TLS supports session setup using a "pre-shared key" (so full handshake not needed):

client | server

$\longleftrightarrow$ full handshake

$\longleftarrow$ NewSessionTicket (nonce, id)

↓

preshared key

derived from session secrets, nonce, and id

$\Longrightarrow$

client | server

first message vulnerable to replay attack: { ClientHello + PresharedKey(id) $\longrightarrow$

$Enc_{AE}(k, \text{data})$ ← "0-RTT data"

↳ derived from preshared key

$\longleftarrow$ server response

fresh key $k_{A \to B}$, $k_{B \to A}$ derived for rest of session (based on initial messages)

Output of AKE protocol: (key, id)

negotiated key — identity of peer

- Authenticity: Only party that knows key is id (i.e., the party identified by id)
- Secrecy: All parties other than client and id cannot distinguish key from random (i.e., key is hidden)
- Consistency: If id also completes protocol, then it outputs (key, id_{client})

↳ if we do not have client authentication, then id_{client} is empty

Often also require forward secrecy: compromise of server in the future cannot affect secrecy of sessions in the past

↳ In TLS, server secret is a signing key — fresh Diffie-Hellman secret used for each session is fresh ("ephemeral")

Compromising signing key allows impersonation of server, but does not break secrecy of past sessions

↳ As we will see, not all AKE protocols provide forward secrecy

Very tricky to get right as we will see... Just use TLS!

AKE from PKE: suppose server has certificate authenticating a public key for a PKE scheme (CCA-secure):

$k \xleftarrow{R} K$ [Alice] — [Bank] sk_{Bank}, $cert_{Bank}$

$\longleftarrow$ r, $cert_{Bank}$ (r: nonce)

$\longrightarrow$ $c \leftarrow Enc(pk_{Bank}, (r, k))$

Alice ↓ k, Bank

Bank ↓ $k, \perp$ ← no client authentication

$(r', k) \leftarrow Decrypt(sk_{Bank}, c)$

check that $r' = r$

Yields statically-secure AKE (no forward secrecy)

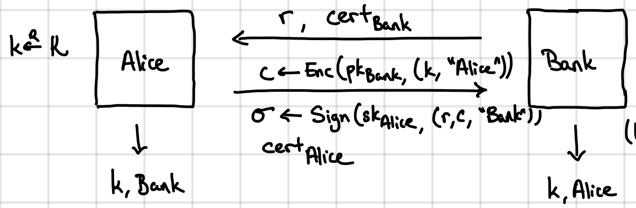
Compromise of sk_{Bank} compromises $\underline{\underline{\text{all}}}$ past sessions

If we do not encrypt the nonce r: replay attack possible (adversary replays messages from past session — e.g., "send Eve \$10")

↳ nonce ensures freshness

Mutual authentication: Bank has certificate identifying public key for PKE scheme

Alice has certificate identifying public key for signature scheme



$(k, \text{Alice}) \leftarrow \text{Dec}(sk_{\text{Bank}}, c)$

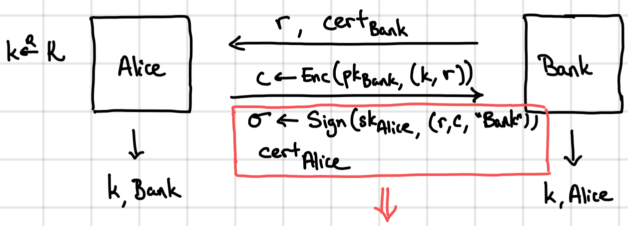
check Alice matches id in certificate

check Alice's signature on $(r, c, \text{"Bank"})$ under pk_{Alice} in $\text{cert}_{\text{Alice}}$

Above protocol provides static (no forward secrecy) mutual authentication

Most variants to this protocol are broken! AKE very delicate:

- Example: Suppose Alice encrypts (k, r) instead of $(k, \text{"Alice"})$ like in the server-auth protocol above
  - Vulnerable to "identity misbinding" attack where Alice thinks she's talking to Bank but Bank thinks it's talking to Eve:



$\sigma \leftarrow \text{Sign}(sk_{\text{Eve}}, (r, c, \text{"Bank"}))$ $\Longrightarrow$ Bank thinks it's talking to Eve

cert_{Eve}

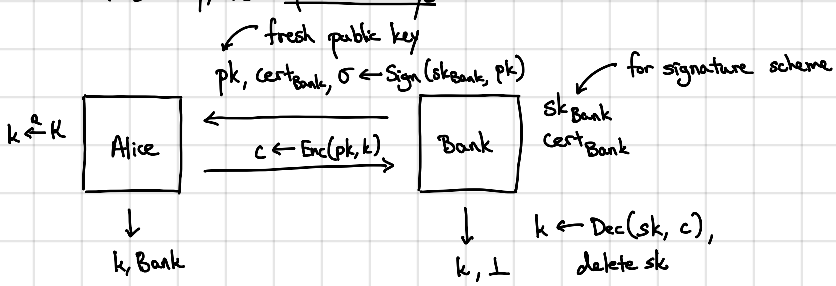
if Alice now sends "deposit this check into my account" to Bank,

Bank deposits it into Eve's account!

↳ observe that Eve did not break secrecy (she does not know k), but nevertheless, broke consistency

Above protocols supported by TLS 1.2, but deprecated in TLS 1.3 due to lack of forward secrecy

To get forward secrecy, use ephemeral keys:



Provides one-sided authentication (signature binds pk to Bank)

Forward secure since each pk used only once and long-term secret is signing key

totally broken without signature, adversary can replace pk with pk' and learn Alice's chosen key

hardware security module (used to protect cryptographic secrets)

Problem: Does not provide "HSM security"

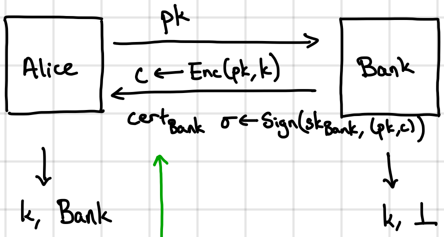
↳ Suppose adversary breaks into the bank and learns a single (pk', sk') pair with $\sigma \leftarrow \text{Sign}(sk_{\text{Bank}}, pk')$

↳ Adversary can now impersonate the bank to any client:

adversary always use the message $(pk', \text{cert}_{\text{Bank}}, \sigma)$

↳ can decrypt keys for all clients that responds!

defending against this requires freshness from client



$k \xleftarrow{R} \mathcal{K}$

Provides HSM security: client chooses <u>fresh</u> pk each time, so signature on pk functions as a "proof" that the other party possesses signing key for id identified by $\mathrm{cert}_{\mathrm{Bank}}$

In many cases, also want to hide the endpoint (the id identified by cert)
Possible by encrypting two keys (k, k') and using k' to encrypt $\mathrm{cert}_{\mathrm{Bank}}$

Diffie-Hellman key-exchange: substitute Diffie-Hellman handshake for the PKE scheme (simpler)
(TLS 1.2, 1.3)