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)
Offline TTP: Certificate Authority (CA)

Registration process:

Alice

\[ v_{CA} \]

verify cert

Bank

choose

\((s_{k_{bank}}, p_{k_{bank}})\)

\[ v_{CA} \]

Cert. Signing Request (CSR)

\[ p_{k_{bank}} \text{ and } \text{proof } \text{"I am Bank.com" and I "know" } s_{k_{bank}} \]

Certificate Authority (CA)

check proof

\[ s_{k_{CA}} \]

Assumptions:

- all parties have a certified \( p_{k} \) (even corrupt users)
- only Bank can get a CA certificate for Bank
Certificates: example

Important fields:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Number</td>
<td>5814744448373890497</td>
</tr>
<tr>
<td>Version</td>
<td>3</td>
</tr>
<tr>
<td>Signature Algorithm</td>
<td>SHA-1 with RSA Encryption (1.2.840.113549.1.1.5)</td>
</tr>
<tr>
<td>Parameters</td>
<td>none</td>
</tr>
<tr>
<td>Not Valid Before</td>
<td>Wednesday, July 31, 2013 4:59:24 AM Pacific Daylight Time</td>
</tr>
<tr>
<td>Not Valid After</td>
<td>Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time</td>
</tr>
<tr>
<td>Public Key Info</td>
<td></td>
</tr>
<tr>
<td>Algorithm</td>
<td>Elliptic Curve Public Key (1.2.840.10045.2.1)</td>
</tr>
<tr>
<td>Parameters</td>
<td>Elliptic Curve secp256r1 (1.2.840.10045.3.1.7)</td>
</tr>
<tr>
<td>Public Key</td>
<td>65 bytes : 04 71 6C DD E0 0A C9 76 ...</td>
</tr>
<tr>
<td>Key Size</td>
<td>256 bits</td>
</tr>
<tr>
<td>Key Usage</td>
<td>Encrypt, Verify, Derive</td>
</tr>
<tr>
<td>Signature</td>
<td>256 bytes : 8A 38 FE D6 F5 E7 F6 59 ...</td>
</tr>
</tbody>
</table>

mail.google.com
- Issued by: Google Internet Authority G2
- Expires: Thursday, July 31, 2014 4:59:24 AM Pacific Daylight Time
- This certificate is valid

Details:
- Subject Name: US
- Country: US
- State/Province: California
- Locality: Mountain View
- Organization: Google Inc
- Common Name: mail.google.com
- Issuer Name: Google Internet Authority G2
- Common Name: Google Inc
Followed by Alice sending $E(k, \text{"data"})$ to Bank

Session-id (sid): public id of session

- Identifies instance of Alice that is talking to instance of Bank

$$\text{sid} = \text{sid}' \iff k = k'$$
Suppose Alice successfully completes an AKE to obtain \((k, \text{sid}, \text{Bank})\). 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 \((k, \text{sid}, \text{Alice})\)
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)
Alice uses same $k$ for **all** key exchanges. Is this a secure one-sided AKE?

No, a single AKE with a corrupt user will reveal $k$

Lots of other problems with this protocol ...
Things to remember ...

Do not design AKE protocol yourself ...

Just use latest version of TLS

(sid for TLS is called channel binding)
Building blocks

**cert\textsubscript{bank}**: contains pk\textsubscript{bank}. Bank has sk\textsubscript{bank}.

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

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

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

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

Informally: if Alice and Bank are not corrupt then we have
(1) secrecy for Alice and (2) authenticity for Alice
Did the AKE complete successfully?

Problem: Alice does not know if Bank received $k$

One-sided: Bank does not know who $k'$ is shared with
Explicit AKE via key confirmation

Key confirmation: if succeeds, Alice is assured Bank has k.

**Diagram:**
- Alice
  - $k \leftarrow K$
  - $(k_0, k_a, k_b) \leftarrow \text{PRG}(k)$
  - check $k_b$
  - $k_0, \text{sid} \leftarrow (c, r), \text{Bank}$

- Bank
  - $r \leftarrow R, \text{cert}_{\text{bank}}$
  - $c \leftarrow \text{E}_{\text{bank}}((k, r)), k_a$
  - $(k_0, k_a, k_b) \leftarrow \text{PRG}(k)$
  - check $k_a$
  - $k_0, \text{sid} \leftarrow (c, r), ??$

**Note:** $k_0$ used as session key (not k)
Insecure variant 1: r not encrypted

Problem: replay attack
Replay attack

$k \leftarrow K$

Alice

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

c \leftarrow \text{E}_{bank}(k)$

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

Bank

Later:

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

c

c_1

Bank
Insecure variant 2: $E_{\text{bank}}$ not CCA-secure

Suppose: from $c = E_{\text{bank}}((k, r))$ can construct $c' = E_{\text{bank}}((k, r'))$ for any $r'$

⇒ replay attack
Replay attack

Alice

\[ k \leftarrow K \]

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

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

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

Bank

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

Later:

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

\[ c = E_{\text{bank}}((k, r)) \rightarrow c' = E_{\text{bank}}((k, r')) \]

\[ c_1 \]

Bank

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

accept \( c' \)
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
Attack: key exposure

\( \text{Alice} \)

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

\( pk', \ text{cert}_{\text{bank}} \) →

\[ \text{bank} \]

\( k \leftarrow K \)

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

\[ \text{Adv. gets} \]

\[ k \text{ and data} \]

\[ \text{E}_{\text{sym}}(k, \ "data") \]
Two-sided AKE

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

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

"Thm": this protocol is a secure AKE
Insecure variant: encrypt $r$ instead of “Alice”

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

Example:
Attack: identity misbinding

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

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

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

\[ c \leftarrow \text{decrypt(c), check } r, \text{ check sig. } \sigma' \]

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

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

Attack: key exposure
Attack: key exposure

\[
\begin{align*}
r &\leftarrow R, \text{ cert}_{\text{bank}} \\
\sigma &\leftarrow S_{\text{alice}}((r, \text{“bank”}), \text{ cert}_{\text{alice}}) \\
c &\leftarrow E_{\text{bank}}((k, \text{“Alice”})) \\
c' &\leftarrow E_{\text{bank}}((k', \text{“Alice”}))
\end{align*}
\]

Adversary can read data

\[E_{\text{sym}}(k', \text{“data”})\]

decrypt(c'), check id, check sig. \sigma
Can we defeat the attack on previous slide with key confirmation?

Yes, the attacker does not know $k$ and cannot send a valid $k_b$. 

$k \leftarrow K$

c, $\sigma$, $\text{cert}_{\text{alice}}$, $k_a$

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

$E_{\text{sym}}(k, \text{“deposit in my account”})$

check $k_a$

cHECK $k_b$

check $k_b$
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

ChangeCipherSpec

Finished

Server

ServerHello (and ext.),
[Certificate],
[ServerKeyExchange],
[CertificateRequest],
ServerHelloDone

secret key

cert

Application Data
Brief overview of SSL/TLS

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

Web server in Russia

US browser

Prefer GOST ciphers (Russian)

Russian browser

Prefer NIST ciphers

Old browser
does not understand ECDHE
Abstract TLS: RSA exchange (simplified)

Client

- verify $cert_s$
- pick random 46-byte PreK

Server

- secret key $cert_s$
- decrypt c to get PreK

ClientHello: $r_C, SID, cipher-list$

ServerHello: $r_S, SID, cipher, cert_s$

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

MasterK $\leftarrow PRF_{ms}(PreK, r_C \| r_S)$

SessionKeys $\leftarrow PRF_{ke}(MasterK, r_S \| r_C)$

Finished (FinishedData)

Key Confirmation: $FinishedData = PRF_{vd}(MasterK, \text{hash(HandshakeMessages)})$
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 \( cert_C \))
Suppose always $r_C = 0$, but $r_S$ is random

Would this be a secure one-sided AKE?

- No, an attacker can replay an old session to the server
- No, an attacker can replay an old session to the client
- Yes, it would be a secure one-sided AKE
- No, a man in the middle can expose PreK
TLS key exchange with forward-secrecy (DHE) (simplified)

Client:
- `ClientHello(r_c)`
- `ClientKeyExchange`: $B \leftarrow g^b(p)$
- `PreK \leftarrow g^{ab}`
- `Delete b`

Server:
- `ServerHello(r_s), Cert, ServerKeyExchange`
- `ServerKeyExchange`
- `PreK \leftarrow g^{ab}`
- `Delete a`

Fix prime $p$ and $g$

`sk_s`: signing key
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: $PreK \leftarrow g^{ab} \in G$

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

retrieve old MasterK
Session resume (simplified)

Client

ClientHello: $r_C, \text{SID}_C$

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

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

ChangeCipherSpec

Finished

Bank

Session Store (ST)

MasterK(Alice)

MasterK(bank)

If $\text{SID}_C = \text{SID}_S$ then resume else full

$\text{SID}_C = 0$: full handshake

$\text{SID}_C \neq 0$: resume old session

$\text{SID}_S \leftarrow \text{SID}_C$ if $\text{SID}_C \in \text{ST}$

$\text{SID}_S \leftarrow$ random, otherwise

If $\text{SID}_C = \text{SID}_S$ then resume

else full

$\text{MasterK}(\text{bank})$

$\text{MasterK}(\text{Alice})$
On a first connection Alice is connected to Server #1. What must the site do to avoid a full handshake on subsequent connections from Alice?

- Ensure that Alice always connects to server #1
- Ensure that SessionStore is shared among all servers in farm
- Do all TLS processing in the load balancer
- All of the above are reasonable solutions
THE END