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

\[
\begin{align*}
\text{Alice} & \xrightarrow{g^x} \text{Bob} \\
& \leftarrow g^y
\end{align*}
\]

} 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) \( \rightarrow \) we need some binding between keys and identities

\[
\begin{align*}
\text{Alice, plain} & \xrightarrow[\text{certAlice}]{} \text{CA} \\
\text{CA} & \xleftarrow{\text{the certificate binds Alice’s public key } pk_{\text{Alice}} \text{ to Alice’s identity}} \text{Alice’s identity}
\end{align*}
\]

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

\[
\begin{align*}
\text{Alice} & \xrightarrow{x \in Z_p} g^x \xrightarrow{g^y, \text{cert}_{\text{Bank}}} (k', \text{cert}_{\text{Bank}}) \\
& \xrightarrow{k, k' \leftarrow H((g,g^x,g^y,g^xg^y))} \\
& \xrightarrow{\sigma \leftarrow \text{Sign}_{\text{cert}_{\text{Bank}}}(g,g^x,g^y,pk_{\text{Bank}})} \\
& \text{check } \sigma \text{ is signature on } (g,g^x,g^y,pk_{\text{Bank}}) \\
& \text{under } pk_{\text{Bank}} \text{ is the public key identified by } \text{cert}_{\text{Bank}} \\
\text{session key } k'
\end{align*}
\]

intuition: \( \text{cert}_{\text{Bank}} \) identifies server as \( \text{Bank} \) (with \( pk_{\text{Bank}} \))

End of protocol: Alice knows she is talking to \( \text{Bank} \) (but not vice versa!)

“one-sided AKE” — most common mode on the web

\[
\begin{align*}
\text{TLS 1.3 handshake ("one-sided" AKE)} \quad & \text{ALWAYS USE TLS 1.3 — Don’t invent your own AKE protocol!}
\end{align*}
\]

.client

<table>
<thead>
<tr>
<th>Client Hello</th>
<th>DH Key-Share</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server Hello</td>
<td>DH Key-Share</td>
</tr>
<tr>
<td>Certificate (encrypted)</td>
<td></td>
</tr>
<tr>
<td>Finished</td>
<td>Application Data</td>
</tr>
</tbody>
</table>

server

ClientHello: list of supported ciphersuites (e.g., AES-GCM-128, AES-GCM-256)

Possible TLS extensions

ServerHello: Chosen ciphersuite

Application layer secured using unidirectional keys \( k_{\text{Alice}} \rightarrow \text{Bob} \) and \( k_{\text{Bob}} \rightarrow \text{Alice} \)
TLS supports session setup using a "pre-shared key" (so full handshake not needed):

```
client ← full handshake server
Neo SessionTicket (nonce, id) ↓ pre-shared key derived from session secrets, nonce, and id
↓ negotiated key ← identity of peer
```

Output of AKE protocol: (key, id)

- **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 claim)

```
if we do not have client authentication, then id claim 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 = \{Alice \rightarrow r, \text{cert}\_Bank \leftarrow \text{Enc}_{\text{sk}\_Bank}(r, k) \rightarrow \text{Bank} \rightarrow \text{Sig}_{\text{sk}\_Bank}(\text{cert}\_Bank) \leftarrow \text{Alice} \rightarrow k, \text{Bank} \downarrow (r', k) \leftarrow \text{ Decrypt} (\text{sk}\_Bank, c) \downarrow k, 1 \check{=} \text{ check that } r' = r \downarrow \text{no client authentication}
```

Yields statically-secure AKE (no forward secrecy)

```
Compromise of sk\_Bank compromises all past sessions
```

If we do not encrypt the nonce c: replay attack possible (adversary replays messages from past session — e.g., "send Enc $t_1""

c nonce ensures freshness
Mutual authentication: Bank has certificate identifying public key for PKE scheme
Alice has certificate identifying public key for signature scheme

\[ k^2 \rightarrow \begin{array}{c|c} \text{Alice} & \text{Bank} \\ \hline \end{array} \]

\[ r, \text{cert}_\text{Bank} \leftarrow c = \text{Enc}(\text{pk}_\text{Bank}, (k, "Alice")) \]
\[ \sigma \leftarrow \text{Sign}(\text{some of Alice's info}), (r, c, "Bank") \]
\[ (k, Alice) \leftarrow \text{Dec}(\text{sk}_\text{Bank}, c) \]

Above protocol provides static (no forward secrecy) mutual authentication

Most variants to this protocol have been broken! AKF very delicate:

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

\[ \rightarrow \begin{array}{c|c} \text{Alice} & \text{Bank} \\ \hline \end{array} \]

\[ r, \text{cert}_\text{Bank} \leftarrow c = \text{Enc}(\text{pk}_\text{Bank}, (k, r)) \]
\[ \sigma \leftarrow \text{Sign}(\text{some of Alice's info}), (r, c, "Bank") \]
\[ \text{check Alice matches id in certificate} \]
\[ (k, Alice) \leftarrow \text{Dec}(\text{sk}_\text{Bank}, c) \]

\[ \text{Alice would think Bank authenticate} \]

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

To get forward secrecy, use ephemeral keys:

\[ k^2 \rightarrow \begin{array}{c|c} \text{Alice} & \text{Bank} \\ \hline \end{array} \]

\[ \text{pk, cert}_\text{Bank}, \sigma \leftarrow \text{Sign}(\text{some of Alice's info}) \]
\[ \text{for signature scheme} \]
\[ \text{sk}_\text{Bank}, \text{cert}_\text{Bank} \]

\[ k \leftarrow \text{Dec}(\text{sk}_\text{Bank}, c) \]
\[ k, \sigma \leftarrow \text{check Alice} \]

Above protocol provides on-demand, one-sided authentication

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

\[ \text{hardware security module (used to protect cryptographic secrets)} \]

Problem: Does not provide "HSM security"

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

- can decrypt keys for all clients that responds!
Diffie-Hellman key-exchange: substitute Diffie-Hellman handshake for the PKI scheme (simpler)

(TLS 1.2, 1.3)

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 certBank