Auth. Key Exchange
Review: key exchange

Alice and Bank want to generate a secret key

- 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 exchange with corrupt users should not “affect” other sessions
- Adversary may corrupt an honest user at time $T$
  - We want sessions established at time $t < T$ to remain “secure”
All AKE protocols require a TTP to certify user identities.

Registration process:

Two types of TTP:

- **Online TTP**: actively participates in **every** key exchange (Kerberos)
  Benefit: security using only symmetric crypto

- **Offline TTP (CA)**: contacted only during registration (... not quite true)
Followed by Alice sending $E(k, \text{"data"})$ to Bank
Suppose Alice successfully completes an AKE to obtain \((k, \text{Bank})\)

If Bank is not corrupt then:

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

**Secrecy** for Alice: \((\text{similarly for Bank})\)
- 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 \((k, \text{Alice})\)
One-sided AKE

Security: authenticity for Alice and secrecy for Alice

- Bank has no guarantees for identity of peer (no consistency)
- Commonly used on the Web (often followed by ID protocol)

\[ vk_{CA}, k, Bank \text{ or } \perp \]

\[ sk_{bank}, cert_{bank}, vk_{CA}, k, ?? \text{ or } \perp \]
Things to remember ...

Do not design AKE protocol yourself ...

Just use latest version of TLS
Building blocks

cert_{\text{bank}}: \text{ contains } pk_{\text{bank}}. \text{ Bank has } sk_{\text{bank}}.

\[ E_{\text{bank}}((m,r)) = E(\text{pk}_{\text{bank}}, (m,r)) \] where E is \textit{chosen-ciphertext secure}

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

\[ S_{\text{alice}}((m,r)) = S(\text{sk}_{\text{alice}}, (m,r)) \] where S is a signing algorithm

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

```
\text{Alice} \quad k \leftarrow K

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

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

\text{Bank} \quad \text{sk}_{\text{bank}}, \quad \text{cert}_{\text{bank}}

decrypt(c), \quad \text{check correct } r

k, ??
```

"Thm": this protocol is a 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

- $k \leftarrow K$
- $k, \text{Bank}$
- $r \leftarrow R, \text{cert}_{\text{bank}}$
- $c \leftarrow E_{\text{bank}}(k)$

Bank

- $\text{sk}_{\text{bank}}$
- $\text{cert}_{\text{bank}}$
- $c$
- no $r$
- $k, ??$
Replay attack

$k \leftarrow K$

Alice

$r \leftarrow R$, $cert_{bank}$

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

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

Bank

Later:

$r' \leftarrow R$, $cert_{bank}$

c

c_1

Bank

Dan Boneh
Protocol #2
Simple one-sided AKE with forward-secrecy

(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

\[
\begin{align*}
\sigma & \leftarrow S_{\text{bank}}(pk) \\
c & \leftarrow E(pk, k)
\end{align*}
\]
Attack: key exposure

\[(pk', sk') \leftarrow Gen\]

\[pk', cert_{bank} \leftarrow Gen\]

\[k \leftarrow K\]

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

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

Adv. gets k and data

Dan Boneh
Two-sided AKE

For now: no forward secrecy
Two-sided AKE (mutual authentication)

```
\[ k \leftarrow K \]

Alice

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

Bank

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

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

Alice

\[ k, \text{ Bank} \]

Bank

\[ k, \text{ Alice} \]

"Thm": this protocol is a secure AKE

Informally: if Alice and Bank are not corrupt then we have
(1) secrecy and (2) authenticity for Alice and for Bank

Dan Boneh
Insecure variant: encrypt r instead of “Alice”

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

Example:
**Attack: identity misbinding**

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

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

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

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

\[ E_{\text{sym}}(k, \text{“deposit this check into my account”}) \]
Insecure variant: do not sign c

Attack: key exposure
Attack: key exposure

Alice

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

\[ c \leftarrow E_{bank}((k, "Alice")) \]

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

\[ c' \leftarrow E_{bank}((k', "Alice")) \]

\[ \sigma, \quad \text{cert}_{alice} \]

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

Bank

Adversary can read data

decrypt(c'), check id, check sig. \sigma
Many more AKE variants

Two-sided AKE with forward secrecy:

AKE with end-point privacy:
- Goal: certificates are not visible to adversary (TLS 1.3)

AKE based on a shared secret between Alice and Bank:
- High entropy shared secret: want forward secrecy
- Password: ensure no offline dictionary attack (PAKE)
Auth. key exchange

TLS v1.2 key exchange
TLS session setup (handshake)

Client

- ClientHello (and extensions)
- [Certificate], ClientKeyExchange, [CertificateVerify]
- ChangeCipherSpec
- Finished

Server

- ServerHello (and ext.), [Certificate], [ServerKeyExchange], [CertificateRequest], ServerHelloDone
- [Certificate]
- ClientKeyExchange
- [CertificateVerify]
- Finished

Application Data
Brief overview of SSL/TLS

- **ClientHello** (cipher-list)
- **ServerHello** (cipher), **ServerCert** (PK)
- **key exchange** (many options): RSA, DHE, ECDHE, ...
  - [ServerKeyExchange]
  - **ClientKeyExchange**
- **Finished** (key & params confirmation)
- HTTP data encrypted with KDF(PreK)

In this diagram: one sided authentication (no client authentication)
The need for negotiating ciphers

Web server in Russia

US browser prefer NIST ciphers

Russian browser prefer GOST ciphers (Russian)

don't understand ECDHE

old browser
Abstract TLS: RSA exchange (simplified)

ClientHello: $r_C, \text{SID}, \text{cipher-list}$

ServerHello: $r_S, \text{SID}, \text{cipher, cert}_S$

ClientKeyExchange: $c \leftarrow E(pk_S, \text{PreK})$

$\text{MasterK} \leftarrow \text{PRF}_{\text{ms}}(\text{PreK}, r_C || r_S)$

$\text{SessionKeys} \leftarrow \text{PRF}_{\text{ke}}(\text{MasterK}, r_S || r_C)$

Finished (FinishedData)

Key Confirmation: $\text{FinishedData} = \text{PRF}_{vd}(\text{MasterK, hash(HandshakeMessages)})$
Properties

\( r_C, r_S \): prevent replay of old session

**RSA key exchange: no forward secrecy**

– Compromise of server secret key exposes old sessions

– Costly RSA decryption on server, easier RSA enc. on client

**One sided identification:**

• Browser identifies server using server-cert

• Server has no guarantees about client’s identity

• TLS has support for mutual auth. (client needs sk_C and cert_C)
TLS key exchange with forward-secrecy (DHE) (simplified)

Client

- ClientHello($r_c$)
- verify cert$_S$ and $\sigma$
- random b in 1..p
- $PreK \leftarrow g^{ab}$
- Delete b

Server

- ServerHello($r_s$), Cert, ServerKeyExchange
- $p, g, A \leftarrow g^a(p)$, $\sigma \leftarrow \text{sign}(sk_S, (r_c, r_s, p, g, A))$
- $ClientKeyExchange: B \leftarrow g^b(p)$
- MasterK $\leftarrow \text{PRF}_{ms}(PreK, r_c \parallel r_s)$
- SessionKeys $\leftarrow \text{PRF}_{ke}(MasterK, r_s \parallel r_c)$
- Delete a

Fix prime $p$ and $g$

$sk_S$: signing key
www.google.com
The identity of this website has been verified by Thawte SGC CA.

Certificate Information

Your connection to www.google.com is encrypted with 128-bit encryption.

The connection uses TLS 1.0.

The connection is encrypted using RC4_128, with SHA1 for message authentication and ECDHE_RSA as the key exchange mechanism.

Prefer ECDHE over DHE

Elliptic curve Diffie-Hellman
Performance: RSA vs. forward-secrecy

Cost of crypto operations on server per handshake:

- **RSA key exchange**: one RSA-2048 decryption (deprecated in TLS 1.3)

- **ECDHE**: Diffie-Hellman in group G with generator $g \in G$
  
  1. One exp. to compute $A \leftarrow g^a \in G$
  2. One sig. on Diffie-Hellman parameters $(G,g,A)$
  3. One exp. to compute DH secret: $\text{PreK} \leftarrow g^{ab} \in G$

Server support (2014): RSA (99.9%), DHE (60%), ECDHE (18%)
Session Resume

Goal: reduce # of full handshakes

Few hours later (new TCP connection)

Abbreviated handshake

reuse old MasterK
Session resume (simplified)

Client

**MasterK(bank)**

- SID$_C$=0: full handshake
- SID$_C$ ≠ 0: resume old session

If SID$_C$ = SID$_S$ then resume
else full

**ClientHello:** $r_C, \ SID_C$

SID$_S$ ← SID$_C$ if SID$_C$ ∈ ST
SID$_S$ ← random, otherwise

**ServerHello:** $r_S, \ SID_S$

SessionKeys ← $\text{PRF}_{ke}(\text{MasterK}, \ r_S \ || \ r_C)$

ChangeCipherSpec

Finished

Bank

**Session Store (ST)**

**MasterK(Alice)**

ChangeCipherSpec

Finished
THE END