Auth. Key Exchange
Review: key exchange

Alice and Bank want to generate a secret key

• So far we saw key exchange secure against eavesdropping

• This lecture: **Authenticated Key Exchange (AKE)**
  key exchange secure against **active** adversaries
Active adversary

Adversary has complete control of the network:
• Can modify, inject and delete packets
• Example: man-in-the-middle

Moreover, some users are honest and others are corrupt
• Corrupt users are controlled by the adversary
  – Key exch. with corrupt users should not “affect” other sessions
All AKE protocols require a TTP to certify user identities.

Registration process:

Two types of TTP: (here, we only consider offline TTP)

- **Offline TTP (CA):** contacted only during registration (and revocation)
- **Online TTP:** actively participates in every key exchange (Kerberos)

Benefit: security using only symmetric crypto
Followed by Alice sending $E(k, \text{“data”})$ to Bank and vice versa.
Basic AKE security (very informal)

Suppose Alice successfully completes an AKE to obtain \((k, \text{Bank})\)

If Bank is not corrupt then:

**Authenticity** for Alice: (similarly for Bank)
- If Alice’s key \(k\) is shared with anyone, it is only shared with Bank

**Secrecy** for Alice: (similarly for Bank)
- To the adversary, Alice’s key \(k\) is indistinguishable from random
  
  (even if adversary sees keys from other instances of Alice or Bank)

**Consistency**: if Bank completes AKE then it obtains \((k, \text{Alice})\)
Three levels of (core) security:

- **Static security**: previous slide

- **Forward secrecy**: static security, and if adv. learns $sk_{bank}$ at time $T$ then all sessions with Bank from time $t<T$ remain secret.

- **HSM security**: if adv. queries an HSM holding $sk_{bank}$ $n$ times, then at most $n$ sessions are compromised. Moreover, forward secrecy holds.

Several other AKE requirements ...
One-sided AKE: syntax

Used when only one side has a certificate.
- Similarly, three security levels.
Things to remember ...

Do not design AKE protocol yourself ...

Just use latest version of TLS
Building blocks

$\text{cert}_{\text{bank}}$: contains $pk_{\text{bank}}$. Bank has $sk_{\text{bank}}$.

$E_{\text{bank}}((m,r)) = E(pk_{\text{bank}}, (m,r))$ where $E$ is chosen-ciphertext secure

- Recall: from $E_{\text{bank}}((m,r))$ adv. cannot build $E_{\text{bank}}((m,r'))$ for $r' \neq r$

$S_{\text{alice}}((m,r)) = S(sk_{\text{alice}}, (m,r))$ where $S$ is a secure signing alg.

$R$: some large set, e.g. $\{0,1\}^{256}$
Protocol #1
Simple one-sided AKE protocol

k ← K

Alice

Bank

"Thm": this protocol is a statically secure one-sided AKE

Informally: if Alice and Bank are not corrupt then we have
(1) secrecy for Alice and (2) authenticity for Alice
Insecure variant 1: $r$ not encrypted

Problem: replay attack

Diagram:

- Alice
  - $k \leftarrow K$
  - $r \leftarrow R, \text{ cert}_{\text{bank}}$

- Bank
  - $c \leftarrow E_{\text{bank}}((k))$
  - $k, \text{ Bank}$
  - $\text{no } r$
  - $\text{sk}_{\text{bank}}$, $\text{cert}_{\text{bank}}$
  - $k, ??$
Replay attack

Later:

$k \leftarrow K$

$r \leftarrow R$, $\text{cert}_{\text{bank}}$

$c \leftarrow E_{\text{bank}}((k))$

$c_1 \leftarrow E_{\text{sym}}(k, \text{"I am Alice, pay Bob 30\$"})$

Later:

$r' \leftarrow R$, $\text{cert}_{\text{bank}}$

$c$

$c_1$
Two-sided AKE (mutual authentication)

```
Alice

r ← R , cert_{bank}

k ← K

c ← E_{bank}((k, "alice"))

σ ← S_{alice}((r, c, "bank")), cert_{alice}

Alice → Bank

Bank

sk_{bank} decrypt(c), check correct id, check sig. σ

cert_{bank}

k , Alice

thm: this protocol is a statically secure AKE

informally: if Alice and Bank are not corrupt then we have
(1) secrecy and (2) authenticity for Alice and for Bank
```
Insecure variant: encrypt $r$ instead of “Alice”

Any change to protocol makes it insecure, sometime in subtle ways

Example:
Dan Boneh

Attack: identity misbinding

\[ r \leftarrow R, \text{cert}_{\text{bank}} \]

\[ c \leftarrow E_{\text{bank}}((k, r)) \]

\[ \sigma \leftarrow S_{\text{alice}}((r, c, \text{"bank"})), \text{cert}_{\text{alice}} \]

\[ c \]

\[ \sigma' \leftarrow S_{\text{evil}}((r, c, \text{"bank"})), \text{cert}_{\text{evil}} \]

\[ E_{\text{sym}}(k, \text{"deposit this check into my account"}) \]
Problem: no forward secrecy

Recall the one-sided AKE:

\[
\begin{align*}
&k \leftarrow K, \text{Bank} \\
&r \leftarrow R, \text{cert}_\text{bank} \\
&c \leftarrow E_{\text{bank}}((k, r)) \\
&k \leftarrow K \\
&\text{decrypt}(c), \text{check correct } r \\
&k, ??
\end{align*}
\]

Suppose a year later adversary obtains \( sk_{\text{bank}} \)

\[
\Rightarrow \text{can decrypt all recorded traffic}
\]

This protocol is used in TLS 1.2, deprecated in TLS 1.3
Protocol #2: forward secrecy
Simple one-sided AKE with forward-secrecy

Alice

Bank

check sig. $\sigma$

$k \leftarrow K$

$pk, cert_{bank}$

$\sigma \leftarrow S_{bank}((pk))$

$c \leftarrow E(pk, k)$

$k \leftarrow D(sk, c)$

delete $sk$

Bank

$(pk, sk) \leftarrow \text{Gen}$

$k, ??$

$(pk, sk)$ are ephemeral: sk is deleted when protocol completes

Compromise of Bank: past sessions are unaffected
Insecure variant: do not sign pk

**Attack:** complete key exposure
Attack: key exposure

Alice

\[ (pk', sk') \leftarrow \text{Gen} \]

\[ \text{pk}', \text{cert}_{\text{bank}} \]

Bank

\[ \text{pk}, \text{cert}_{\text{bank}} \]

\[ (pk, sk) \leftarrow \text{Gen} \]

\[ k \leftarrow K \]

\[ c \leftarrow E(pk', k) \]

\[ E_{\text{sym}}(k, \text{"data"}) \]

Adv. gets \( k \) and data
Problem: not HSM secure

Suppose attacker breaks into Bank and queries HSM once
⇒ complete key exposure forever!
Problem: not HSM secure

Single HSM query:

\((pk', sk') \leftarrow \text{Gen}\)

\(\sigma' \leftarrow S_{\text{bank}}((pk'))\)

\(c \leftarrow E(pk', k)\)

Attacker gets Alice’s data encrypted with \(k\)

Alice

Bank

HSM

\(sk_{\text{bank}}\)

\(cert_{\text{bank}}\)

\(k, Bank\)

\(k\)
Protocol #3: HSM Security
Simple HSM security (one-sided)

Main point: HSM needed to sign ephemeral pk from client ⇒ past access to HSM will not compromise current session
Final variant: end-point privacy

Protocol #3: eavesdropper learns that Alice wants to talk to Bank.
Solution: hide $\text{cert}_{\text{bank}}$

$(pk, sk) \leftarrow \text{Gen}$
$k, k' \leftarrow D(sk, c)$
decrypt $c'$
check sig. $\sigma$
delete $sk$

$(pk, sk) \leftarrow \text{Gen}$
$k, k' \leftarrow K$

$c \leftarrow E(pk, (k, k'))$
$c' \leftarrow E_{\text{sym}}(k', (\text{cert}_{\text{bank}}, \sigma))$
Using Diffie-Hellman: DHAKE (simplified)

We can use Diffie-Hellman instead of general public-key encryption.

\[
\begin{align*}
\alpha & \leftarrow \mathbb{Z}_q \\
k, k' & \leftarrow H(g^{\alpha \beta}) \\
deencrypt c' & \\
check sig. \sigma & \\
delete \alpha & \\
\end{align*}
\]

\[
\begin{align*}
\text{Alice} & \\
g^\alpha & \in G \\
\text{Bank} & \\
g^\beta & \in G \\
c' & \leftarrow E_{sym}(k', (\text{cert}_{\text{bank}}, \sigma)) \\
\end{align*}
\]

[variant of TLS 1.3]
Many more AKE variants

AKE based on a shared secret between Alice and Bank:
• High entropy shared secret: ensure forward secrecy
• Password: ensure no offline dictionary attack (PAKE)

Deniable:
• Both sides can claim they did not participate in protocol
• In particular, parties do not sign public messages
Auth. key exchange

TLS 1.3 Session Setup

RFC 8446 (Aug. 2018)
TLS 1.3 Session Setup

Generate unidirectional keys: \( k_{b\rightarrow s} \) and \( k_{s\rightarrow b} \)

Security goals:
- Support for one-sided and two-sided AKE
- HSM security (including forward secrecy and static security)
- End-point privacy against an eavesdropper

Protocol is related to the Diffie-Hellman protocol DHAKE above
TLS 1.3 session setup (full handshake, simplified)

Client

ClientHello (cipherSuites, extensions)
KeyShare (Diffie-Hellman)

ServerHello (chosen cipherSuite),
KeyShare (Diffie-Hellman),
Encrypted Certificate

Finished

Server

secret key

cert

Application Data
The need for negotiating ciphers

US browser

prefer
NIST ciphers

Web server in Russia

Russian browser

 Prefer GOST ciphers (Russian)

does not understand
ECDHE

old browser
Session setup from pre-shared keys

Goal: reduce # of full handshakes

Full handshake

NewSessionTicket(nonce, ID)

PreSharedKey

PreSharedKey

derived from session secrets and nonce

Later (new TCP connection)

ClientHello w/PreSharedKey(ID)

Abbreviated handshake

$k_b \rightarrow s$ and $k_s \rightarrow b$

Session Store

Bank

retrieve old

PreSharedKey

or

recompute

from ID
**PSK 0-RTT**

ClientHello w/PreSharedKey(ID)

$E_{sym}(k_{ce}, \text{ 0-RTT application data})$

Abbreviated handshake

$k_{b\rightarrow s}$ and $k_{s\rightarrow b}$

$k_{CE}$ : client early key-exchange key. derived from PSK (and other CH data)

Problem: 0-RTT app data is vulnerable to replay.
THE END