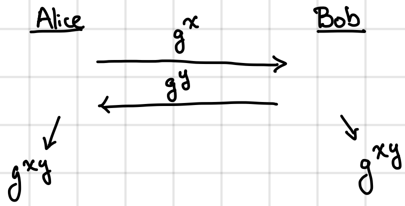
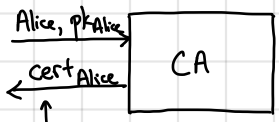
Now that we have digital signatures, let's revisit the question of key exchange (with active security)



} completely vulnerable to an active network adversary that can intercept and inject packets

In addition, should guarantee that one compromised session should not affect other honest sessions
- Alice ⟷ Eve should not compromise security of Alice ⟷ Bob

Authenticated key exchange (AKE): provides security against active adversaries
- Requires a "root of trust" (certificate authority) ⟶ we need some binding between keys and identities



(one-time setup, at least for duration of validity period)

└ the certificate binds Alice's public key pk_{Alice} to Alice's identity

- Certificates typically have the following format (X509):
  - Subject (entity being authenticated)
  - Public key (public key for subject for signature scheme)
  - CA: identity of the CA issuing the certificate
  - Validity dates for certificate
  - CA's signature on certificate ⟵ the browser and operating system have a set of hard-coded certificate authorities and their respective public keys (usually several hundred authorities) [public-key infrastructure (PKI)]

Basic flow of Diffie-Hellman based AKE:

Alice: $x \xleftarrow{R} \mathbb{Z}_p$

Bank: $y \xleftarrow{R} \mathbb{Z}_p$

Alice → Bank: g^x

Bank: $k, k' \leftarrow H(g, g^x, g^y, g^{xy})$

Bank: $\sigma \leftarrow \text{Sign}(sk_{Bank}, (g, g^x, g^y, pk_{Bank}))$

Bank → Alice: g^y, $c \leftarrow \text{Enc}_{AE}(k', (cert_{Bank}, \sigma))$

Alice: derive $k, k' \leftarrow H(g, g^x, g^y, g^{xy})$
check σ is signature on (g, g^x, g^y, pk_{Bank})
under pk_{Bank} is the public key identified by $cert_{Bank}$

Bank: session key k'

} intuition: $cert_{Bank}$ identifies server as Bank (with pk_{Bank})
σ binds the session parameters (g, g^x, g^y) to the public key identified by $cert_{Bank}$

End of protocol: Alice knows she is talking to Bank (but not vice versa!)
"one-sided AKE" — most common mode on the web

└→ Basis of TLS 1.3 handshake ("one-sided" AKE) ALWAYS USE TLS 1.3 — Don't invent your own AKE protocol!

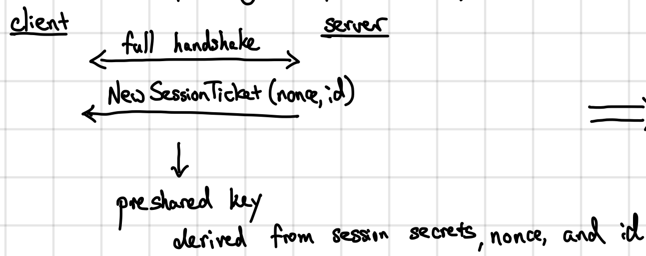
ClientHello: List of supported ciphersuites (e.g., AES-GCM-128, AES-GCM-256) ← older systems / foreign systems may prefer different ciphers
Possible TLS extensions

Server Hello: Chosen ciphersuite

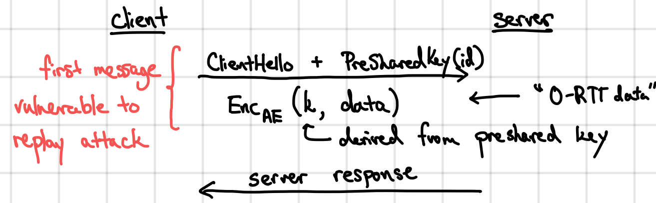
older versions of TLS vulnerable to cipher downgrade attacks

Application layer secured using unidirection keys $k_{A \to B}$ and $k_{B \to A}$

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



$\Longrightarrow$



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)

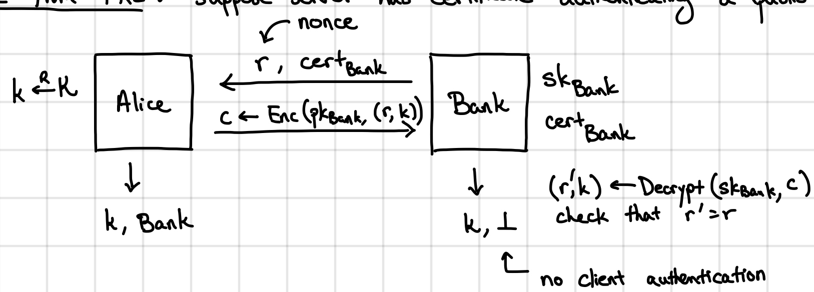
- Authenticity: Only party that knows key is id (i.e., the party identified by id)
- Secrecy: All parties other than client and id cannot distinguish key from random (i.e., key is hidden)
- Consistency: If id also completes protocol, then it outputs (key, id_{client})
  - if we do not have client authentication,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

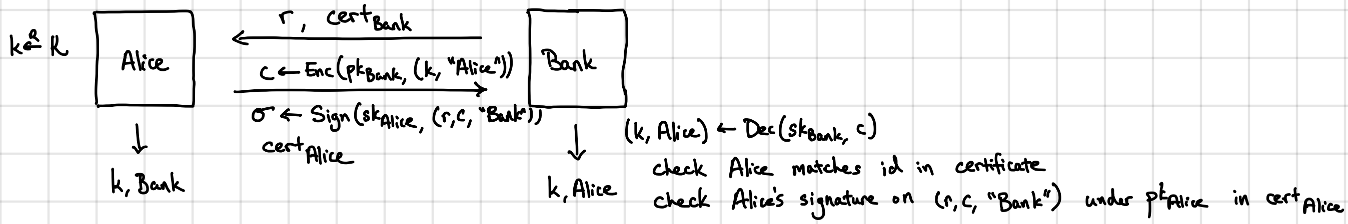


Yields statically-secure AKE (no forward secrecy)

Compromise of sk_{Bank} compromises 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)

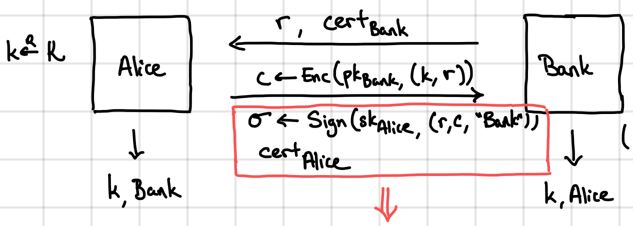
Mutual authentication: Bank has certificate identifying public key for PKE scheme
Alice has certificate identifying public key for signature scheme



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:



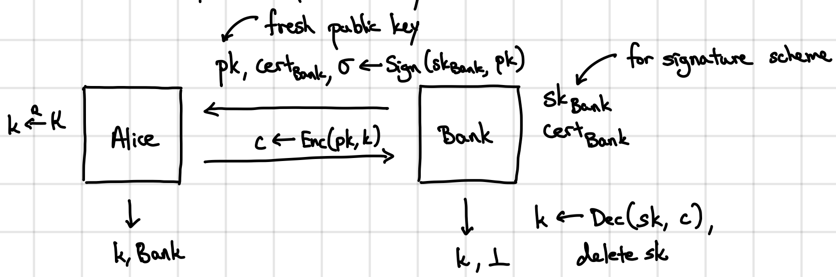
$\Downarrow$

$\sigma \leftarrow \text{Sign}(sk_{Eve}, (r, c, \text{"Bank"}))$ $\Longrightarrow$ 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 authenticity

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

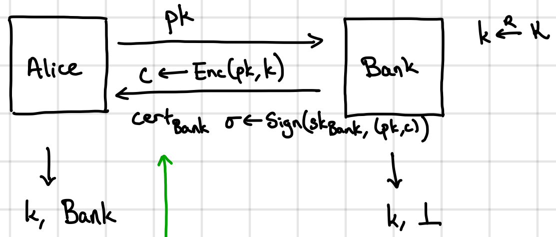
To get forward secrecy, use ephemeral keys:



Provides one-sided authentication (signature binds pk to Bank) ← totally broken without signature, adversary can replace pk with pk' and learn Alice's chosen key
Forward secure since each pk used only once and long-term secret is signing key

Problem: Does not provide "HSM security" (← hardware security module (used to protect cryptographic secrets))

- ↳ Suppose adversary breaks into the bank and learns a single (pk', sk') pair with $\sigma \leftarrow \text{Sign}(sk_{Bank}, pk')$
- ↳ Adversary can now impersonate the bank to any client:
  - adversary always use the message $(pk', \text{cert}_{Bank}, \sigma)$
    - ↳ can decrypt keys for all clients that responds!
  - } defending against this requires freshness from client



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 $cert_{Bank}$

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 $cert_{Bank}$

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