1. Alice and Bob share a key
   - How do they determine that they do?
   - Challenge-response protocols

2. How do they establish the shared secret in the first place?
   - Key distribution
   - PKI, Kerberos, Other key distribution schemes
Challenge-Response Protocols

1. Alice issues to challenge to Bob
2. Bob responds
3. Bob challenges Alice
4. Alice responds
5. The challenge: “prove that you have the key”
Two-Way Authentication

First Attempt

\[
\begin{align*}
A & \rightarrow A \\
R_b & \leftrightarrow R_b \\
E(K, R_b) & \rightarrow E(K, R_b) \\
R_a & \leftarrow R_a \\
E(K, R_a) & \leftrightarrow E(K, R_a)
\end{align*}
\]

Alice \hspace{1cm} Bob

Improved

\[
\begin{align*}
A, R_a & \rightarrow A, R_a \\
R_b, E(K, R_a) & \leftrightarrow R_b, E(K, R_a) \\
E(K, R_b) & \rightarrow E(K, R_b)
\end{align*}
\]

Alice \hspace{1cm} Bob
Reflection Attack

Alice? \[
\begin{align*}
&\Rightarrow A, R_a \\
&\Leftarrow R_b, E(K, R_a) \\
&\Rightarrow A, R_b \\
&\Leftarrow R_b', E(K, R_b) \\
&\Rightarrow E(K, R_b)
\end{align*}
\]

Bob

Watch-out for Parallel Sessions!

Trick the challenger into responding to his own challenge in a parallel session
Some Thumb Rules

1. Have initiator prove who he/she is before responder has to
2. Initiator and responder should use different keys
   1. Different keys may be derived from the same key $K$ as (for example) $K_1 = h(K \parallel 1)$ and $K_2 = h(K \parallel 2)$.
3. Initiator and responder should draw challenges from different sets (odd/even)
Authentication Using HMACs

Alice

⇒ \( A, R_a \)

⇐ \( R_b, \text{HMAC}(R_a \parallel R_b \parallel A \parallel B \parallel K) \)

⇒ \( \text{HMAC}(R_a \parallel R_b \parallel K) \)

Bob
**Diffie Helman Key Exchange**

Alice

\[
\begin{align*}
\Rightarrow & \quad p, g, \alpha \equiv g^a \mod p \\
\Leftarrow & \quad \beta \equiv g^b \mod p
\end{align*}
\]

Bob

**Common Secret**

Both Alice and Bob compute a common secret

\[K_{AB} = \beta^a \equiv \alpha^b \equiv g^{ab} \mod p\]
Man-in-the-Middle

\[
\begin{align*}
A & \quad \Rightarrow \quad p, g, \alpha \equiv g^a \mod p \\
& \quad \Leftarrow \quad \xi \equiv g^x \mod p \\
O & \quad \Rightarrow \quad p, g, \omega \equiv g^w \mod p \\
& \quad \Leftarrow \quad \beta \equiv g^b \mod p \\
B &
\end{align*}
\]

**MiM Attack**

1. Oscar tells B that \( \omega \) is A’s public key
2. An A that \( \xi \) is B’s public key
3. If Oscar is lazy he can even choose \( \omega = \xi \) (he needs to generate only one key pair instead of 2)
1. Alice and Bob exchange their public keys
2. Bob chooses a random secret $K_A$, encrypts it using Alice’s public key $\alpha$
3. Alice chooses a random secret $K_B$, encrypts it using Bob’s public key $\beta$
4. shared secret can be any function of $K_A$ and $K_B$.
5. The problem is
   - Bob has no way of confirming that $\alpha$ is Alice’s public key
   - Alice has no way of confirming that $\beta$ is Bob’s public key
6. Need some way to *bind* “Alice” with “$\alpha$” (and “Bob” with “$\beta$”).
Certificate Authority

1. CA certifies bindings
2. CA generates a key-pair
3. CA’s public key publicized widely
4. CA signs certificates using its private key
5. Anyone with a legitimate copy of the CAs public key can verify the certificate
6. X.509: standard format for public key certificates
## X.509 Certificate Format

<table>
<thead>
<tr>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial Number</td>
</tr>
<tr>
<td>Signature Algorithm Identifier</td>
</tr>
<tr>
<td>Issuer Name</td>
</tr>
<tr>
<td>Validity Period</td>
</tr>
<tr>
<td>Subject Name</td>
</tr>
<tr>
<td>Public Key Information</td>
</tr>
<tr>
<td>Issuer Unique ID</td>
</tr>
<tr>
<td>Subject Unique ID</td>
</tr>
<tr>
<td>Extensions</td>
</tr>
</tbody>
</table>

The X.509 certificate format includes a version number, serial number, issuer name, validity period, subject name, public key information, issuer and subject unique IDs, and extensions. The version is specified by the version number, which can be 1, 2, or 3.
X.509 Certificate Format

(a) X.509 certificate

Version
Certificate serial number
Algorithm Parameters
Issuer name
Period of validity
Not before
Not after
Subject name
Subject's public-key info
Algorithms Parameters
Key
Issuer unique identifier
Subject unique identifier
Extensions
Signature
Algorithms Parameters
Encrypted
Signature
Algorithm Parameters
Issuer name
This update date
Next update date
Revoked certificate
User certificate serial #
Revocation date
...(b) Certificate revocation list
Public Key Infrastructure

1. Need to avoid a single CA
2. Alice’s public key signed by Verisign. Bob’s by SecureNet
3. Alice may not recognize SecureNet (Bob may not recognize Verisign)
4. Alice accepts SecureNet’s public key only if it is authenticated by Verisign?
5. Does every CA need to sign public keys of all other CAs?
Forward and Reverse Certificates

1. X signs public key of A represented as $X << A >>$
2. If A signs public key of X then we also have $A << X >>$
3. Forward and reverse certificates.
4. Does not matter which is forward and which is reverse.
5. In practice the certificate issued by the better known CA is considered as forward certificate?
Why Reverse Certificates?

1. Let $Y$ be a lesser known CA
2. $Y$ gets its public key signed by better known CA $X$ (or $X << Y >>$ is created)
3. In turn $Y$ signs a reverse certificate $Y << X >>$
4. This reverse certificate helps customers of $Y$ (who trust only $Y$)
5. $Y$’s customers can now gain a trusted copy of $X$’s public key
Why Reverse Certificates?

1. A new CA has to get its key signed by just one existing CA (and sign the reverse certificate)
2. A chain of forward/reverse certificates exist from any CA to any CA
3. Avoids having every CA sign the public key of every other CA.