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

\[
\begin{align*}
\text{Alice} & \quad g^x \quad \rightarrow \quad \text{Bob} \\
\downarrow & \quad g^{xy} \quad \downarrow \quad g^{xy} \\
\end{align*}
\]

\{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 \leftrightarrow Eve should not compromise security of Alice \leftrightarrow 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

\[
\begin{center}
\text{Alice, pk}\text{Alice} \rightarrow \text{CA} \\
\text{cert}\text{Alice} \rightarrow \text{Bob} \\
\end{center}
\]

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

- Certificates typically have the following format (X509):
  - Subject (entity being authenticated)
  - Public key (public key for subject for signature schema)
  - 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)

Basic flow of Diffie-Hellman based AKE:

\[
\begin{align*}
\text{Alice} & \quad x \in \mathbb{Z}_p \\
\rightarrow & \quad g^x \quad \rightarrow \quad \text{Bank} \\
\leftarrow & \quad g^{y} \\
\text{cert}_\text{Bank} & \rightarrow \text{Alice} \\
\end{align*}
\]

\[
\begin{align*}
\text{k, k}' & \leftarrow H(g, g^x, g^y, g^z) \\
\sigma & \leftarrow \text{Sign}(\text{cert}_\text{Bank}, (g, g^x, g^y, \text{pk}\text{Bank})) \\
\end{align*}
\]

\[
\begin{align*}
\text{check } \sigma \text{ is signature on } (g, g^x, g^y, \text{pk}\text{Bank}) \\
\text{under pk}\text{Bank is the public key identified by cert}\text{Bank} \\
\end{align*}
\]

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!

- Older systems/foreign systems may prefer different cipher suites

- Other versions of TLS vulnerable to cipher downgrade attacks

\[
\begin{align*}
\text{Client Hello} & \rightarrow \text{Server Hello} \\
\text{DH Key Share} & \rightarrow \text{Server Hello} \\
\text{Certificate} & \rightarrow \text{Client Hello} \\
\text{Finished} & \rightarrow \text{Application Data} \\
\end{align*}
\]

\[
\begin{align*}
\text{Client Hello: list of supported cipher suites} \\
\text{e.g., AES-GCM-128, AES-GCM-256} \\
\text{Possible TLS extensions} \\
\text{Server Hello: Chosen cipher suite} \\
\text{Application layer secured using unidirectional keys } k_{A\rightarrow B} \text{ and } k_{B\rightarrow A} \\
\end{align*}
\]
TLS supports session setup using a “pre-shared key” (so full handshake not needed):

\[
\begin{array}{c}
\text{client} \xrightarrow{\text{full handshake}} \text{server} \\
\text{New SessionTicket (nonce, id)} \quad \Rightarrow \\
\text{pre-shared key} \quad \text{derived from session secrets, nonce, and id} \\
\end{array}
\]

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, idents)

- if we do not have client authentication, then idents is empty

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

\[
\begin{align*}
\text{In TLS, server secret is a signing key — fresh Diffie-Hellman secret used for each session is fresh (“ephemeral”)} \\
\text{Compromising signing key allows impersonation of server, but does not break secrecy of past sessions} \\
\text{As we will see, not all AKE protocols provide forward secrecy}
\end{align*}
\]

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):

\[
\begin{align*}
&\begin{array}{c}
k \leftarrow K \\
A \leftarrow (r, \text{cert}_\text{Bank}) \\
\text{Alice} \\
\end{array} \\
&\begin{array}{c}
\text{c} \leftarrow \text{Enc}_\text{Bank}(r, k) \\
\text{Bank} \\
\end{array} \\
&\begin{array}{c}
k, \text{Bank} \\
\downarrow \\
\end{array} \\
&\begin{array}{c}
(r', k) \leftarrow \text{Dec}_\text{Bank}(\text{cert}_\text{Bank}, c) \\
\text{Alice} \\
\end{array} \\
&\begin{array}{c}
k = 1 \\
\text{check that } r' = r \\
\end{array} \\
&\begin{array}{c}
\text{no client authentication} \\
\end{array} \\
\end{align*}
\]

Yields statically-secure AKE (no forward secrecy)

Compromise of certBank 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 $r'$”)

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

\[ k^2 \rightarrow Alice \quad \begin{array}{c} r, \text{cert}_{\text{Bank}} \rightarrow c \leftarrow \text{Enc}(p_{\text{Bank}}, (k, \text{Alice})) \downarrow k, \text{Bank} \\ \sigma \leftarrow \text{Sign}(\text{Alice}, (r, c, \text{Bank})) \downarrow k, \text{Alice} \end{array} \quad \begin{array}{c} (k, Alice) \rightarrow \text{Dec}(sk_{\text{Bank}}, c) \downarrow k, Alice \end{array} \]

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 mistaking" attack where Alice thinks she's talking to Bank but Bank thinks it's talking to Eve:

\[ k^2 \rightarrow Alice \quad \begin{array}{c} r, \text{cert}_{\text{Bank}} \rightarrow c \leftarrow \text{Enc}(p_{\text{Bank}}, (k, r)) \downarrow k, \text{Bank} \\ \sigma \leftarrow \text{Sign}(\text{Alice}, (r, c, \text{Bank})) \downarrow k, Alice \end{array} \quad \begin{array}{c} \sigma \leftarrow \text{Sign}(sk_{\text{Bank}}, (r, c, \text{Bank})) \end{array} \]

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:

\[ k^2 \rightarrow Alice \quad \begin{array}{c} \text{fresh public key} \quad pk, \text{cert}_{\text{Bank}}, \sigma \leftarrow \text{Sign}(sk_{\text{Bank}}, pk) \downarrow \text{for signature scheme} \quad sk_{\text{Bank}} \quad \text{cert}_{\text{Bank}} \end{array} \quad \begin{array}{c} sk_{\text{Bank}} \quad \text{cert}_{\text{Bank}} \quad \text{for signature scheme} \end{array} \]

Provides one-sided authentication (signature binds pk to Bank)
Forward secure since each pk used only once and long-term secret is signing 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!
Diffie-Hellman key-exchange: substitute Diffie-Hellman handshake for the PKE scheme (simpler)

(TLS 1.2, 1.3)