Try #1:
 Alice  \[ g^a \rightarrow Bob \]
 k = g^a
 \[ \text{sign}_A(k) \rightarrow \text{sign}_B(k) \]

Problem: How do Bob and Alice share public keys?

Public key infrastructure:
- Web of trust (e.g., PGP)
- Trust on first use (e.g., SSH)
- Public key infrastructure (e.g., SSL)

Problem #2: What if \text{sign}(\cdot) leaks into on k?

Try #2:
 Alice  \[ g^a \rightarrow Bob \]
 \[ \text{sign}_A(g^a) \rightarrow g^b \]
 \[ \text{sign}_B(g^a \cdot g^b) \]
 k = g^{ab}

Try #3: \text{randomness}
 Alice  \[ r^A \rightarrow Bob \]
 \[ g^b \cdot \text{sign}(r^A \cdot g^b) \]
 \[ k = g^{ab} \]

Problem: k: 1024-bit algebraic value?
Solution: k = H(g^{ab})
SSL/TLS

Client

hello, random (24 bytes and 4 bytes)

list of cipher-suits
TLS-ECDHE-WITH-AES-256-CBC-SHA

hello, random

pick a cipher-suite

certificate
- issuer (Verisign, GeoTrust, ...
- validity (dates)
- common name (e.g., www.google.com)
- public key
- CA signature
  possibly chain of certs to root

client certificate
- Is signature valid?
- Is signing key trusted/trusted?
- Does validity include today?
- Does CN match URL?

RSA key exchange
client generates 48 byte "pre-master-secret" PS
PKCS#1 v.1.5 padded

PS + server-random + client-random


PRF = based on HMAC, SHA1, MD5

master-secret

encryption
(mac IV values)

c = C

MAC of (data)

MAC of (data)
Diffie-Hellman TLS key exchange

Client

\[ \text{cert} \]

\[ g, p, g^a \mod p, \text{sign}_{p, g^a \mod p}(\text{something}) \]

\[ g^b \mod p \]

\[ pS = g^{ab} \mod p \]

Server

\[ \text{cert} \]

\[ g, p, g^a \mod p, \text{sign}_{p, g^a \mod p}(\text{something}) \]

\[ g^b \mod p \]

\[ pS = g^{ab} \mod p \]

SSL record protocol

- Type
- Version
- Length
- Data
- MAC over data
- Pad
- Pad len

POODLE (Padding Oracle on Downgraded Legacy Encryption)

POST /path Cookies: auth=cookie | MAC | Pad | Pad len

CBC padding oracle:

Attacker replaces \( C_n = C_i \):

Victim accepts cipher text if \( \text{Dec}(C_i) \oplus C_{n-1} + \text{last byte} = \text{pad len} \)

Victim rejects if \( \text{next last byte} \).

Attack! Repeat \( 2^{256} \) times, \( 256 \) requests until one accepts, then learn one byte of \( C_i \).

\( \text{Dec}(C_i)[15 \text{ XOR } C_{15}] = 15 \)

CRIME-style shifting:

Attacker changes request length (POST /AAAAA) until next byte is boundary full block pad.

TLS: Pads 1 length for all bytes.
SSH

Client

hello

hello

list algorithms

list of algorithms

preferred group

group params $(g,p)$

$e = g^x$

server hostkey

$f = g^y$

server hostkey($H$(transcript))

2. Verify signature

switch/decrypt/decrypt

if "password"

password

if "public key"

client pub

sign($H$(transcript))