CS 772/872: Advanced
Computer and Network Security
Fall 2022

Course Link:

https://shhaos.github.io/courses/CS872/netsec-fall2022.html

Instructor: Shuai Hao

shao@odu.edu

www.cs.odu.edu/~haos

OLD DOMINION UNIVERSITY

Network Security – Cryptography

- TCP/IP
- DoS Attacks
- DNS
- BGP
- CDN

- **Applied Cryptography**
- **PKI**
- **TLS/SSL and HTTPS**

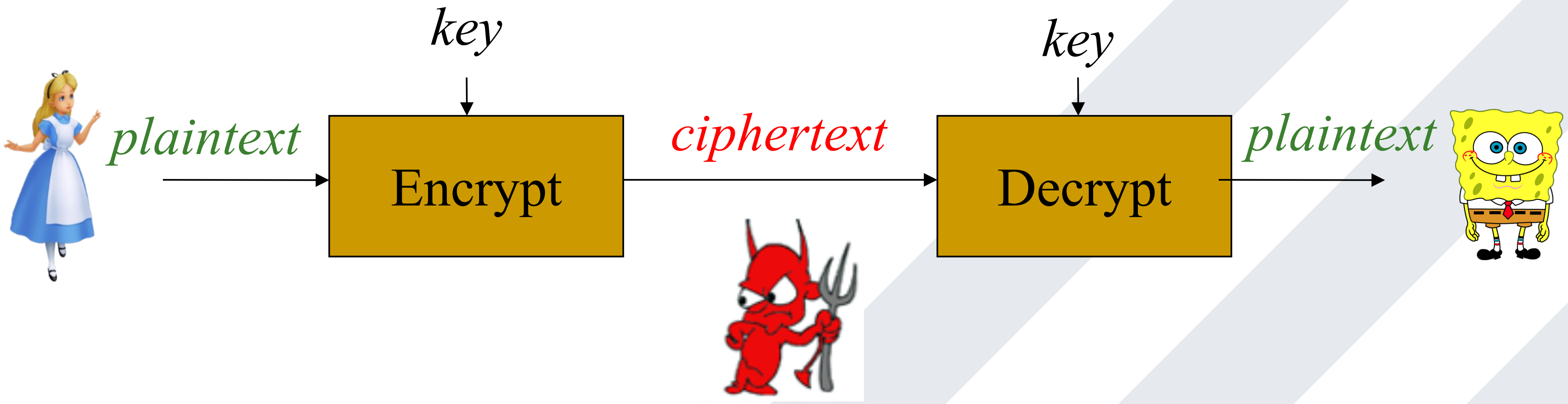
- **DNSSEC** (*USENIX Security'17*)
- **RPKI** (*NDSS'17*)
- **HTTPS/CDN** (*IEEE S&P'14*)

Cryptography Foundations

- **Ensuring secrecy of the communication between two parties**
- **Classical Cryptography**
  - Always assumed that two parties shared some secret information (Key)
    - Private-kay or symmetric-key
- **“Modern” Cryptography**
  - No pre-shared secret is requited for two parties
    - Public-key or asymmetric-key

Cryptography Foundations

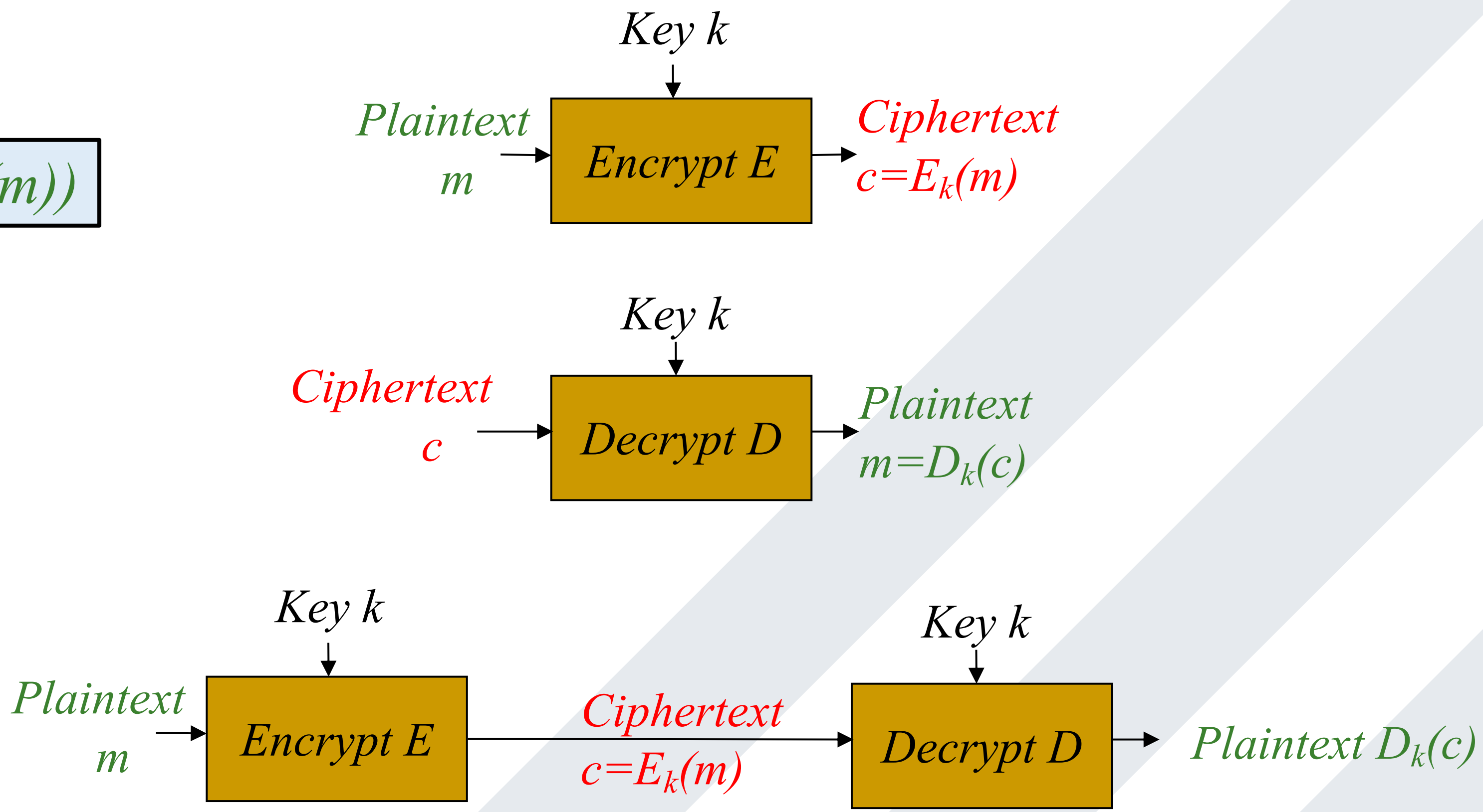
- **(Symmetric Key) Encryption**
  - Encrypt (encode) plaintext into ciphertext
    - Only legit-recipient can decrypt ciphertext to plaintext



Cryptography Foundations

- **Correctness**

Correctness: $m=D_k(E_k(m))$



Cryptography Foundations

- **Threat model for encryption**
  - Describe the assumption on what the capability an attack can gain
  - Ciphertext-only attack
  - Known-plaintext attack
  - Chosen-plaintext attack
    - Attacker was able to obtain some cipher text, encrypted using the same key, corresponding to plaintext of the attacker's choice (an oracle)

Cryptography Foundations

- **Threat model for encryption**
  - Describe the assumption on what the capability an attack can gain
  - Ciphertext-only attack
  - Known-plaintext attack
  - Chosen-plaintext attack
  - Chosen-ciphertext attack
    - Attacker is able to get a party to decrypt certain cipher texts of that attacker's choice.

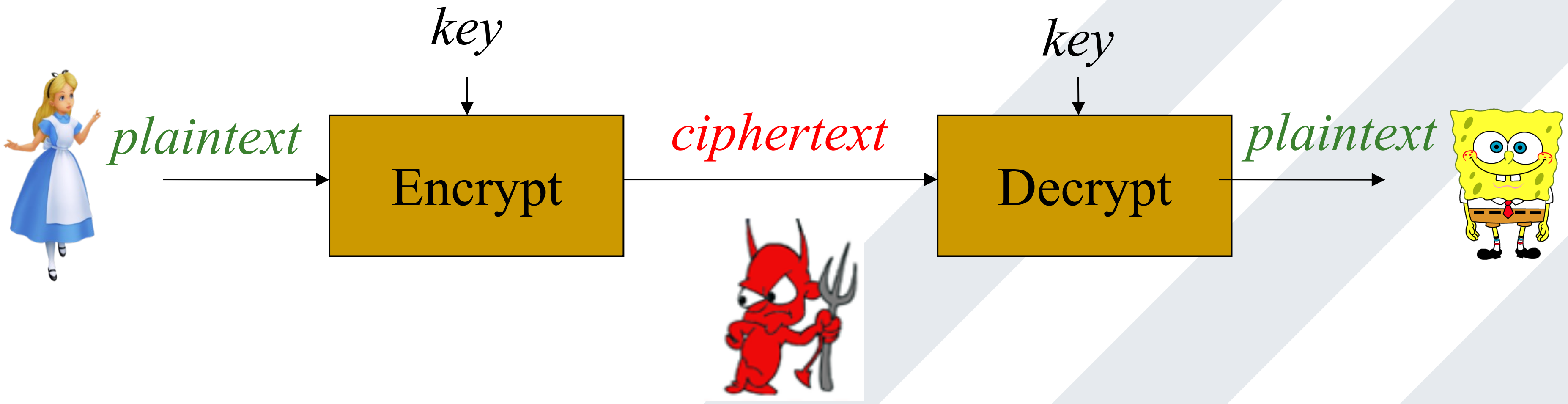
Cryptography Foundations

- **Threat model for encryption**
  - Describe the assumption on what the capability an attack can gain
  - Ciphertext-only attack
  - Known-plaintext attack
  - Chosen-plaintext attack
  - Chosen-ciphertext attack

Regardless of any prior information the attacker has about the plaintext, the ciphertext observed by the attacker should leak no additional information about the plaintext.

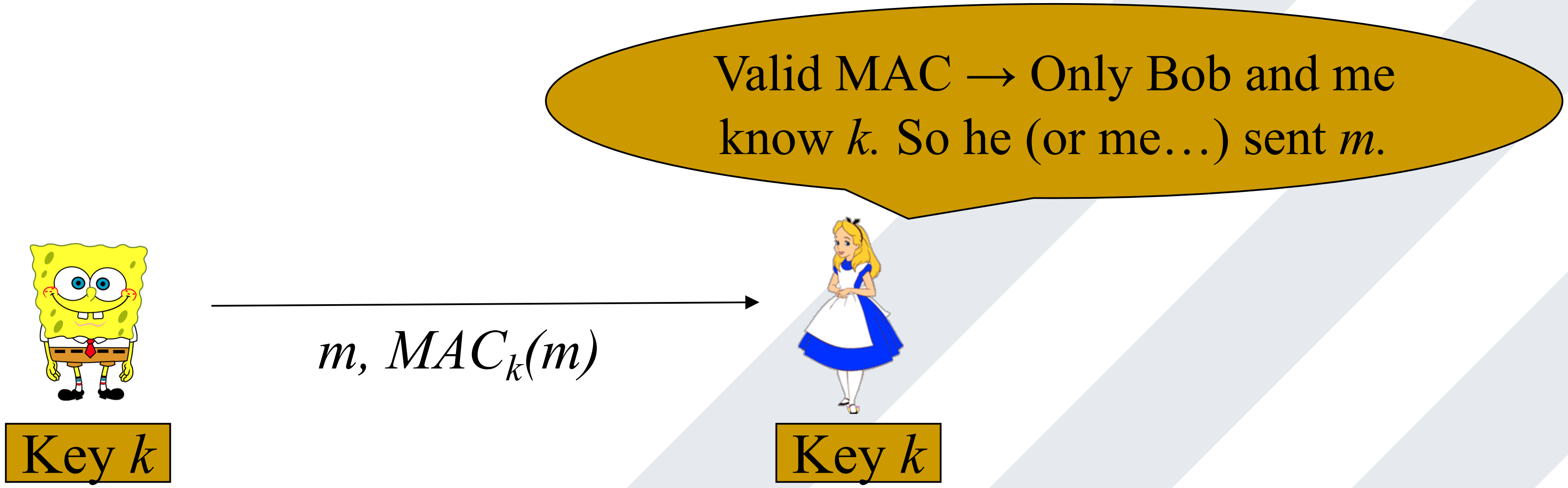
Cryptography Foundations

- **Authentication**
  - Encryption ensures Confidentiality
  - What about Integrity and Authentication
    - Does ***Alice*** send ***this*** message?



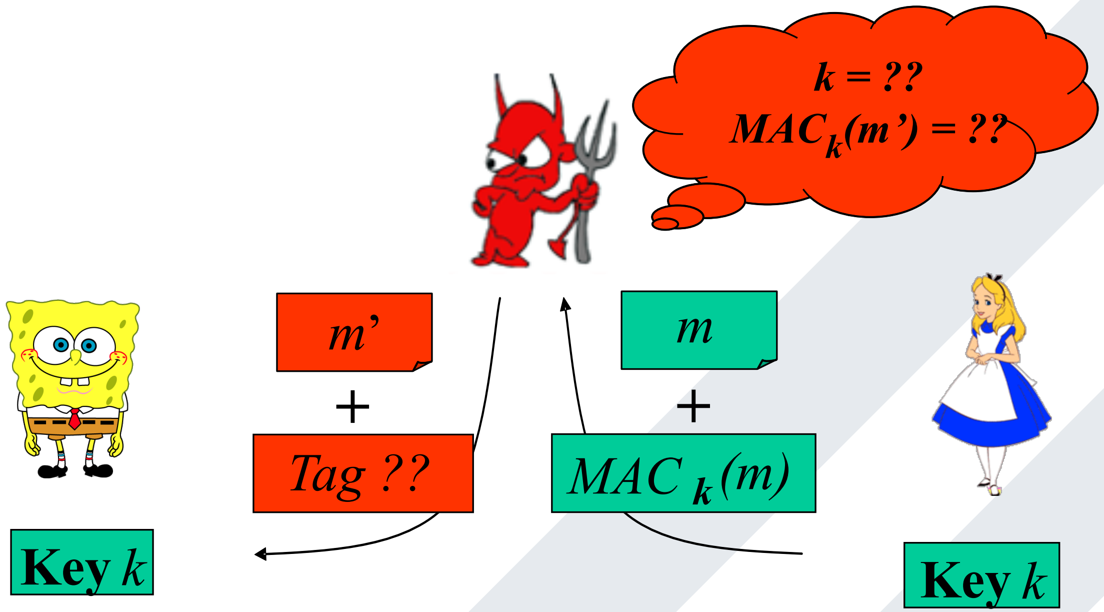
Cryptography Foundations

- **Message Authentication Code (MAC)**
  - Allow a recipient to validate that a message was sent by a key holder
  - Use shared key k to authenticate messages



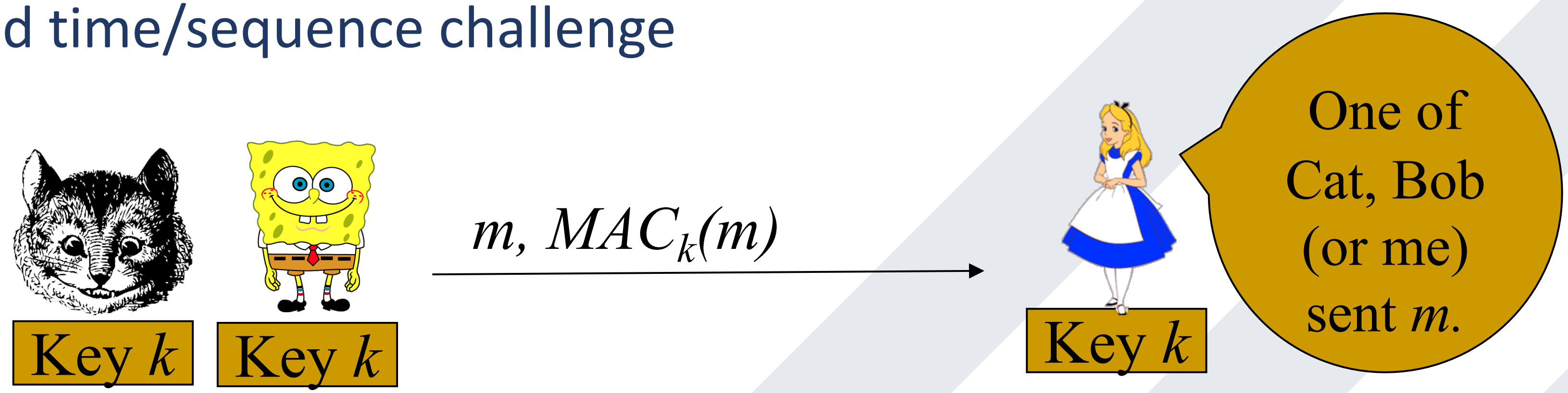
Cryptography Foundations

- **Message Authentication Code (MAC)**
  - Allow a recipient to validate that a message was sent by a key holder
  - (m, Tag) is valid iif $Tag = MAC_k(m)$



Cryptography Foundations

- **Message Authentication Code (MAC)**
  - Allow a recipient to validate that a message was sent by a key holder
  - Sender could be any key-holder including recipient
    - Specify sender and recipient in the message
  - Could be re-transmission (replay attack)
    - Add time/sequence challenge



Cryptography Foundations

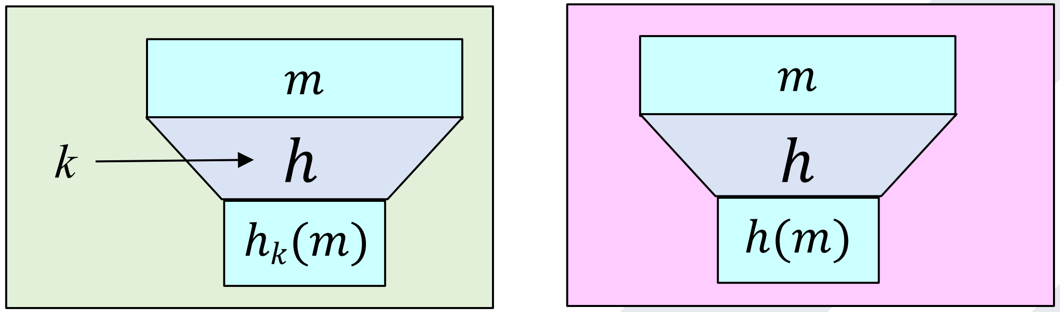
- **Hash**
  - Hash function $h(m)$ allow verification of message: **Integrity**
  - Also confidentiality: one-way function
    - Hash value $h(m)$ does not expose m
  - Collision-resistance
    - $h(m) \neq h(m')$
    - Pseudo-randomness
    - Every hash has collisions: $|input| >> |output|$
    - But hard to find collisions

Cryptography Foundations

- **Hash**
  - Hash function $h(m)$ allow verification of message: **Integrity**
  - Also confidentiality: one-way function
    - Hash value $h(m)$ does not expose m
  - Practical hash functions
    - MD5: 128-bit output; collisions found in 2004
    - SHA-1: 160-bit; theoretical analysis indicates weakness
    - SHA-2: 256/512-bit output
    - SHA-3: different design than previous SHAs; results of a public competition

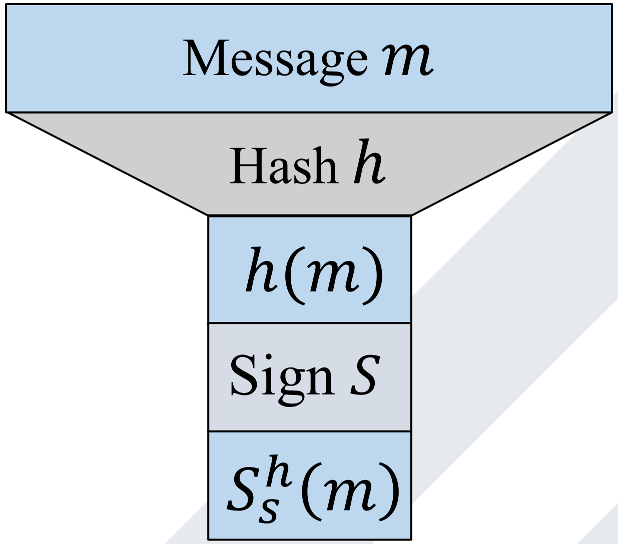
Cryptography Foundations

- **Hash**
  - Hash functions: maps arbitrary length inputs to a fixed length output
    - Input: message m (binary strings)
    - Output: (short) binary strings n (message digest)
  - Keyed or unkeyed



Cryptography Foundations

- **Hash-then-Sign Paradigm**
  - Sign long messages efficiently
    - Signature scheme $\boldsymbol{S}$ (inefficient)
    - Collision-resistant hash h: $m \neq m'$ then $h(m) \neq h(m')$
  - First hash, then sign
    - $S^h(m) = S(h(m))$



Public Key Cryptography

- **Private-key cryptography allows two users who share a secret key to establish a secure channel**
- **The need to share this secret key incurs drawbacks**
  - Key distribution problem
    - How do users share a key in the first place?
    - Need to share the key using a secure channel
    - Trusted carrier

Public Key Cryptography

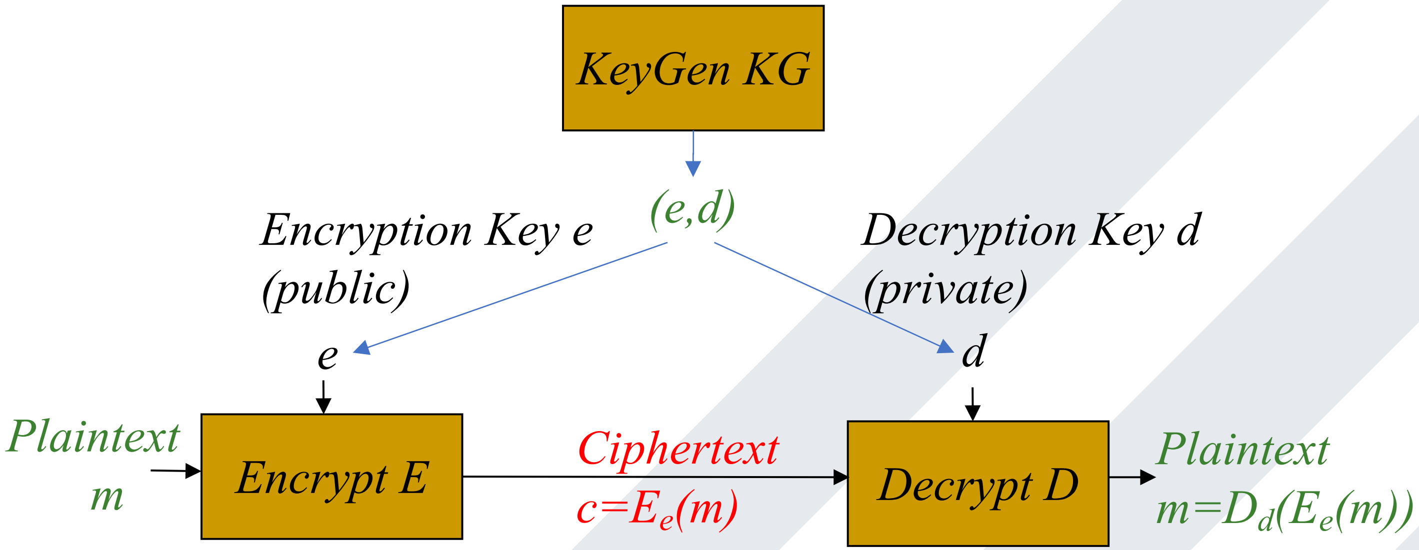
- **Private-key cryptography allows two users who share a secret key to establish a secure channel**
- **The need to share this secret key incurs drawbacks**
  - Key distribution problem
  - Key management problem
    - When each pair of users might need to communicate securely
    - $O(N^2)$ keys overall

Public Key Cryptography

- **Private-key cryptography allows two users who share a secret key to establish a secure channel**
- **The need to share this secret key incurs drawbacks**
  - Key distribution problem
  - Key management problem
  - Lack of ”open systems”
    - Two users who have no prior relationship want to communicate securely

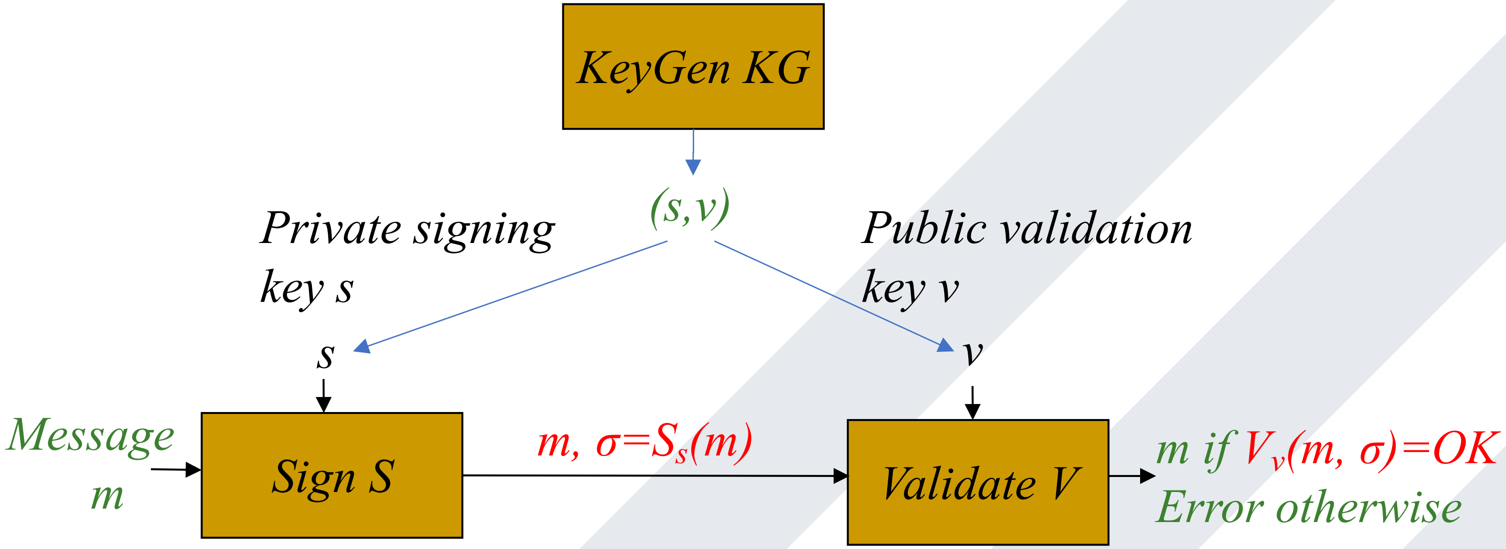
Public Key Cryptography

- **New direction: can encryption key be *public*?**
  - Anyone can encrypt the message using public encryption key
  - Decryption key will be different (and private)
    - only the key-holder can decrypt it



Public Key Cryptography

- **Public Key Cryptosystem**
  - Encryption: Public key encrypts, private key decrypts
  - Also: Authentication Signature
    - Sign with private ley, validate with public key



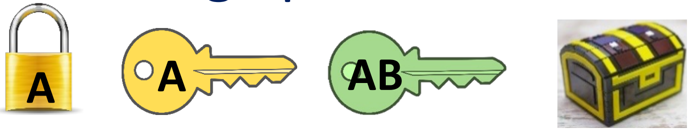
Public Key Cryptography

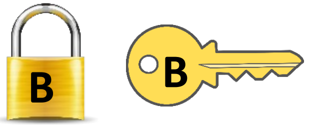
- **Public Key Cryptosystem**
  - Encryption: Public key encrypts, private key decrypts (reversable)
  - Also: Authentication Signature
    - Sign with private ley, validate with public key
  - Public key cryptosystem also has drawbacks: significantly expensive and slow
    - Public key cryptosystem: exchange a shared, private key
    - Private key encryption: establish a secure communication channel

Public Key Cryptography

- **Key-exchange protocol**
  - Alice and Bob want to agree on secret (key)
    - Secure against eavesdropping
    - No prior shared secrets

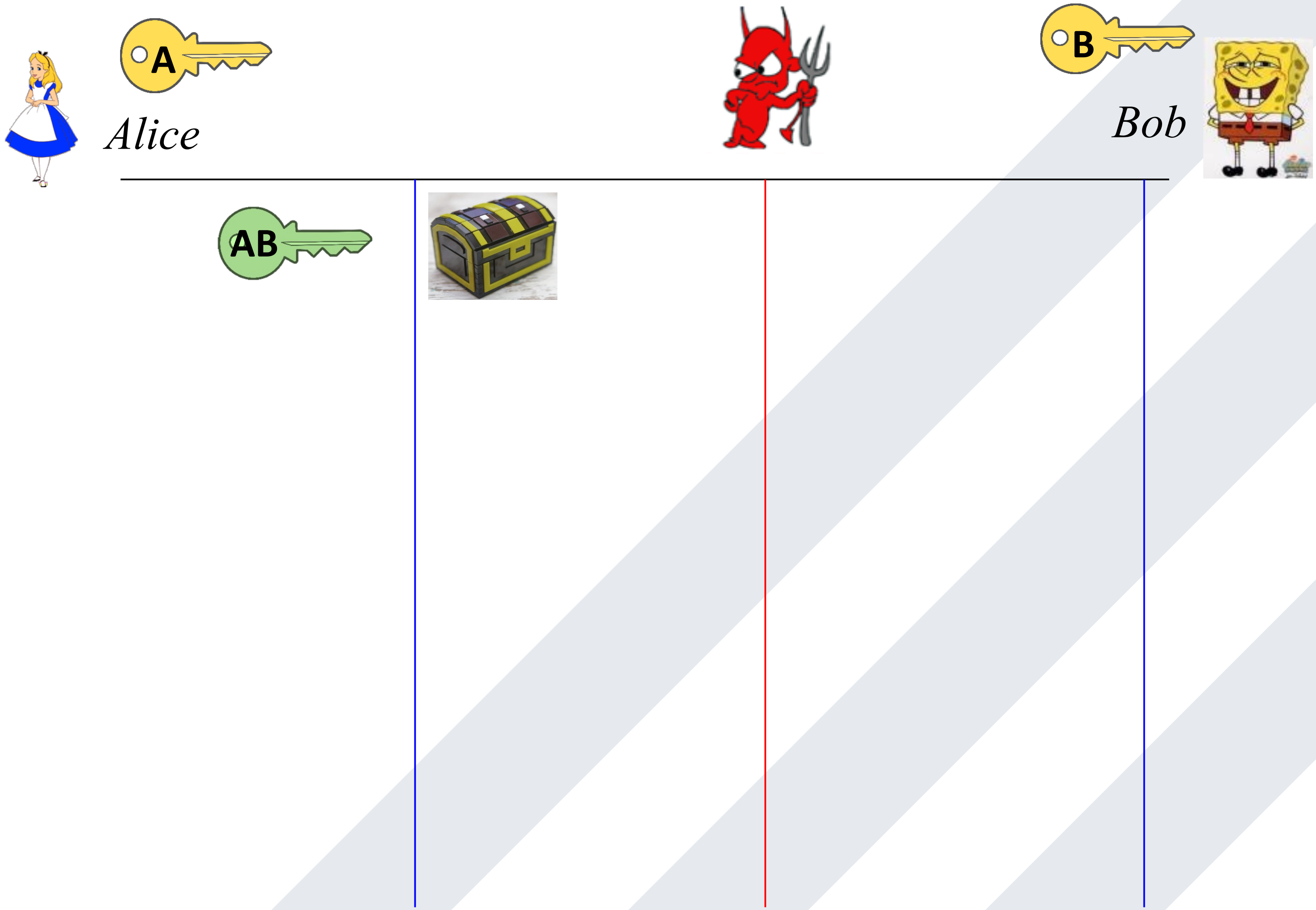
Public Key Cryptography

- **Key-exchange protocol**
  - Alice and Bob want to agree on secret (key)
    - Secure against eavesdropping
    - No prior shared secrets
  - A *physical* key-exchange problem
    - Alice has: 

    - Bob has 

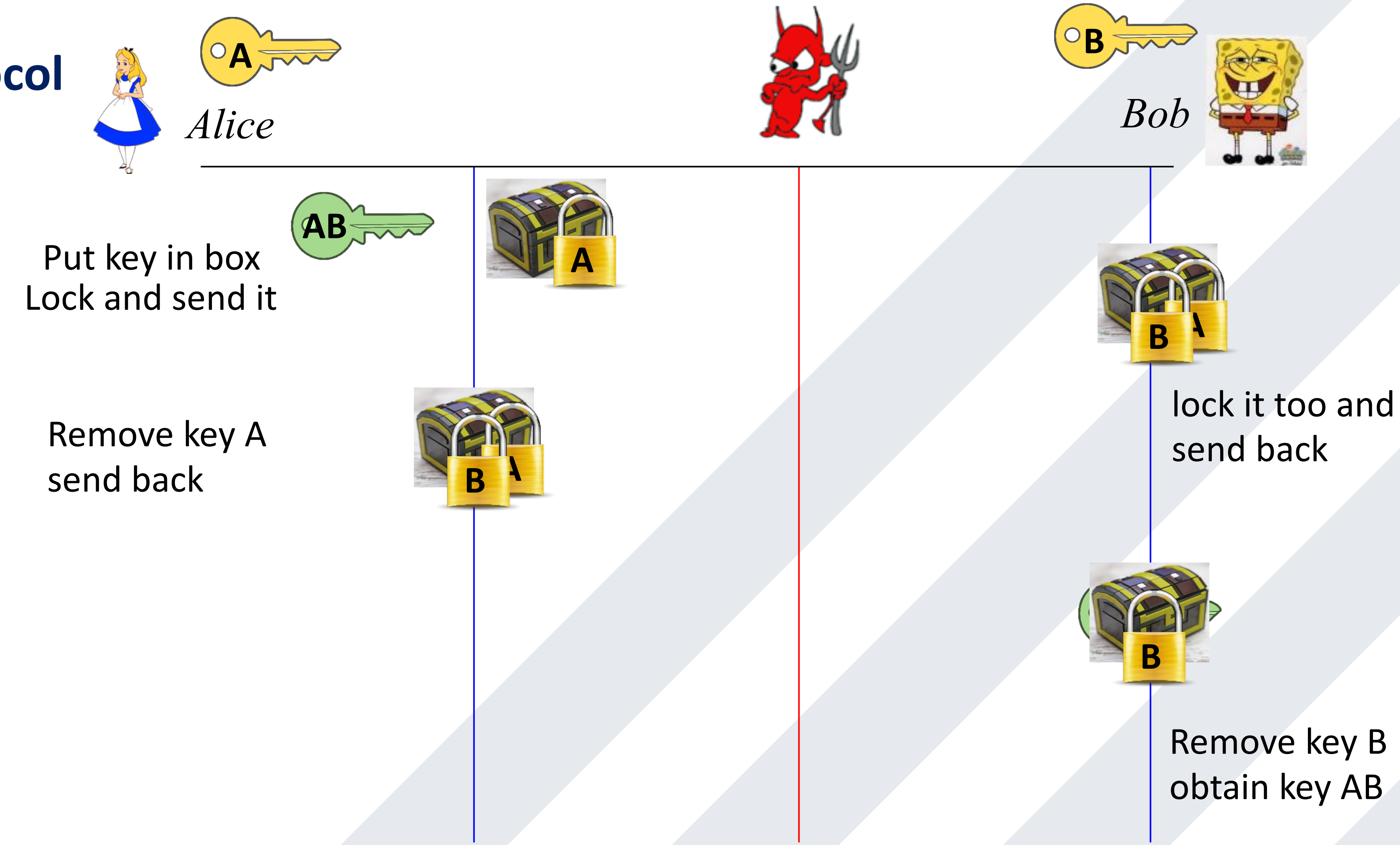
Public Key Cryptography

- **Key-exchange protocol**



Public Key Cryptography

- **Key-exchange protocol**



Public Key Cryptography

- **Public Key Cryptosystem**
  - Encryption: Public key encrypts, private key decrypts

$$m = D_{Private\_Key}(E_{Public\_Key}(m))$$

  - Public key is roughly 2-3 orders of magnitude slower than private-key crypto
    - Hybrid encryption: using public key to exchange a private key and establish a secure communication channel

Public Key Cryptography

- **Diffie-Hellman key-exchange**
  - Alice and Bob want to agree on secret (key)
    - Secure against eavesdropping
    - No prior shared secrets
  - Security goal: even after observing the messages, the shared key k should be undisguisable from a uniform key

Public Key Cryptography

- **Diffie-Hellman key-exchange**
  - Alice and Bob want to agree on secret (key)
    - Secure against eavesdropping
    - No prior shared secrets
  - Security goal: even after observing the messages, the shared key k should be indisguisable from a uniform key
  - Discrete-logarithm problem
    - Given prime p and q, and X
    - It would be easy to have $Y = p^X \mod q$
    - But it is very hard to compute X when giving Y

Public Key Cryptography

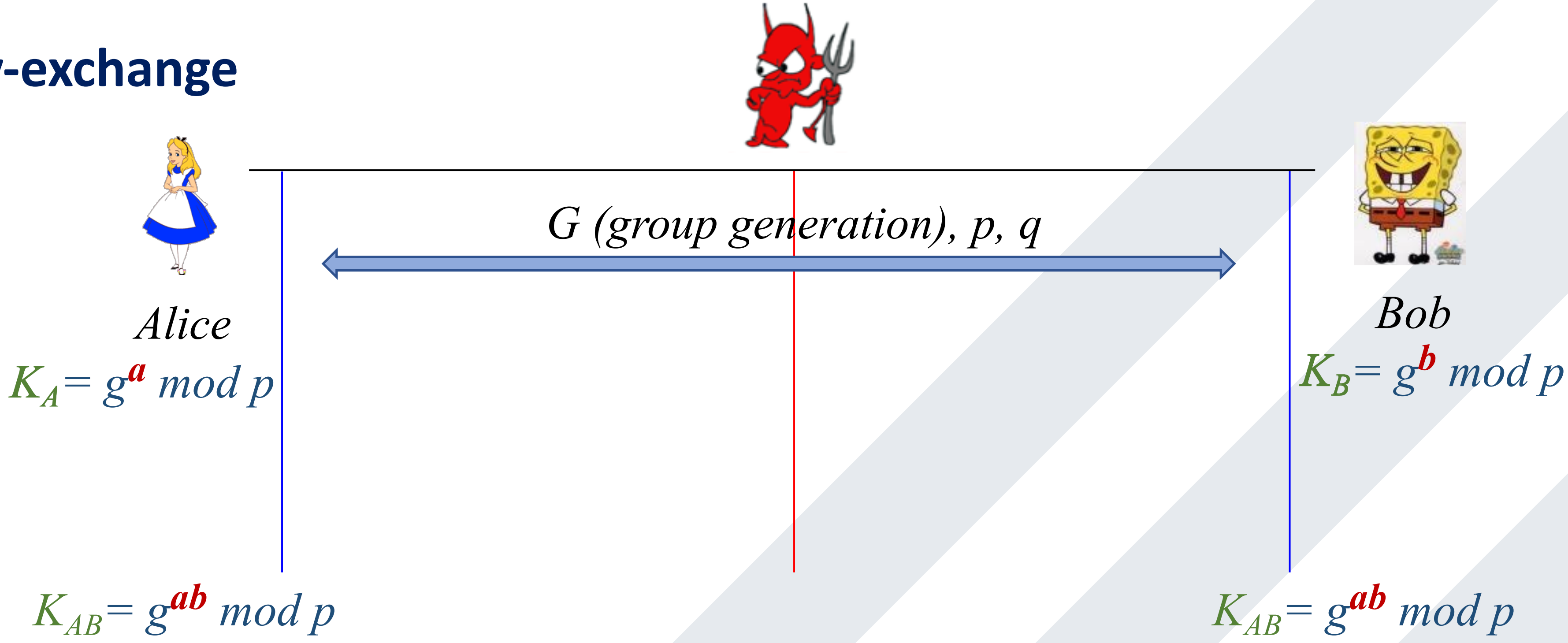
- **Diffie-Hellman key-exchange**
  - Alice and Bob want to agree on secret (key)
    - Alice and Bob agree on a random safe prime p (modulo) and a base g (which is a primitive root modulo p)
    - Alice chooses a secret key $\boldsymbol{a}$ $\longrightarrow$ public key $K_A = g^a \; mod \; p$
    - Bob chooses a secret key $\boldsymbol{b}$ $\longrightarrow$ public key $K_B = g^b \; mod \; p$
    - Alice and Bob set up a shared key

$$(g^b)^a \; mod \; p = (g^a)^b \; mod \; p = g^{ab} \; mod \; p$$

  - Only $\boldsymbol{a}$ and $\boldsymbol{b}$ are keeping secret

Public Key Cryptography

- **Diffie-Hellman key-exchange**



Does Diffie-Hellman secure the communication channel?

Public Key Cryptography

- **Diffie-Hellman key-exchange**

G (group generation), p, q

Alice

$K_A = g^{a} \bmod p$

Bob

$K_B = g^{b} \bmod p$

K_T

$K_T = g^{T} \bmod p$

$K_{AB} = g^{aT} \bmod p$

$K_{AB} = g^{Tb} \bmod p$

Does Diffie-Hellman secure the communication channel?

Authenticate the public key

Public Key Cryptography

- **Public Key distribution**
  - Encryption: Public key encrypts, private key decrypts

$$m = D_{Private\_Key}(E_{Public\_Key}(m))$$

  - Assume the parties are able to obtain the correct copies of (each other's) public key

Public Key Cryptography

- **Public Key Certificate**
  - Use signatures for secure key distribution
    - Assume a trusted party providing authentication
    - **Certificate Authority** (CA)
    - Public key $CA.e$
    - Private key $CA.s$

Public Key Cryptography

- **Public Key Certificate**
  - Use signatures for secure key distribution
    - Assume a trusted party providing authentication
    - **Certificate Authority** (CA)
    - Public key $CA.e$
    - Private key $CA.s$
  - Bob asks the CA to sign the binding $(Bob, P_{Bob.e})$
    - $Cert_{CA \rightarrow Bob} = Sign_{CA.s}(Bob, P_{Bob.e})$
    - CA must verify Bob’s identity out of band

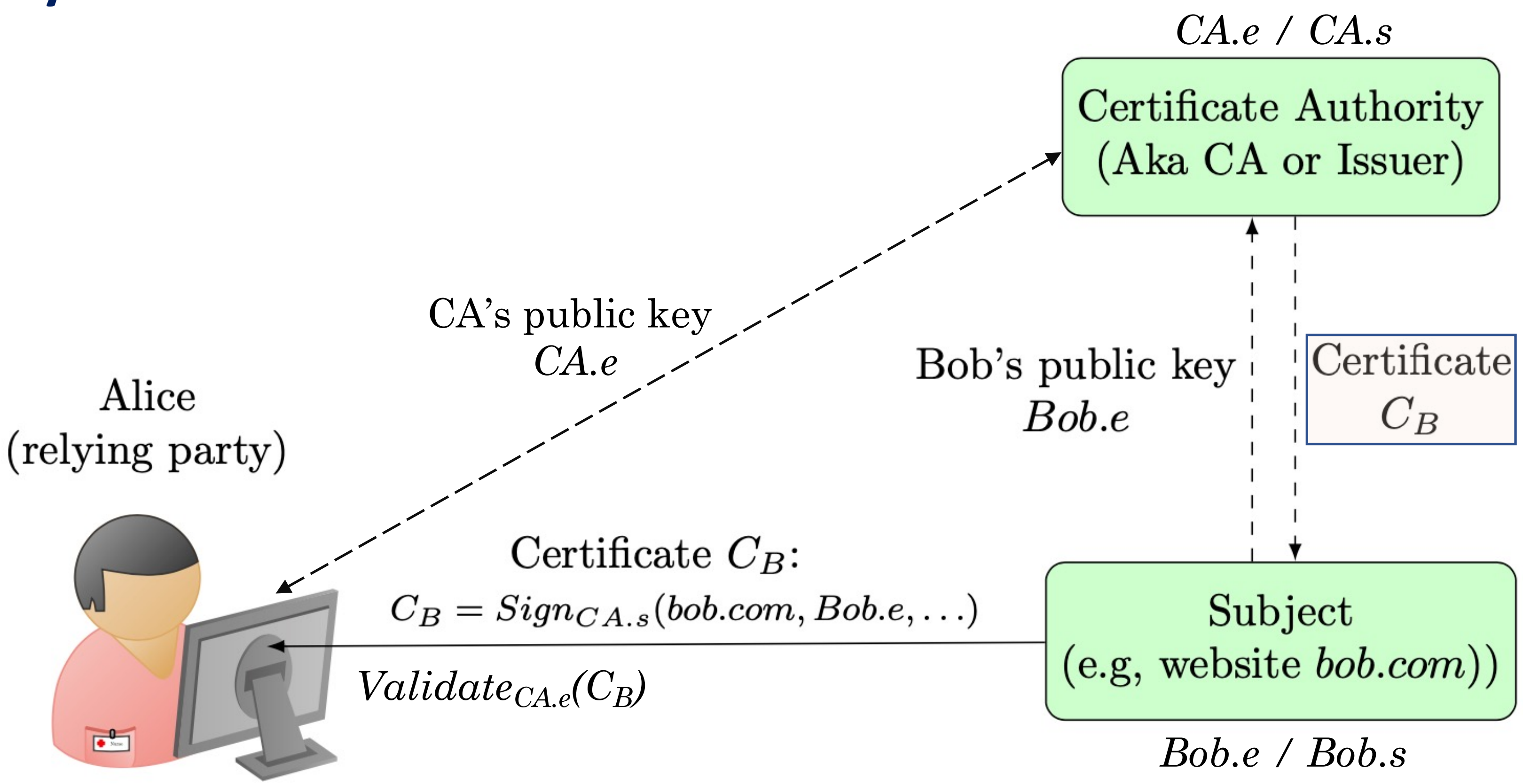
Public Key Cryptography

- **Public Key Certificate**
  - Bob asks the CA to sign the binding $(Bob, P_{Bob.e})$
    - $Cert_{CA\to Bob} = Sign_{CA.s}(Bob, P_{Bob.e})$
    - CA must verify Bob's identity out of band
  - Alice obtains and wants to verify $(Bob, P_{Bob.e})$
    - Alice obtains $Cert_{CA\to Bob}$
    - Alice verifies that $Validate_{CA.e}(Bob, P_{Bob.e}, Cert_{CA\to Bob})$

$$Validate_{CA.e}(Cert_{CA\to Bob})$$

Public Key Infrastructure (PKI)

- **Public Key Certificate**

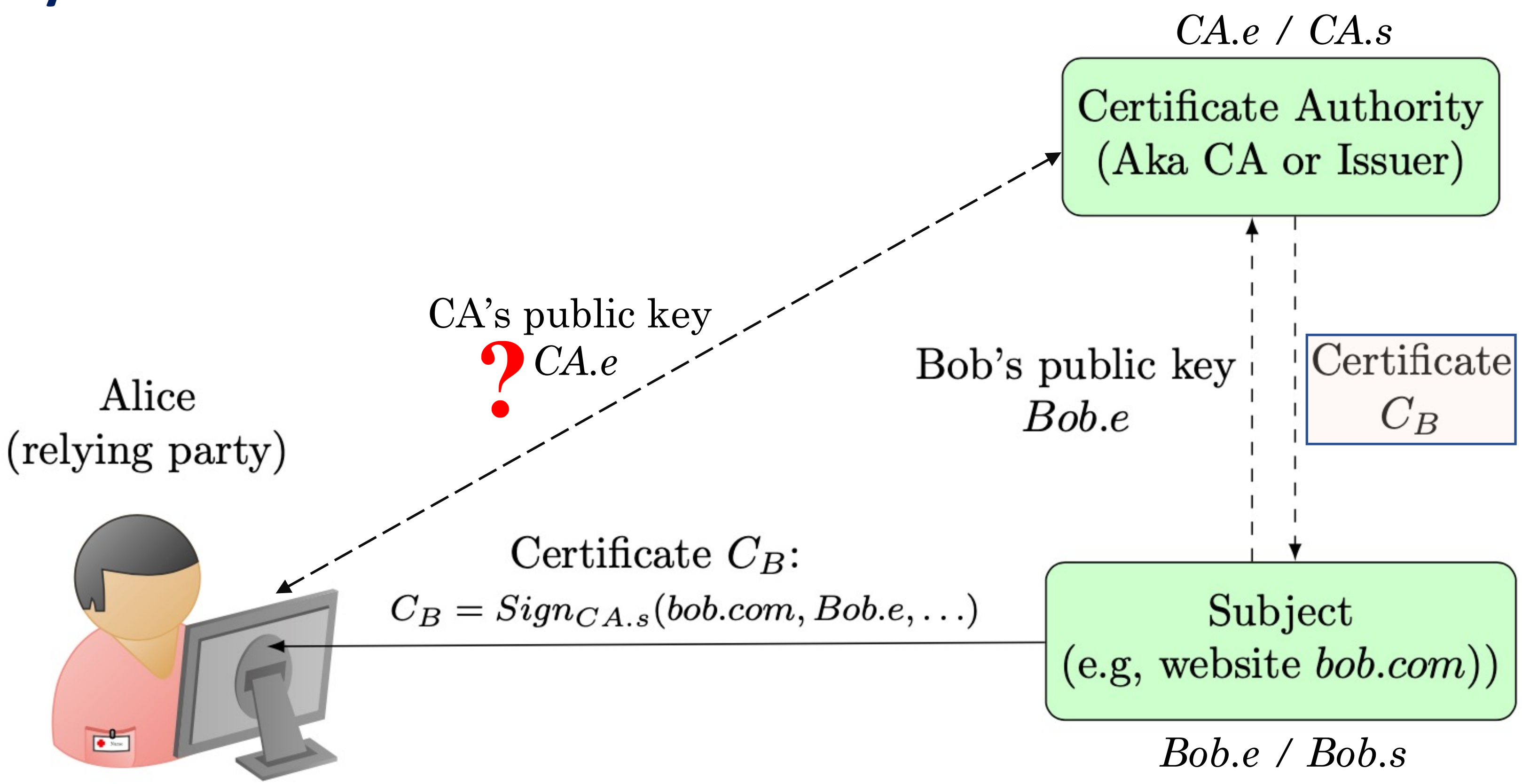


Public Key Infrastructure (PKI)

- **Public Key Certificate**
  - Bob asks the CA to sign the binding $(Bob, P_{Bob.e})$
    - $Cert_{CA \to Bob} = Sign_{CA.s}(Bob, P_{Bob.e})$
    - CA must verify Bob's identity out of band
  - Alice obtains and wants to verify $(Bob, P_{Bob.e})$
    - Alice obtains $Cert_{CA \to Bob}$
    - Alice verifies that $Validate_{CA.e}(Bob, P_{Bob.e}, Cert_{CA \to Bob})$
  - As long as …
    - CA is trustworthy and CA's key pair has not been compromised

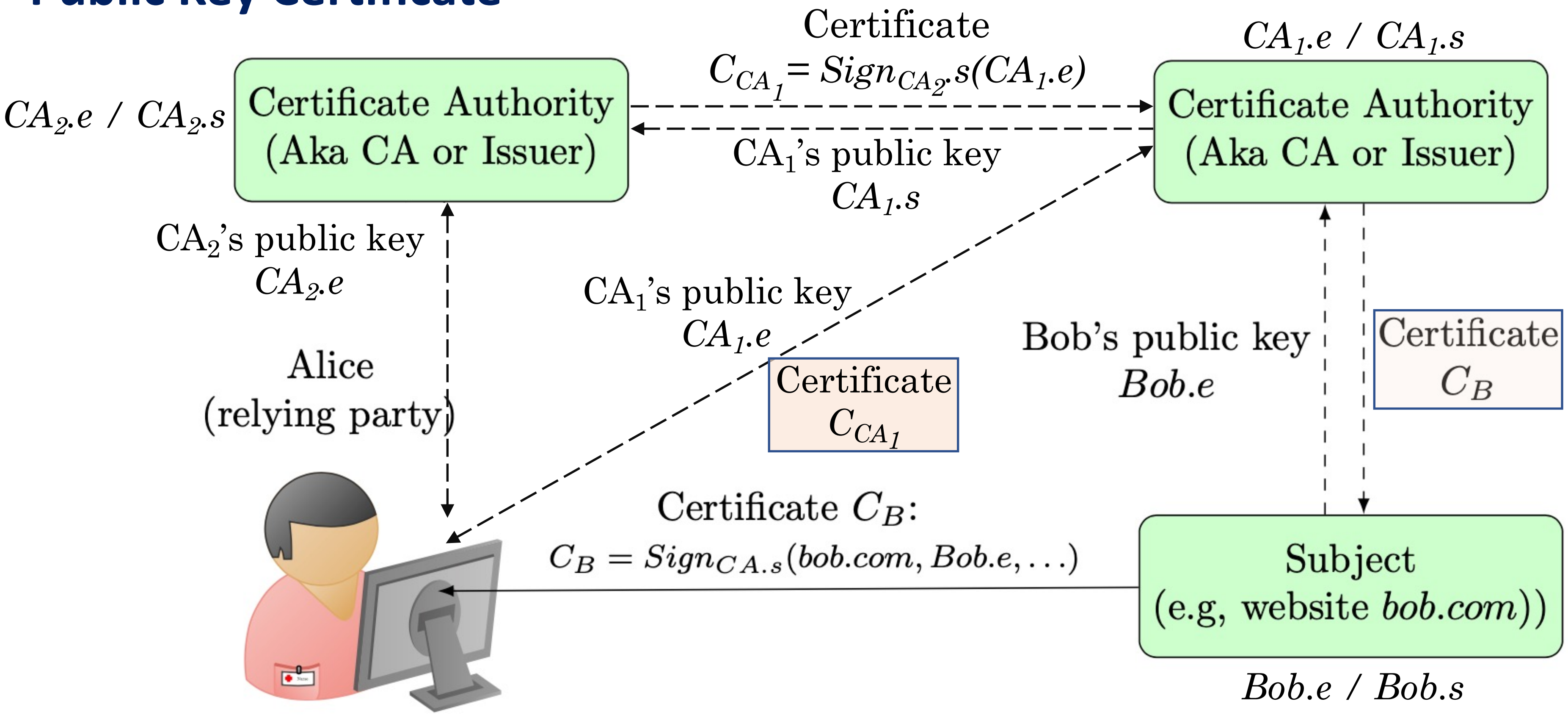
Public Key Infrastructure (PKI)

- **Public Key Certificate**



Public Key Infrastructure (PKI)

- **Public Key Certificate**



Public Key Infrastructure (PKI)

- **Root-of-Trust**
  - Alice will only need to securely obtain a small number of Public key $CA.e$
    - Ensure secure distribution for few initial $CA.e$
  - Root CAs
    - Root CAs issues Certificate for intermediate CA $Cert_{Root\_CA.s \rightarrow CA}$

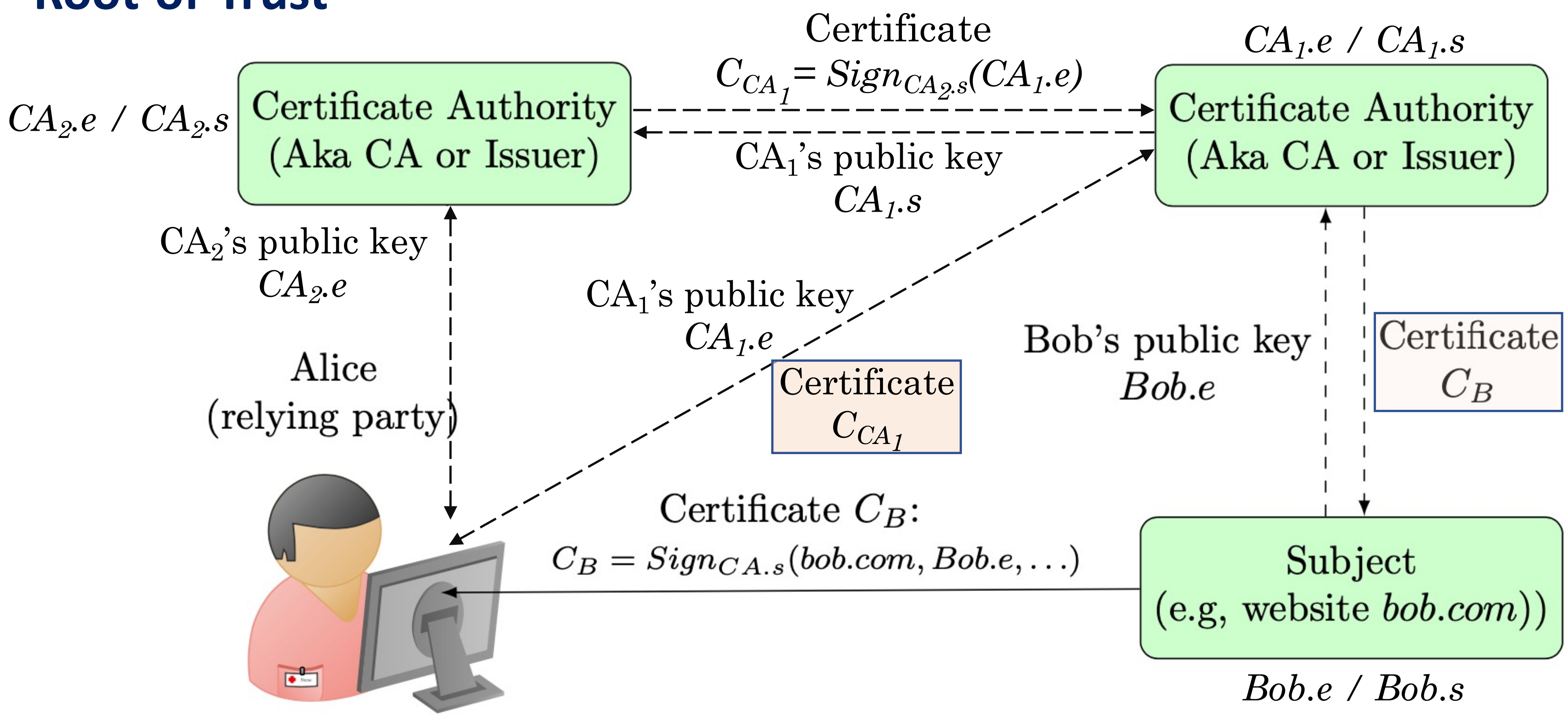
$$Validate_{Root\_CA.e}(Cert_{Root\_CA.s \rightarrow CA})$$

    - Intermediate CAs issue Certificate for subject (website)

$$Validate_{CA.e}(Cert_{CA.s \rightarrow Bob})$$

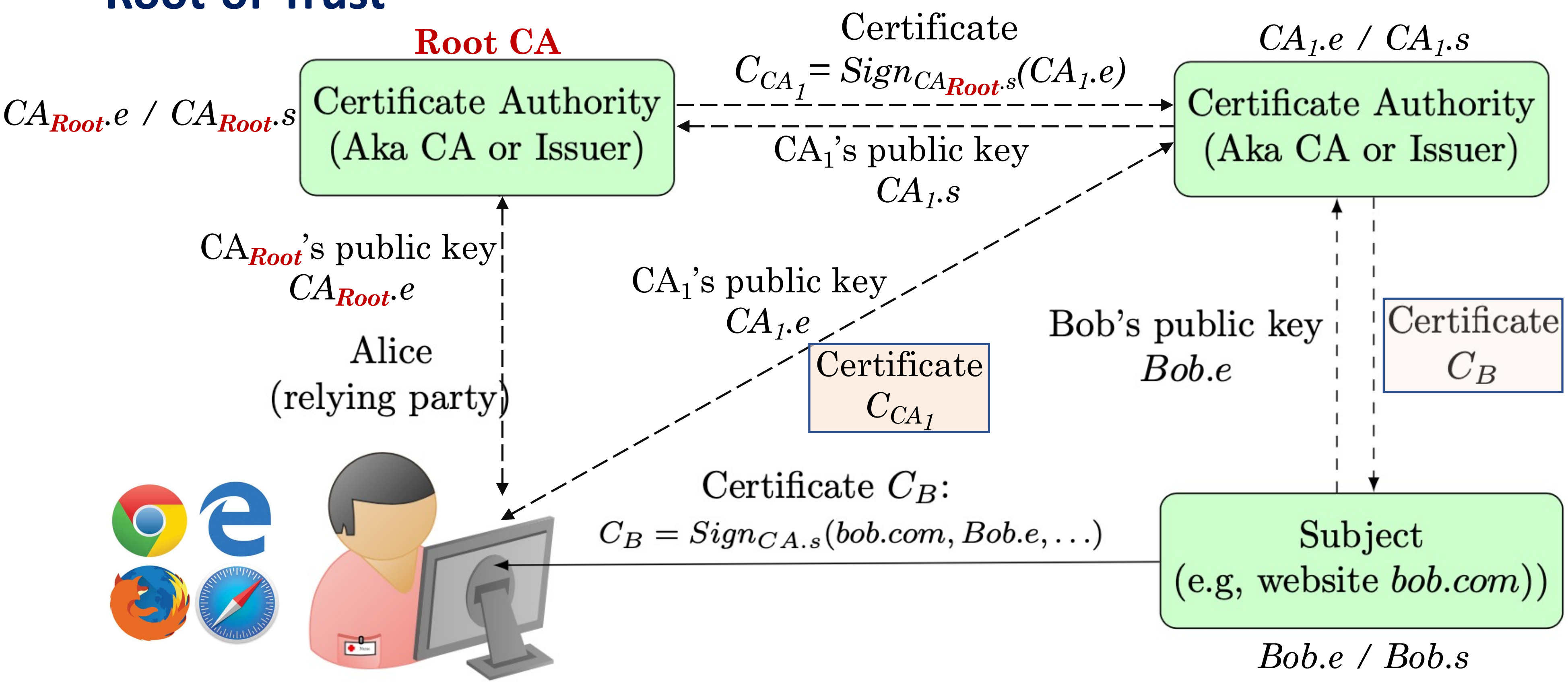
Public Key Infrastructure (PKI)

- **Root-of-Trust**

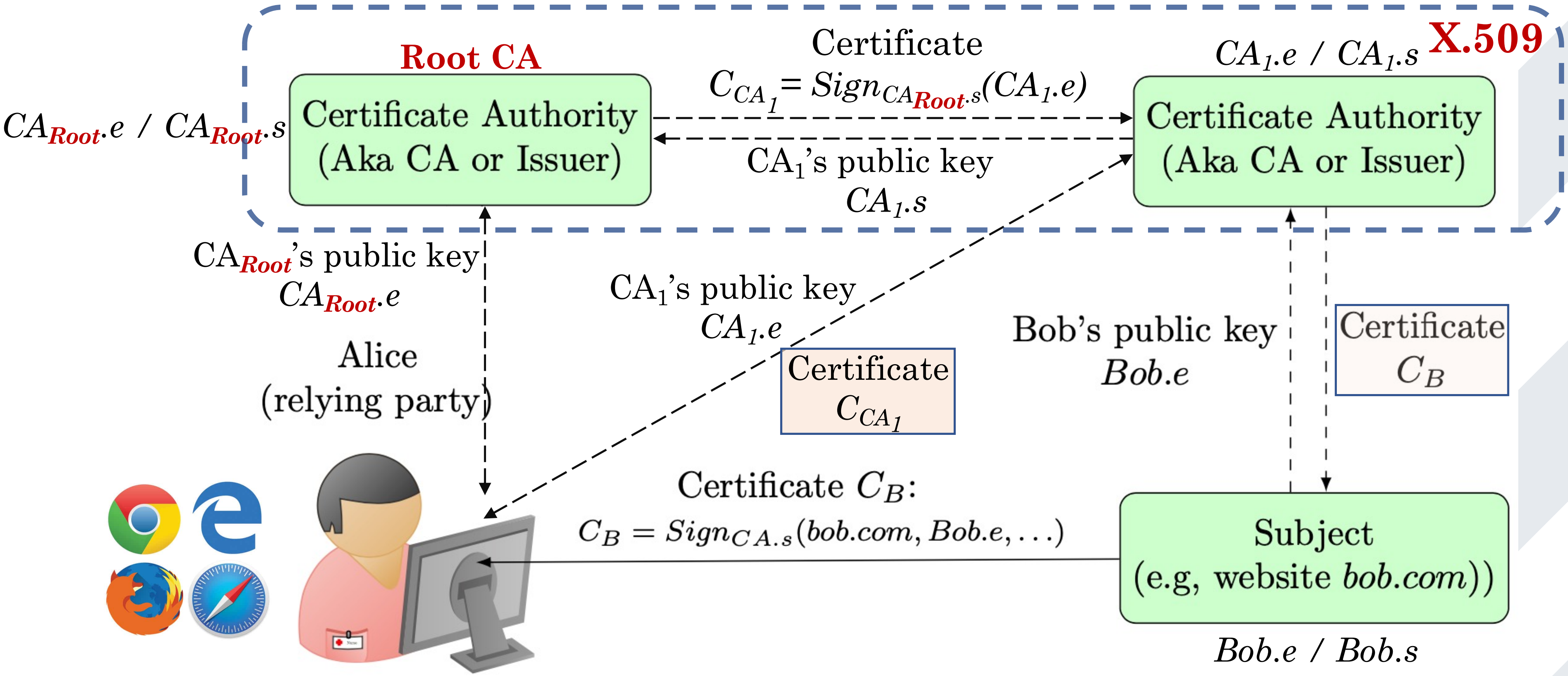


Public Key Infrastructure (PKI)

- **Root-of-Trust**



Public Key Infrastructure (PKI)



Public Key Infrastructure (PKI)

- **Dealing with CA failures**
  - Certificates are all about Trust

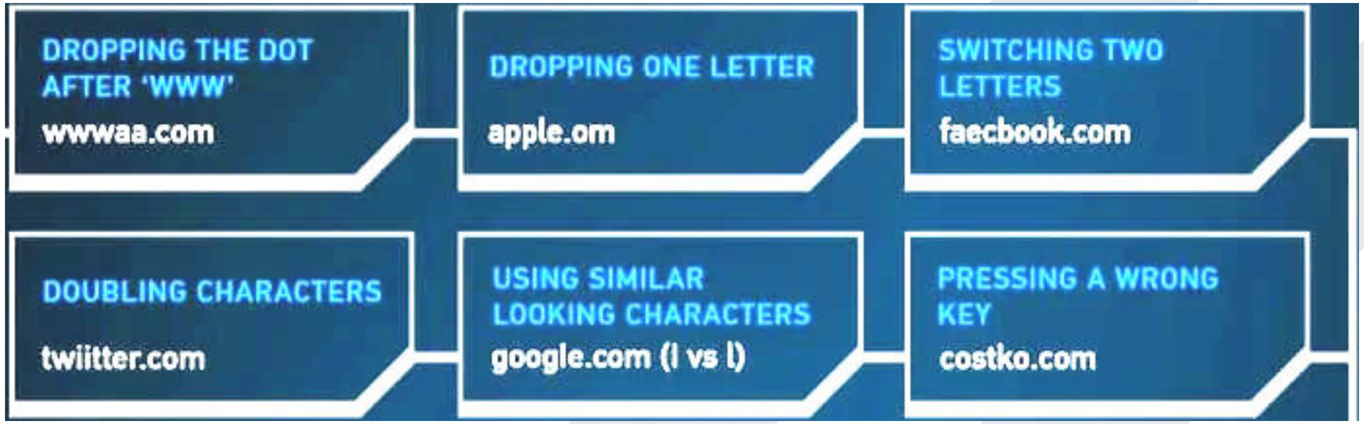
$$Cert_{CA \rightarrow Bob} = Sign_{CA.s}(Bob,\ P_{Bob.e})$$

Public Key Infrastructure (PKI)

- **Dealing with CA failures**
  - Certificates are all about Trust

$$Cert_{CA \rightarrow Bob} = Sign_{CA.s}(Bob,\ P_{Bob.e})$$

  - Equivocating or misleading (domain) name (Rogue Certificates)
    - Intentionally signed and issued by malicious CAs Certificates
    - Squatting misleading names

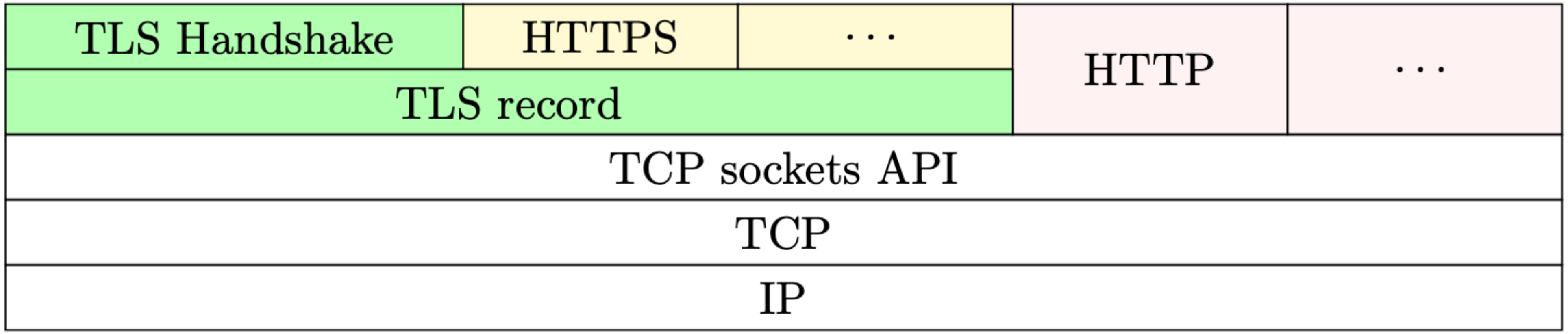


TLS and SSL

- **Securing the Web in practice**
  - SSL: Secure Socket Layer (Netscape, mid-'90s)
  - TLS: Transport Layer Security
    - For standardizing SSL
    - TLS 1.0 (1999)
    - TLS 1.2 (2008, current)
    - TLS 1.3 (adopting)
    - Used by every web browser for HTTPS connections

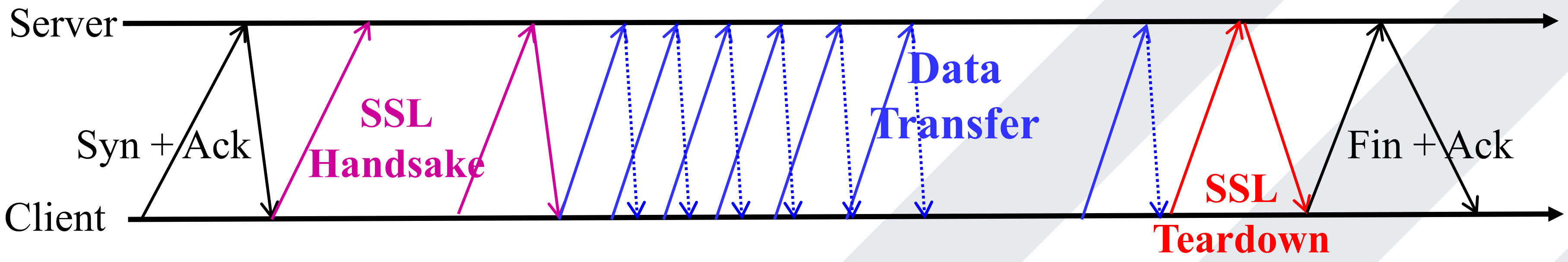
TLS and SSL

- **Securing the Web in practice**
  - SSL: Secure Socket Layer (Netscape, mid-’90s)
  - TLS: Transport Layer Security



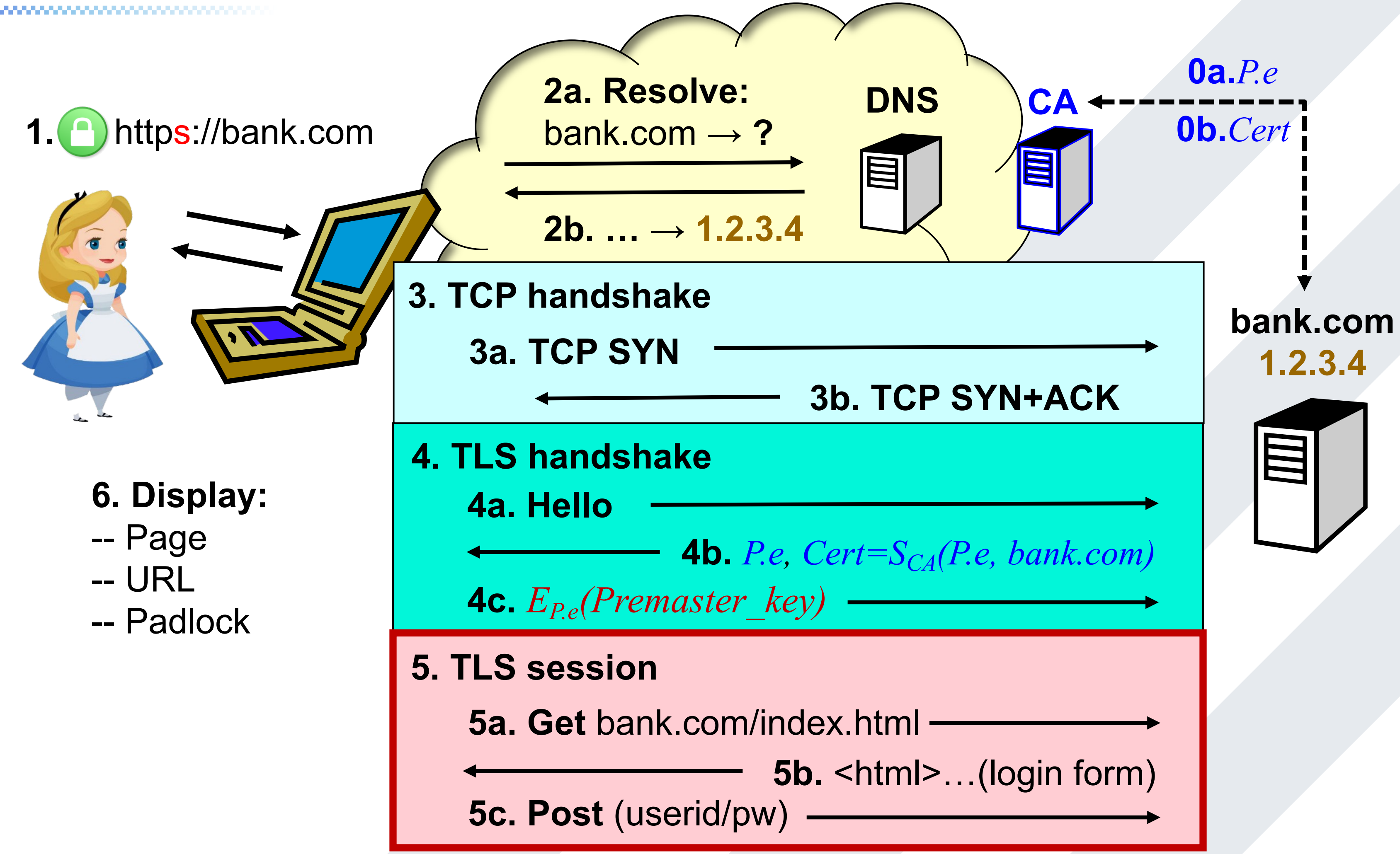
TLS and SSL

- **TLS/SSL Operations**
  - Handshake layer
    - Server/client authentication, cipher suite negotiation, key exchange
  - Record layer
    - Secure communications between client and server using exchanged session keys



TLS and SSL

- **TLS/SSL Operations**
  - HTTPS



TLS and SSL

- **TLS/SSL Operations**
  - HTTPS

4. TLS handshake

4a. Hello ⟶

⟵ **4b.** *P.e, Cert=*S_{CA}*(P.e, bank.com)*

4c. $E_{P.e}$*(Premaster_key)* ⟶

TLS and SSL

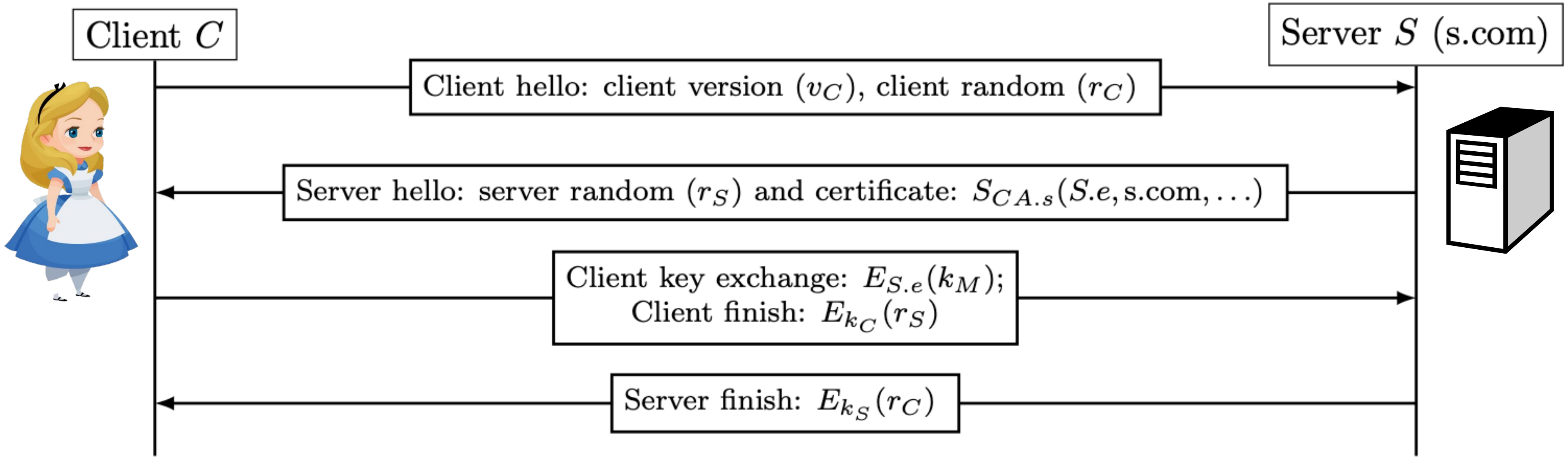
- **Handshake Layer**

4. TLS handshake

4a. Hello →

← **4b.** *P.e, Cert=S_{CA}(P.e, bank.com)*

4c. *$E_{P.e}$(Premaster_key)* →



- r_C and r_S: Nonces for protecting against replay

TLS and SSL

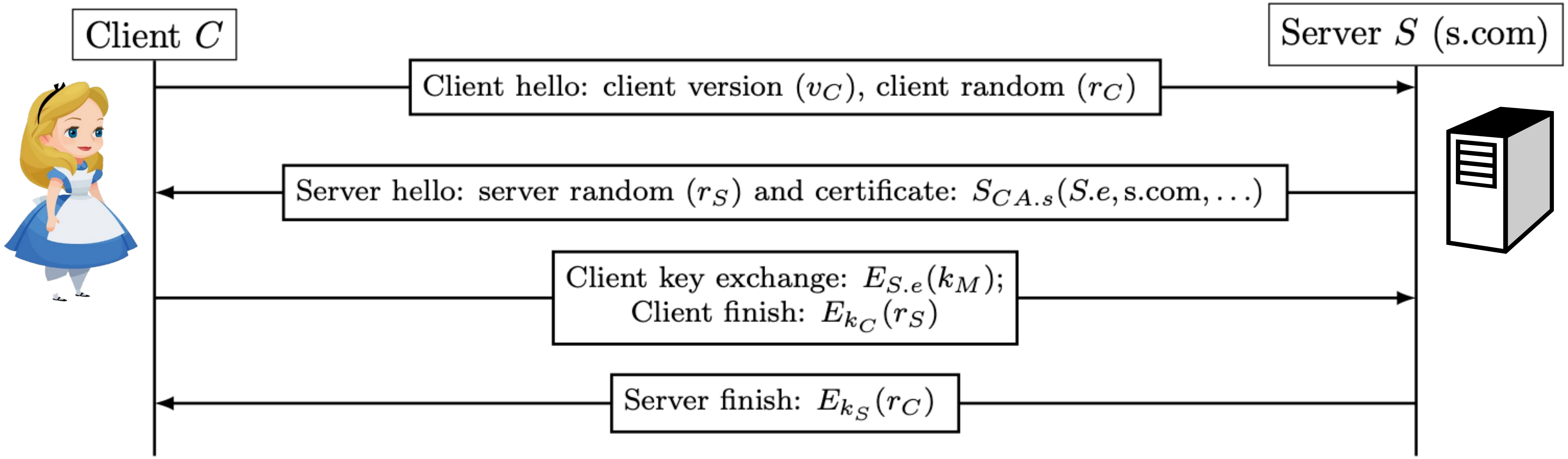
- **Handshake Layer**

4. TLS handshake

4a. Hello →

← **4b.** *P.e, Cert=S_{CA}(P.e, bank.com)*

4c. *$E_{P.e}$(Premaster_key)* →



- k_C and k_S: derived from the master key k_M

TLS and SSL

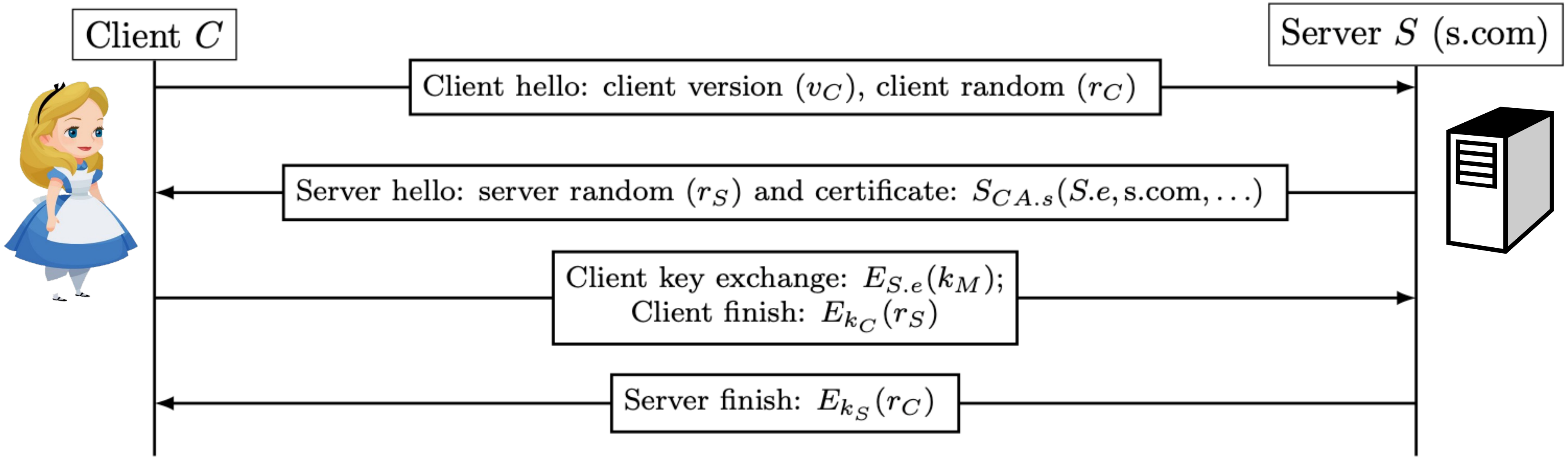
- **Handshake Layer**

4. TLS handshake

4a. Hello →

← **4b.** $P.e$, $Cert{=}S_{CA}(P.e,\ bank.com)$

4c. $E_{P.e}(Premaster\_key)$ →



- k_C: protecting traffic from client to server
- k_S: protecting traffic from server to client

TLS and SSL

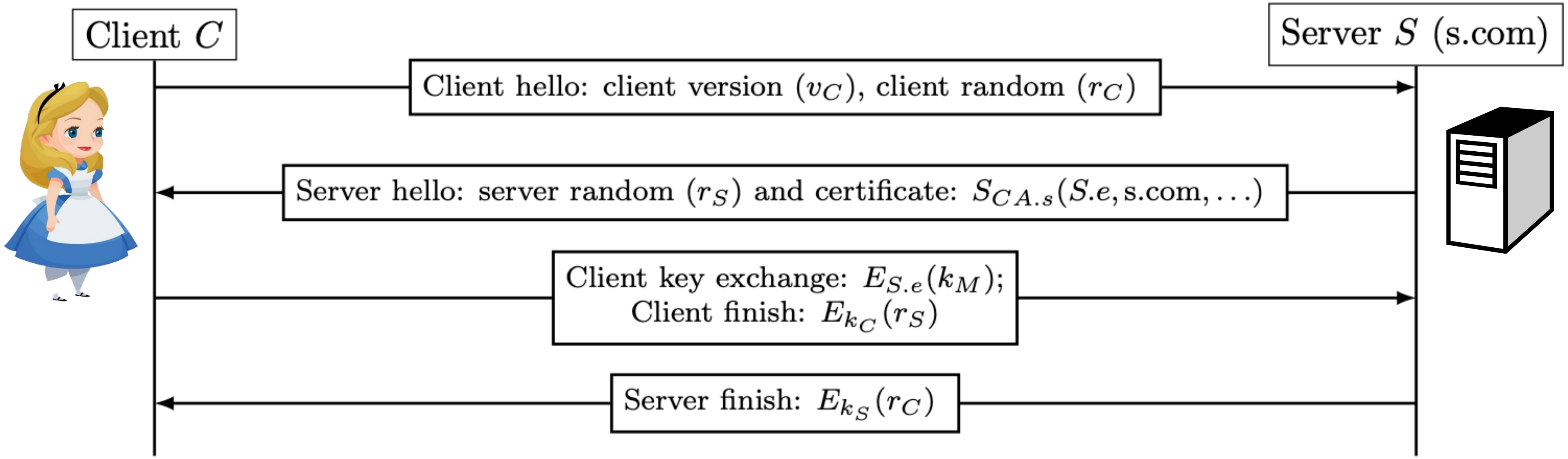
- **Handshake Layer**

4. TLS handshake

4a. Hello $\longrightarrow$

$\longleftarrow$ **4b.** *P.e, Cert=S_{CA}(P.e, bank.com)*

4c. *$E_{P.e}$(Premaster_key)* $\longrightarrow$



- Attacker cannot use the large amount of known plaintext sent from server to client, to cryptanalyze the ciphertext sent from client to server

TLS and SSL

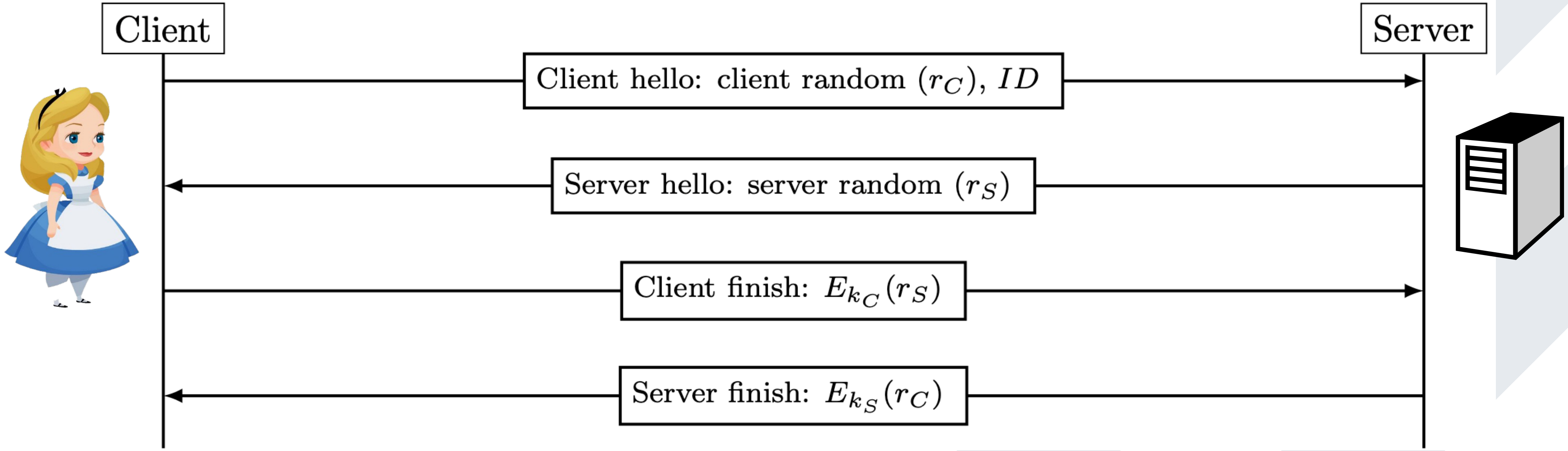
4. TLS handshake

4a. Hello →

← 4b. *P.e, Cert=*S_{CA}*(P.e, bank.com)*

4c. $E_{P.e}$*(Premaster_key)* →

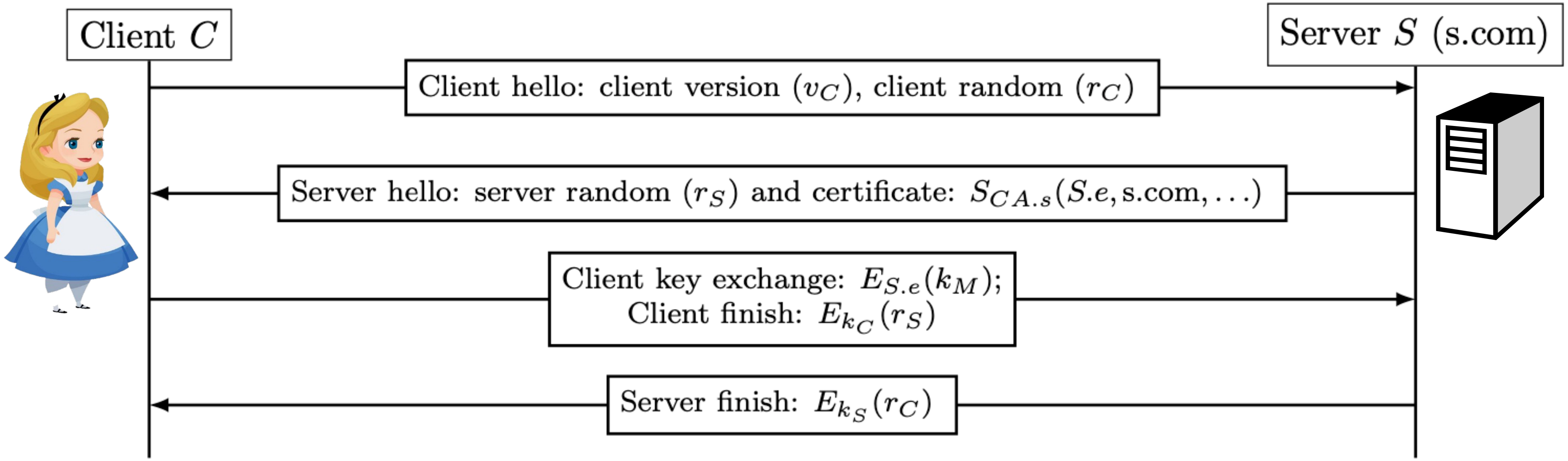
- **Handshake Layer**
  - Public key reuse



- Client identifies cached key by rendering session ID
- Server will send only Nonce and new ID

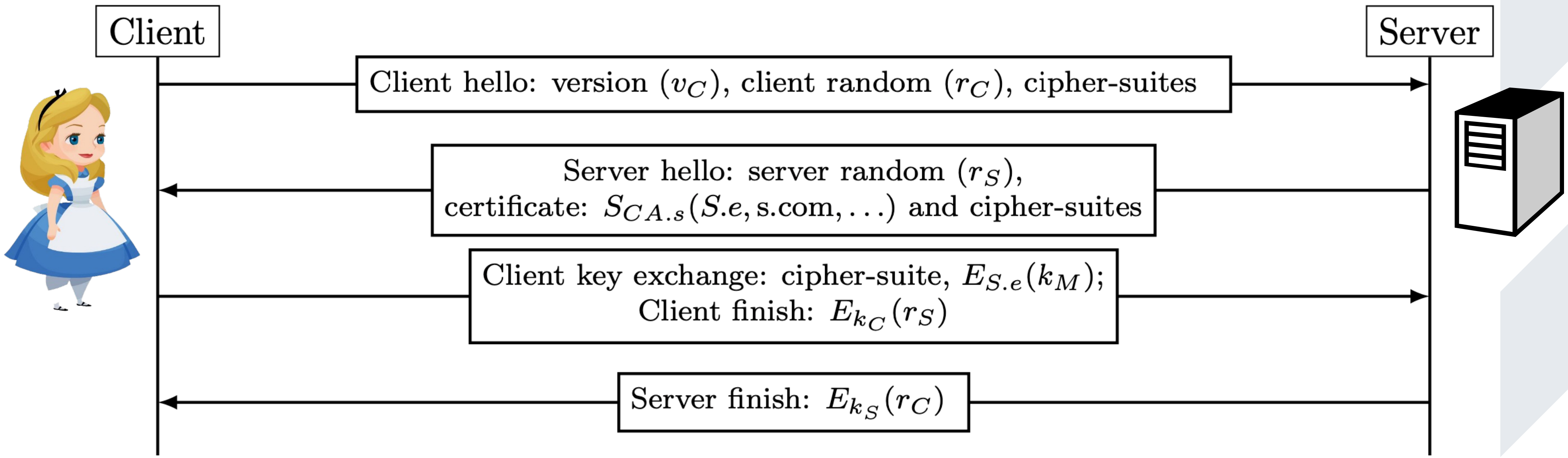
TLS and SSL

- **Handshake Layer**



TLS and SSL

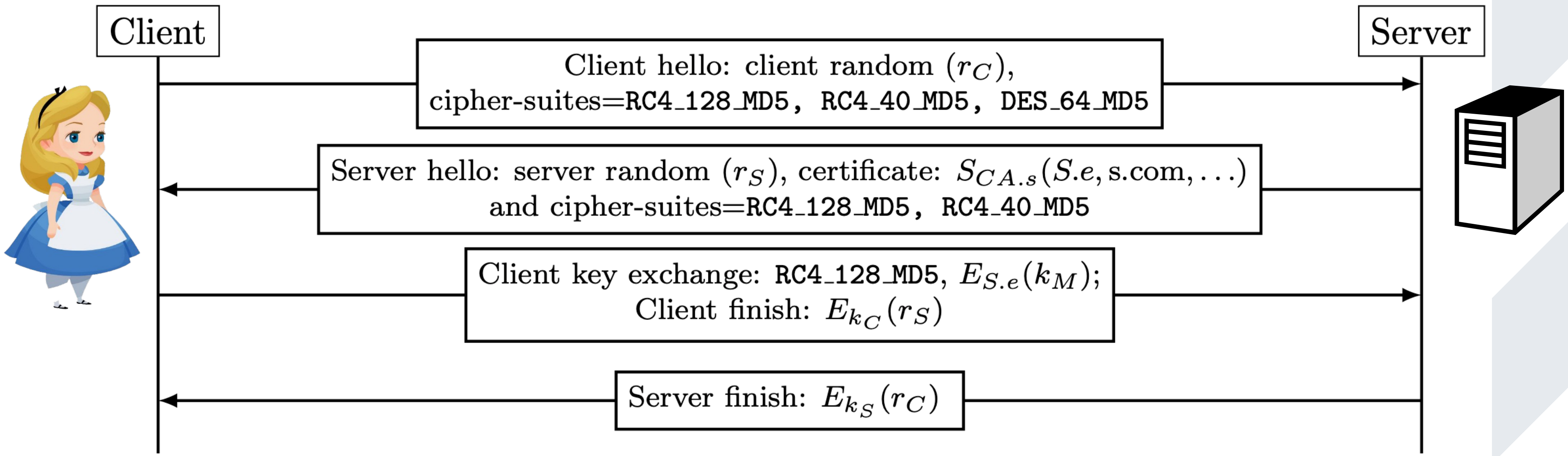
- **Cipher-suite negotiation** (SSLv2)



- Vulnerable to downgrade attack

TLS and SSL

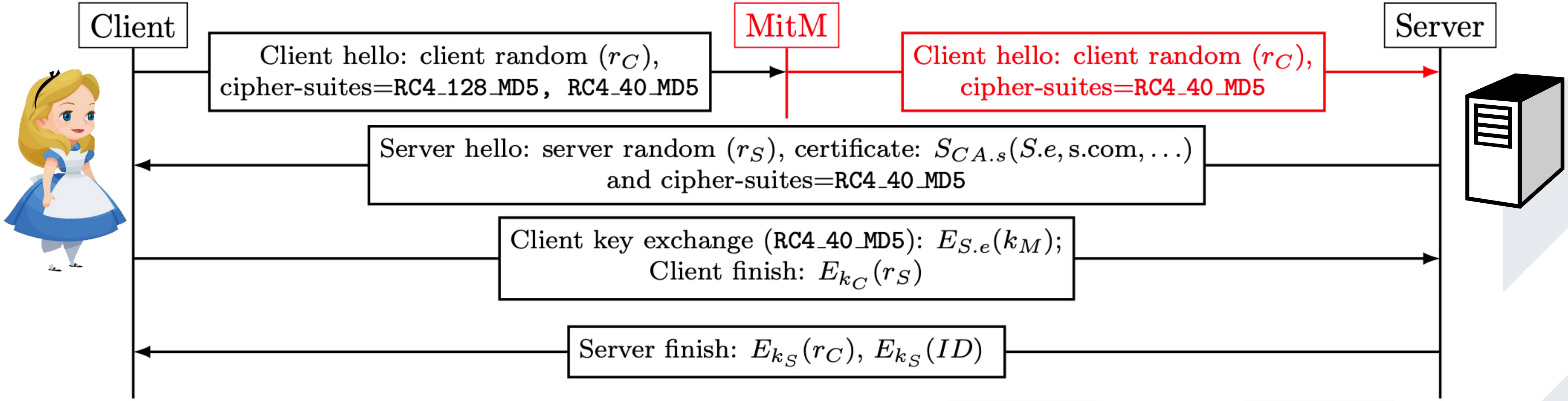
- **Cipher-suite negotiation** (SSLv2)



  - Vulnerable to downgrade attack

TLS and SSL

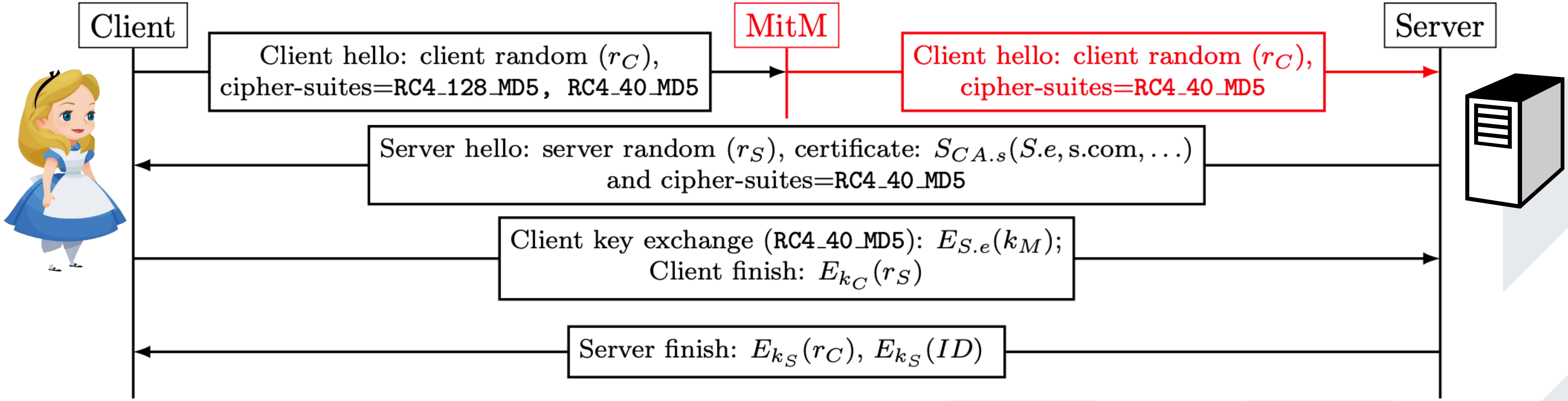
- **Cipher-suite negotiation** (SSLv2)



  - Vulnerable to downgrade attack

TLS and SSL

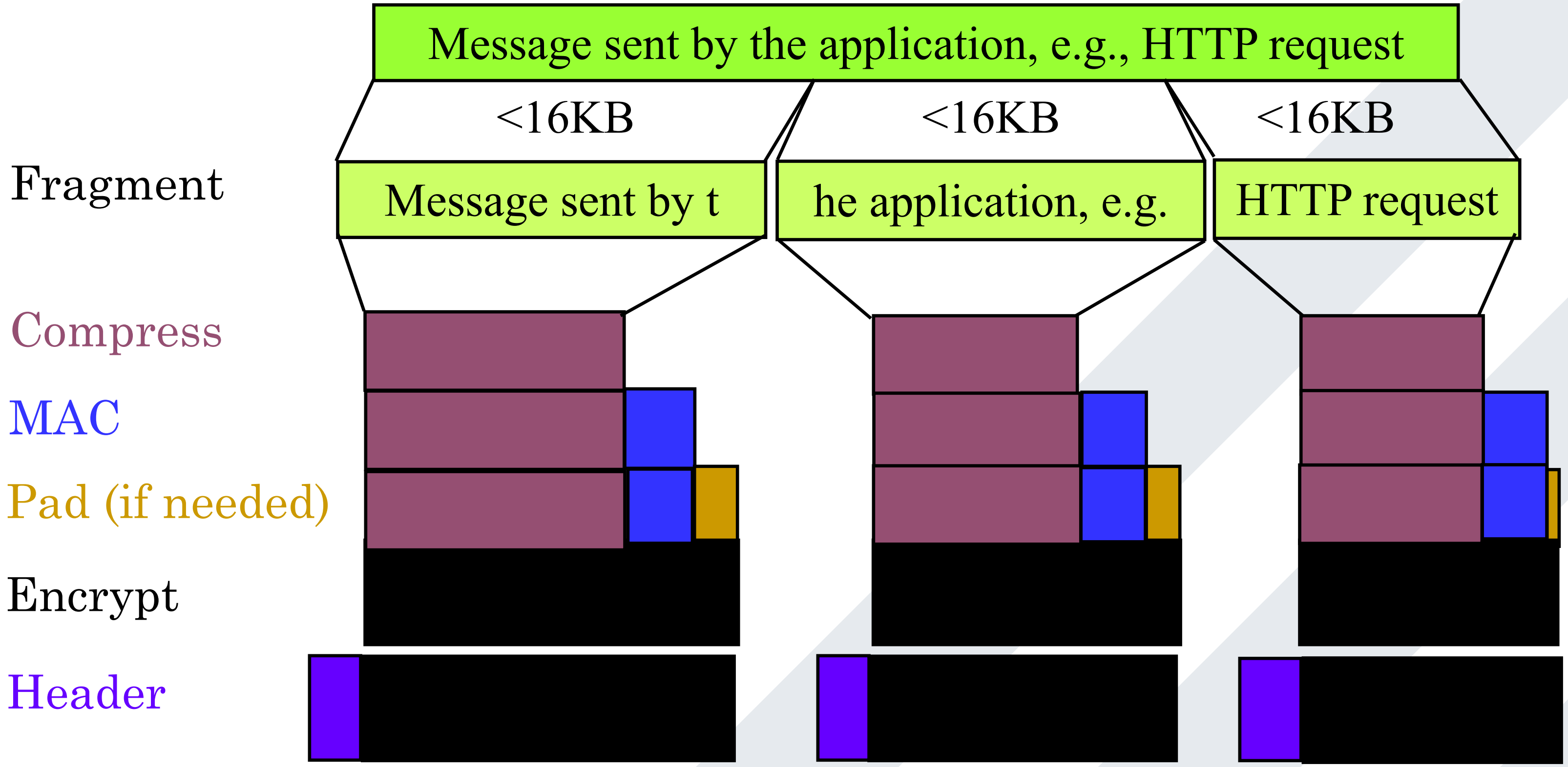
- **Cipher-suite negotiation** (SSLv2)



- SSLv3 improvement: authenticate the handshake message with the *finish* message

TLS and SSL

- **Record layer**
  - Assume reliable underlying communication (TCP)



PKI: Revoking certificates

- **Reasons for revoking certificate**
  - Key compromise
  - CA compromise
  - Affiliation changed - Object names or attribute)
  - Cessation – no longer needed
- **How to inform replying parties?**
  - Wait for end of validity period
  - Distribute **Certificate Revocation List**
  - Online Certificate Status Protocol

PKI: Revoking certificates

- **Reasons for revoking certificate**
  - Key compromise
  - CA compromise
  - Affiliation changed - Object names or att
  - Cessation – no longer needed
- **How to inform replying parties?**
  - Wait for end of validity period
  - Distribute **Certificate Revocation List**
  - Online Certificate Status Protocol

X.509 CRL

Signed fields

Version of CRL format			
Signature Algorithm Object Identifier (OID)			
CRL Issuer Distinguished Name (DN)			
This update (date/time)			
Next update (date/time) - optional			
Subject (user) Distinguished Name (DN)			
CRL Entry	Certificate Serial Number	Revocation Date	CRL entry extensions
CRL Entry…	Serial…	Date…	extensions
….			
CRL Extensions			
Signature on the above fields			

PKI: Revoking certificates

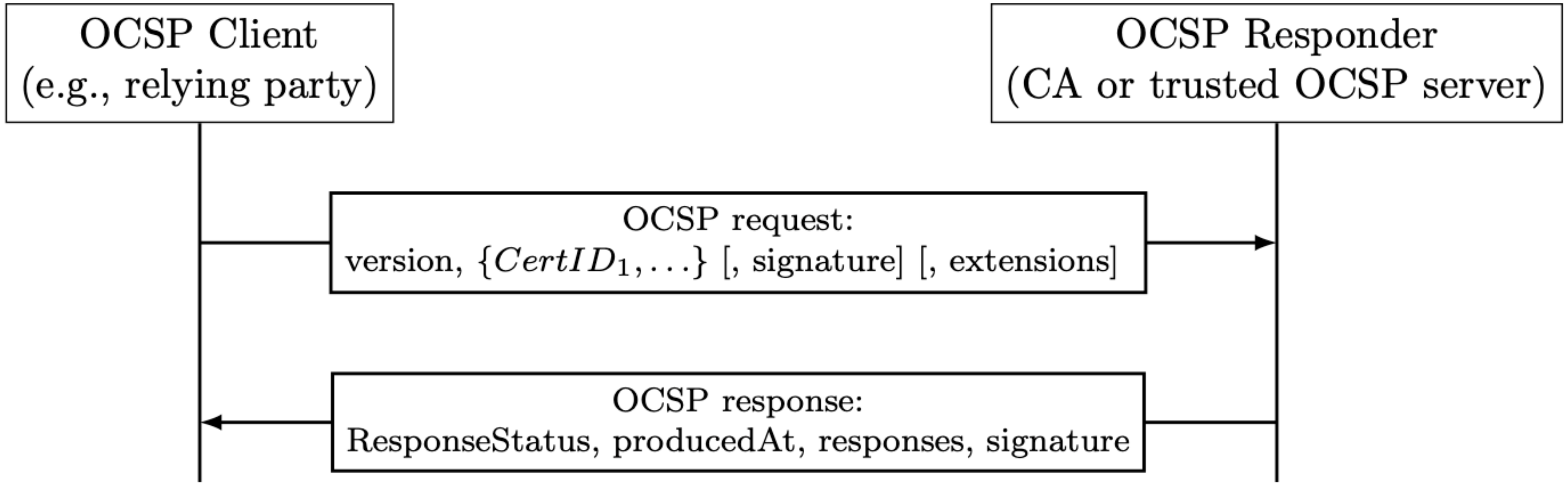
- **Revocation is hard**
  - CRLs contain all revoked certificates – huge!
  - CRLs are not immediate
    - Affiliation changed - Object names or attribute
    - Frequent CRLs – more overhead
- **Solutions**
  - Distributed CRLs - split certificates to several CRLs
  - Delta CRLs – only new revocation since last “base” CRL
  - Short validity for certificates – no need to revoke them

PKI: Revoking certificates

- **Online Certificate Status Protocol (OCSP)**
  - Most browsers don’t use CRLs
    - Efficiency
    - Frequent CRLs – more overhead
- **OCSP**
  - Check validity of certificates as needed

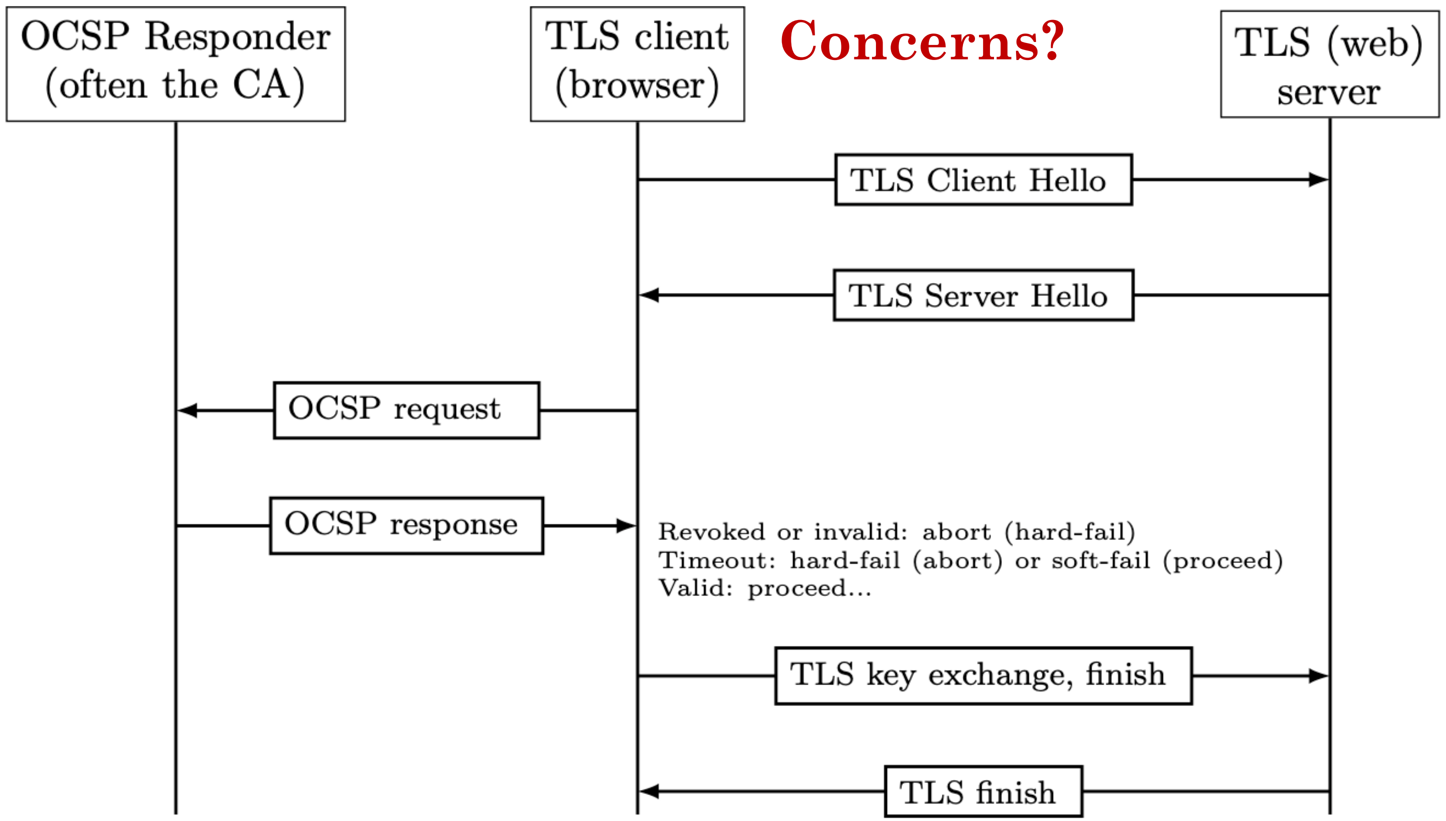
PKI: Revoking certificates

- **Online Certificate Status Protocol (OCSP)**
  - Most browsers don't use CRLs
    - Efficiency
    - Frequent CRLs – more overhead
- **OCSP**



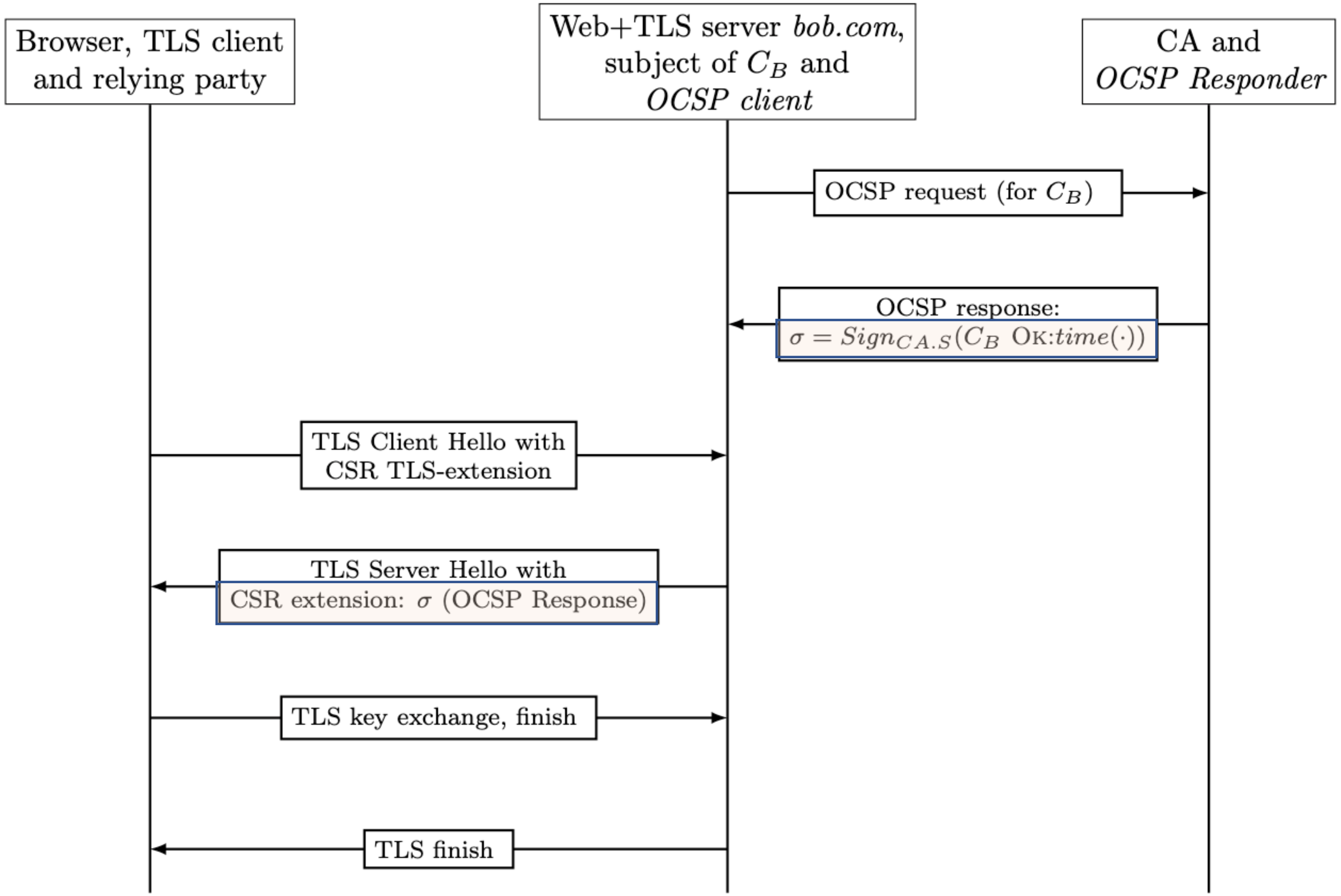
PKI: Revoking certificates

- **Online Certificate Status Protocol (OCSP)**



PKI: Revoking certificates

- **OCSP Stapling**



PKI: Certificate Transparency (CT)

- **Why and How CAs fail**
  - (Root) CAs trusted in browsers
  - Every CA can certify any domain (name)
  - Bad certificates
    - Equivocation: rogue certificates
    - Misleading certificates (e.g., squatting names)
- **How to improve defense against bad CAs/certificates**

PKI: Certificate Transparency (CT)

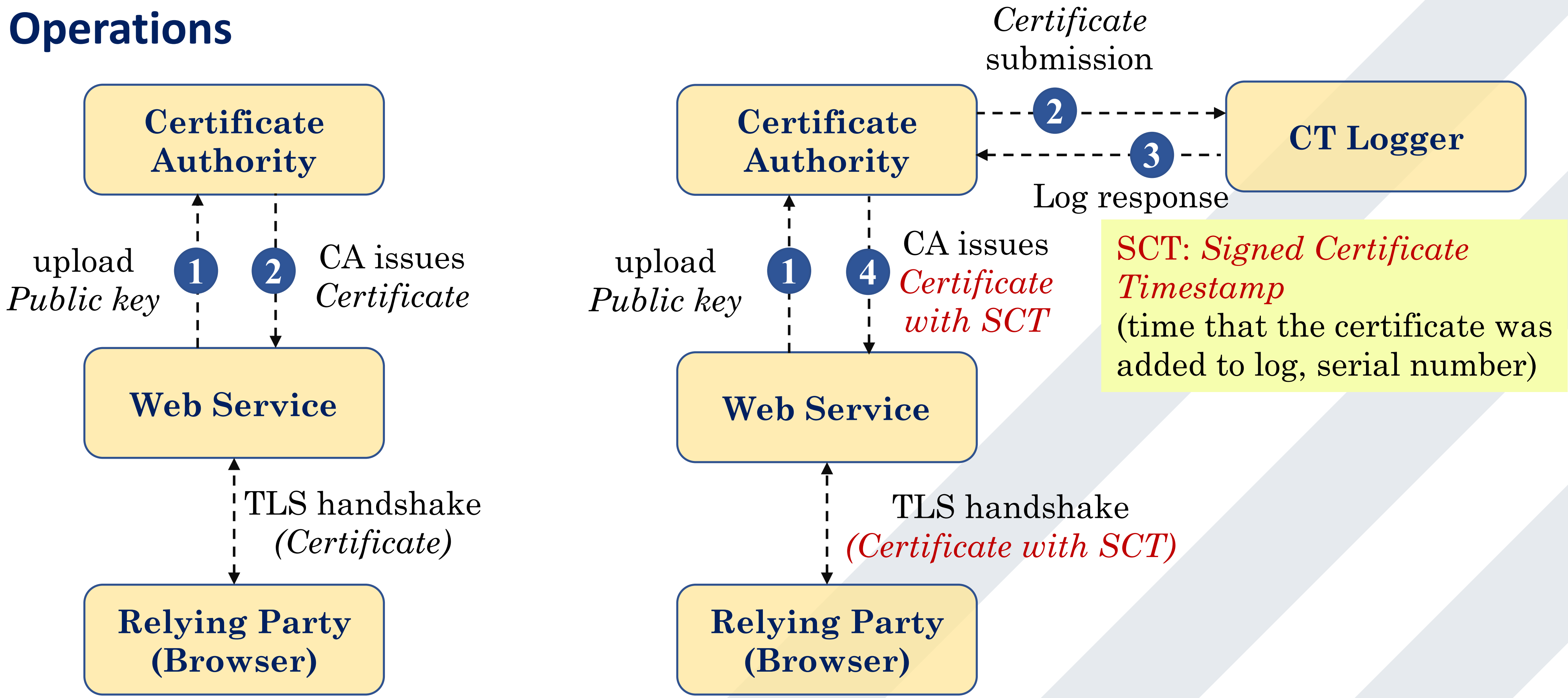
- **Certificate Transparency (CT)**
  - A proposal originating from Google, for improving the transparency and security of the (Web) PKI
  - Goals
    - Detecting equivocating certificates by monitoring specific domain name
    - Detecting suspect CAs/certificates
  - An extensive standardization
    - already enforced by Chrome browser
    - many websites and CAs deploy CT, making CT the most important development in PKI since X.509

PKI: Certificate Transparency (CT)

- **CT Entities**
  - Loggers: provide public logs of certificates
    - CAs send each certificate to loggers, who add the certificate to the log
    - Loggers provide accountability for the public availability of certificates
    - Google and few CAs operate loggers
  - Monitors: monitor the certificates logged by (many) loggers
    - Detect (suspicious) changes of certificates for domain owners
    - Operated by Facebook and few other CAs and companies
  - Auditors: ensure the logger provides exactly the same log to all parties
    - Typically implemented and performed by relying parties (browsers)

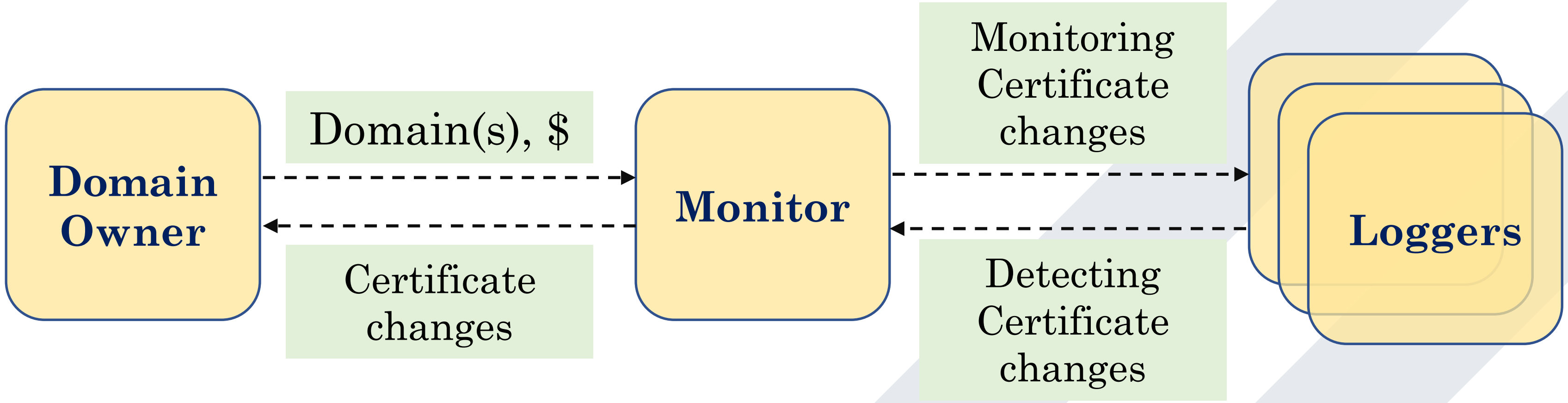
PKI: Certificate Transparency (CT)

- **CT Operations**



PKI: Certificate Transparency (CT)

- **Certificate Transparency (CT)**
  - Goals
    - Detecting equivocating certificates by monitoring specific domain name



Network Security - Cryptography

- TCP/IP
- DoS Attacks
- DNS
- BGP
- CDN

- Applied Cryptography
- PKI
- TLS/SSL and HTTPS

- DNSSEC (*USENIX Security'17*)
- RPKI (*NDSS'17*)
- HTTPS/CDN (*IEEE S&P'14*)

Major Reference

- Amir Herzberg, *Foundations of Cybersecurity, Volume I: An Applied Introduction to Cryptography*, 2021 (Draft).
- Jonathan Katz, Yehuda Lindell. *Introduction to Modern Cryptography,* 2nd Edition.

CS 772/872: Advanced
Computer and Network Security

Fall 2022

Course Link:

https://shhaos.github.io/courses/CS872/netsec-fall2022.html