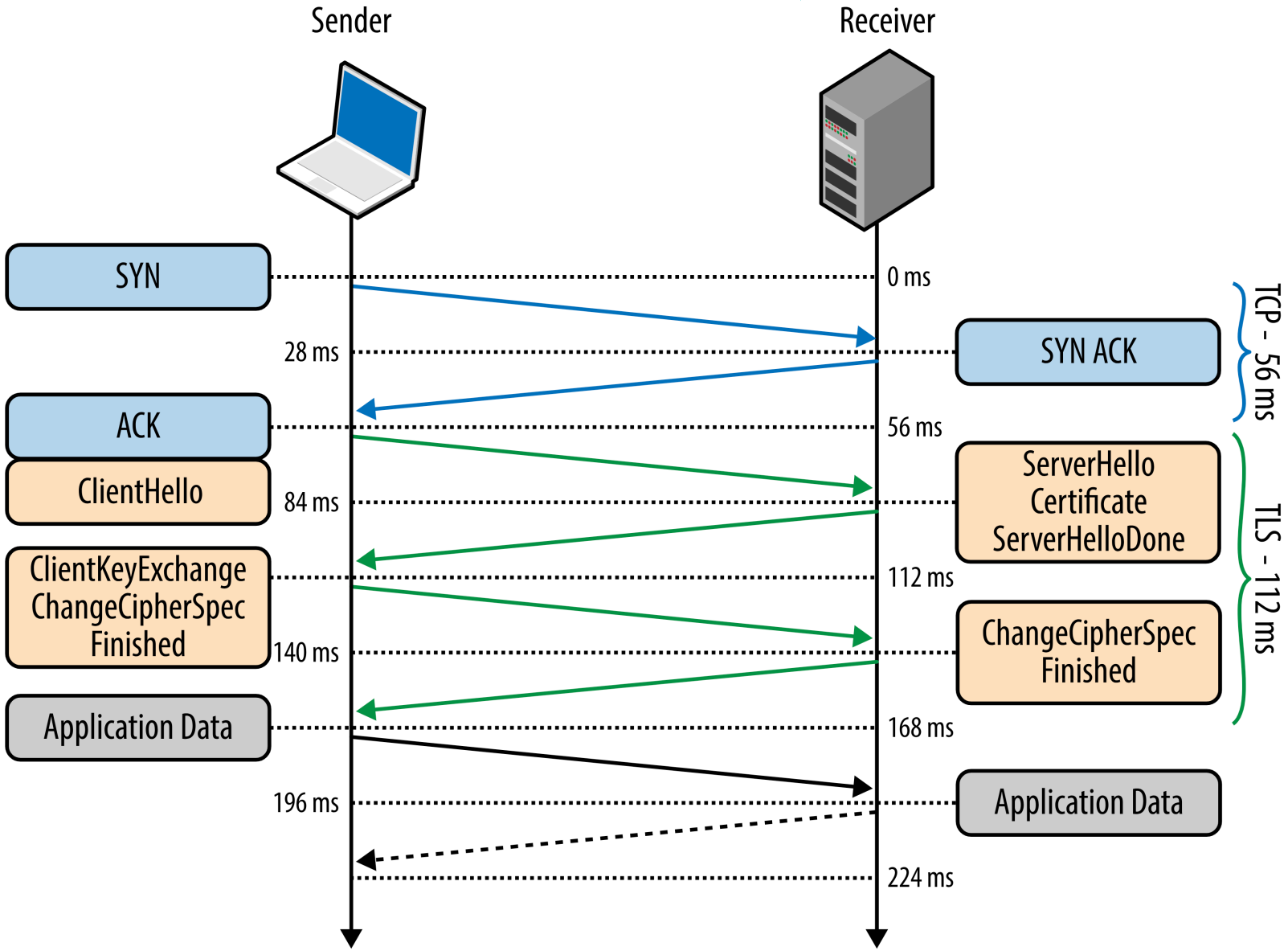
Ref: Message Authentication Codes (MAC)



SSL Record Protocol Operation



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?

    - 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

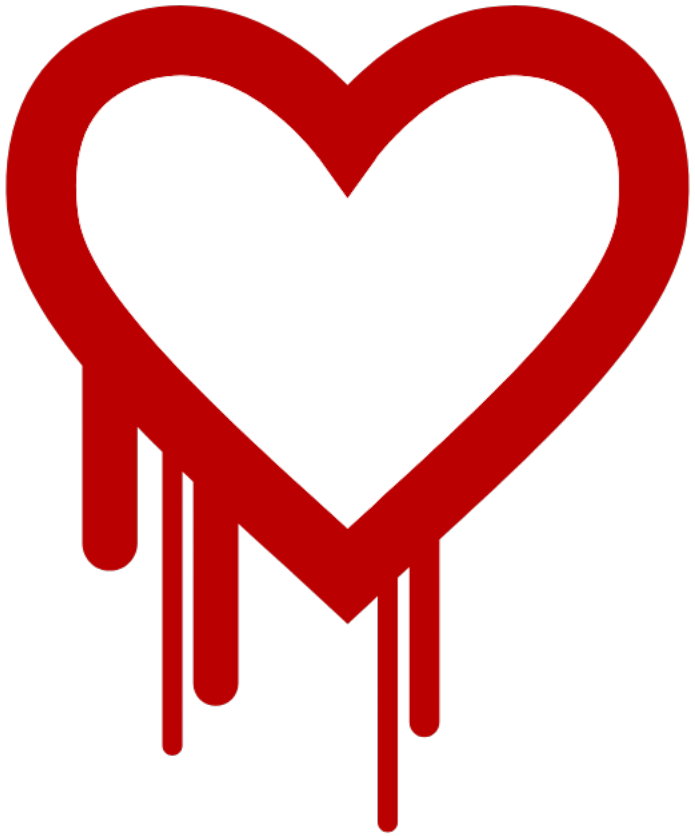
SSL/TLS Implementations

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

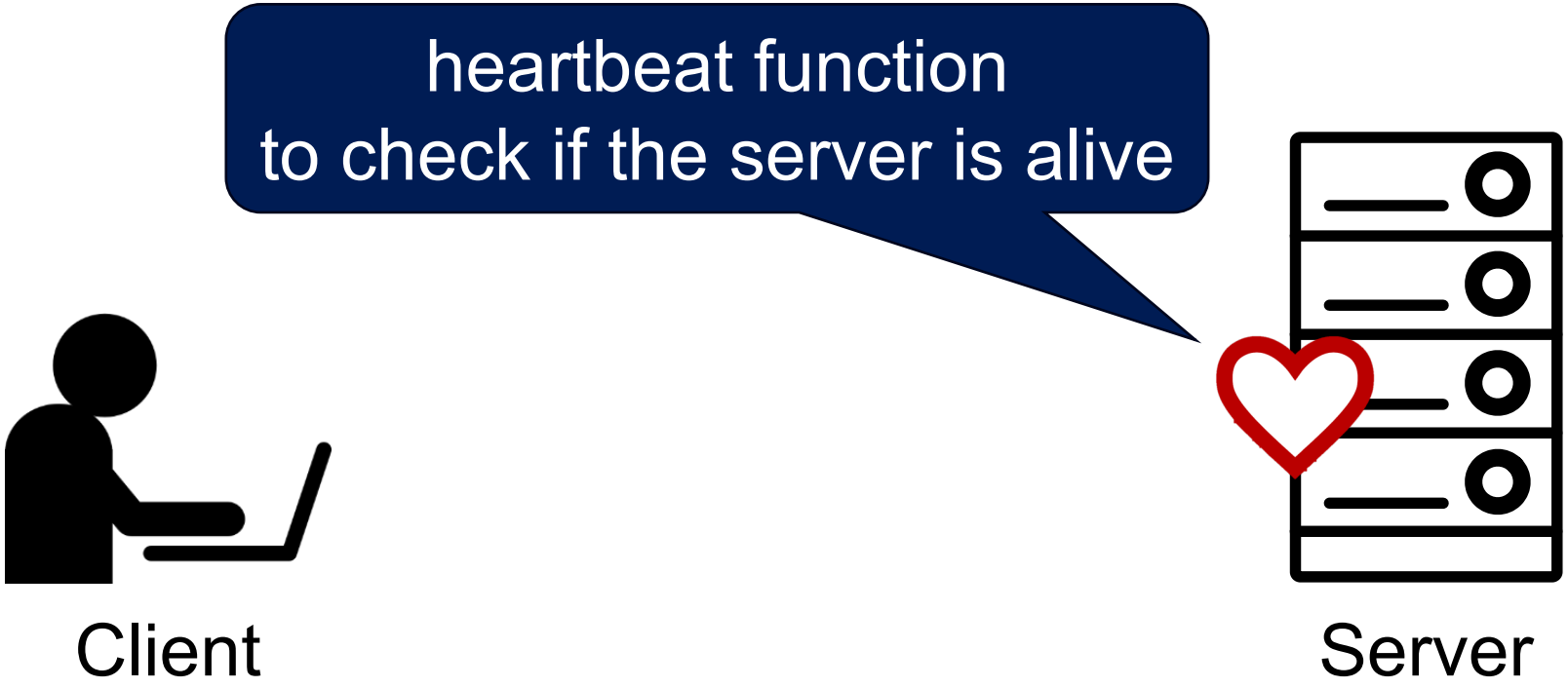
Can We Believe the SSL/TLS Implementations?

Heartbleed Bug (in 2014)

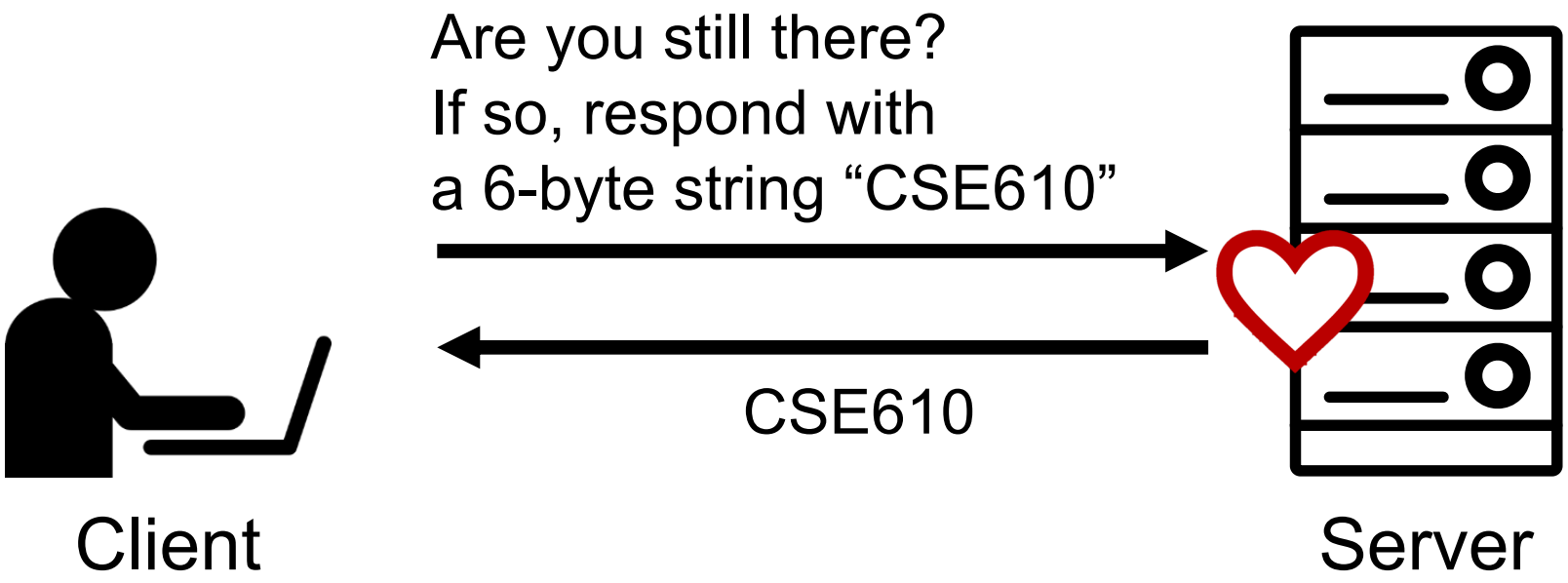
- Famous bug in OpenSSL (in TLS ***heartbeat***)
- An attacker can steal private keys



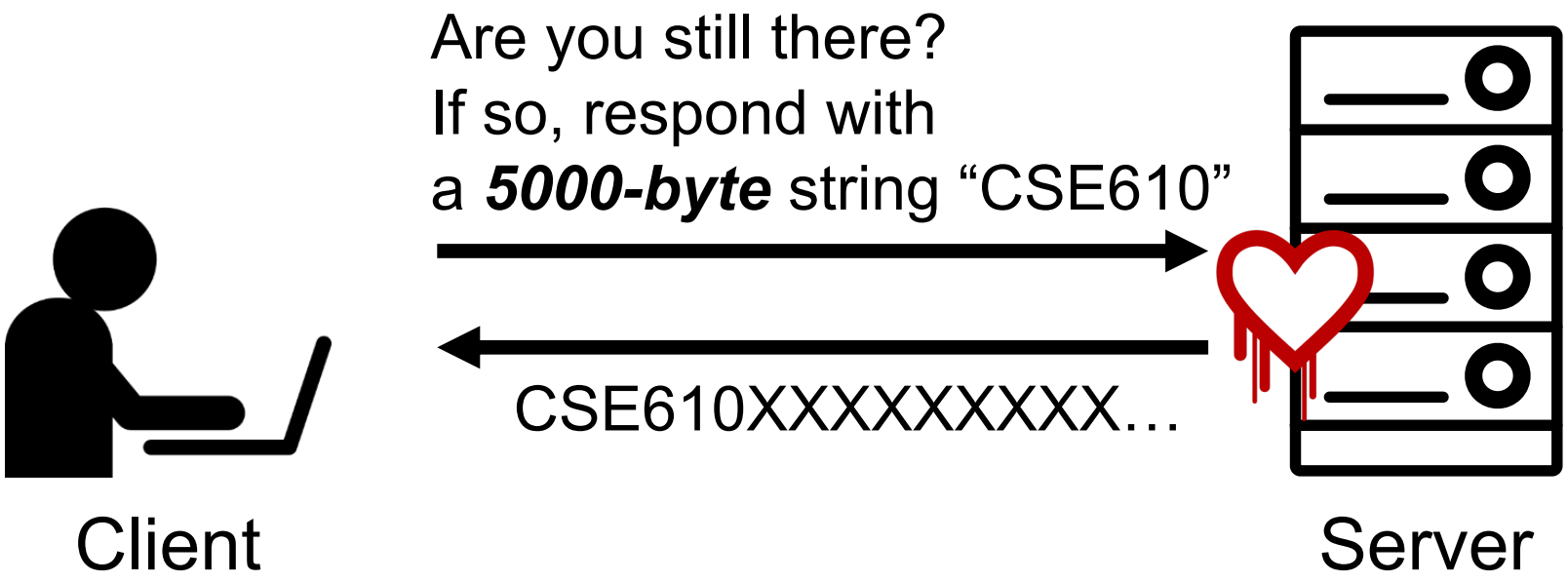
Heartbleed Bug: High-level Workflow



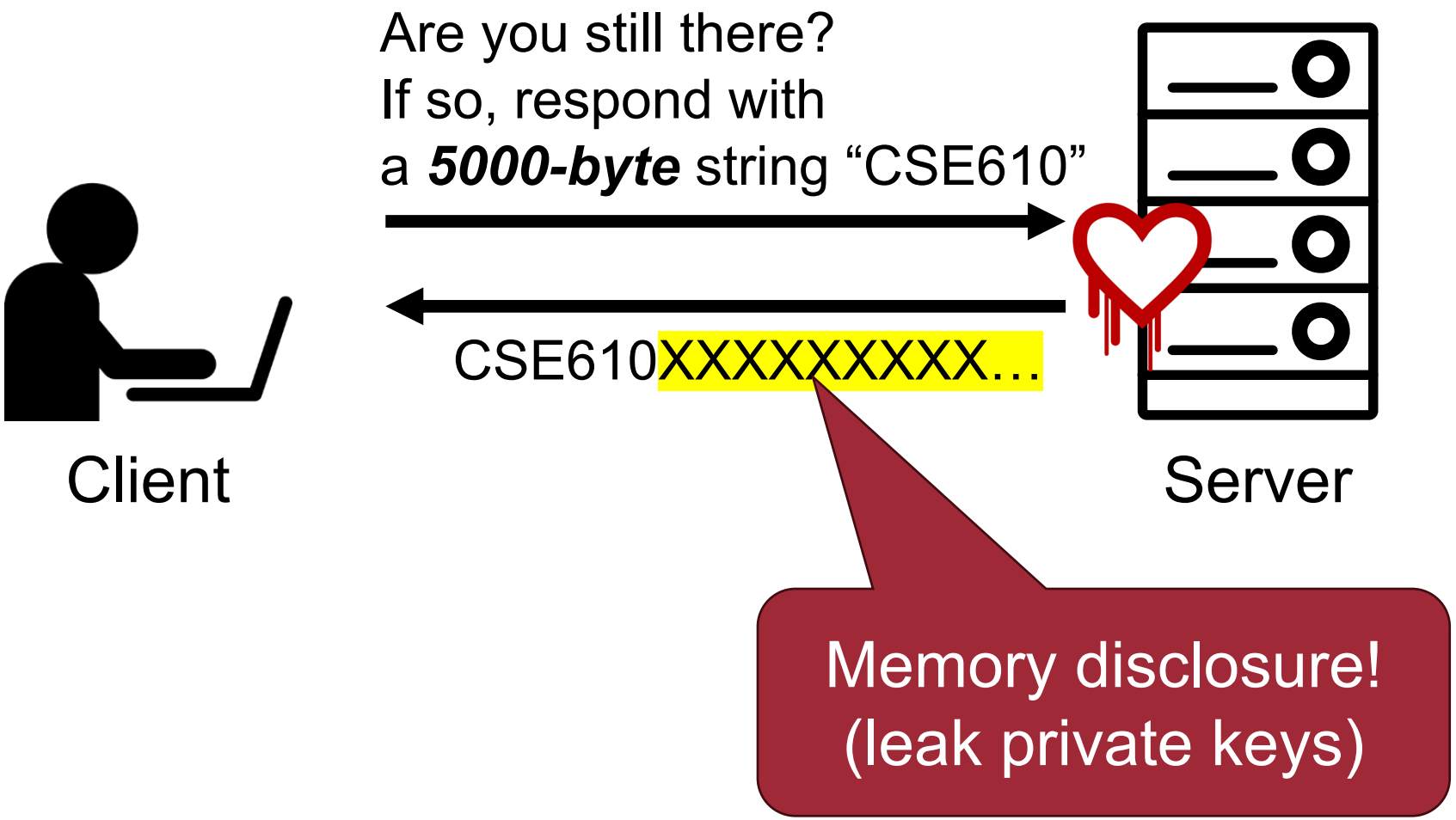
Heartbleed Bug: High-level Workflow



Heartbleed Bug: High-level Workflow



Heartbleed Bug: High-level Workflow



Frankencert, S&P'2014

- Design the first automated method for large-scale testing of **certificate validation logic** in SSL/TLS implementations

Using Frankencerts for Automated Adversarial Testing of Certificate Validation in SSL/TLS Implementations

Chad Brubaker * † Suman Jana† Baishakhi Ray‡ Sarfraz Khurshid† Vitaly Shmatikov†
*Google
†The University of Texas at Austin
‡University of California, Davis

***Abstract*—Modern network security rests on the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols. Distributed systems, mobile and desktop applications, embedded devices, and all of secure Web rely on SSL/TLS for protection against network attacks. This protection critically depends on whether SSL/TLS clients correctly validate X.509 certificates presented by servers during the SSL/TLS handshake protocol.**

many open-source implementations of SSL/TLS are available for developers who need to incorporate SSL/TLS into their software: OpenSSL, NSS, GnuTLS, CyaSSL, PolarSSL, MatrixSSL, cryptlib, and several others. Several Web browsers include their own, proprietary implementations.

In this paper, we focus on *server authentication*, which

SSL/TLS Objectives

- To protect
  - Confidentiality
  - Integrity
  - Authenticity

SSL/TLS Objectives

- To protect
  - Confidentiality
  - Integrity
  - **Authenticity** = certificate validation!!
    - Focus on **server authentication**
    - Protection against man-in-the-middle and other server impersonation attacks

The focus of this paper

Current State-of-the-Art

- Uses several pre-generated X.509 certificates intended for testing.

Implementation	Certificate count
NSS	64
GnuTLS	51
OpenSSL	44
PolarSSL	18
CyaSSL	9
MatrixSSL	9

Testing with a handful of valid certificates is unlikely to uncover vulnerabilities in the certificate validation logic

How to generate test certificates?

- Challenge:
  - X.509 certificates are **structurally complex** data with intricate semantic and syntactic constraints!
- Requirements
  - Should be **syntactically correct**, otherwise won't exercise most of the cert validation code
  - Must generate ***semantically bad*** certificates including unusual combinations of features and extensions
  - Must scale to millions of certs

Frankencert

1. Gather a corpus of real SSL/TLS certificates by scanning the Internet

Frankencert

1. Gather a corpus of real SSL/TLS certificates by scanning the Internet

2. Break them down into parts

Frankencert

1. Gather a corpus of real SSL/TLS certificates by scanning the Internet

2. Break them down into parts

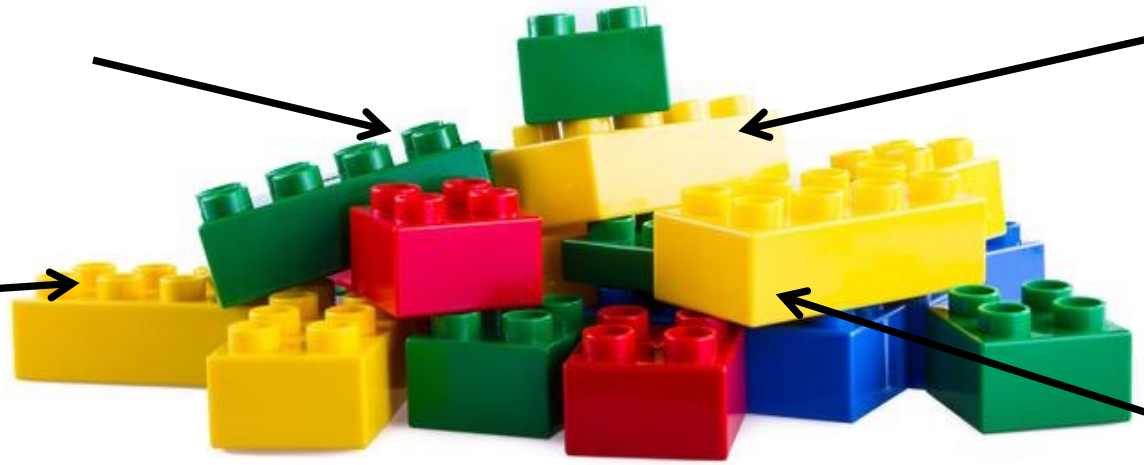
3. Generate frankencerts by mutating random combinations of these parts

Collecting Seed Certificates

- **Scan** the internet and attempt an SSL connection to every host listening 443
- Total **243,246 certificates**
- Purpose: Use corpus of real certificates as the source of syntactically valid certificate parts

Generating Frankencerts

- Generate a certificate by choosing parts randomly from seeds.
  - Assembles parts into random combinations

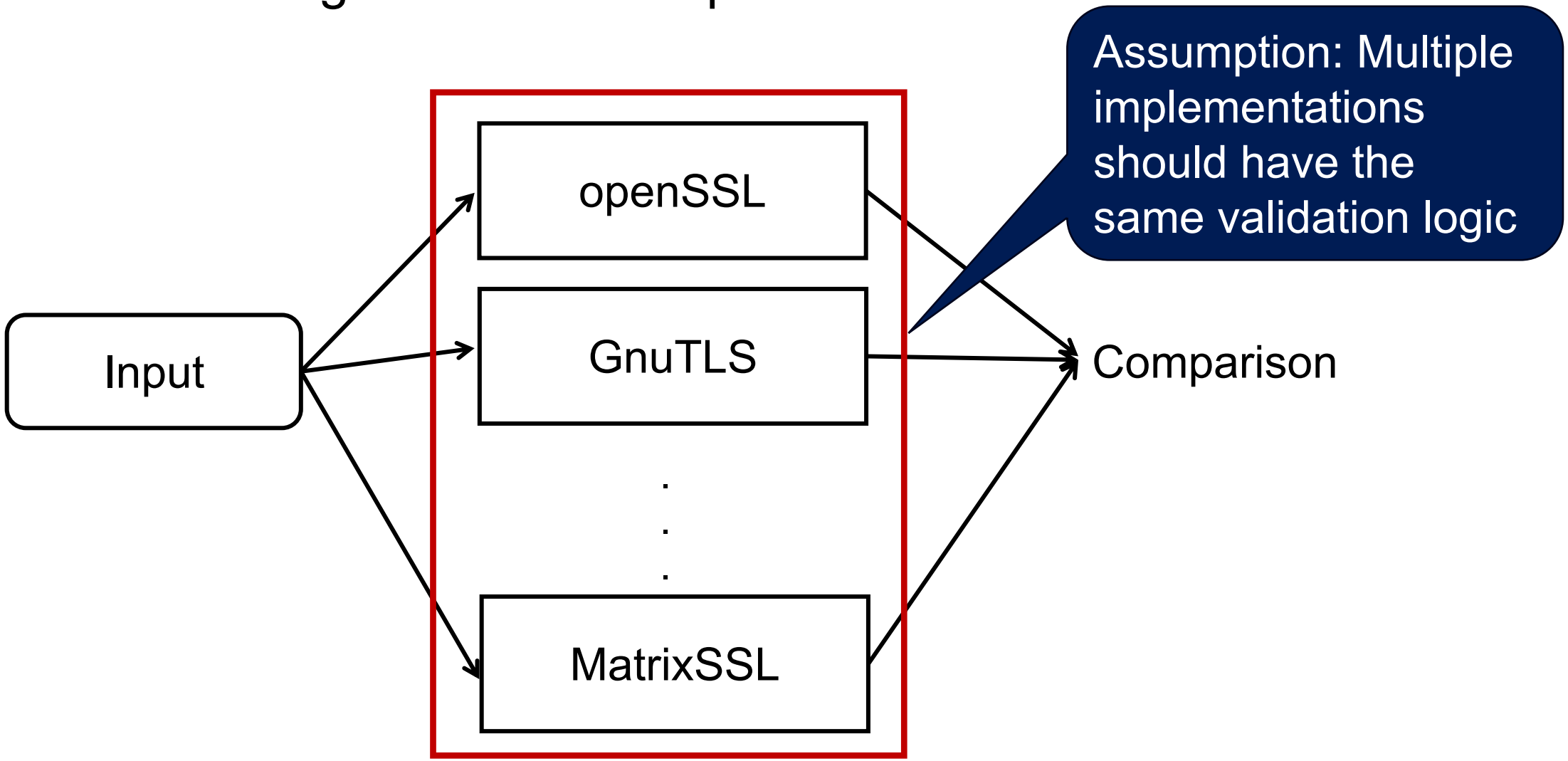


How to interpret frankencert test results?

Differential Testing!

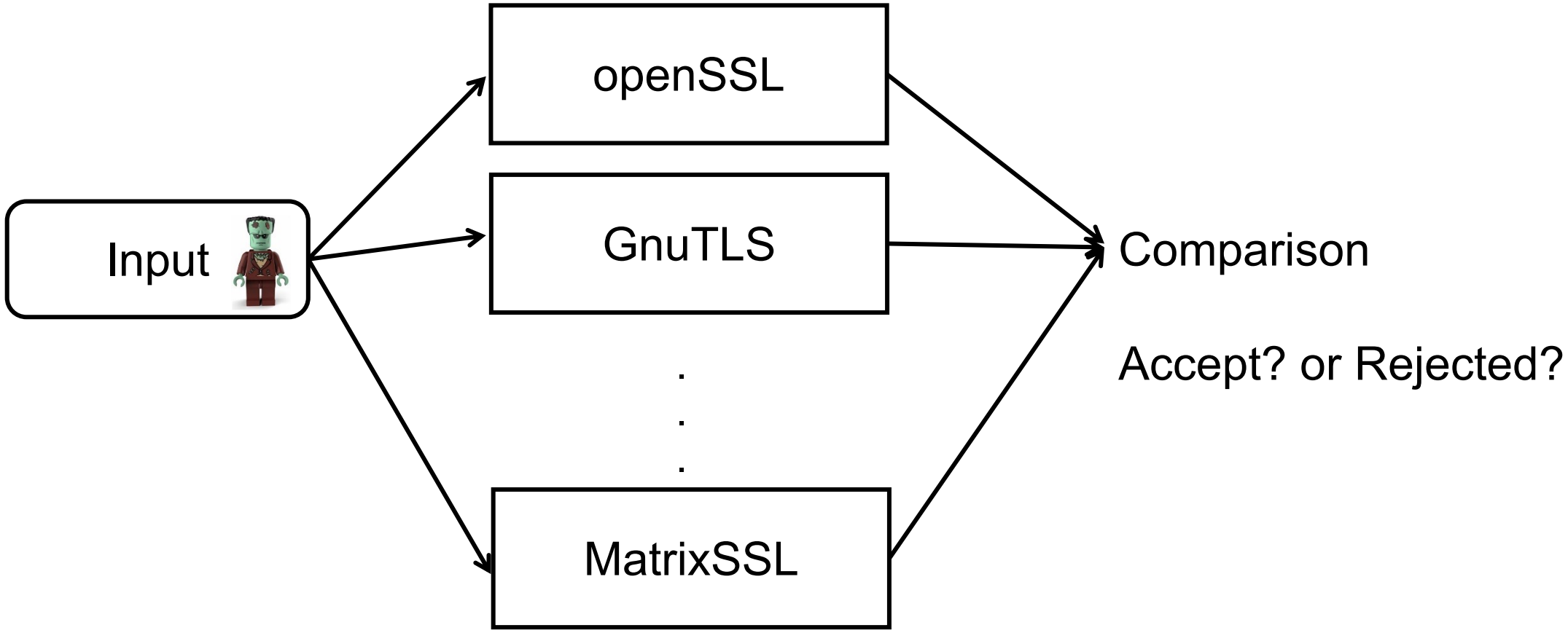
Testing SSL/TLS Implementations

- Differential testing of SSL/TLS implementations



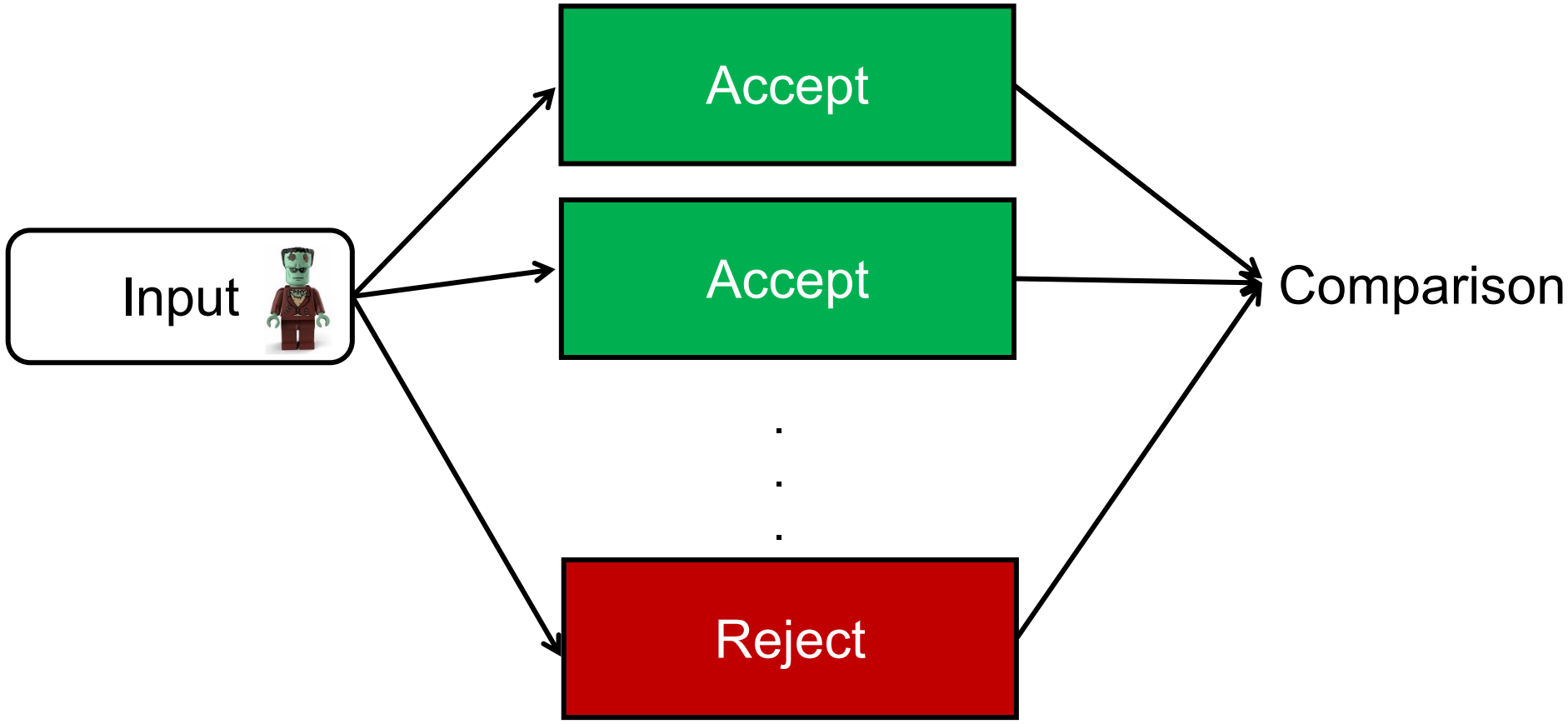
Testing SSL/TLS Implementations

- Differential testing of SSL/TLS implementations



Testing SSL/TLS Implementations

- Differential testing of SSL/TLS implementations



Testing SSL/TLS Implementations

- Differential testing of SSL/TLS implementations

- Target:
  - SSL/TLS libraries: OpenSSL, PolarSSL, GnuTLS, CyaSSL, matrixSSL, NSS...
  - Web browsers: FireFox, Opera, Chrome

- **208** discrepancies are found by testing **8,127,600** frankencerts

Case Study: MatrixSSL

- Incorrect checking **whether version 1 intermediate certificate** in the chain belongs to a valid CA

Just skip checking for version 1 or 2

```
/* Certificate authority constraint only available i
     version 3 certs */

if ((ic->version > 1) && (ic->extensions.bc.ca<= 0)) {
    psTraceCrypto("Issuer does not have basicConstraint
          CA permissions\n");
    sc->authStatus = PS_CERT_AUTH_FAIL_BC;
    return PS_CERT_AUTH_FAIL_BC;
}
```

Man-in-the-middle attacks: Any server certified by the same root **can act as rogue CA** and issue fake certificates for other domains

Review

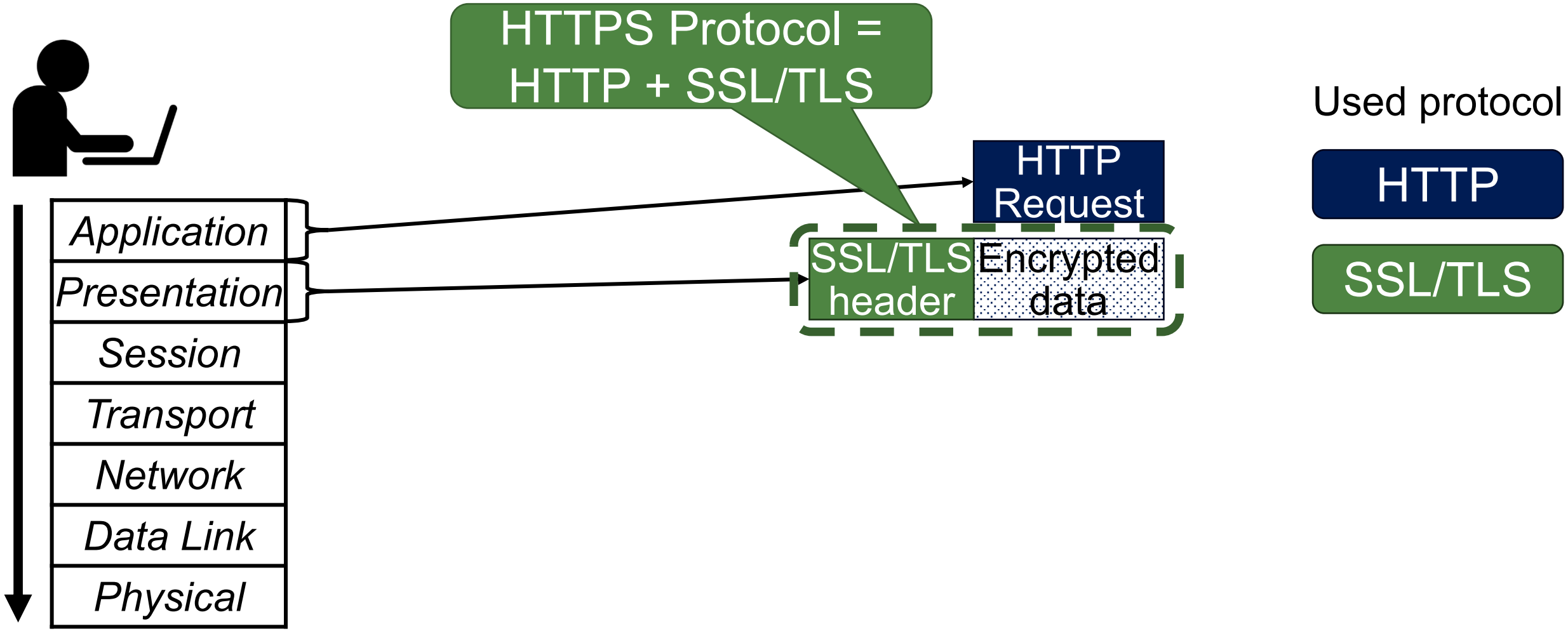
- ***"Blind" nature of frankencert makes it cost-ineffective***: testing an enormous number of frankencerts are very **resource-intensive**, but most of the frankencerts do not trigger any discrepancies

- By adding **existing pre-generated test suite** as corpus, the generated certificate will be more diverse and meaningful in terms of testing

HTTPS

HTTPS

- Adding a protocol layer for secure communication!



HTTPS – The Lock Icon

- 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

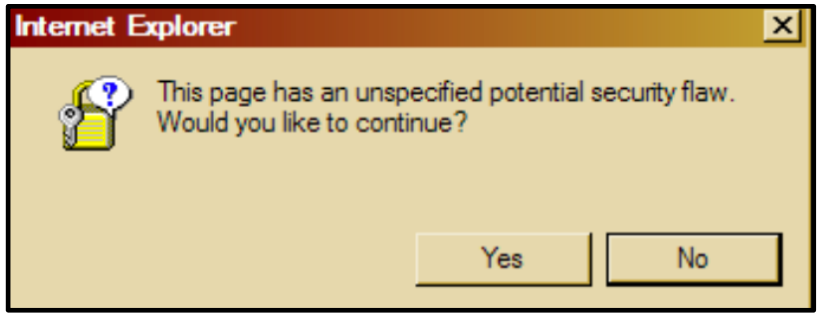
- Goal: the client (Human) can identify s
  - SSL/TLS is being used to protect against

What happens if page served over HTTPS but contains HTTP?

- 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!"

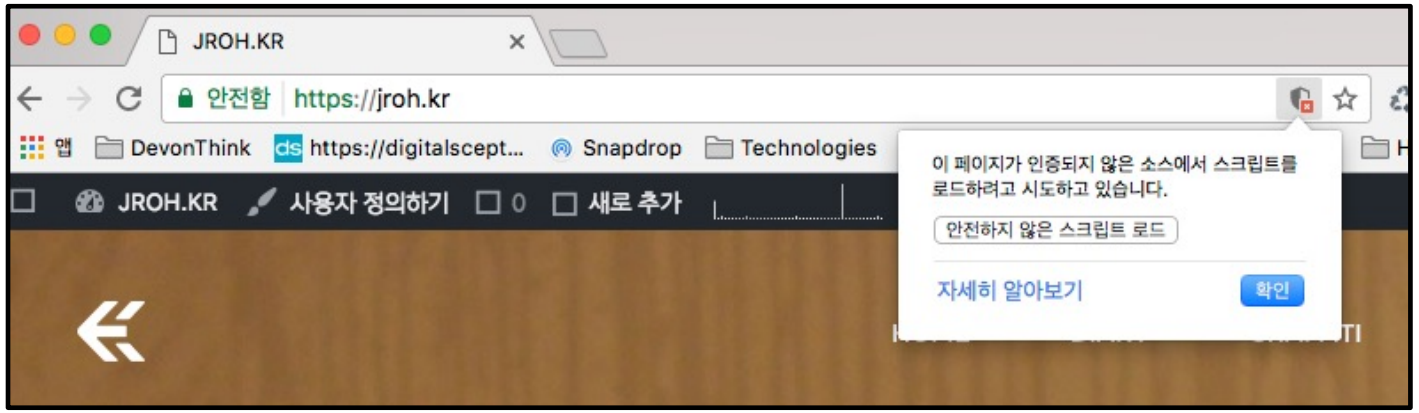
Mixed Content: Combining HTTPS and HTTP

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



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

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



Mixed Content and Network Attacks

https://bank.com/attack.html

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

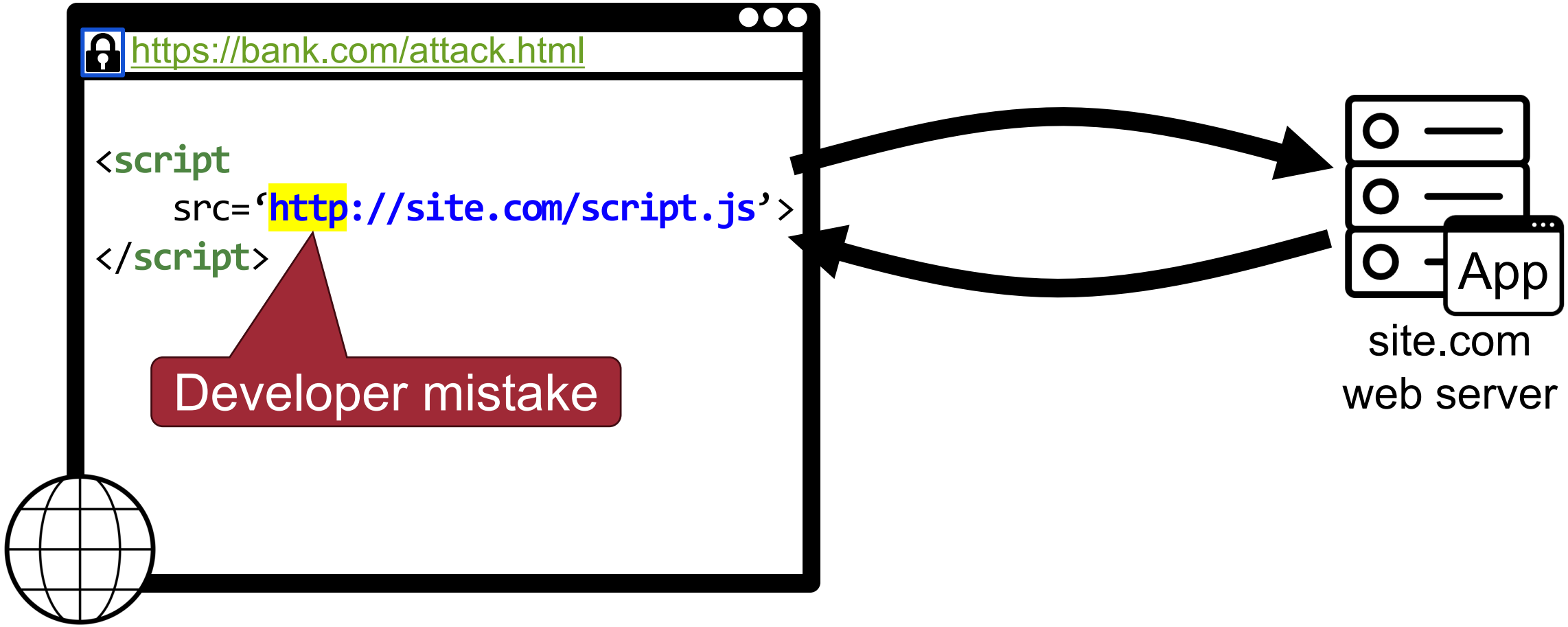
Mixed Content and Network Attacks

https://bank.com/attack.html

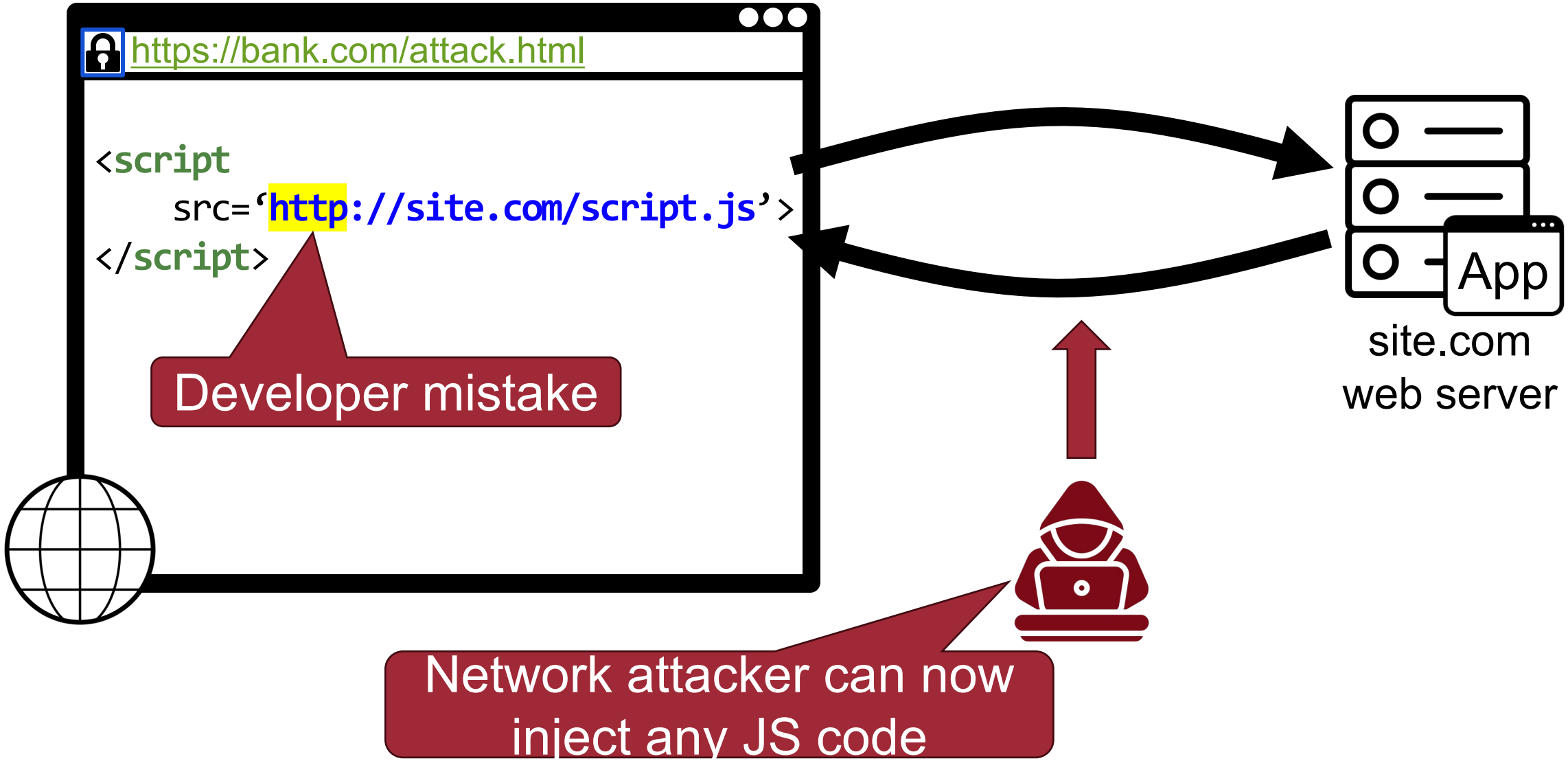
```
<script
    src='http://site.com/script.js'>
</script>
```

Developer mistake

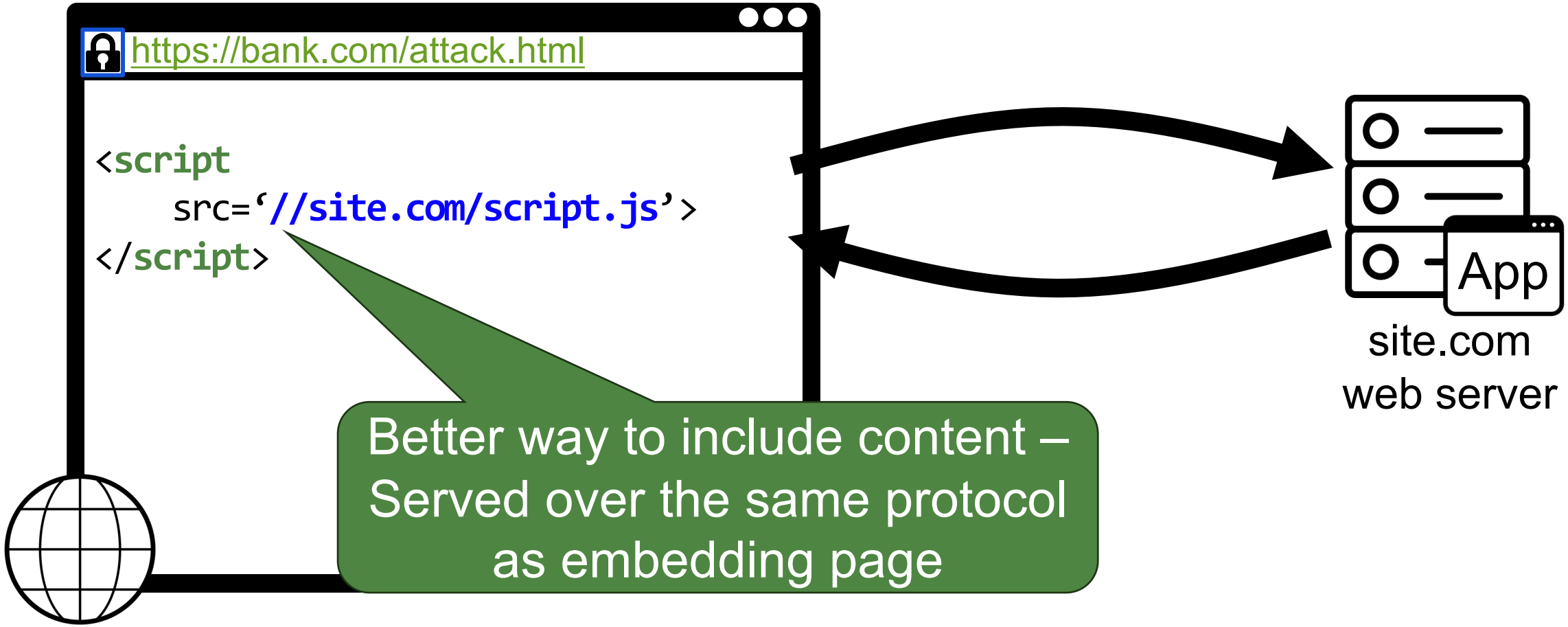
Mixed Content and Network Attacks



Mixed Content and Network Attacks

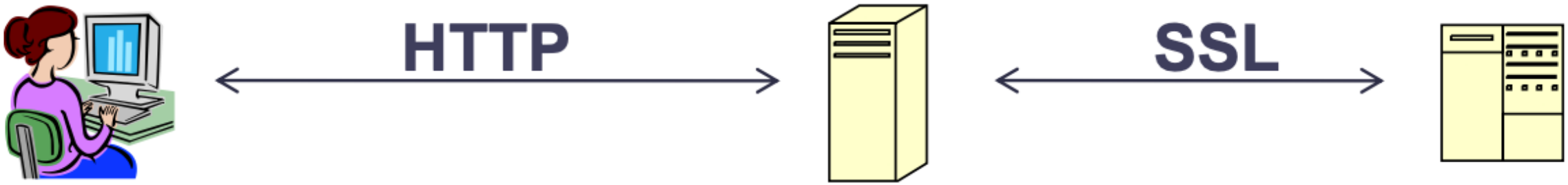


Mixed Content and Network Attacks



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

Summary

- SSL/TLS protocol
  - Satisfy confidentiality
  - Satisfy integrity
  - Satisfy authentication

- HTTPS: HTTP + SSL/TLS protocol

Question?