KEY DISTRIBUTION:
PKI and SESSION-KEY EXCHANGE
The public key setting

$B$’s secret key is $sk[B]$ and its associated public key is $pk[B]$. The public key setting **assumes** $A$ is in possession of $pk[B]$.

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

$$C \leftarrow^S E_{pk[B]}(M) \quad \quad C \rightarrow M \leftarrow D_{sk[B]}(C)$$

$$V_{pk[B]}(M, \sigma) \leftarrow M, \sigma \quad \quad \sigma \leftarrow^S S_{sk[B]}(M)$$

Now $A$ can encrypt a message $M$ under $pk[B]$ to get a ciphertext $C$ that $B$ can decrypt using $sk[B]$.

$B$ can sign a message $M$ using $sk[B]$ to get signature $\sigma$ that $A$ can verify using $pk[B]$.

But how does $A$ get $pk[B]$?
But who exactly are “A” and “B”? 

Typically, as in most uses of TLS, B is a server. Its identity B is an associated domain name or ip address, for example $B = \text{google.com}$. A is a client, also with an associated ip address.
How does $A$ get $B$’s public key?

**How about:** $B$ runs a prescribed key-generation algorithm $\mathcal{K}$ to generate $(pk[B], sk[B])$. It sends $(B, pk[B])$ to $A$.

$$
\begin{align*}
A & \quad B \\
& \quad \underline{B, pk[B]} \\
& \quad (pk[B], sk[B]) \overset{\$}{\leftarrow} \mathcal{K}
\end{align*}
$$
Entity-in-the-middle attack

Adversary $E$ can decrypt ciphertexts intended for $B$ and can forge $B$’s signatures. Adversary effectively becomes $B$. 
Goal: A gets an **authentic** copy of B’s public key, meaning if \( pk \) claims to come from \( B \), then \( A \) has a proof to that effect.

Popular Solution: The PKI (Public Key Infrastructure).

Certificate authority: Trusted entity that provides the above proof.

Certificate: The proof

Note: There are other ways to reach the goal: \( B \) could post its public key on its Facebook; post it on its personal or corporate webpage; include it as an attachment in its emails; put it on a keyserver like openpgp SKS; hand it to \( A \) in person; ...
Some other certificate authorities

<table>
<thead>
<tr>
<th>Rank</th>
<th>Issuer</th>
<th>Usage</th>
<th>Market share</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IdenTrust</td>
<td>20.4%</td>
<td>39.7%</td>
</tr>
<tr>
<td>2</td>
<td>Comodo</td>
<td>17.9%</td>
<td>34.9%</td>
</tr>
<tr>
<td>3</td>
<td>DigiCert</td>
<td>6.3%</td>
<td>12.3%</td>
</tr>
<tr>
<td>4</td>
<td>GoDaddy</td>
<td>3.7%</td>
<td>7.2%</td>
</tr>
<tr>
<td>5</td>
<td>GlobalSign</td>
<td>1.8%</td>
<td>3.5%</td>
</tr>
<tr>
<td>6</td>
<td>Certum</td>
<td>0.4%</td>
<td>0.7%</td>
</tr>
<tr>
<td>7</td>
<td>Actalis</td>
<td>0.2%</td>
<td>0.3%</td>
</tr>
<tr>
<td>8</td>
<td>Entrust</td>
<td>0.2%</td>
<td>0.3%</td>
</tr>
<tr>
<td>9</td>
<td>Secom</td>
<td>0.1%</td>
<td>0.3%</td>
</tr>
<tr>
<td>10</td>
<td>Let's Encrypt</td>
<td>0.1%</td>
<td>0.2%</td>
</tr>
<tr>
<td>11</td>
<td>Trustwave</td>
<td>0.1%</td>
<td>0.1%</td>
</tr>
<tr>
<td>12</td>
<td>WiSeKey Group</td>
<td>&lt;0.1%</td>
<td>0.1%</td>
</tr>
<tr>
<td>13</td>
<td>StartCom</td>
<td>&lt;0.1%</td>
<td>0.1%</td>
</tr>
<tr>
<td>14</td>
<td>Network Solutions</td>
<td>&lt;0.1%</td>
<td>0.1%</td>
</tr>
</tbody>
</table>
Certificate process

- $B$ generates $(pk, sk) \leftarrow \mathcal{K}$ by running a key-generation algorithm $\mathcal{K}$
- $B$ sends its identity $B$, and $pk$, to CA
- CA does identity check to ensure $pk$ is $B$’s
- $B$ proves knowledge of $sk$ to CA
- CA issues certificate to $B$
- $B$ sends certificate to $A$
- $A$ verifies certificate and extracts $B$’s public key $pk$
RSA Key generation with openssl

Generating a private RSA key

1. Generate an RSA private key, of size 2048, and output it to a file named key.pem:

```
$ openssl genrsa -out key.pem 2048
Generating RSA private key, 2048 bit long modulus
..........+++ 
..........................................................+++ 
e is 65537 (0x10001)
```

2. Extract the public key from the key pair, which can be used in a certificate:

```
$ openssl rsa -in key.pem -outform PEM -pubout -out public.pem
writing RSA key
EC Key generation with openssl

Generating a private EC key

1. Generate an EC private key, of size 256, and output it to a file named key.pem:

   $ openssl ecparam -name prime256v1 -genkey -noout -out key.pem

2. Extract the public key from the key pair, which can be used in a certificate:

   $ openssl ec -in key.pem -pubout -out public.pem
   read EC key
   writing EC key

After running these two commands you end up with two files: key.pem and public.pem. These files are referenced in various other guides on this page when dealing with key import.
Checks

$B$ sends its identity $B$ (domain name, ip address, email address, ...) and its public key $pk$ to the certificate authority (CA).

Upon receiving $(B, pk)$ the CA performs some checks to ensure $pk$ is really $B$’s key.

**Example:** If $B$ is a domain name, then the CA sends $B$ a challenge and checks that it can put it on the webpage of the domain name.

**Example:** If $B$ is an email address, then the CA sends an email to that address with a link for $B$ to click to verify that it owns the address.

**Example:** If $B$ is a passport or driver’s license, the CA may be able to verify it physically, out of band.

**Proof of knowledge of secret key:** The CA might have $B$ sign or decrypt something under $sk$ to ensure that $B$ knows $sk$. This ensures $B$ has not copied someone else’s public key.
Certificate Issuance

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

$$\text{CERT}[B] = (\text{CERTDATA}, \sigma),$$

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

- $B$’s public key $pk$, and its type (RSA, EC, ...)
- Identity $B$ of $B$
- Name of CA
- Expiry date of certificate
- ...

The certificate CERT[$B$] is returned to $B$. 
Certificate usage

$B$ can send $\text{CERT}[B]$ to $A$, who is assumed to have the CA’s public key $pk[CA]$, and now will:

- Parse $\text{CERT}[B]$ as $(\text{CERTDATA}, \sigma) \leftarrow \text{CERT}[B]$
- Check that $\nu_{pk[CA]}(\text{CERTDATA}, \sigma) = 1$
- Extract $(pk, B, \text{expiry}, \ldots) \leftarrow \text{CERTDATA}$
- Check certificate has not expired
- Check that $B$ is the desired identity
- ...

If all is well, $A$ accepts the certificate and is ready to use the public key $pk$ therein.

How does $A$ get $pk[CA]$? CA public keys are embedded in software such as your browser, or, on Apple, in the keychain.
Certificate hierarchies

\[
\text{CERT}[\text{Nadia}] = ( (pk[Y], Y, \ldots), S_{sk[X]}((pk[Y], Y, \ldots)) )
\]

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

- It is easier for CA(UCSD) to check Nadia’s identity (and issue a certificate) than for CA(USA) since Nadia is on UCSD’s payroll and UCSD already has a lot of information about her.
- 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.)
Certificates on Mac: keychain

Keychain Access

Keychains
- login
- Local Items
- System
- System Roots

Name
- AddTrust External CA Root
- auth.resnet.ucsd.edu
- auth.ucsd.edu
- auth.ucsd.edu
- com.apple.idms.appleid.pr.d.46414e6a564e2f746958484d6d4475473070582b2b513d3d
- com.apple.idms.appleid.pr.d.46414e6a564e2f746958484d6d4475473070582b2b513d3d
- InCommon RSA Server CA
- InCommon Server CA
- member: B3F2F72E-E369-43F3-97C6-B61C88991470 76EF51EC-49C1-47F6-A069-EF8FED3300FF
- ucsb-secure.wireless.ucsd.edu
- USERTrust RSA Certification Authority
- USERTrust RSA Certification Authority
- www.schlossbensberg.com

Category
- All Items
- Passwords
- Secure Notes
- My Certificates
- Keys
- Certificates
A particular certificate

auth.ucsd.edu
Issued by: InCommon RSA Server CA
Expires: Sunday, April 19, 2020 at 4:59:59 PM Pacific Daylight Time
This certificate is marked as trusted for this account

Trust
Details
Subject Name
Country or Region US
Postal Code 92093
State/Province CA
Locality La Jolla
Street Address 9500 Gilman Drive
Organization University of California, San Diego
Organizational Unit UCSD
Common Name auth.ucsd.edu

Issuer Name
Country or Region US
State/Province MI
Locality Ann Arbor
Organization Internet2
Organizational Unit InCommon
Common Name InCommon RSA Server CA
Serial Number 00 B1 2B 07 A2 0D 0B E2 27 6E A0 9C 97 47 D0 DF 87
Version 3
Signature Algorithm SHA-256 with RSA Encryption (1.2.840.113549.1.1.1)
Parameters None
Not Valid Before Thursday, April 19, 2018 at 5:00:00 PM Pacific Daylight Time
Not Valid After Sunday, April 19, 2020 at 4:59:59 PM Pacific Daylight Time
Public Key Info
Algorithm RSA Encryption (1.2.840.113549.1.1.1)
Parameters None
Public Key 256 bytes: C4 AD 44 82 D1 A1 84 0F ...
Exponent 65537
Key Size 2048 bits
Key Usage Encrypt, Verify, Wrap, Derive
Signature 256 bytes: 41 01 7D F8 D1 B0 AC E8 ...

Extension Key Usage (2.5.29.15)
Critical YES
Usage Digital Signature, Key Encipherment
Suppose $B$ wishes to revoke its certificate $\text{CERT}[B] = (\text{CERTDATA}, \sigma)$, perhaps because its secret key $sk$, corresponding to the $pk$ in CERTDATA, was compromised. Then:

1. $B$ sends $\text{CERT}[B]$ and revocation request to CA, signed under $sk$
2. CA verifies the signature under $pk$
3. CA puts $(\text{CERT}[B], \text{RevocationDate})$ on its Certificate Revocation List (CRL)
4. This list is disseminated.

Before $A$ accepts $B$’s certificate, $A$ should check that it is not on the CRL. The OCSP (Online Certificate Status Protocol) is one way to do this.
Revocation Issues

- November 22: $B$’s secret key compromised
- November 24: $B$’s $\text{CERT}[B]$ revoked
- November 25: $A$ sees CRL

$\text{CERT}[B]$ might be used in the November 22-25 range, compromising security.

In practice, CRLs are large and revocation is a problem.
Certificate transparency

Who watches the watchers?

Historically, user agents determined if CAs were trustworthy through audits by credentialled third parties. But these tended to look at operational practices and historical performance rather than technical correctness. Such audits can’t catch everything. Before CT, there could be a significant time lag between a certificate being wrongly issued, and a CA doing something about it.

That’s where Certificate Transparency comes in.

Independent, reliable logs

CT depends on independent, reliable logs because it is a distributed ecosystem. Built using Merkle trees, logs are publicly verifiable, append-only, and tamper-proof.
PGP SKS keyservers

**SKS OpenPGP Key server**

**Extract a key**
You can find a key by typing in some words that appear in the userid (name, email, etc.) of the key you're looking for, or by typing in the keyid in hex format ("0x..."

---

**Search for a public key**

<table>
<thead>
<tr>
<th>String</th>
<th>Show PGP Fingerprints</th>
<th>Show SKS full-key hashes</th>
<th>Get regular index of matching keys</th>
<th>Get verbose index of matching keys</th>
<th>Retrieve ascii- armored keys</th>
<th>Retrieve keys by full-key hash</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**Submit a key**
You can submit a key by simply pasting in the ASCII- armored version of your key and clicking on submit.

---

**SKS** is a new OpenPGP keyserver. The main innovation of SKS is that it includes a highly-efficient reconciliation algorithm for keeping the keyservers synchronized.

**SKS statistics**

Nadia Heninger

UCSD
A large part of secure communication over the Internet is through protocols like TLS (https).

Here, public-key cryptography is not used to directly secure data.

Rather, public-key cryptography is used in a session-key exchange that provides (client) $A$ and (server) $B$ with a shared (symmetric) session key $K$.

Data is then secured under $K$ using an authenticated encryption scheme $AE = (K, E, D)$:

\[
\begin{align*}
A^K & \quad & B^K \\
M & \leftarrow D_K(C) & C & \leftarrow E_K(M)
\end{align*}
\]
Why session keys, as opposed to directly securing data with public-key cryptography?

One reason is performance: symmetric cryptography is more efficient than asymmetric cryptography.

More fundamentally, it reflects the Internet architecture in which $A$ and $B$ will engage in multiple, sometimes concurrent communication sessions.

The session key exchange paradigm gives each such session a fresh session key, making its security independent of that of other sessions.
Recall Diffie-Hellman Key Exchange

Let $G = \langle g \rangle$ be a cyclic group of order $m$ in which the CDH problem is hard. Let $H: \{0, 1\}^* \to \{0, 1\}^k$ be a hash function.

\[
\begin{align*}
A & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \frac{Y^x}{L} = (g^y)^x = g^{xy} = (g^x)^y = X^y
\end{align*}
\]

This enables $A$ and $B$ to agree on the common $k$-bit key $K = H(L) = H(g^{xy})$.

So is this a suitable session key exchange protocol? Are we done?
DH Key Exchange is secure under Passive Attack

A passive adversary is one that observes the communication, acquiring $X = g^x$ and $Y = g^y$, and wants to compute $K = H(g^{xy})$. But to do so requires solving the CDH problem, which is here assumed hard.
DH Key Exchange is insecure under Active Attack

Entity-in-the-middle attack:

\[
\begin{align*}
  A & \quad \leftarrow Z_m; \quad X \rightarrow g^x \\
  L & \longrightarrow Y^x \\
  E & \quad y \leftarrow Z_m; \quad Y \rightarrow g^y \\
  B, Y & \longrightarrow X^y
\end{align*}
\]

Adversary \( E \) impersonates \( B \). \( A \) thinks it shares \( K = H(L) \) with \( B \), but in fact \( A \) shares \( K \) with \( E \).

If \( A \) now encrypts, under \( K \), a message intended for \( B \), then \( E \) can decrypt the ciphertext and recover the message.

So DH key exchange does not solve the session key exchange problem. However, we will see that it will be a useful tool in achieving forward security in session key exchange ...
Session key exchange requirements

We consider the unilateral, public-key setting. Here $B$ has a certificate $\text{CERT}[B]$ and corresponding public and secret keys $pk[B], sk[B]$. $A$ is not assumed to have a certificate or corresponding keys.

This is the most common setting for TLS, where $B$ is a server like google.com and $A$ is a client.

The session key exchange should result in a session key $K$, known to both $A$ and $B$, and satisfying:

- Authenticity: $A$ really shares $K$ with $B$, not some other entity
- Secrecy: The adversary does not know $K$.

This must hold even if the adversary knows session keys of other sessions and is active, meaning in complete control of the communication.

These basic requirements are supplemented by various others including forward secrecy, anonymity, ...
Session key exchange secrecy

Secrecy: The adversary $E$ cannot distinguish the true session key $K$ from a random string of the same length.

Suppose the protocol terminates and a party $X$ outputs a session key $K$. Now we let

$$b \leftarrow \{0, 1\}; \ K_1 \leftarrow K; \ K_0 \leftarrow \{0, 1\}^{|K|}; \ b' \leftarrow E(K_b)$$

Then the adversary's advantage $2 \Pr[b = b'] - 1$ should be small.

This must hold even if the adversary has obtained the session key of all other instances except the one partnered with $X$, and when the adversary is active, in charge of all communication.

Warning: This is not a formal definition, just a glimpse of it.
Session-key exchange is a subtle problem.

Easy to specify protocols, hard to get them right.

Many security requirements, many proposed protocols, many attacks.

Definitions and provable security treatment started with [BR93] and continued with [BCK98,BPR00,CK01,CK02,...].

Today, standards look for proof-based support.

The TLS 1.3 session key exchange protocol is based on the Sigma protocol of [Kr03].
Protocol KE1

\[
\begin{array}{c}
\text{A} \\
A, R_A \\
B, R_B, \text{CERT}[B], \text{Sig}_B(R_A \parallel R_B) \\
C \leftarrow \text{Enc}_B(L) \\
\end{array}
\quad
\begin{array}{c}
\text{B} \\
A, R_A \\
B, R_B, \text{CERT}[B], \text{Sig}_B(R_A \parallel R_B) \\
C, \text{MAC}_M(R_A \parallel R_B \parallel C) \\
\end{array}
\]

\(R_A, R_B,\) called *nonces*, are randomly chosen by the parties.

\(\text{Sig}_B(X)\) is \(B\)'s signature on \(X\), computed under \(sk[B]\) and verifiable under the \(pk[B]\) that is in \(\text{CERT}[B]\).

\(L\) is randomly chosen by \(A\). Session key is \(K = H_1(L)\) and MAC key is \(M = H_2(L)\) where \(H_1, H_2\) are public hash functions.

\(\text{Enc}_B(L)\) is encryption of \(L\) under \(B\)'s public key \(pk[B]\). Decryption uses \(sk[B]\).
Identity mis-binding attack on KE1

A accepts B and thinks it shares K with B.

But B accepts E and thinks it shares K with E.

This is viewed as a problem, even though E does not know K, because there is a mis-binding of identities.

A good definition would view this as a successful attack.

A good protocol should ensure that if A accepts B with K, then B either accepts A with K, or accepts nobody with K or a key related to K.
Identity mis-binding is circumvented by inclusion of identities in the signature and the MAC, and addition of a MAC from the server:

$$A$$

\[A, R_A\]

\[B, R_B, CERT[B], \text{Sig}_B(A\|B\|R_A\|R_B)\]

\[C \leftarrow \text{Enc}_B(L)\]

\[C, \text{MAC}_M(0\|A\|B\|R_A\|R_B\|C)\]

\[\text{MAC}_M(1\|A\|B\|R_A\|R_B)\]

Session key is \(K = H_1(A\|B\|R_A\|R_B\|L)\) and MAC key is \(M = H_2(A\|B\|R_A\|R_B\|L)\).
KE2 is not forward secure

\[ A \quad \xrightarrow{A, R_A} \quad B \]

\[ B, R_B, \text{CERT}[B], \text{Sig}_B(A \parallel B \parallel R_A \parallel R_B) \]

\[ C, \text{MAC}_M(0 \parallel A \parallel B \parallel R_A \parallel R_B \parallel C) \]

\[ \text{MAC}_M(1 \parallel A \parallel B \parallel R_A \parallel R_B) \]

\[ C_B \]

\[ C_B \xleftarrow{\$} \text{Enc}_K(X) \]

Nov. 20: Adversary \( E \) records above flows.
Dec. 18: \( E \) compromises \( B \)'s system and obtains \( sk[B] \)
Dec. 19: \( B \) revokes \( \text{CERT}[B] \), and thus \( pk[B] \)

However, at any time after Dec. 18, \( E \) can obtain session key \( K \) and decrypt \( C_B \) to obtain \( X \) via: \( K \leftarrow \text{Dec}_{sk[B]}(C) \); \( X \leftarrow \text{Dec}_K(C_B) \).

This is a violation of what's called forward secrecy.
**Forward secrecy**

*Forward secrecy* asks that exposure of $sk[B]$ does not allow recovery of session keys $K$ exchanged prior to the time of exposure.

This is achieved using the DH key exchange inside the session key exchange protocol.

Forward secrecy is considered necessary in modern session key exchange, and is present in the TLS 1.3 protocol.

Session-key exchange protocols using DH for forward secrecy are often called authenticated DH key exchange protocols.
Protocol KE3

Let \( G = \langle g \rangle \) be a cyclic group of order \( m \) in which the CDH problem is hard.

\[
\begin{array}{c}
A \\
\hline
A, g^a \\
B, g^b, \text{CERT}[B], \text{Sig}_B(A \| B \| g^a \| g^b), \text{MAC}_M(1 \| A \| B \| g^a \| g^b) \\
\hline
\text{MAC}_M(0 \| A \| B \| g^a \| g^b) \\
B
\end{array}
\]

Here \( a, b \xleftarrow{\$} Z_m \) are chosen by \( A, B \), respectively, and \( g^a, g^b \) play the role of nonces.

\( \text{Sig}_B(X) \) is \( B \)'s signature on \( X \), computed under \( sk[B] \) and verifiable under the \( pk[B] \) that is in \( \text{CERT}[B] \).

Let \( L = g^{ab} \) be the DH key. Then session key is \( K = H_1(A \| B \| g^a \| g^b \| L) \) and MAC key is \( M = H_2(A \| B \| g^a \| g^b \| L) \) where \( H_1, H_2 \) are as before.
There is no public-key encryption used here, only signatures.

Compromise of $sk[B]$ only gives $E$ the ability to forge signatures. Even given $sk[B]$, it cannot recover the DH key $L = g^{ab}$ from a prior exchange, and thus cannot distinguish from random the session key $K = H_1(A \parallel B \parallel g^a \parallel g^b \parallel L)$.

Accordingly this provides forward secrecy.

This is roughly the core of the unilateral session-key exchange in the TLS 1.3 handshake. It is based on Sigma [Kr03].
A password is a human-memorizable key.

Attackers can form a set $D$ of possible passwords called a dictionary such that

- If the target password $pwd$ is in $D$, and also
- The attacker knows $pwd = f(pwd)$, the image of $pwd$ under some public function $f$,

then the target password $pwd$ can be found via:

For all $pwd' \in D$ do
  If $f(pwd') = pwd$ then return $pwd'$

This is called a dictionary, or brute-force, attack.
Passwords are in widespread use for client authentication to Internet services and servers like \texttt{gmail}, Amazon, Internet banking, ... Most of us have more passwords than we can remember.

Passwords are communicated over TLS. The main threat is dictionary attacks arising from the adversary obtaining the image $pwd = f(pwd)$ of the target password pwd under some public function $f$.

Studies show that many users select poor passwords, meaning ones that fall into attacker dictionaries. And attackers get better and better at making dictionaries. So preventing dictionary attacks is important for security.
In 2016, the 25 most common passwords made up more than 10% of surveyed passwords, with the most common making up 4%.

### Top 25 most common passwords by year according to SplashData

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>password</td>
<td>password</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
<td>123456</td>
</tr>
<tr>
<td>2</td>
<td>123456</td>
<td>123456</td>
<td>password</td>
<td>password</td>
<td>password</td>
<td>password</td>
<td>password</td>
<td>password</td>
</tr>
<tr>
<td>3</td>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
<td>12345678</td>
<td>123456789</td>
<td>123456789</td>
</tr>
<tr>
<td>4</td>
<td>qwerty</td>
<td>abc123</td>
<td>qwerty</td>
<td>12345678</td>
<td>qwerty</td>
<td>12345678</td>
<td>qwerty</td>
<td>12345678</td>
</tr>
<tr>
<td>5</td>
<td>abc123</td>
<td>qwerty</td>
<td>abc123</td>
<td>qwerty</td>
<td>12345</td>
<td>football</td>
<td>12345</td>
<td>12345</td>
</tr>
<tr>
<td>6</td>
<td>monkey</td>
<td>monkey</td>
<td>123456789</td>
<td>123456789</td>
<td>123456789</td>
<td>qwerty</td>
<td>123456789</td>
<td>111111</td>
</tr>
<tr>
<td>7</td>
<td>1234567</td>
<td>letmein</td>
<td>111111</td>
<td>1234</td>
<td>football</td>
<td>1234567890</td>
<td>letmein</td>
<td>1234567</td>
</tr>
<tr>
<td>8</td>
<td>letmein</td>
<td>dragon</td>
<td>1234567</td>
<td>baseball</td>
<td>1234</td>
<td>1234567</td>
<td>1234567</td>
<td>sunshine</td>
</tr>
<tr>
<td>9</td>
<td>trustno1</td>
<td>111111</td>
<td>iloveyou</td>
<td>dragon</td>
<td>1234567</td>
<td>princess</td>
<td>football</td>
<td>qwerty</td>
</tr>
<tr>
<td>10</td>
<td>dragon</td>
<td>baseball</td>
<td>adobe123[a]</td>
<td>football</td>
<td>baseball</td>
<td>1234</td>
<td>iloveyou</td>
<td>iloveyou</td>
</tr>
<tr>
<td>11</td>
<td>baseball</td>
<td>iloveyou</td>
<td>123123</td>
<td>1234567</td>
<td>welcome</td>
<td>login</td>
<td>admin</td>
<td>princess</td>
</tr>
<tr>
<td>12</td>
<td>111111</td>
<td>trustno1</td>
<td>admin</td>
<td>monkey</td>
<td>1234567890</td>
<td>welcome</td>
<td>welcome</td>
<td>admin</td>
</tr>
<tr>
<td>13</td>
<td>iloveyou</td>
<td>1234567</td>
<td>1234567890</td>
<td>letmein</td>
<td>abc123</td>
<td>solo</td>
<td>monkey</td>
<td>welcome</td>
</tr>
<tr>
<td>14</td>
<td>master</td>
<td>sunshine</td>
<td>letmein</td>
<td>abc123</td>
<td>111111</td>
<td>abc123</td>
<td>login</td>
<td>666666</td>
</tr>
<tr>
<td>15</td>
<td>sunshine</td>
<td>master</td>
<td>photoshop[a]</td>
<td>111111</td>
<td>1qaz2wsx</td>
<td>admin</td>
<td>abc123</td>
<td>abc123</td>
</tr>
<tr>
<td>16</td>
<td>ashley</td>
<td>123123</td>
<td>1234</td>
<td>mustang</td>
<td>dragon</td>
<td>121212</td>
<td>starwars</td>
<td>football</td>
</tr>
<tr>
<td>17</td>
<td>bailey</td>
<td>welcome</td>
<td>monkey</td>
<td>access</td>
<td>master</td>
<td>flower</td>
<td>123123</td>
<td>123123</td>
</tr>
<tr>
<td>18</td>
<td>passw0rd</td>
<td>shadow</td>
<td>shadow</td>
<td>shadow</td>
<td>monkey</td>
<td>passw0rd</td>
<td>dragon</td>
<td>monkey</td>
</tr>
<tr>
<td>19</td>
<td>shadow</td>
<td>ashley</td>
<td>sunshine</td>
<td>master</td>
<td>letmein</td>
<td>dragon</td>
<td>passw0rd</td>
<td>654321</td>
</tr>
<tr>
<td>20</td>
<td>123123</td>
<td>football</td>
<td>12345</td>
<td>michael</td>
<td>login</td>
<td>sunshine</td>
<td>master</td>
<td>l@#$%^&amp;*</td>
</tr>
<tr>
<td>21</td>
<td>654321</td>
<td>jesus</td>
<td>password1</td>
<td>superman</td>
<td>princess</td>
<td>master</td>
<td>hello</td>
<td>charlie</td>
</tr>
<tr>
<td>22</td>
<td>superman</td>
<td>michael</td>
<td>princess</td>
<td>696969</td>
<td>qwertyuiop</td>
<td>hottie</td>
<td>freedom</td>
<td>aa123456</td>
</tr>
<tr>
<td>23</td>
<td>qazwsx</td>
<td>ninja</td>
<td>azerty</td>
<td>123123</td>
<td>solo</td>
<td>loveme</td>
<td>whatever</td>
<td>donald</td>
</tr>
<tr>
<td>24</td>
<td>michael</td>
<td>mustang</td>
<td>trustno1</td>
<td>batman</td>
<td>passw0rd</td>
<td>zaq1zaq1</td>
<td>qazwsx</td>
<td>password1</td>
</tr>
<tr>
<td>25</td>
<td>Football</td>
<td>password1</td>
<td>000000</td>
<td>trustno1</td>
<td>starwars</td>
<td>password1</td>
<td>trustno1</td>
<td>qwerty123</td>
</tr>
</tbody>
</table>
A protocol for Password Authenticated Key Exchange (PAKE) assumes client $A$ has a password $pwd$ and server $B$ has either $pwd$ or its hash under a public hash function.

The parties interact to arrive at a common session key $K$ satisfying authenticity, secrecy, forward secrecy and also security against off-line dictionary attacks.

This means the protocol never reveals an image $pwd = f(pwd)$ of $pwd$ under a public function $f$. So even if the password is in the dictionary, the off-line dictionary attack is infeasible.

Roughly, one adversary interaction with one of the parties can eliminate at most one candidate password from the dictionary.

Authentication here is mutual, and no PKI / certificates are assumed.
Protocol KE4

\[
\begin{align*}
A & \quad B \\
A, g^a & \quad B, g^b, \text{MAC}_M(1|A|B|g^a|g^b) \\
\text{MAC}_M(0|A|B|g^a|g^b) & 
\end{align*}
\]

Client A has password pwd that is known to server B.

Let \( L = g^{ab} \) be the DH key. Then the session key and MAC keys are \( K = H_1(A|B|g^a|g^b|L|pwd) \) and \( M = H_2(A|B|g^a|g^b|L|pwd) \), respectively.

Is this secure against dictionary attack?
A successful dictionary attack by adversary $E$ is possible, as follows:

$E$ has $A, B, g^a, g^b$ and also $L = g^{ab} = (g^b)^a$. Let

$$f(\text{pwd}) = \text{MAC}_{H_2}(A\|B\|g^a\|g^b\|L\|\text{pwd})(A\|B\|g^a\|g^b).$$

This $f$ is a public function of the password, allowing $E$ to mount the dictionary attack.
History and status of PAKE

The first protocols were by Bellovin and Merrit, 1992.
Definitions and proven-secure protocols begin with [BPR00].
Large literature.
A representative modern PAKE protocol is OPAQUE [JKX18].