CS120: Computer Networks

Lecture 28. Network Security 2

Zhice Yang

Example Systems

- TLS/SSL
- SSH
- Wi-Fi Security

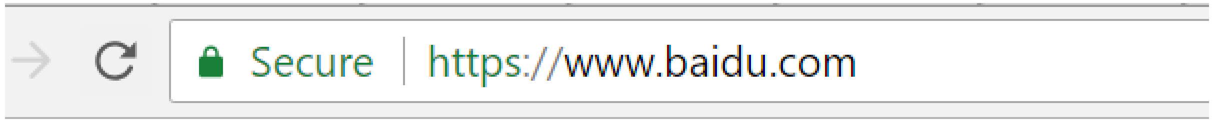
SSL: A Secure Transportation Layer Protocol

- SSL: Secure Sockets Layer
  - Deprecated [2015]
- TLS: Transport Layer Security
  - TLS 1.3: RFC 8846 [2018]
- Security for applications that use TCP
  - HTTPS (HTTP over SSL)
  - Some VPN
- Be able to handle threats:
  - Eavesdropping
    - Confidentiality
  - Manipulation
    - Integrity
  - Impersonation
    - Authentication

Application (e.g., HTTP)
Secure transport layer
TCP
IP
Subnet

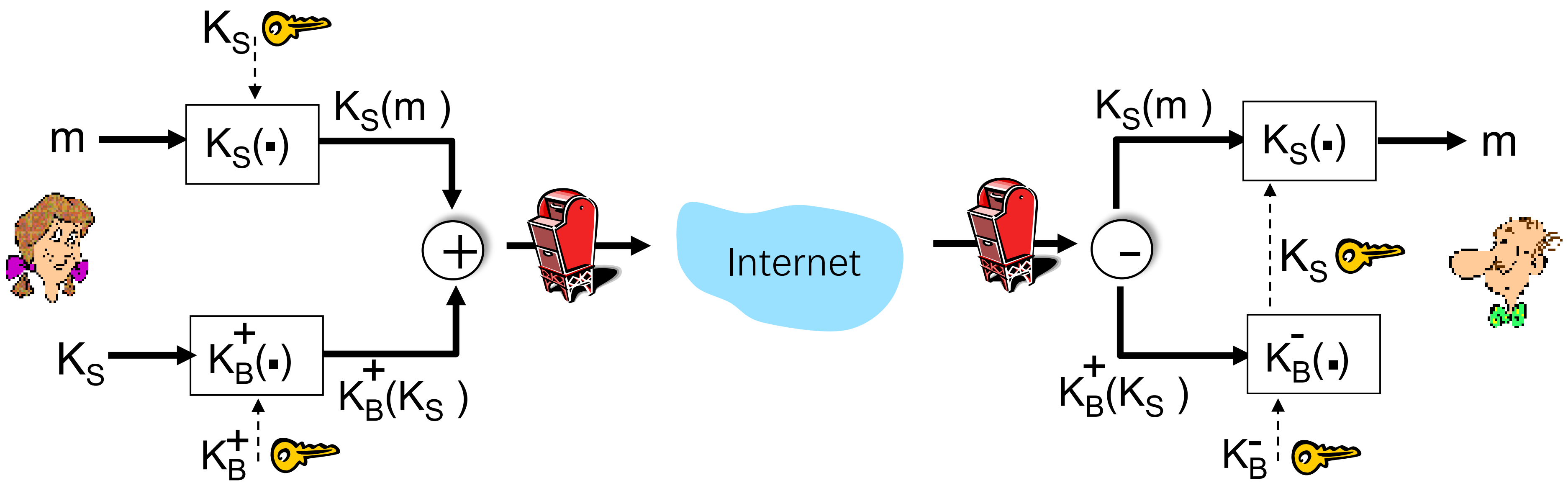
HTTPS

- Suppose a browser (client) wants to connect to a server who has a certificate from a trusted CA



Secure Message: Confidentiality

Alice wants to send *confidential* Message, m, to Bob.



Alice:

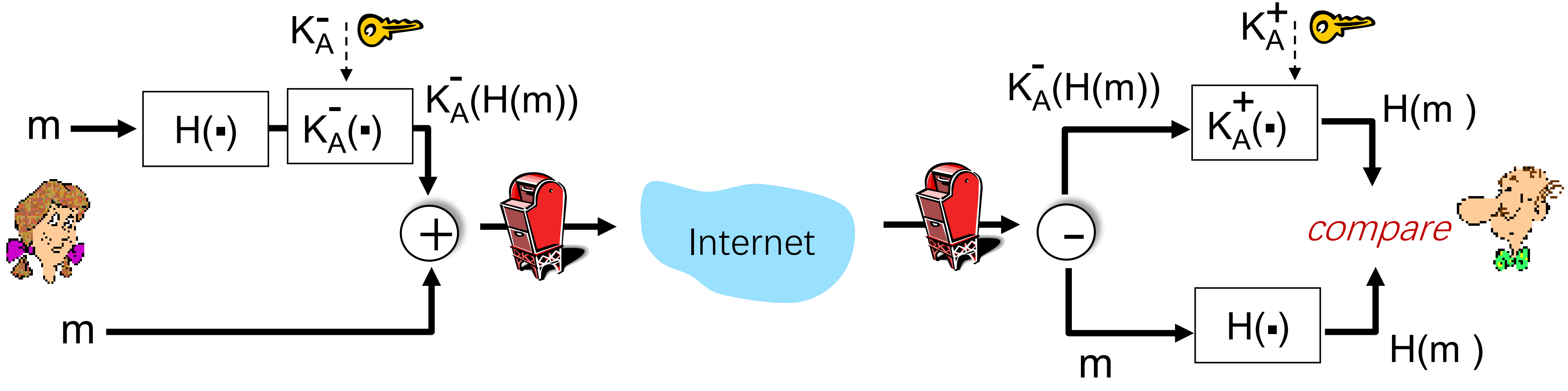
- generates random *symmetric* private key, K_S
- encrypts message with K_S (for efficiency)
- also encrypts K_S with Bob's public key
- sends both $K_S(m)$ and $K^+{}_B(K_S)$ to Bob

Bob:

- uses his private key to decrypt and recover K_S
- uses K_S to decrypt $K_S(m)$ to recover m

Secure Message: Integrity + Authentication

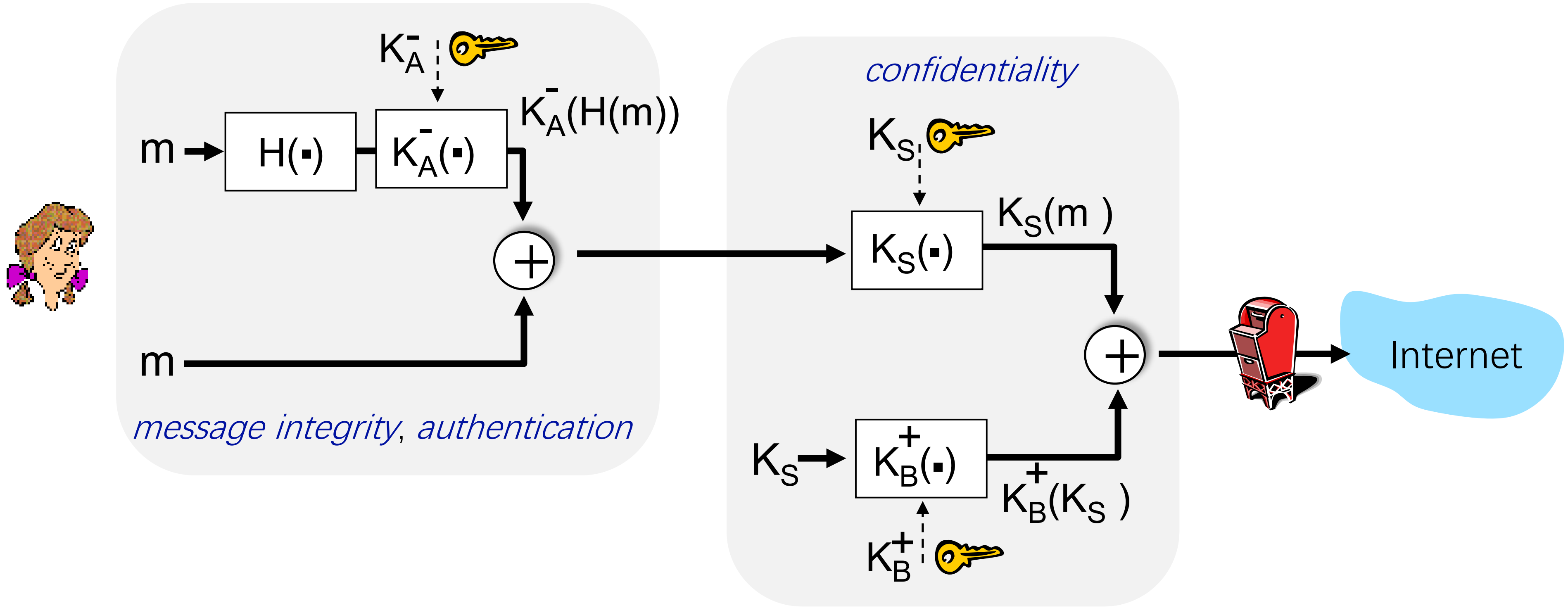
Alice wants to send m to Bob, with *message integrity*, *authentication*



- Alice digitally signs hash of her message with her private key, providing integrity and authentication
- Alice sends both message (unencrypted) and digital signature to Bob

Secure Message: ALL

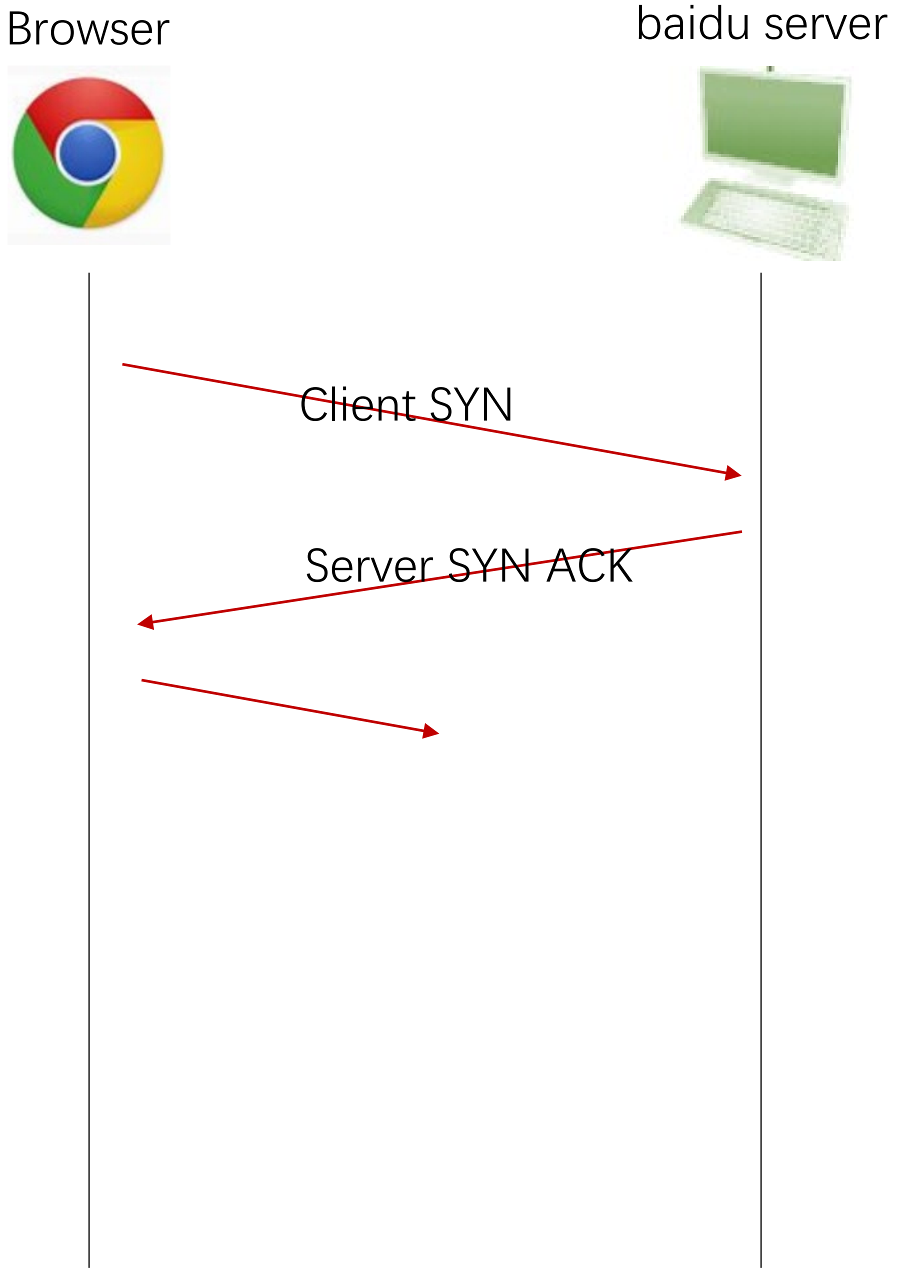
Alice sends m to Bob, with *confidentiality, message integrity, authentication*



Alice uses three keys: her private key, Bob's public key, new symmetric key

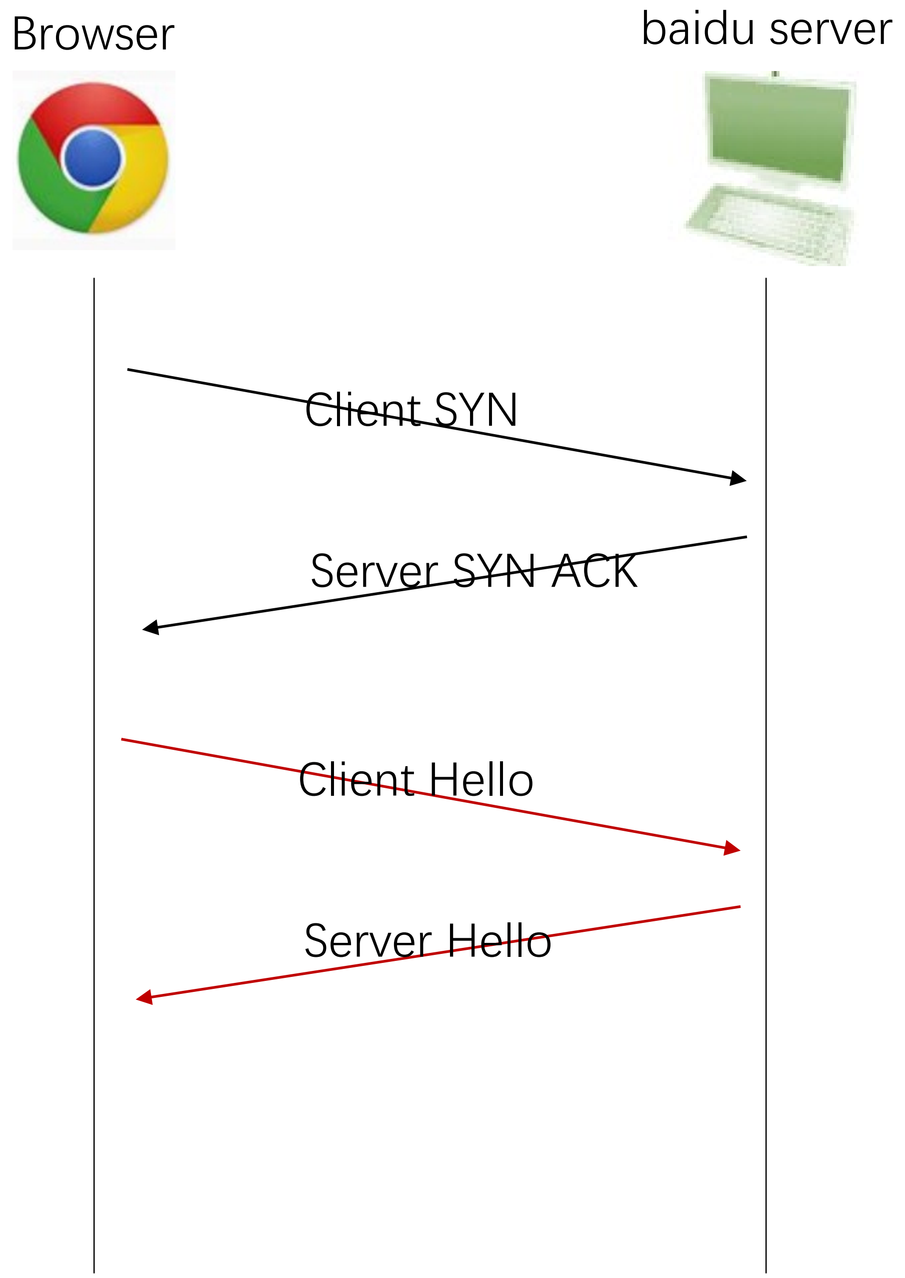
HTTPS via RSA

- Browser obtains the IP of the domain name www.baidu.com
- Browser connects to Baidu's HTTPS server (port 443) via TCP



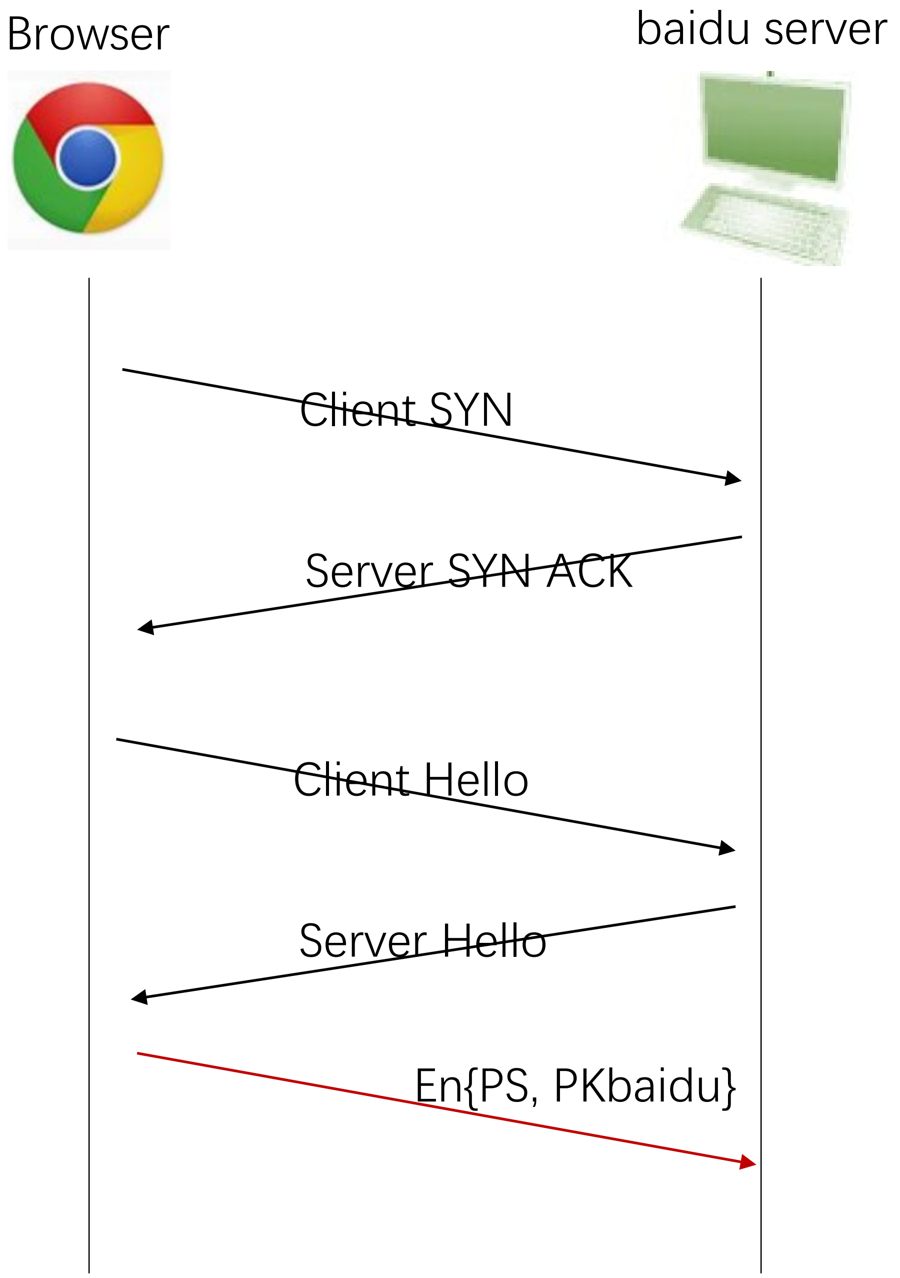
HTTPS via RSA

- Client Hello contains
  - 256-bit random number R_B
  - list of crypto algorithms it supports
- Server Hello contains
  - 256-bit random number R_S
  - Selects algorithms to use for this session
  - Server's certificate
- Browser validates server's cert
  - According to CAs



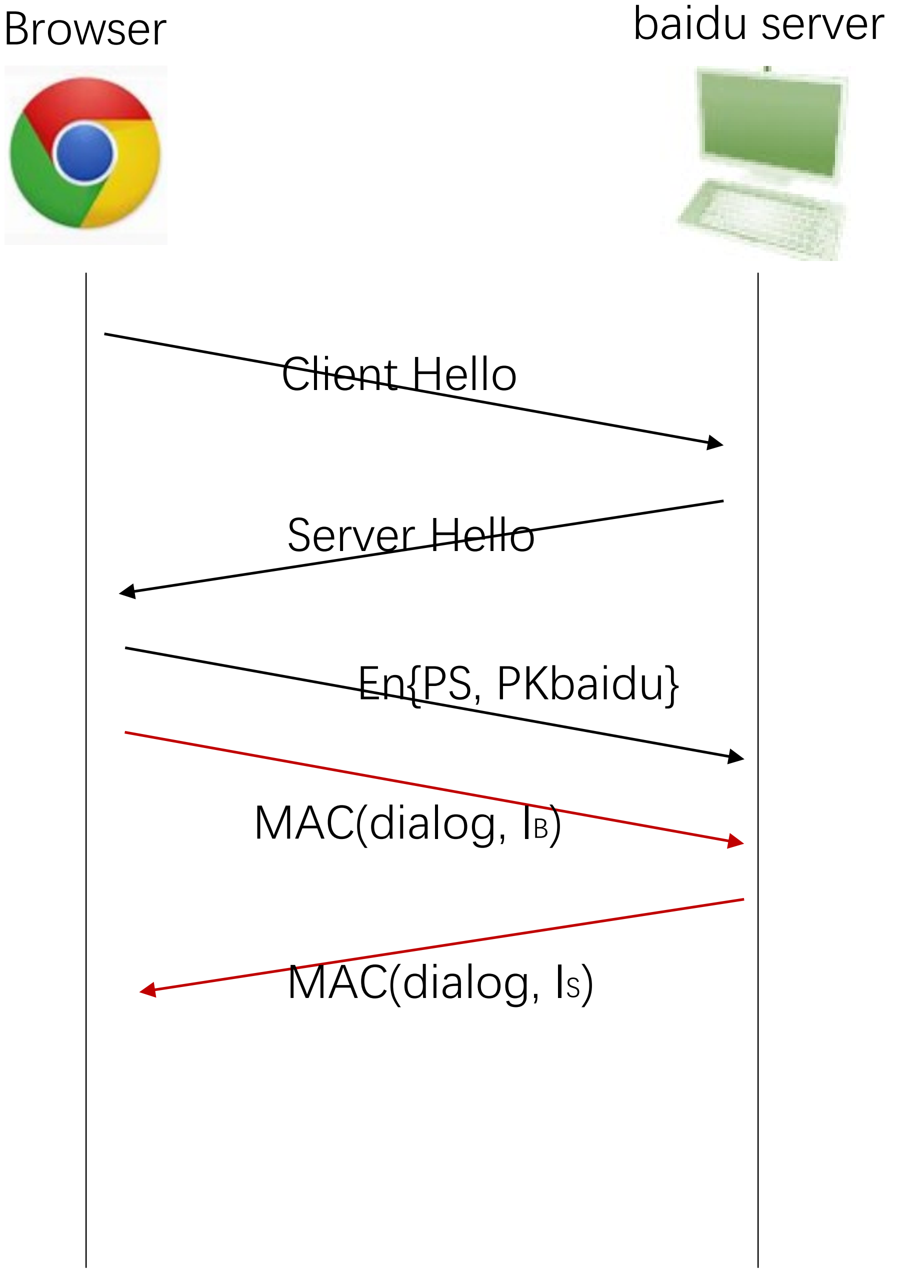
HTTPS via RSA

- Browser constructs “Premaster Secret” **PS**.
  - Uses R_B, R_S
- Browser sends **PS** encrypted using Baidu’s public RSA key: PKbaidu
- Using **PS**, R_B, and R_S, browser & server derive symmetric cipher keys (C_B, C_S) & MAC integrity keys (I_B, I_S)
  - One pair to use in each direction
  - Considered bad to use same key for more than one cryptographic function
    - i.e., I and C should be different



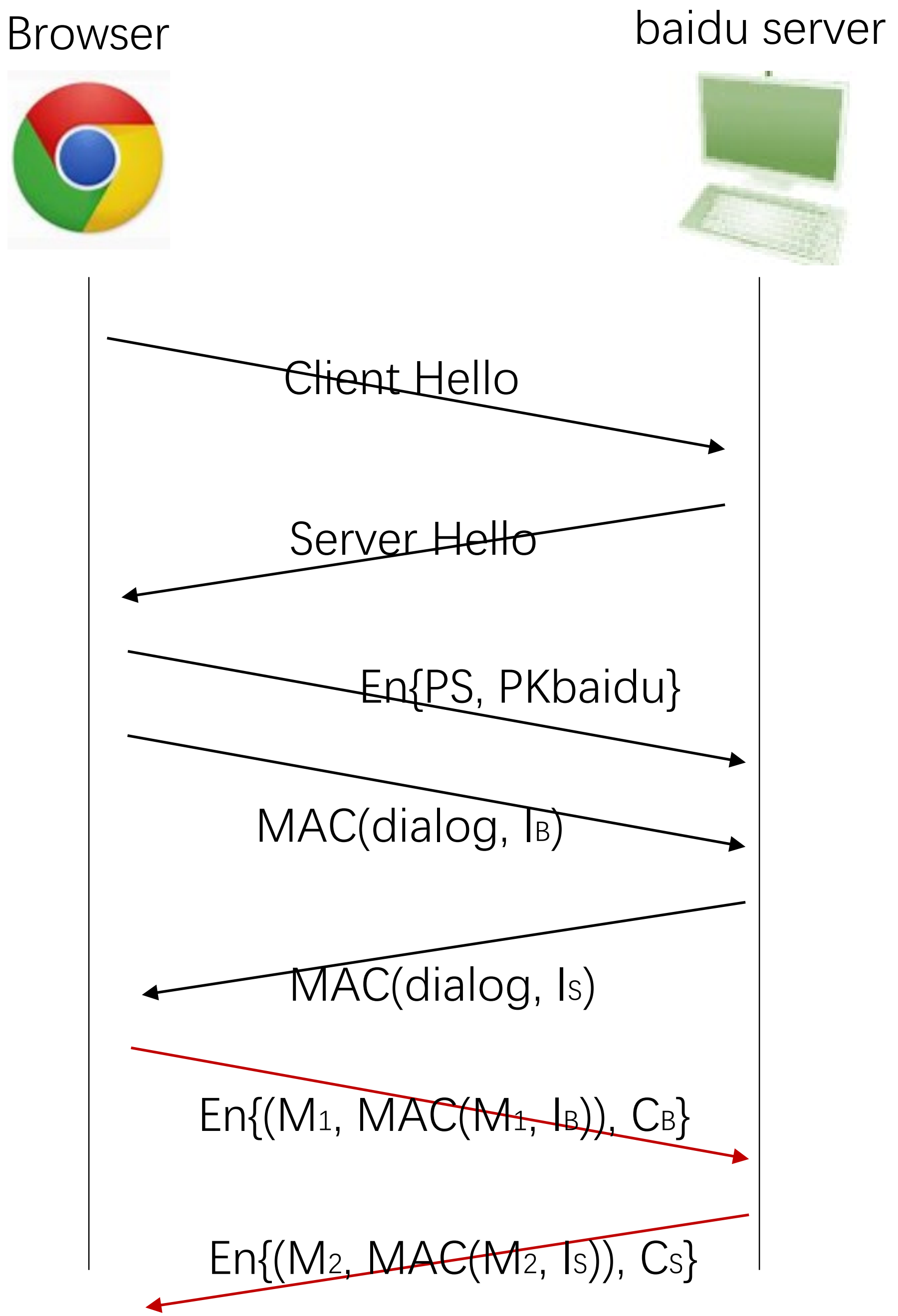
HTTPS via RSA

- Browser & server exchange MACs computed over entire dialog so far
  - Verify that (C_B, C_S) (I_B, I_S) are calculated correctly
- If the MAC is verified correctly, Browser displays 



HTTPS via RSA

- Browser & server exchange MACs computed over entire dialog so far
- If good MAC, Browser displays 🔒 Secure
- All subsequent communication encrypted with symmetric cipher (AES, 3DES, etc.)

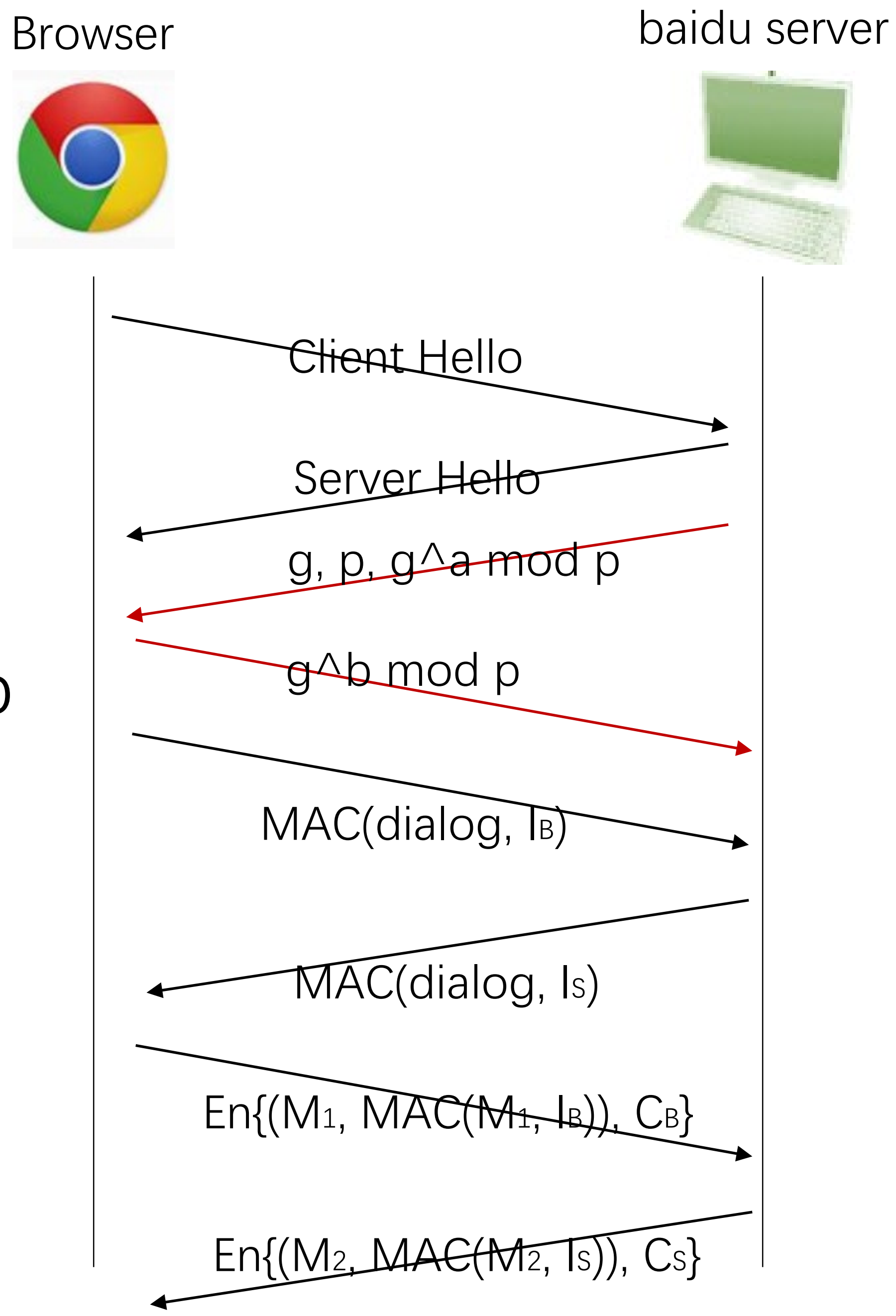


HTTPS via Diffie-Hellman Key Exchange

- Forward Secrecy
  - Assumptions:
    - The attacker can log all the traffic.
    - Assume PKbaidu is known to the attacker (some day in the future the private key of the server might be compromised)
    - Since in RSA, **PS** is encrypted by Pkbaidu. **R_B** and **R_S** are not encrypted
  - Attacker can calculate session keys (C_B, C_S) (I_B, I_S) and decode the logged conversations
- Solution
  - Diffie-Hellman Key exchange
    - Secure the conversations even with the above assumptions.

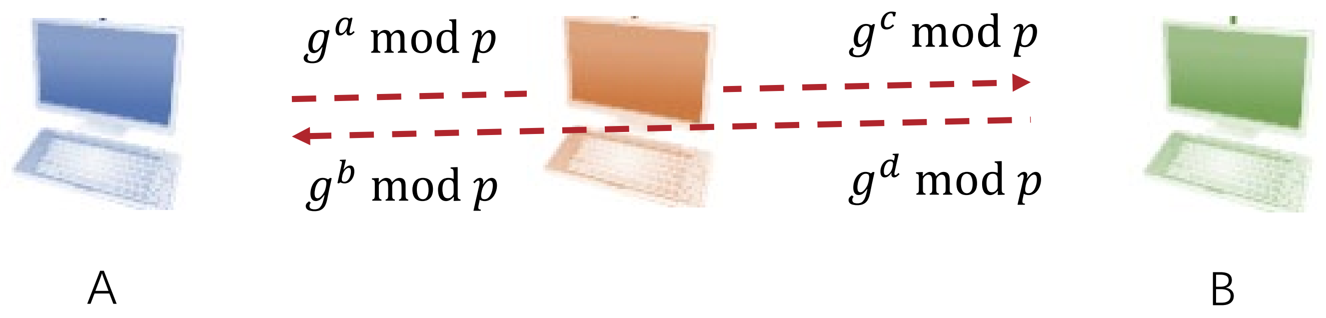
HTTPS via DH

- Server generates a random number **a**, sends public parameters (g, and p) and g^a mod p
- Browser generates a random number b, computes **PS** = g^ab mod p, sends g^b mod p to server
- Server computes **PS** = g^ab mod p

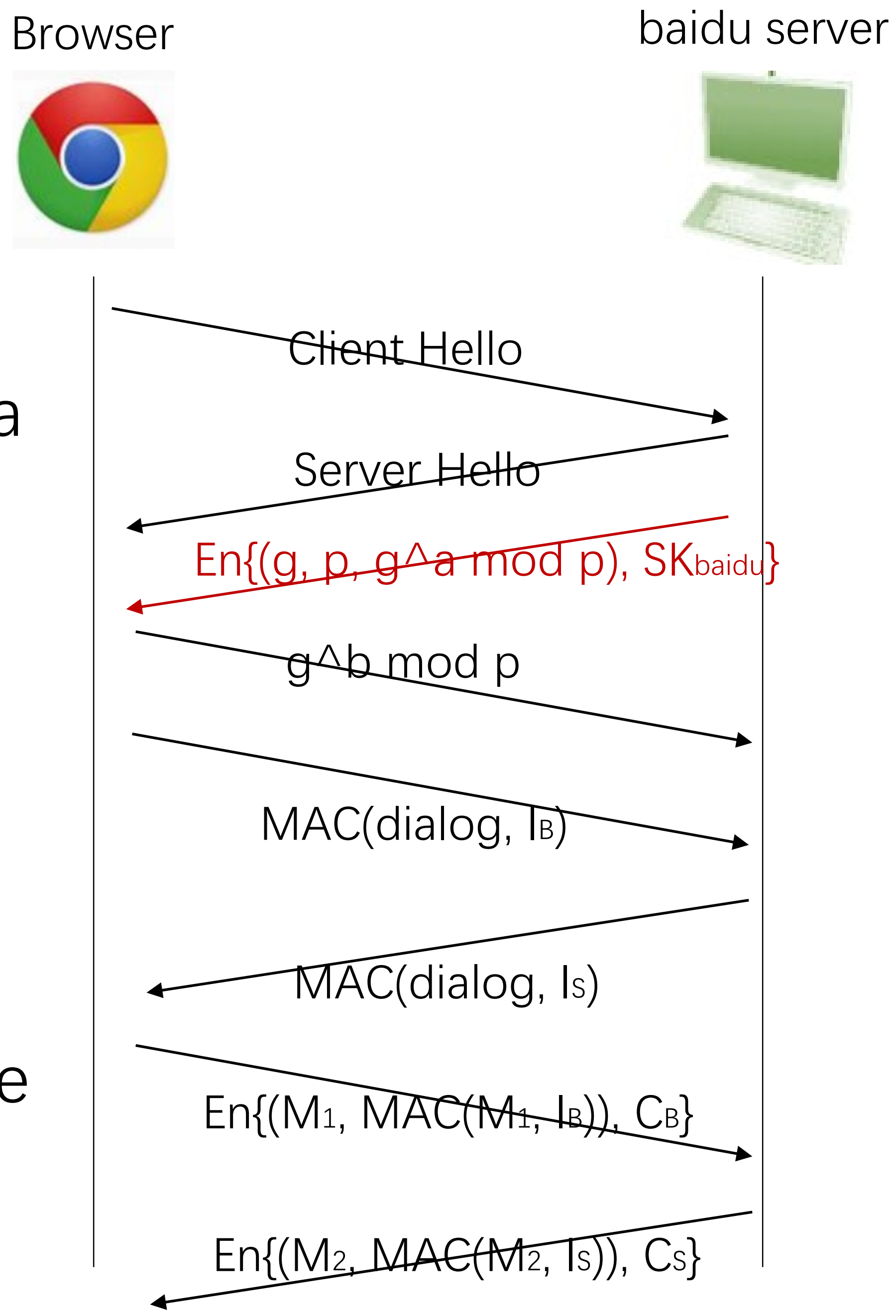


Diffie-Hellman Key Exchange

- Man in the middle attack
  - A cannot authenticate he is talking with B
- Diffie-Hellman Key Exchange is not secure without authentication



HTTPS via DH

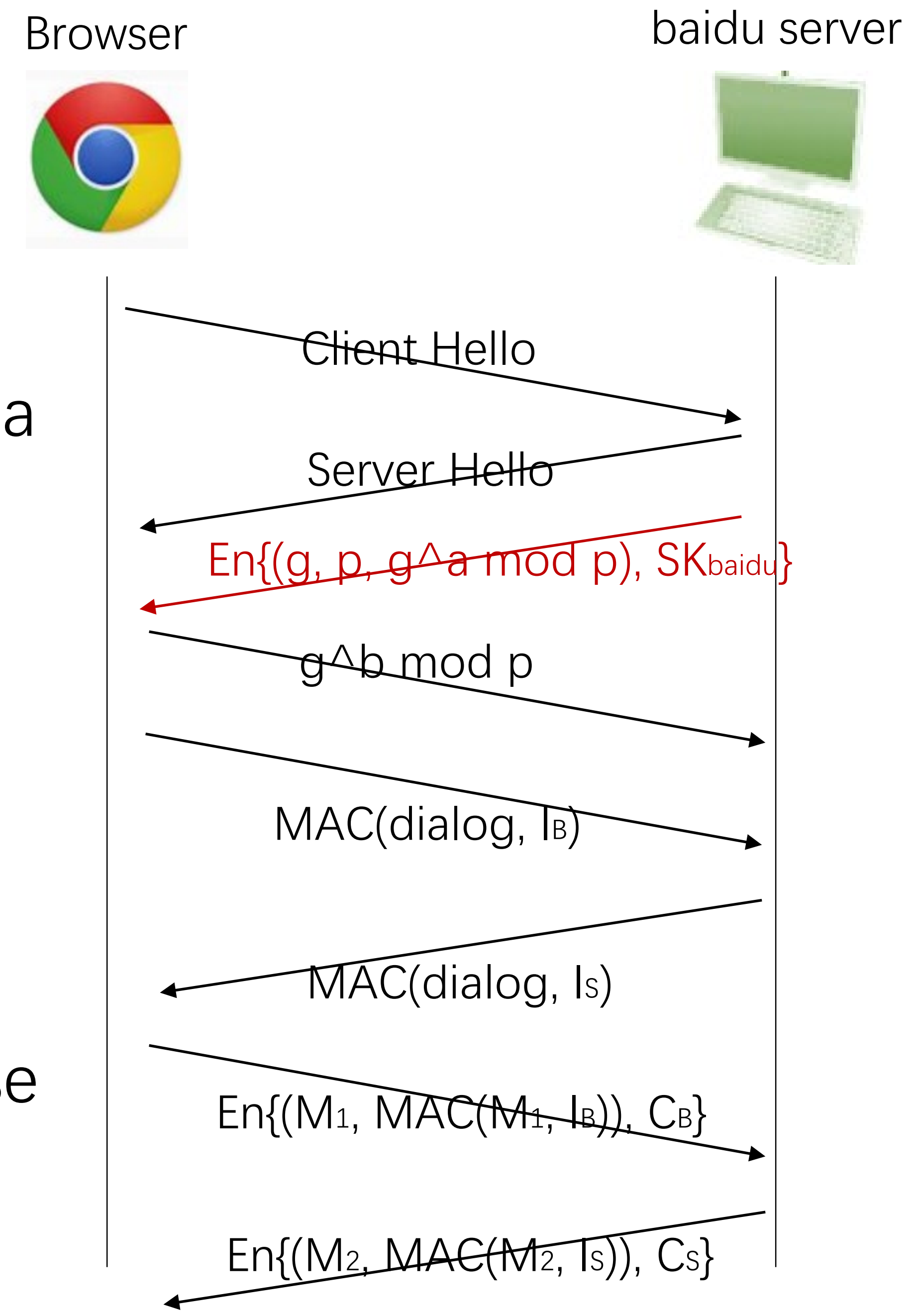


- Server generates a random number **a**, sends public parameters (g, and p) and g^a mod p
  - Sign the content with servers' private key **SKbaidu**
- Browser generates a random number b, computes **PS** = g^ab mod p, sends g^b mod p to server
- Server computes **PS** = g^ab mod p
- Attacker is not able to calculate **PS**, because **a** and b have not been transmitted!

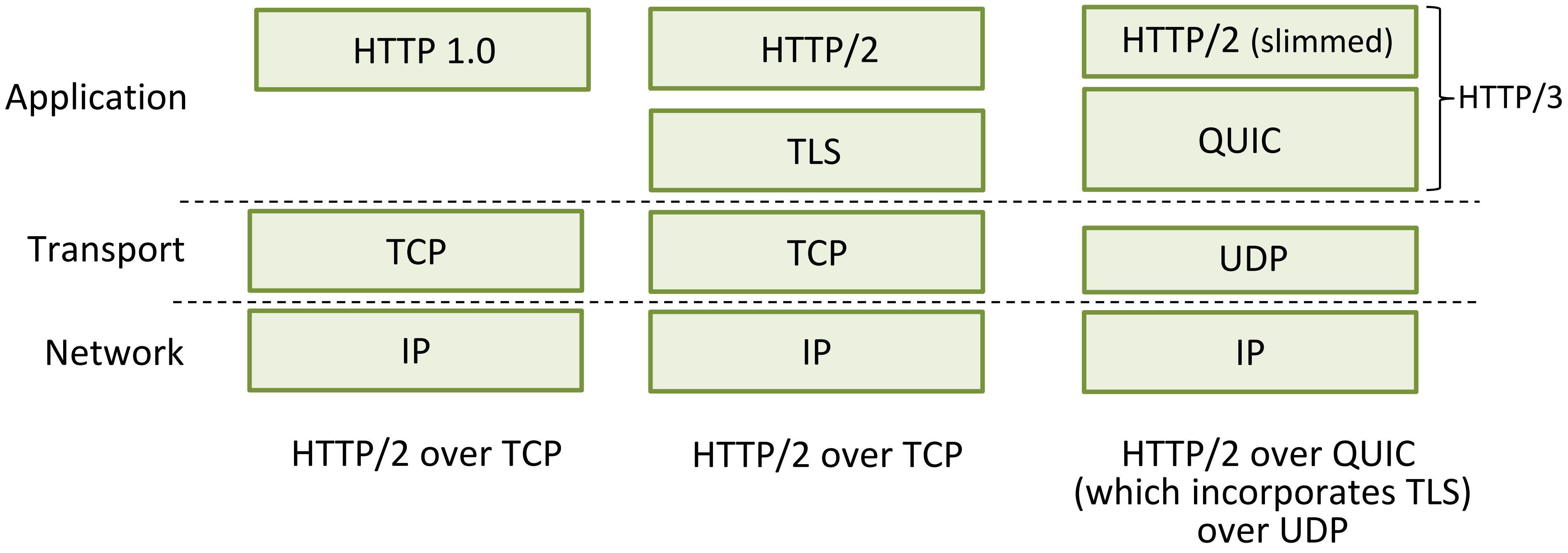
HTTPS via DH

- Server generates a random number **a**, sends public parameters (g, and p) and g^a mod p
  - Sign the content with servers' private key **SKbaidu**
- Browser generates a random number **b**, computes **PS** = g^ab mod p, sends g^b mod p to server
- Server computes **PS** = g^ab mod p
- Attacker is not able to calculate PS, because

RSA and Diffie-Hellman Key Exchange are combined to improve security



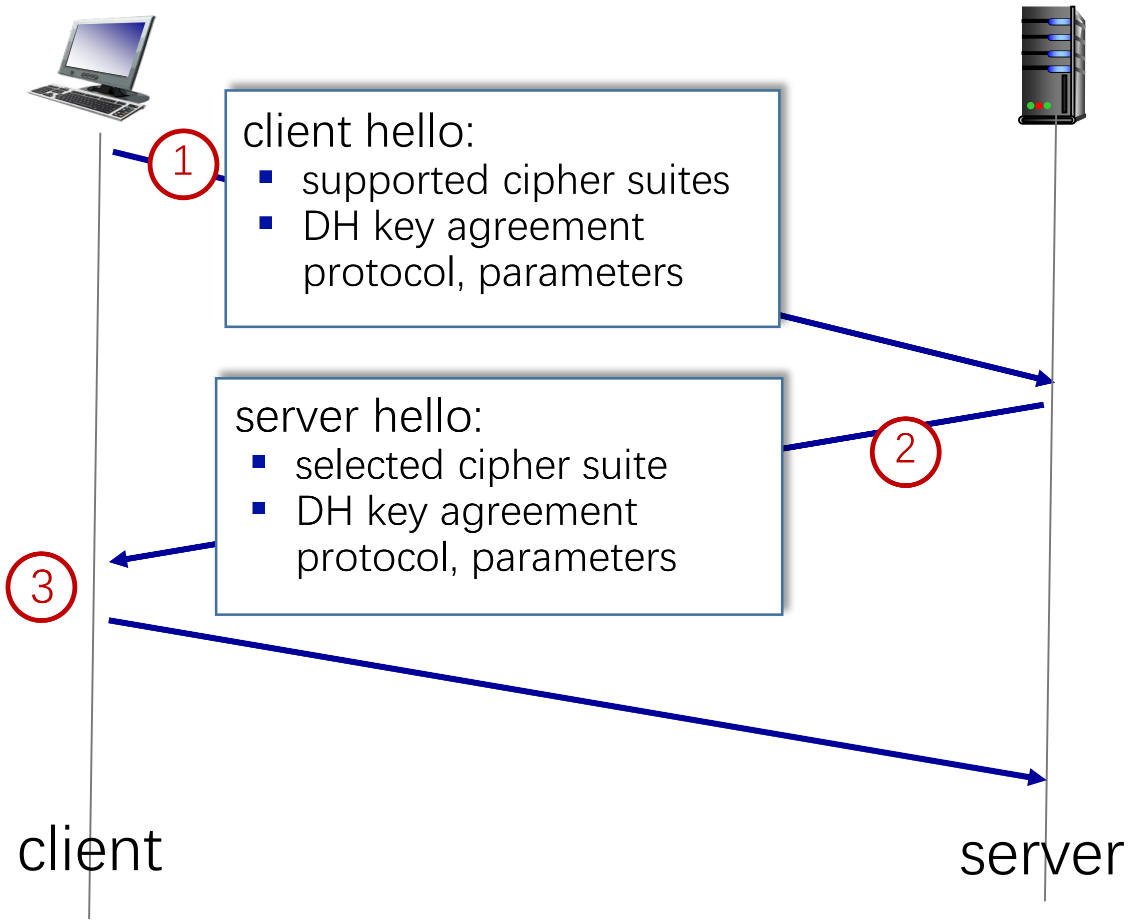
An HTTP view of TLS:



TLS: 1.3

- TLS: 1.3 (2018)
  - only 5 cipher choices, rather than 37 choices (TLS 1.2)
  - requires Diffie-Hellman (DH) for key exchange, rather than DH or RSA
  - combined encryption and authentication algorithm ("authenticated encryption") for data rather than serial encryption, authentication
  - HMAC uses SHA (256 or 284) cryptographic hash function

TLS 1.3 Handshake: 1 RTT

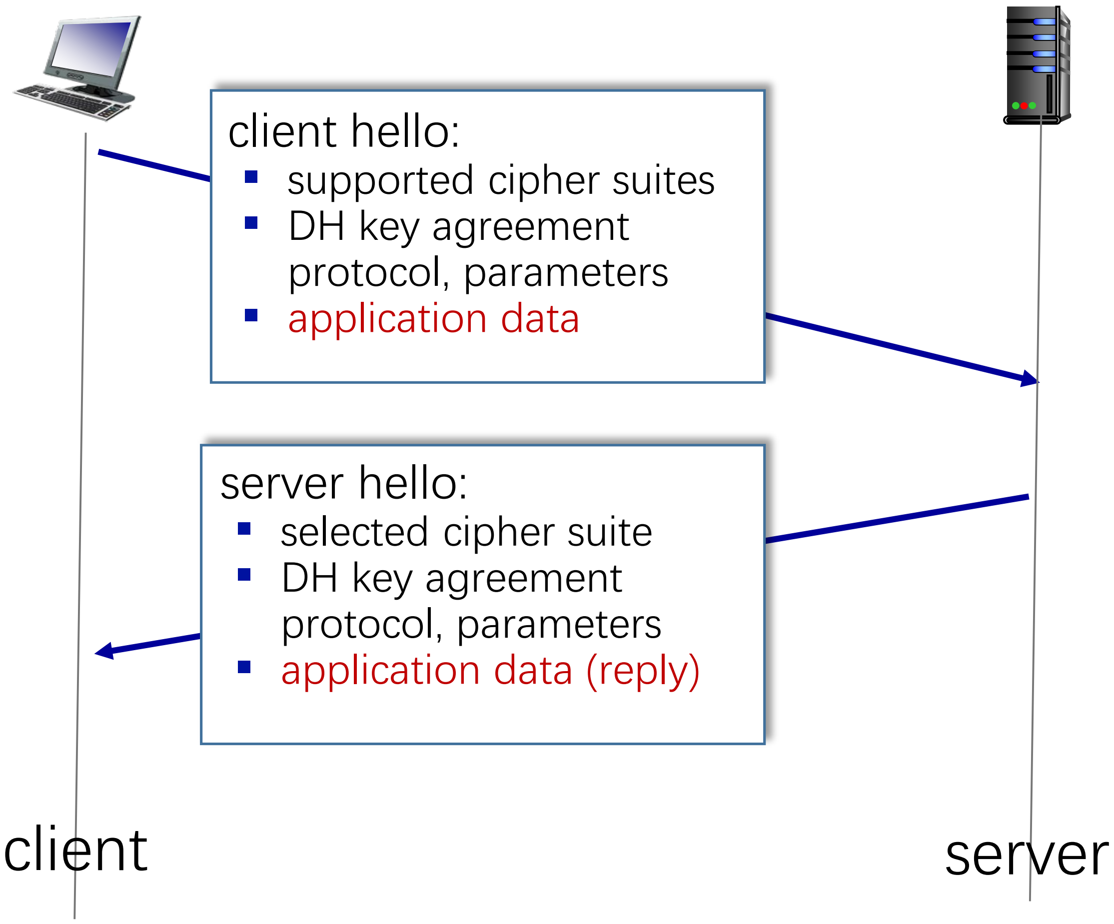


1 client TLS hello msg:
- *guesses* key agreement protocol, parameters
- indicates cipher suites it supports

2 server TLS hello msg chooses
- key agreement protocol, parameters
- cipher suite
- server-signed certificate

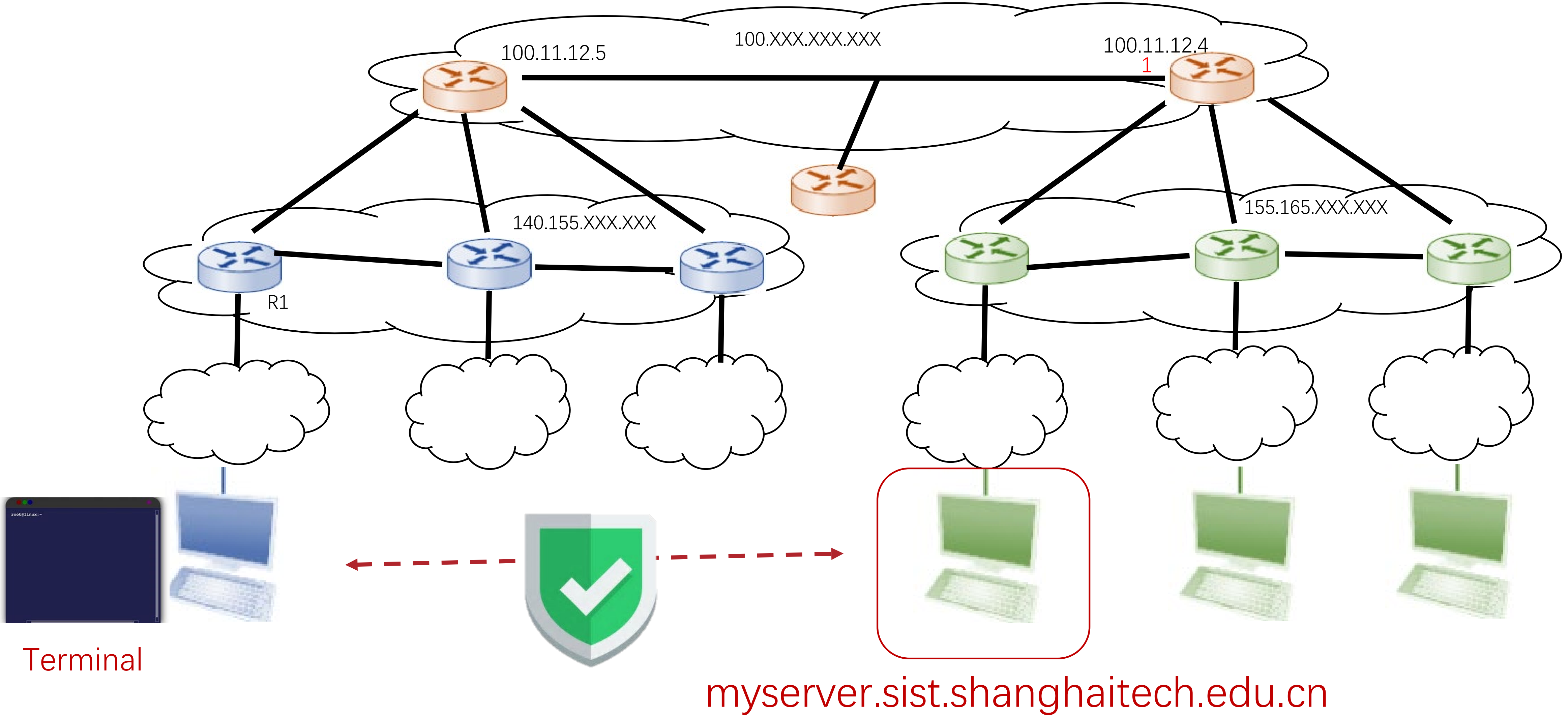
3 client:
- checks server certificate
- generates key
- can now make application request (e.g.., HTTPS GET)

TLS 1.3 Handshake: 0 RTT



- initial hello message contains encrypted application data!
  - "resuming" earlier connection between client and server
  - application data encrypted using "resumption master secret" from earlier connection
- vulnerable to replay attacks!
  - maybe OK for get HTTP GET or client requests not modifying server state

The Secure Shell (SSH)

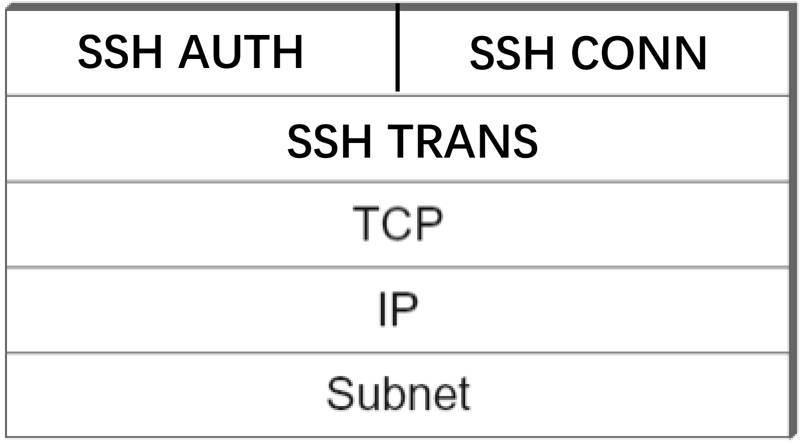


The Secure Shell (SSH)

- Developed by Tatu Ylönen, Helsinki University of Technology, Finland in 1995
- A Secure Version of Telnet
  - Message confidentiality
  - Message integrity
  - Client/server authentication

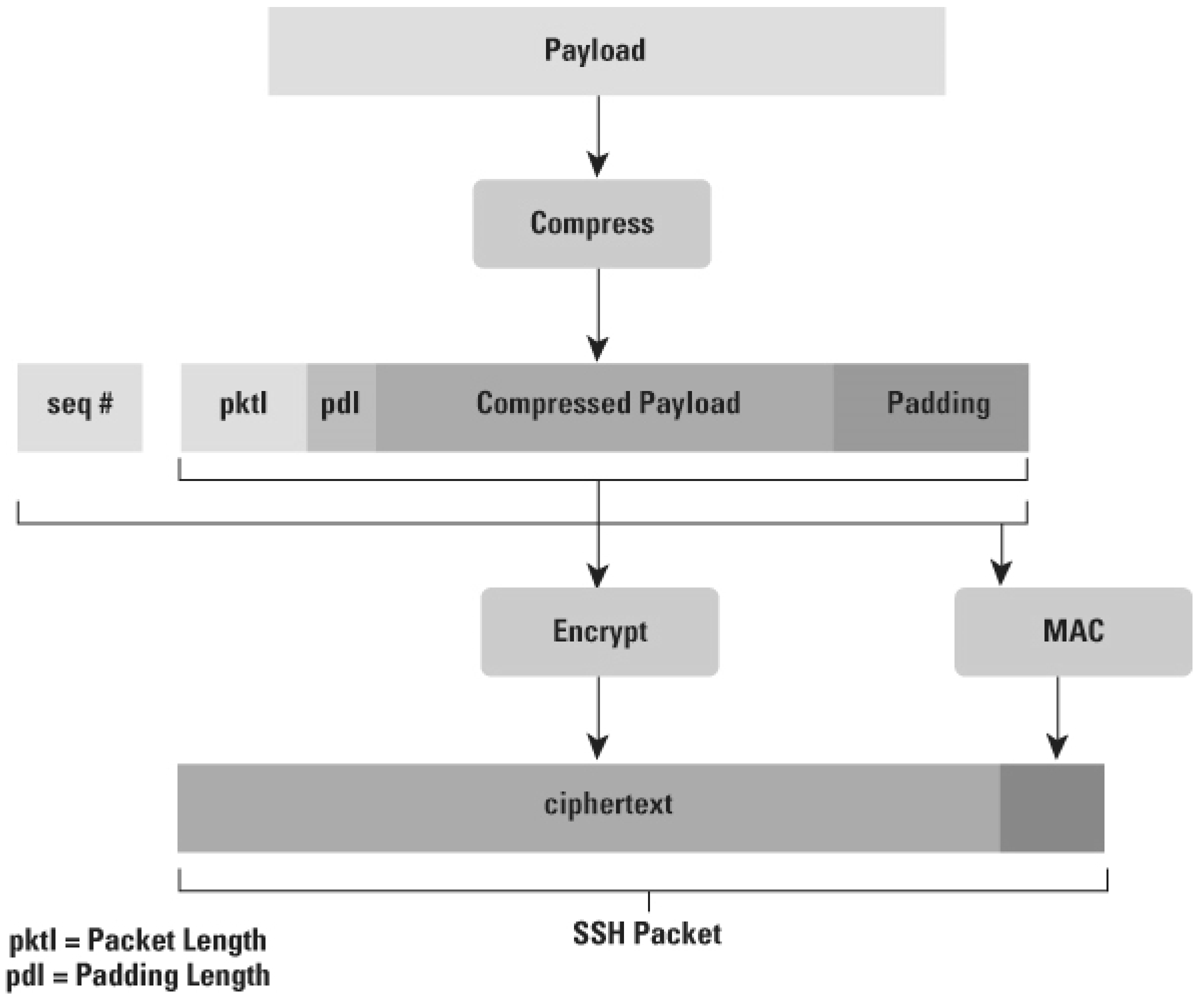
SSH v2 Protocols

- SSH Transportation Layer Protocol
  - Establish secure channel between client and server
  - Client authorizes server
- SSH User Authentication Protocol
  - Server authorizes client
- SSH Connection Protocol
  - Tunnel over secure channel



SSH-TRANS

- Protocol Steps
  - Establish TCP Connection
  - Exchange SSH Parameters
    - Distribution of server's public key
      - Manually through offline channel
      - Trust the first time
  - Key Exchange
  - Messages



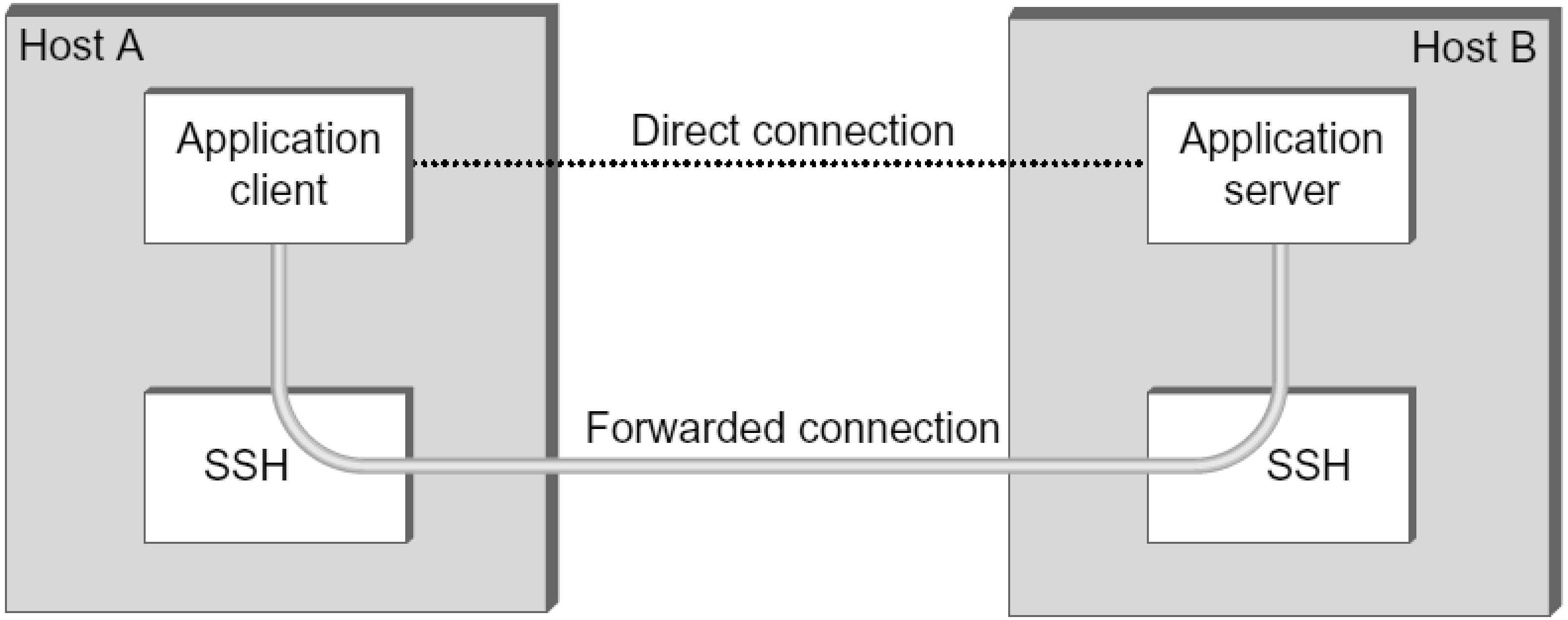
https://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-46/124-ssh.html

SSH AUTH

- Server Authorizes Client
  - User Name + Password
  - RSA
  - Host-based Authentication

SSH CONN

- Examples
  - SFTP
  - SSH Tunnel



SSL v.s. SSH

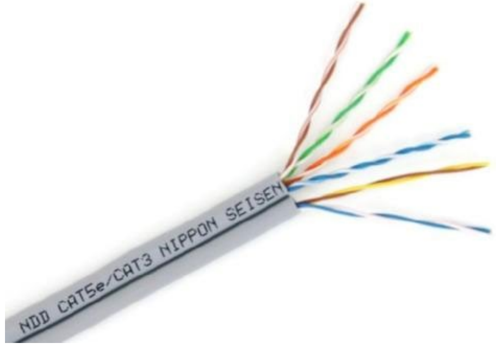
- Applications: Quite Different
  - SSL: browsers
  - SSH: remote consoles
- Techniques: Very Similar
  - Data integrity
    - HMAC (MD5, SHA-1)
  - Confidentiality
    - Symmetric-key ciphers: 3DES, AES, etc.
  - Session Key Establishment
    - RSA, DH, RSA+DH, etc.

Example Systems

- TLS/SSL
- SSH
- ➢ Wi-Fi Security

Wi-Fi Security

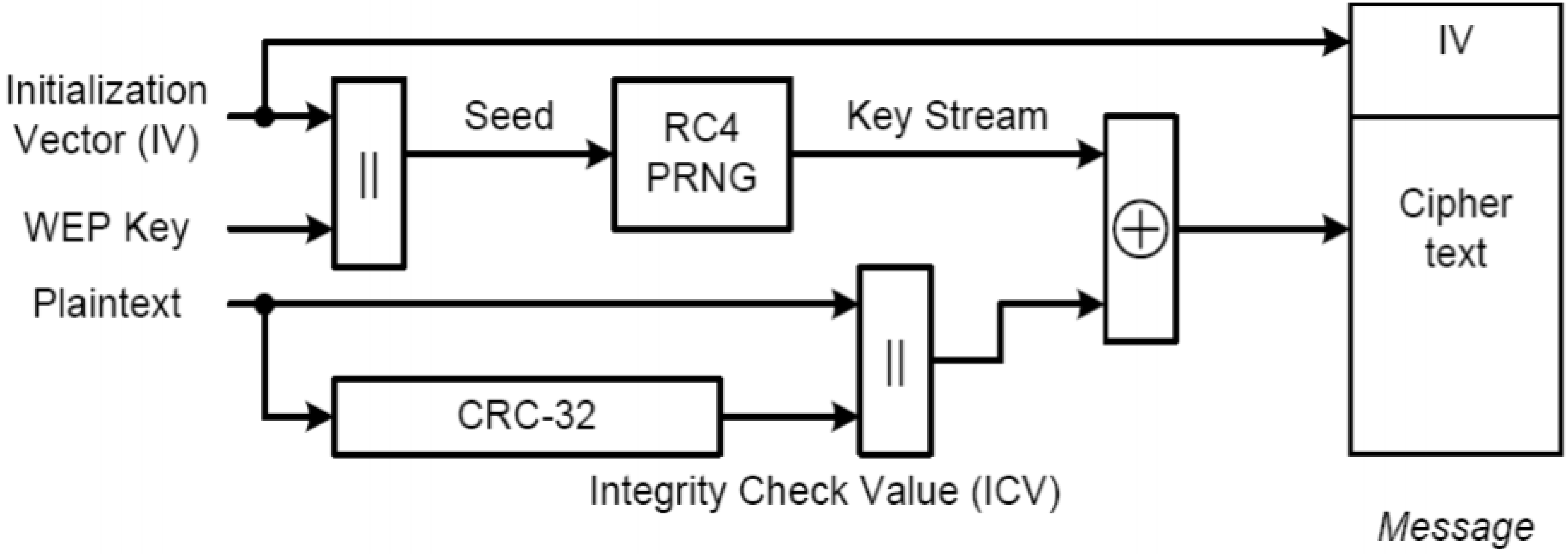
- Why ?
  - The broadcast nature of the wireless medium



Wi-Fi Security

- Authentication Method
  - Wired Equivalent Privacy (WEP)
    - Not secure
  - Wi-Fi Protected Access (WAP)

Wired Equivalent Privacy (WEP)

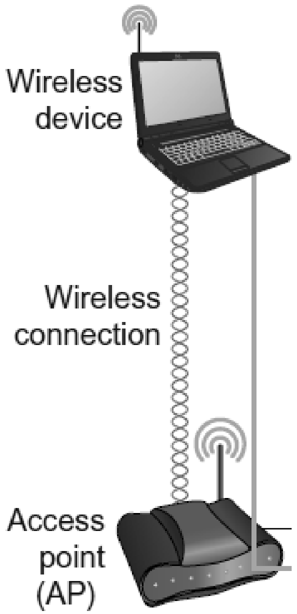


WEP Weakness

- Fluhrer-Mantin-Shamir (FMS) Attack
  - 24 bit IV, reuse very soon
  - Leverage the first two bytes of the plaintext
    - 0xAA
  - Collecting multiple messages to exploit the leakage

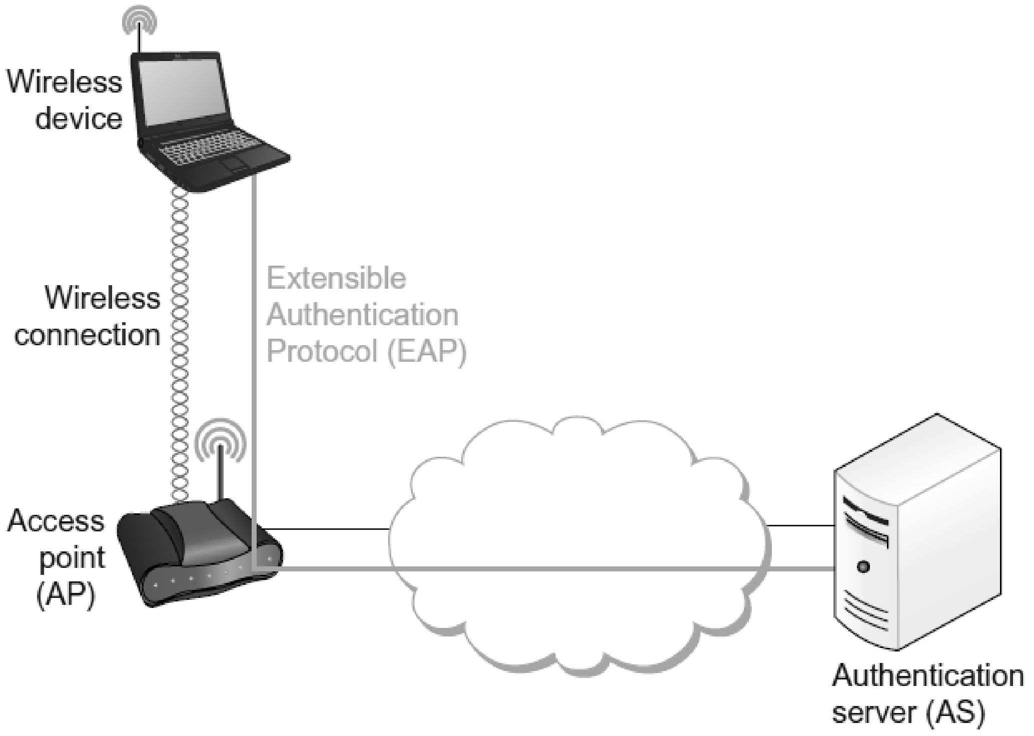
Authentication Directly

- Personal Mode



Authentication through EAP

- Enterprise Mode



Reference

- Textbook 8.4
- Some slides are adapted from http://www-net.cs.umass.edu/kurose_ross/ppt.htm by Kurose Ross
- http://inst.eecs.berkeley.edu/~cs161/sp18/