KEY DISTRIBUTION
The public key setting

Alice

\[ M \leftarrow D_{sk[A]}(C) \]

Bob \textsuperscript{pk[A]}

\[ C \leftarrow E_{pk[A]}(M) \]

\[ \sigma \leftarrow S_{sk[A]}(M) \]

\[ M, \sigma \rightarrow V_{pk[A]}(M, \sigma) \]

Bob can:

- send encrypted data to Alice
- verify her signatures

as long as he has Alice’s public key \( pk[A] \).

But how does he get \( pk[A] \)?
Distributing public keys

How about:

\[ (pk[A], sk[A]) \overset{\$}{\leftarrow} K \]

Alice \hspace{1cm} Bob

\[ M \leftarrow D_{sk[A]}(C) \]

\[ C \leftarrow \overset{\$}{E_{pk[A]}}(M) \]

\[ \sigma \leftarrow S_{sk[A]}(M) \]

\[ \overset{\$}{V_{pk[A]}(M, \sigma)} \]
Man-in-the-middle attack

Messages that Bob encrypts to Alice are obtained by $E$, and $E$ can forge Alice’s signatures.
Bob needs an authentic copy of Alice’s public key.

The PKI (Public Key Infrastructure) is responsible for ensuring this.

Usually it is done via certificates.
Certificate Process

- Alice generates $pk$ and sends it to CA
- CA does identity check
- Alice proves knowledge of secret key to CA
- CA issues certificate to Alice
- Alice sends certificate to Bob
- Bob verifies certificate and extracts Alice’s $pk$
Generate key and send to CA

Key generation: Alice generates her keys locally via $(pk, sk) \leftarrow \mathcal{K}$

Send to CA: Alice sends $(Alice, pk)$ to a certificate authority (CA).
Upon receiving \((Alice, pk)\) the CA performs some checks to ensure \(pk\) is really Alice’s key:

- Call Alice by phone
- Check documents

These checks are out-of-band.
Proof of knowledge

The CA might have Alice sign or decrypt something under $pk$ to ensure that Alice knows the corresponding secret key $sk$.

This ensures Alice has not copied someone else’s key.
Certificate Issuance

Once CA is convinced that $pk$ belongs to Alice it forms a certificate

$$CERT_A = (CERTDATA, \sigma),$$

where $\sigma$ is the CA’s signature on $CERTDATA$, computed under the CA’s secret key $sk[CA]$.

$CERTDATA$:

- $pk$, $ID$ (Alice)
- Name of CA
- Expiry date of certificate
- Restrictions
- Security level
- ...

The certificate $CERT_A$ is returned to Alice.
Certificate usage

Alice can send $\text{CERT}_A$ to Bob who will:

- $(\text{CERTDATA}, \sigma) \leftarrow \text{CERT}_A$
- Check $\nu_{pk[CA]}(\text{CERTDATA}, \sigma) = 1$ where $pk[CA]$ is CA’s public key
- $(pk, Alice, expiry, \ldots) \leftarrow \text{CERTDATA}$
- Check certificate has not expired
- ...

If all is well we are ready for usage.
How does Bob get $pk[CA]$?

CA public keys are embedded in software such as your browser.
Certificate hierarchies

\[
\begin{align*}
\text{CA(USA)} & \rightarrow \text{CA(Calif)} \rightarrow \text{CA(SD)} \rightarrow \text{CA(UCSD)} \rightarrow \text{Mihir} \\
\text{CERT}_\text{Mihir} &= \text{CERT[CA(USA) : CA( Calif) ]} \\
&= \text{CERT[CA( Calif) : CA(SD) ]} \\
&= \text{CERT[CA(SD) : CA(UCSD) ]} \\
&= \text{CERT[CA(UCSD) : Mihir]} \\
\text{CERT}[X : Y] &= (pk[Y], Y, \ldots, S_{sk[X]}(pk[Y], Y, \ldots))
\end{align*}
\]

To verify \(\text{CERT}_\text{Mihir}\) you need only \(pk_{\text{CA[USA]}}\).
Why certificate hierarchies?

- It is easier for CA(UCSD) to check Mihir’s identity (and issue a certificate) than for CA(USA) since Mihir is on UCSD’s payroll and UCSD already has a lot of information about him.
- Spreads the identity-check and certification job to reduce work for individual CAs
- Browsers need to have fewer embedded public keys. (Only root CA public keys needed.)
Suppose Alice wishes to revoke her certificate $CERT_A$, perhaps because her secret key was compromised.

- Alice sends $CERT_A$ and revocation request to CA
- CA checks that request comes from Alice
- CA marks $CERT_A$ as revoked
Certificate revocation lists (CRLs)

CA maintains a CRL with entries of form

\[(CERT, \text{Revocation date})\]

This list is disseminated.

Before Bob trusts Alice’s certificate he should ensure it is not on the CRL.
Revocation Issues

- November 22: Alice’s secret key compromised
- November 24: Alice’s $CERT_A$ revoked
- November 25: Bob sees CRL

In the period Nov. 22-25, $CERT_A$ might be used and Bob might be accepting as authentic signatures that are really the adversary’s. Also Bob might be encrypting data for Alice which the adversary can decrypt.
The On-line Certificate Status Protocol (OCSP) enables on-line checks of whether or not a certificate has been revoked.

Bob \rightarrow \text{CERT}_A \quad \text{CA} \quad \rightarrow \text{CERT}_A

ok / not

But on-line verification kind of defeats the purpose of public-key cryptography!
Revocation in practice

- VeriSign estimates that 20% of certificates are revoked
- In practice, CRLs are huge

Revocation is a big problem and one of the things that is holding up widespread deployment of a PKI and use of public-key cryptography.
In PGP, there are no CAs. You get Alice’s public key from Carol and decide to what extent you want to trust it based on your feelings about Carol. Requires user involvement.
## Certificate Examples

![Certificate Management Interface]

### Certificate Intended Purposes

<table>
<thead>
<tr>
<th>Issued To</th>
<th>Issued By</th>
<th>Expiration</th>
<th>Friendly Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DigiCert Global CA</td>
<td>Entrust.net Secure Server</td>
<td>7/14/2014</td>
<td>&lt;None&gt;</td>
</tr>
<tr>
<td>GlobalSign Root CA</td>
<td>Root SGC Authority</td>
<td>1/27/2014</td>
<td>&lt;None&gt;</td>
</tr>
<tr>
<td>GTE CyberTrust Root</td>
<td>Root SGC Authority</td>
<td>2/22/2006</td>
<td>&lt;None&gt;</td>
</tr>
<tr>
<td>Microsoft Windows ...</td>
<td>Microsoft Root Authority</td>
<td>12/30/2002</td>
<td>&lt;None&gt;</td>
</tr>
<tr>
<td>Microsoft Windows ...</td>
<td>Microsoft Root Authority</td>
<td>12/30/2002</td>
<td>&lt;None&gt;</td>
</tr>
<tr>
<td>MS SGC Authority</td>
<td>Root SGC Authority</td>
<td>12/31/2009</td>
<td>&lt;None&gt;</td>
</tr>
<tr>
<td>Root Agency</td>
<td>Root Agency</td>
<td>12/31/2039</td>
<td>&lt;None&gt;</td>
</tr>
<tr>
<td>SecureNet CA SGC ...</td>
<td>Root SGC Authority</td>
<td>10/15/2009</td>
<td>&lt;None&gt;</td>
</tr>
<tr>
<td>Thawte Premium Server</td>
<td>Root SGC Authority</td>
<td>7/16/2004</td>
<td>&lt;None&gt;</td>
</tr>
</tbody>
</table>
Certificate Examples

[Image of a certificate management software interface showing various certificates with details like Issued To, Issued By, Expiration Date, Friendly Name, and options to Import, Export, Remove, and View.]
Certificate Examples

Certificate Information

This certificate is intended for the following purpose(s):
- Protects e-mail messages
- Proves your identity to a remote computer
- 2.16.840.1.113733.1.7.1.1

* Refer to the certification authority's statement for details.

**Issued to:** VeriSign Class 1 CA Individual Subscriber-Persona
Not Validated

**Issued by:** Class 1 Public Primary Certification Authority

**Valid from** 5/11/1998 to 5/12/2008

Issuer Statement
Certificate Examples
Certificate Examples

Certificate Information

This certificate is intended for the following purpose(s):

- Protects e-mail messages
- Proves your identity to a remote computer
- All issuance policies

Issued to: Class 1 Public Primary Certification Authority

Issued by: Class 1 Public Primary Certification Authority

Valid from 1/28/1996 to 8/1/2028
Alice and Bob can

- send each other encrypted data
- verify each other’s MACs

Can be preferable to public key setting because computation costs are lower.

But how do Alice and Bob get a shared key?
Diffie-Hellman Key Exchange

Let $G = \langle g \rangle$ be a cyclic group of order $m$ and assume $G, g, m$ are public quantities.

$$x \xleftarrow{\$} \mathbb{Z}_m; \quad X \leftarrow g^x$$

$$K \leftarrow Y^x$$

Alice

Bob

$$y \xleftarrow{\$} \mathbb{Z}_m; \quad Y \leftarrow g^y$$

$$K \leftarrow X^y$$

$$Y^x = (g^y)^x = g^{xy} = (g^x)^y = X^y$$

This enables Alice and Bob to agree on a common key $K$ which can subsequently be used, say to encrypt:

Alice

$$M \leftarrow D_K(C)$$

Bob

$$C \leftarrow C \xleftarrow{\$} E_K(M)$$
Security of DH Key Exchange under Passive Attack

Alice

\[ x \leftarrow Z_m; \quad X \leftarrow g^x \]

\[ K \leftarrow Y^x \]

Bob

\[ y \leftarrow Z_m; \quad Y \leftarrow g^y \]

\[ K \leftarrow X^y \]

Eavesdropping adversary gets \( X = g^x \) and \( Y = g^y \) and wants to compute \( g^{xy} \). But this is the (presumed hard) CDH problem.

Conclusion: DH key exchange is secure against passive (eavesdropping) attack.
Security of DH Key Exchange under Active Attack

Man-in-the-middle attack:

Adversary $E$ impersonates Alice so that:
- Bob thinks he shares $K$ with Alice but $E$ has $K$
- $E$ can now decrypt ciphertexts Bob intends for Alice

Conclusion: DH key exchange is insecure against active attack
When is key agreement possible?

In the presence of an active adversary, it is impossible for Alice and Bob to

- start from scratch, and
- exchange messages to get a common key unknown to the adversary

Why? Because there is no way for Bob to distinguish Alice from the adversary.

Alice and Bob need some a priori “information advantage” over the adversary. This typically takes the form of long-lived keys.
Settings and long-lived keys

- Public key setting: $A$ has $pk_B$ and $B$ has $pk_A$
- Symmetric setting: $A, B$ share a key $K$
- Three party setting: $A, B$ each share a key with a trusted server $S$.

These keys constitute the long-lived information.
In practice, Alice and Bob will engage in multiple communication “sessions.” For each, they

• First use a session-key distribution protocol, based on their long-lived keys, to get a fresh, authentic session key;
• Then encrypt or authenticate data under this session key for the duration of the session
Session key distribution

- Hundreds of protocols
- Dozens of security requirements
- Lots of broken protocols
- Protocols easy to specify and hard to get right
- Used ubiquitously: SSL, TLS, SSH, ...
Why session keys?

- In public-key setting, efficient cryptography compared to direct use of long-lived keys
- Security attributes, in particular enabling different applications to use keys in different ways and not compromise security of other applications
A party may concurrently be engaged in many different communication sessions.

The requirement that one session’s usage of its session key not compromise another is captured by asking that even exposure of a session key from one session should not compromise session keys of other sessions.
Three party setting

- $S$ is a trusted authentication server
- $A$ shares a key $K[A]$ with $S$
- $B$ shares a key $K[B]$ with $S$
- At any time, $A$, $B$, $S$ can engage in a 3-party protocol to provide $A$, $B$ a (shared) session key.

Model of the Kerberos system.
Notation and conventions in this area

- \( \{X\}_K \) denotes an encryption of \( X \) under key \( K \)
- \( N_A \) denotes a “nonce” chosen by party \( A \)

A nonce is a non-repeating quantity such as a counter or a value drawn at random from a large domain.
Needham-Schroeder (NS) 78 Protocol

\[ A \rightarrow S : A, B, N_A \]
\[ S \rightarrow A : \{ N_A, B, \alpha, \{ \alpha, A \}_K[B] \}_K[A] \]
\[ A \rightarrow B : \{ \alpha, A \}_K[B] \]
\[ B \rightarrow A : \{ N_B \}_\alpha \]
\[ A \rightarrow B : \{ N_B - 1 \}_\alpha \]

Session key \( \alpha \) is chosen by \( S \). Last two flows are for “key-confirmation.”

When \( A \) receives second flow it checks that \( N_A, B \) are correct. When \( B \) receives last flow it checks that the decryption is \( N_B - 1 \).
A → S : A, B, NA
S → A : \{NA, B, \alpha, \{\alpha, A\}K[B]\}K[A]
A → B : \{\alpha, A\}K[B]
B → A : \{NB\}_\alpha
A → B : \{NB - 1\}_\alpha

Assume E obtains all the flows from the execution depicted above and also learns \alpha.

What E does:
E → B : \{\alpha, A\}K[B]
Known-key attack [DS81]

\[
\begin{align*}
A \to S &: A, B, N_A \\
S \to A &: \{N_A, B, \alpha, \{\alpha, A\}_K[B]\}_K[A] \\
A \to B &: \{\alpha, A\}_K[B] \\
B \to A &: \{N_B\}_\alpha \\
A \to B &: \{N_B - 1\}_\alpha
\end{align*}
\]

Assume \( E \) obtains all the flows from the execution depicted above and also learns \( \alpha \).

What \( E \) does:
\[
\begin{align*}
E \to B &: \{\alpha, A\}_K[B] \\
B \to A
\end{align*}
\]
Known-key attack [DS81]

\[ A \rightarrow S : A, B, N_A \]
\[ S \rightarrow A : \{ N_A, B, \alpha, \{ \alpha, A \}_K[B] \}_K[A] \]
\[ A \rightarrow B : \{ \alpha, A \}_K[B] \]
\[ B \rightarrow A : \{ N_B \}_\alpha \]
\[ A \rightarrow B : \{ N_B - 1 \}_\alpha \]

Assume \( E \) obtains all the flows from the execution depicted above and also learns \( \alpha \).

What \( E \) does:
\[ E \rightarrow B : \{ \alpha, A \}_K[B] \]
\[ B \rightarrow E \]
Known-key attack [DS81]

\[A \rightarrow S : A, B, N_A\]
\[S \rightarrow A : \{N_A, B, \alpha, \{\alpha, A\}_K[B]\}_K[A]\]
\[A \rightarrow B : \{\alpha, A\}_K[B]\]
\[B \rightarrow A : \{N_B\}_\alpha\]
\[A \rightarrow B : \{N_B - 1\}_\alpha\]

Assume \(E\) obtains all the flows from the execution depicted above and also learns \(\alpha\).

What \(E\) does:
\[E \rightarrow B : \{\alpha, A\}_K[B]\]
\[B \rightarrow E : \{N'_B\}_\alpha\]
Known-key attack [DS81]

\[ A \rightarrow S : A, B, N_A \]
\[ S \rightarrow A : \{N_A, B, \alpha, \{\alpha, A\}_{K[B]}\}_{K[A]} \]
\[ A \rightarrow B : \{\alpha, A\}_{K[B]} \]
\[ B \rightarrow A : \{N_B\}_\alpha \]
\[ A \rightarrow B : \{N_B - 1\}_\alpha \]

Assume \( E \) obtains all the flows from the execution depicted above and also learns \( \alpha \).

What \( E \) does:
\[ E \rightarrow B : \{\alpha, A\}_{K[B]} \]
\[ B \rightarrow E : \{N'_B\}_\alpha \]
\[ E \rightarrow B : \{N'_B - 1\}_\alpha \]

Now \( B \) thinks it has a fresh session with \( A \) with key \( \alpha \). But \( E \) knows \( \alpha \) and any use of it by \( B \) is insecure.
$T$ will denote a time stamp.

When a party receives a flow with some $T$, it rejects unless $T$ is "current."

Inclusion of a time-stamp thus helps prevent replay.

“Current” means $T$ is “close” to local time. There will always be some chance of successful replay due to this, but for our purposes assume time-stamping is perfect and replay is impossible.
A → S : A, B
S → A : \{ T, \alpha, B, \{ T, \alpha, A \} K[B] \} K[A]

A → B : t_B, \{ A, T \} \alpha
B → A : \{ T + 1 \} \alpha

Session key \alpha and time-stamp T are selected by S. When B receives third flow it rejects unless time stamps in two parts match.
Known-key attack?

\[ A \rightarrow S : A, B \]
\[ S \rightarrow A : \{ T, \alpha, B, \{ T, \alpha, A \} \}_{K[B]} \}_{K[A]} \]
\[ A \rightarrow B : t_B, \{ A, T \}_\alpha \]
\[ B \rightarrow A : \{ T + 1 \}_\alpha \]

Assume \( E \) obtains all of the above flows and also learns \( \alpha \).

- \( E \rightarrow B : t_B, \{ A, T \}_\alpha \)
Known-key attack?

\[ A \rightarrow S : A, B \]
\[ S \rightarrow A : \{T, \alpha, B, \{T, \alpha, A\}_{K[B]}\}_{K[A]} \]
\[ A \rightarrow B : t_B, \{A, T\}_\alpha \]
\[ B \rightarrow A : \{T + 1\}_\alpha \]

Assume \( E \) obtains all of the above flows and also learns \( \alpha \).

- \( E \rightarrow B : t_B, \{A, T\}_\alpha \); \( B \) will reject because \( T \) is not current
Known-key attack?

\[\begin{align*}
A &\rightarrow S : A, B \\
S &\rightarrow A : \{T, \alpha, B, \{T, \alpha, A\}_K[\beta]\}_K[\alpha] \\
A &\rightarrow B : t_B, \{A, T\}_\alpha \\
B &\rightarrow A : \{T + 1\}_\alpha
\end{align*}\]

Assume \(E\) obtains all of the above flows and also learns \(\alpha\).

- \(E \rightarrow B : t_B, \{A, T\}_\alpha\); \(B\) will reject because \(T\) is not current
- \(E \rightarrow B : t_B, \{A, T'\}_\alpha\)
Known-key attack?

\[ A \rightarrow S : A, B \]
\[ S \rightarrow A : \{ T, \alpha, B, \{ T, \alpha, A \}_{K[B]} \}_{K[A]} \]
\[ A \rightarrow B : t_B, \{ A, T \}_\alpha \]
\[ B \rightarrow A : \{ T + 1 \}_\alpha \]

Assume \( E \) obtains all of the above flows and also learns \( \alpha \).

- \( E \rightarrow B : t_B, \{ A, T \}_\alpha \); \( B \) will reject because \( T \) is not current
- \( E \rightarrow B : t_B, \{ A, T' \}_\alpha \); \( B \) will reject because \( T' \) does not match time-stamp \( T \) in \( t_B \)
Known-key attack?

\[
\begin{align*}
  A &\rightarrow S : A, B \\
  S &\rightarrow A : \{ T, \alpha, B, \{ T, \alpha, A \}_K[B] \}_K[A] \\
  A &\rightarrow B : t_B, \{ A, T \}_\alpha \\
  B &\rightarrow A : \{ T + 1 \}_\alpha
\end{align*}
\]

Assume \( E \) obtains all of the above flows and also learns \( \alpha \).

- \( E \rightarrow B : t_B, \{ A, T \}_\alpha ; B \) will reject because \( T \) is not current
- \( E \rightarrow B : t_B, \{ A, T' \}_\alpha ; B \) will reject because \( T' \) does not match time-stamp \( T \) in \( t_B \)
- \( E \rightarrow B : t'_B = \{ T', \alpha, A \}_K[B], \{ A, T' \}_\alpha \)
Known-key attack?

\[ A \rightarrow S : A, B \]
\[ S \rightarrow A : \{ T, \alpha, B, \{ T, \alpha, A \}_K[B] \}_K[A] \]
\[ A \rightarrow B : t_B, \{ A, T \}_\alpha \]
\[ B \rightarrow A : \{ T + 1 \}_\alpha \]

Assume \( E \) obtains all of the above flows and also learns \( \alpha \).

- \( E \rightarrow B : t_B, \{ A, T \}_\alpha \); \( B \) will reject because \( T \) is not current
- \( E \rightarrow B : t_B, \{ A, T' \}_\alpha \); \( B \) will reject because \( T' \) does not match time-stamp \( T \) in \( t_B \)
- \( E \rightarrow B : t'_B = \{ T', \alpha, A \}_K[B], \{ A, T' \}_\alpha \); \( E \) can't create \( t'_B \) because it doesn't have \( K[B] \)
How should we implement \( \{ X \}_K \)?

\[
S \rightarrow A : \{ T, \alpha, B, \{ T, \alpha, A \}_K[B] \}_K[A]
\]

The protocols refer to \( \{ X \}_K \) as encryption of \( X \) under \( K \). How would you implement it?

• Question: What information in above flow needs to be kept private?
  • Answer: \( \alpha \) only (\( T, B, A \) are known!)

• Question: Then why are \( T, B, A \) encrypted?
  • Answer: For integrity

• Question: So how should we implement \( \{ X \}_K \)?
  • Answer: AEAD
How should we implement $\{X\}_K$?

$$S \rightarrow A : \{T, \alpha, B, \{T, \alpha, A\}_K\}_K$$

The protocols refer to $\{X\}_K$ as encryption of $X$ under $K$. How would you implement it?

- Question: What information in above flow needs to be kept private?

  - Answer: $\alpha$ only ($T, B, A$ are known!)

- Question: Then why are $T, B, A$ encrypted?

  - Answer: For integrity

- Question: So how should we implement $\{X\}_K$?

  - Answer: AEAD
How should we implement $\{X\}_K$?

$$S \rightarrow A : \{T, \alpha, B, \{T, \alpha, A\}_K[B]\}_K[A]$$

The protocols refer to $\{X\}_K$ as encryption of $X$ under $K$. How would you implement it?

- Question: What information in above flow needs to be kept private?
  - Answer: $\alpha$ only ($T, B, A$ are known!)

- Question: Then why are $T, B, A$ encrypted?
  - Answer: For integrity

- Question: So how should we implement $\{X\}_K$?
  - Answer: AEAD
How should we implement $\{X\}_K$?

$$S \rightarrow A : \{ T, \alpha, B, \{ T, \alpha, A \}_K[B] \}_K[A]$$

The protocols refer to $\{X\}_K$ as encryption of $X$ under $K$. How would you implement it?

- Question: What information in above flow needs to be kept private?
- Answer: $\alpha$ only ($T, B, A$ are known!)
- Question: Then why are $T, B, A$ encrypted?

$\text{Answer: For integrity}$
How should we implement $\{X\}_K$?

$S \rightarrow A : \{T, \alpha, B, \{T, \alpha, A\}_K[B]\}_K[A]$  

The protocols refer to $\{X\}_K$ as encryption of $X$ under $K$. How would you implement it?

- Question: What information in above flow needs to be kept private?
  - Answer: $\alpha$ only ($T, B, A$ are known!)

- Question: Then why are $T, B, A$ encrypted?
  - Answer: For integrity
How should we implement $\{X\}_K$?

\[ S \to A : \{ T, \alpha, B, \{ T, \alpha, A \}_K[B] \}_K[A] \]

The protocols refer to $\{X\}_K$ as encryption of $X$ under $K$. How would you implement it?

- Question: What information in above flow needs to be kept private?
- Answer: $\alpha$ only ($T, B, A$ are known!)
- Question: Then why are $T, B, A$ encrypted?
- Answer: For integrity
- Question: So how should we implement $\{X\}_K$?
How should we implement \( \{X\}_K \)?

\[ S \rightarrow A : \{ T, \alpha, B, \{ T, \alpha, A \}_K[B] \}_K[A] \]

The protocols refer to \( \{X\}_K \) as encryption of \( X \) under \( K \). How would you implement it?

- **Question:** What information in above flow needs to be kept private?
  - **Answer:** \( \alpha \) only (\( T, B, A \) are known!)
- **Question:** Then why are \( T, B, A \) encrypted?
  - **Answer:** For integrity
- **Question:** So how should we implement \( \{X\}_K \)?
  - **Answer:** AEAD
Recall AEAD

Provides:
- Privacy and integrity of the message $M$
- Integrity of the associated data $AD$

Sender
- $C \leftarrow \mathcal{E}_K(N, AD, M)$
- Send $(N, AD, C)$

Receiver
- Receive $(N, AD, C)$
- $M \leftarrow \mathcal{D}_K(N, AD, C)$

Sender must never re-use a nonce.
Implementing \( \{X\}_K \) using AEAD

\[
S \rightarrow A : \{T, \alpha, B, \{T, \alpha, A\}_K[B]\}_K[A]
\]

can be implemented as

\[
S \rightarrow A : N, N', T, C_B, C_A
\]

where

- \( C_B \leftarrow \mathcal{E}_{K[B]}(N, T || A, \alpha) \)
- \( C_A \leftarrow \mathcal{E}_{K[A]}(N', T || B || C_B, \alpha) \)
S → A : \{ T, \alpha, B, \{ T, \alpha, A \} \_K[B] \} \_K[A]

can be implemented as

S → A : T, C_A, C_B, \tau_B, C_A, \tau_A

where we use an IND-CPA encryption scheme and a UF-CMA MAC to compute these quantities as follows:

- \( C_A \leftarrow \mathcal{E}_{K[A]}(\alpha) \)
- \( C_B \leftarrow \mathcal{E}_{K[B]}(\alpha) \)
- \( \tau_A \leftarrow \text{MAC}_{K[A]}(T, B, C_A) \)
- \( \tau_B \leftarrow \text{MAC}_{K[B]}(T, A, C_B) \)

Encryption and MAC should use separate keys!
Key Confirmation

\[ A \rightarrow B : t_B, \{A, T\}_\alpha \]
\[ B \rightarrow A : \{T + 1\}_\alpha \]

What is required here?
Key Confirmation

\[ A \rightarrow B : t_B, \{ A, T \}_\alpha \]
\[ B \rightarrow A : \{ T + 1 \}_\alpha \]

What is required here? Seems to be integrity so we might implement as:

\[ A \rightarrow B : t_B, \text{MAC}_\alpha(A, T) \]
\[ B \rightarrow A : \text{MAC}_\alpha(T + 1) \]
Question: What is desired security attribute of session key?
Question: What is desired security attribute of session key?

Answer: It should be indistinguishable from random to adversary

At end of protocol:

$$b \leftarrow \{0, 1\}; \alpha_0 \leftarrow \alpha; \alpha_1 \leftarrow \{0, 1\}^{|\alpha|}$$

$\alpha_b \rightarrow A \rightarrow b$?
Session key security under key confirmation

\[ A \rightarrow B : t_B, \{ A, T \}_\alpha \]

\[ B \rightarrow A : \{ T + 1 \}_\alpha \]

Session key is not indistinguishable from random. Adversary given challenge \( \alpha_b \) can decrypt \( C \) under \( \alpha_b \) and check whether it gets back \( A, T \). Or, if a MAC, can re-compute MAC and check.

Key confirmation destroys session key security and is unnecessary anyway!
BR95 Protocol

\[
\begin{align*}
A & \rightarrow B : R_A \\
B & \rightarrow S : R_A, R_B \\
S & \rightarrow A : C_A, \text{MAC}_{K[A]}(A, B, R_A, C_A) \\
S & \rightarrow B : C_B, \text{MAC}_{K[B]}(A, B, R_B, C_B)
\end{align*}
\]

where \( C_A \leftarrow E_{K[A]}(\alpha); \ C_B \leftarrow E_{K[B]}(\alpha) \)

NO key confirmation: \( \alpha \) never used!

This protocol can be proven to satisfy a strong, formal notion of session key distribution security assuming standard properties of \( E, \text{MAC} \) [BR95].
Session key exchange in public key setting

Most important type of session key exchange in practice, used in all communication security protocols: SSL, SSH, TLS, IPSEC, 802.11, ...
Protocol KE1

\[ A^{pk[B]} \]

\[ \frac{A, R_A}{R_B, C, B, \text{Sign}_B(R_A, R_B, C)} \]

\[ \frac{A, \text{Sign}_A(R_A, R_B)}{} \]

\[ B^{pk[A]} \]

\[ C \leftarrow E_A(K) \]

- Session key \( K \) chosen by \( B \)
- \( \text{Sign}_P(M) \) is \( P \)'s signature of \( M \), created under \( sk[P] \) and verifiable given \( pk[P] \).
- \( R_A, R_B \) are random nonces
- \( E_A(\cdot) \) is encryption under \( A \)'s public key \( pk[A] \), decryptable by \( A \) using \( sk[A] \)
A thinks it shares $K$ with $B$, but $B$ records $K$ as a key shared with $E$. This is generally acknowledged to be a problem even though $E$ does not know $K$.

A good example of exactly why this is a problem is, however, lacking.
Protocol KE2

\[ A^{pk[B]} \]

\[ A, R_A \]

\[ R_B, C, B, \text{Sign}_B(A, B, R_A, R_B, C) \]

\[ A, \text{Sign}_A(A, B, R_A, R_B) \]

\[ B^{pk[A]} \]

\[ C \leftarrow^s \mathcal{E}_A(K) \]

Identities are included in scopes of signatures, thwarting the binding attack. Protocol KE2 can be shown to meet a strong, formal notion of secure session key exchange.
Forward secrecy

Can we achieve forward secrecy: Privacy of communication done prior to exposure of $sk[A]$ is not compromised?
Session key is $K = H(A, B, g^a, g^b, g^{ab})$.

Adversary $E$ records above flows on Nov. 20. On Dec. 18, $sk[A]$ is exposed. This allows $E$ to forge $A$’s signatures, but $A$ can address this by revoking $pk[A]$. But $sk[A]$ does not help $E$ obtain $K$.

There is no public-key encryption here, only signatures.

All standard protocols use DH to get forward security.
Anonymity

The requirement here is that the protocol flows do not allow the adversary to identify the participants.

This might be desirable when $B$ is a mobile client, communicating with base station $A$; $B$ does not want her location known to $E$.

The protocols we have seen so far send the identities in the clear

\[ \begin{array}{c}
A \\
\rightarrow \\
A, g^a \\
\leftarrow \\
B, \ldots
\end{array} \]

Such protocols do not provide anonymity.
KE4: Targeting anonymity for $B$

\[
\begin{align*}
A & \xrightarrow{A, g^a} B \\
& \quad \sigma \\
& \quad g^b, \{B\}_{K_e}, \text{Sign}_B(A, B, g^a, g^b) \\
& \quad \text{Sign}_A(A, B, g^a, g^b)
\end{align*}
\]

where $K_e = H(1, A, B, g^a, g^b, g^{ab})$ and the session key is $K = H(0, A, B, g^a, g^b, g^{ab})$. 


KE4: Targeting anonymity for $B$

But if $B \in \{B_1, \ldots, B_n\}$ then $E$ can identify $B$ via:

For $i = 1, \ldots, n$ do:
   if $\mathcal{V}_{pk[B_i]}((A, B_i, g^a, g^b), \sigma) = 1$ then return $B_i$

Signatures reveal identity!

Question: so why don’t we send $g^b$ encrypted too?
KE4: Targeting anonymity for $B$

But if $B \in \{B_1, \ldots, B_n\}$ then $E$ can identify $B$ via:

For $i = 1, \ldots, n$ do:
  if $\mathcal{V}_{pk[B_i]}((A, B_i, g^a, g^b), \sigma) = 1$ then return $B_i$

Signatures reveal identity!

Question: so why don’t we send $g^b$ encrypted too?
Answer: How would $A$ decrypt?
KE5: Anonymity for $B$

$$A, g^a$$

$$g^b, \{B, \text{Sign}_B(A, B, g^a, g^b)\}_{K_e}$$

$$\{\text{Sign}_A(A, B, g^a, g^b)\}_{K_e}$$

where $K_e = H(1, A, B, g^a, g^b, g^{ab})$ and the session key is $K = H(0, A, B, g^a, g^b, g^{ab})$
A password is a human-memorizable key.

Attackers are capable of forming a set $D$ of possible passwords called a dictionary such that

- If the target password $pw$ is in $D$ and
- The attacker knows $\overline{pw} = f(pw)$, the image of $pw$ under some public function $f$.

then the target password can be found via

$$\text{for all } pw' \in D \text{ do}\quad \text{if } f(pw') = \overline{pw} \text{ then return } pw'$$

This is called a dictionary attack.
Fact is that in spite of all the great crypto around, a significant fraction of our security today resides in passwords: bank ATM passwords; login passwords; passwords for different websites; ...

Few of us today have cryptographic keys; but we all have more passwords than we can remember!

Passwords are convenient and entrenched.

Preventing dictionary attacks is an important concern.
Preventing dictionary attacks: Password selection

Systems try to force users to select “good” passwords, meaning ones not in the dictionary. But studies show that a significant fraction of user passwords end up being in the dictionary anyway.

Attackers get better and better at building dictionaries.

Good password selection helps, but it is unrealistic to think that even the bulk of passwords are well selected, meaning not in the dictionary.
An alternative approach is to ensure that usage of a password $pw$ never reveals an image $\overline{pw} = f(pw)$ of $pw$ under a public function $f$. Then, even if the password is in the dictionary, the dictionary attack cannot be mounted.
Password-based session-key exchange

A, B share a password $pw$.

They want to interact to get a common session key.

The protocol should resist dictionary attack: adversary should be unable to obtain an image of $pw$ under a public function.
Protocol KE6

\[
\begin{array}{c}
A^{pw} \\
A, g^a \\
\sigma \\
\underline{B, g^b, MAC_{pw}(1, A, B, g^a, g^b)} \\
A, MAC_{pw}(0, A, B, g^a, g^b) \\
B^{pw}
\end{array}
\]

Session key is \( K = H(A, B, g^a, g^b, g^{ab}). \)

Dictionary attack is possible: Let \( f \) be defined by

\[
f(x) = MAC_x(1, A, B, g^a, g^b)
\]

Then get \( pw \) via

for all \( pw' \in D \) do

if \( f(pw') = \sigma \) then return \( pw' \)
Protocol KE7

\[ A^{pw} \]

\[ \begin{array}{c}
A, g^a \\
B, g^b, \text{MAC}_{K_m}(1, A, B, g^a, g^b) \\
A, \text{MAC}_{K_m}(0, A, B, g^a, g^b)
\end{array} \]

\[ B^{pw} \]

where \( K_m = H(1, A, B, g^a, g^b, g^{ab}, pw) \) and the session key is \( K = H(0, A, B, g^a, g^b, g^{ab}) \).

Does protocol transcript reveal \( f(pw) \) for some public \( f \)? Defining

\[ f(x) = \text{MAC}_{H(1, A, B, g^a, g^b, g^{ab}, x)}(1, A, B, g^a, g^b) \]

is the natural idea.
Protocol KE7

\[ A^{pw} \]

\[ A, g^a \]

\[ B, g^b, MAC_{K_m}(1, A, B, g^a, g^b) \]

\[ B^{pw} \]

\[ A, MAC_{K_m}(0, A, B, g^a, g^b) \]

where \( K_m = H(1, A, B, g^a, g^b, g^{ab}, pw) \) and the session key is \( K = H(0, A, B, g^a, g^b, g^{ab}) \).

Does protocol transcript reveal \( f(pw) \) for some public \( f \)? Defining

\[ f(x) = MAC_{H(1, A, B, g^a, g^b, g^{ab}, x)}(1, A, B, g^a, g^b) \]

is the natural idea but \( f \) is not public because \( E \) cannot compute \( g^{ab} \)!

Dictionary attack does not seem possible ... at least under a passive attack.
Active attack on KE7

\[ E \xrightarrow{A, g^a} B, g^b, \text{MAC}_{K_m}(1, A, B, g^a, g^b) \xrightarrow{\sigma} B^{pw} \]

where \( K_m = H(1, A, B, g^a, g^b, g^{ab}, pw) \). But now \( E \) has \( a \) and can compute \( g^{ab} = (g^b)^a \) so

\[ f(x) = \text{MAC}_{H(1, A, B, g^a, g^b, g^{ab}, x)}(1, A, B, g^a, g^b) \]

becomes public and a dictionary attack is possible.
Security goal

We cannot prevent $E$ from eliminating one candidate password per interaction with $A$ or $B$ in an active attack.

Our goals are

- A protocol transcript should not reveal the image of $pw$ under a public function.
- An interaction with $A$ or $B$ should not allow $E$ to eliminate more than a small number $d$ (ideally $d = 1$) of candidate passwords.
Protocol KE8: EKE2 [BPR00]

\[ A^{pw} \]

\[ A, E_{pw}(g^a) \]

\[ B, E_{pw}(g^b), H(1, A, B, g^a, g^b, g^{ab}) \]

\[ A, H(2, A, B, g^a, g^b, g^{ab}) \]

\[ B^{pw} \]

\[ E : PW \times G \rightarrow G \] is a block cipher over group G and keyspace PW of all possible passwords; the session key is \( K = H(0, A, B, g^a, g^b, g^{ab}) \).

This prevents the previous active attack because the adversary cannot compute \( E_{pw}(g^a) \) while knowing a.

This protocol has a proof [BPR00].