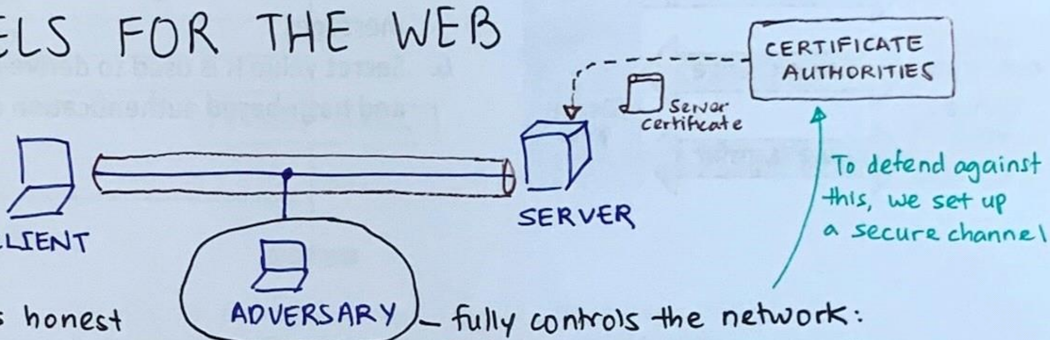
SECURE COMMUNICATIONS:

TLS

→ are cryptographic protocols designed to provide communications security over a computer network.

SECURE CHANNELS FOR THE WEB

→ Our setting:



→ As long as the client is honest & the adversary doesn't know server's private key, it cannot:

- ↳ Inject forged data into the data stream — Integrity
- ↳ Distinguish the data stream from random bytes. — Confidentiality

ADVERSARY — fully controls the network:

e.g.
- it can redirect traffic to own server (DNS Rebinding)
- it can passively read all data (IP Monitoring)
- it can play an active MITM (TCP Hijacking)

GOALS OF SSL/TLS

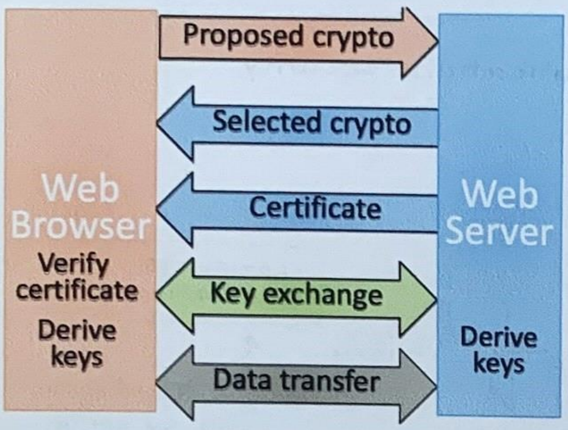
1) End-to-end confidentiality
   - encrypt communication between client & server apps
2) End-to-end integrity
   - detect corruption of communication between C&S apps
3) Required server authentication
   - identity of server always proved to client
4) Optional client authentication
5) Modular deployment

TLS BUILDING BLOCKS

	Confidentiality	Integrity	Authentication
SETUP	Public-key based key-exchange (RSA & DH)	Public-key Digital Signature (RSA)	Public-key Digital Signature
DATA TRANSMISSION	Symmetric Encryption (AES in CBC mode)	Hash-based MACs (HMAC using SHA-256)	

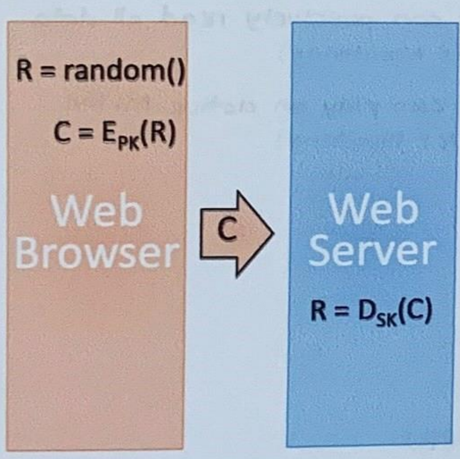
After we make sure that we're talking to the server we're meant to talk to

TLS OVERVIEW



1. Browser sends supported crypto algos
2. Server picks strongest algo it supports
3. Server sends certificate
4. Client verifies certificate → We get public key of server
5. Client and server agree on secret value R by exchanging messages
6. Secret value R is used to derive keys for symmetric encryption and hash-based authentication of subsequent data transfer

BASIC KEY EXCHANGE (RSA)

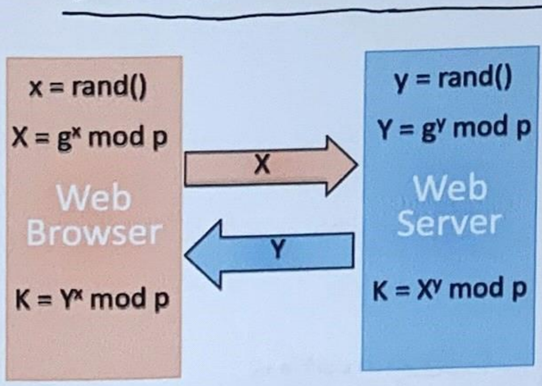


1. Client generates random secret value R
2. Client encrypts R with public key PK of server
3. Client sends ciphertext C to server
4. Server decrypts C with private key SK of server

TLS with basic key exchange does NOT provide forward secrecy.

- If server's SK is compromised, attacker finds secret value R in key exchange and derives encryption keys.
- The problem comes from the fact that we're using the same secret and public key for all our communication.

DIFFIE-HELLMAN KEY EXCHANGE (DHKE)



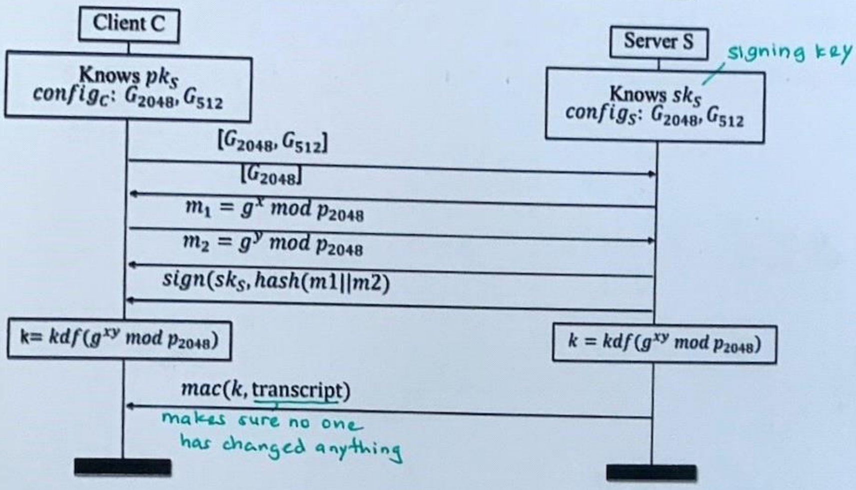
1. Public params: large prime p and generator g of Z_p
2. Client generates random x and computes X
3. Server generates random y and computes Y
4. Client sends X to server
5. Server sends Y to client
6. Client and server compute $K = g^{xy} \bmod p$

TLS with DH key exchange achieves forward secrecy, but is vulnerable to MiTM attack (See Note 7). Solution:

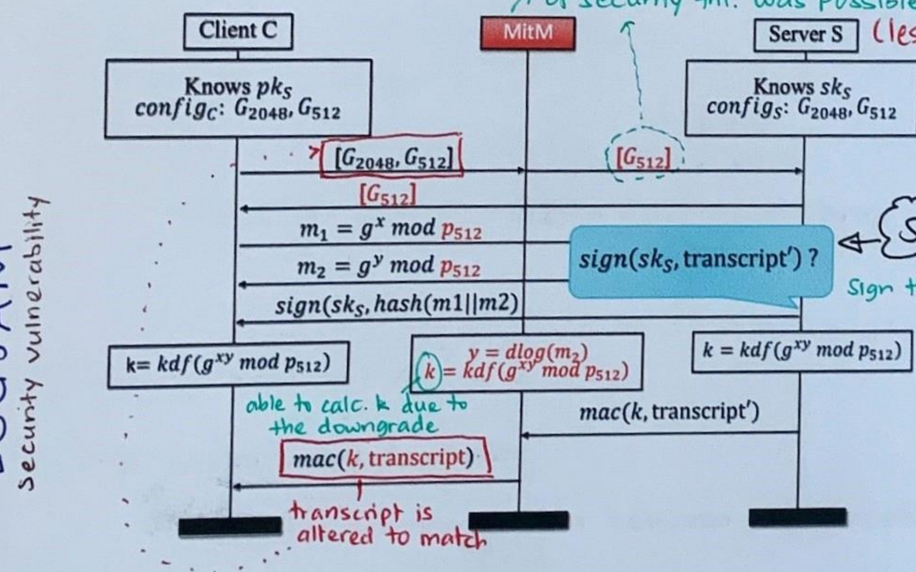
- Add a signature to X and Y they are sending. This requires each to know the PK of the other; certificates will solve this.

kdf – key derivation fn

SIGNED DH KEY EXCHANGE



is able to downgrade the level of security tht. was possible ; use less secure crypto suite (less bit-size keys)



SOLN

Sign the transcript with signing key, so even if an adversary intercepts it, bc. it doesn't know the signing key, the client will know

BLEICHENBACHER ATTACK [B98]

→ RSA: $E(m) = m^e \bmod n$

is <u>homomorphic</u>: $(m^e \cdot s^e)^d = (m \cdot s)^{ed} = m \cdot s$

→ PKCS#1 standard for RSA:

$$E_{(n,e)}(\text{message}, \text{padding}) = M^e \text{ where } M = (00\ 02\ \text{padding}\ 00\ \text{message})$$

→ The attack involves picking s_i adaptively.

For accepted $c \times s_i$, atacker knows that

$$M \times s_i = (00\ 02\ {*}{*}{*}\ 00\ {*}{*}{*})$$

So, build a system of inequalities

$$(00\ 02\ 00 \ldots 00) \le M \times s_i \le (00\ 02\ FF \ldots FF)$$