KEY DISTRIBUTION:
PKI and SESSION-KEY EXCHANGE

The public key setting

$$\begin{array}{ccc} \text{Alice} & & \text{Bob}^{pk[A]} \\ M \leftarrow \mathcal{D}_{sk[A]}(C) & \xleftarrow{\quad C \quad} & C \overset{\$}{\leftarrow} \mathcal{E}_{pk[A]}(M) \\ \sigma \overset{\$}{\leftarrow} \mathcal{S}_{sk[A]}(M) & \xrightarrow{\quad M,\sigma \quad} & \mathcal{V}_{pk[A]}(M,\sigma) \end{array}$$

Bob can:

- send encrypted data to Alice
- verify her signatures

as long as he has Alice's public key $pk[A]$.

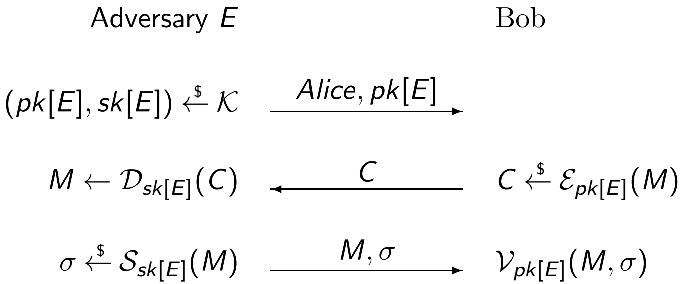
But how does he get $pk[A]$?

Distributing public keys

How about:

$$\begin{array}{ccc}
\text{Alice} & & \text{Bob} \\
(pk[A], sk[A]) \overset{\$}{\leftarrow} \mathcal{K} & \xrightarrow{\hspace{2em} Alice,\, pk[A] \hspace{2em}} & \\
M \leftarrow \mathcal{D}_{sk[A]}(C) & \xleftarrow{\hspace{4em} C \hspace{4em}} & C \overset{\$}{\leftarrow} \mathcal{E}_{pk[A]}(M) \\
\sigma \overset{\$}{\leftarrow} \mathcal{S}_{sk[A]}(M) & \xrightarrow{\hspace{3em} M,\, \sigma \hspace{3em}} & \mathcal{V}_{pk[A]}(M, \sigma)
\end{array}$$

Man-in-the-middle attack



Adversary E Bob

$(pk[E], sk[E]) \xleftarrow{\$} \mathcal{K}$ $\xrightarrow{\quad Alice, pk[E] \quad}$

$M \leftarrow \mathcal{D}_{sk[E]}(C)$ $\xleftarrow{\quad C \quad}$ $C \xleftarrow{\$} \mathcal{E}_{pk[E]}(M)$

$\sigma \xleftarrow{\$} \mathcal{S}_{sk[E]}(M)$ $\xrightarrow{\quad M, \sigma \quad}$ $\mathcal{V}_{pk[E]}(M, \sigma)$

Messages that Bob encrypts to Alice are obtained by E, and E can forge Alice's signatures.

The authenticity problem and PKI

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 $(pk, sk) \xleftarrow{\$} \mathcal{K}$

Send to CA: Alice sends $(Alice, pk)$ to a certificate authority (CA).

Identity check

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 σ 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 $CERT_A$ to Bob who will:

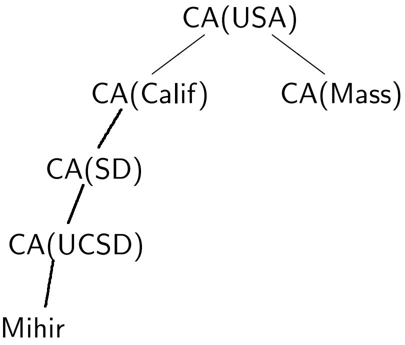
- $(CERTDATA, \sigma) \leftarrow CERT_A$
- Check $\mathcal{V}_{pk[CA]}(CERTDATA, \sigma) = 1$ where $pk[CA]$ is CA's public key
- $(pk, Alice, expiry, \ldots) \leftarrow CERTDATA$
- Check certificate has not expired
- $\ldots$

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



$\underline{CERT_{Mihir}}$

$CERT[CA(USA) : CA(Calif)]$
$CERT[CA(Calif) : CA(SD)]$
$CERT[CA(SD) : CA(UCSD)]$
$CERT[CA(UCSD) : Mihir]$

$$CERT[X : Y] = (pk[Y], Y, \ldots, \mathcal{S}_{sk[X]}(pk[Y], Y, \ldots))$$

To verify $CERT_{Mihir}$ you need only $pk_{CA[USA]}$.

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.)

Revocation

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.

Revocation in practice

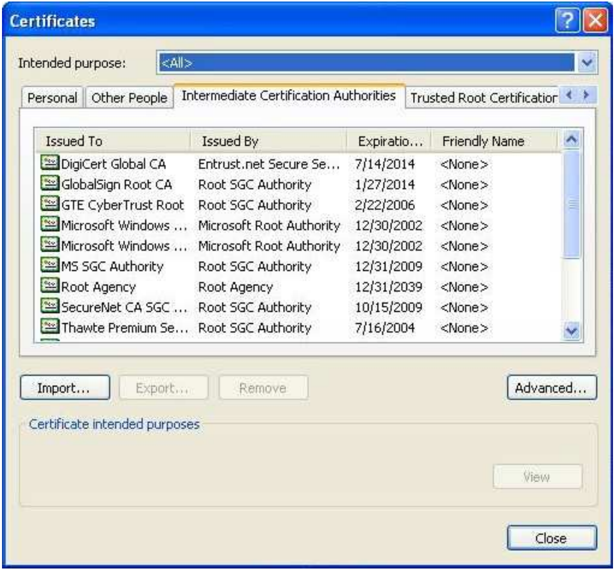
- VeriSign estimates that 20% of certificates are revoked
- In practice, CRLs are huge

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.

PGP

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

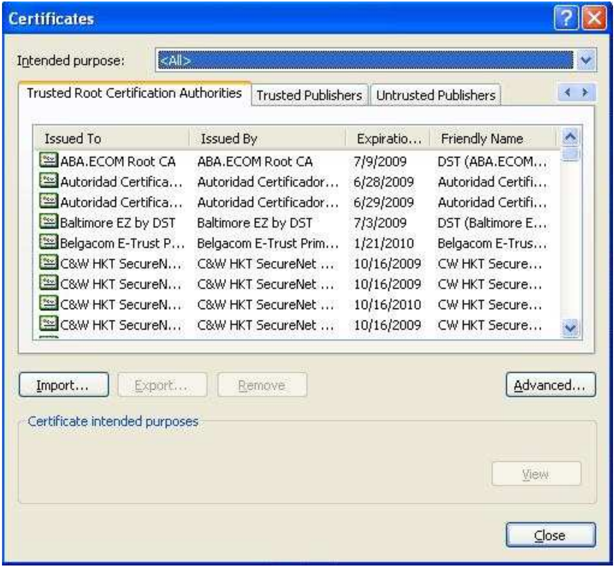


Intended purpose: <All>

Personal | Other People | Intermediate Certification Authorities | Trusted Root Certificatior

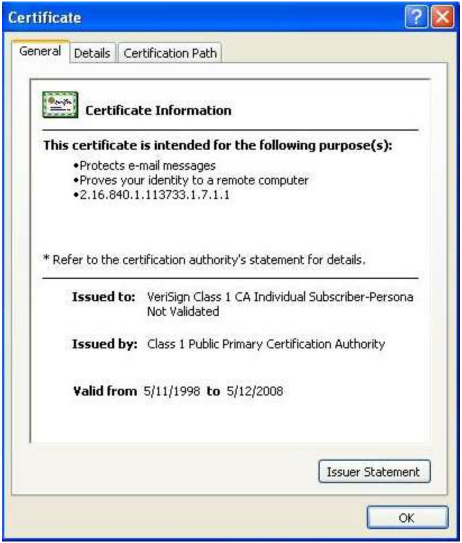
Issued To	Issued By	Expiratio...	Friendly Name
DigiCert Global CA	Entrust.net Secure Se...	7/14/2014	<None>
GlobalSign Root CA	Root SGC Authority	1/27/2014	<None>
GTE CyberTrust Root	Root SGC Authority	2/22/2006	<None>
Microsoft Windows ...	Microsoft Root Authority	12/30/2002	<None>
Microsoft Windows ...	Microsoft Root Authority	12/30/2002	<None>
MS SGC Authority	Root SGC Authority	12/31/2009	<None>
Root Agency	Root Agency	12/31/2039	<None>
SecureNet CA SGC ...	Root SGC Authority	10/15/2009	<None>
Thawte Premium Se...	Root SGC Authority	7/16/2004	<None>

Certificate Examples



Issued To	Issued By	Expiratio...	Friendly Name
ABA.ECOM Root CA	ABA.ECOM Root CA	7/9/2009	DST (ABA.ECOM...
Autoridad Certifica...	Autoridad Certificador...	6/28/2009	Autoridad Certifi...
Autoridad Certifica...	Autoridad Certificador...	6/29/2009	Autoridad Certifi...
Baltimore EZ by DST	Baltimore EZ by DST	7/3/2009	DST (Baltimore E...
Belgacom E-Trust P...	Belgacom E-Trust Prim...	1/21/2010	Belgacom E-Trus...
C&W HKT SecureN...	C&W HKT SecureNet ...	10/16/2009	CW HKT Secure...
C&W HKT SecureN...	C&W HKT SecureNet ...	10/16/2009	CW HKT Secure...
C&W HKT SecureN...	C&W HKT SecureNet ...	10/16/2010	CW HKT Secure...
C&W HKT SecureN...	C&W HKT SecureNet ...	10/16/2009	CW HKT Secure...

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

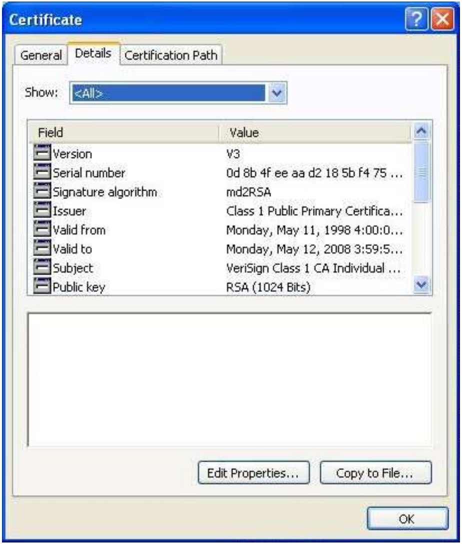
* Refer to the certification authority's statement for details.

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

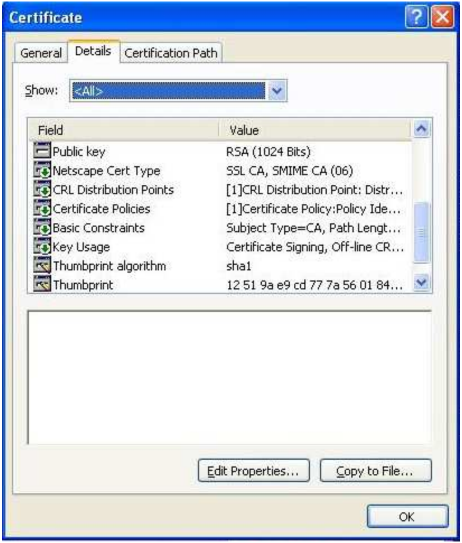
Certificate Examples



Show: <All>

Field	Value
Version	V3
Serial number	0d 8b 4f ee aa d2 18 5b f4 75 ...
Signature algorithm	md2RSA
Issuer	Class 1 Public Primary Certifica...
Valid from	Monday, May 11, 1998 4:00:0...
Valid to	Monday, May 12, 2008 3:59:5...
Subject	VeriSign Class 1 CA Individual ...
Public key	RSA (1024 Bits)

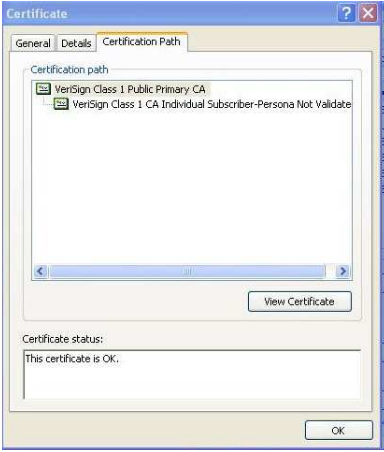
Certificate Examples



Show: <All>

Field	Value
Public key	RSA (1024 Bits)
Netscape Cert Type	SSL CA, SMIME CA (06)
CRL Distribution Points	[1]CRL Distribution Point: Distr...
Certificate Policies	[1]Certificate Policy:Policy Ide...
Basic Constraints	Subject Type=CA, Path Lengt...
Key Usage	Certificate Signing, Off-line CR...
Thumbprint algorithm	sha1
Thumbprint	12 51 9a e9 cd 77 7a 56 01 84...

Certificate Examples

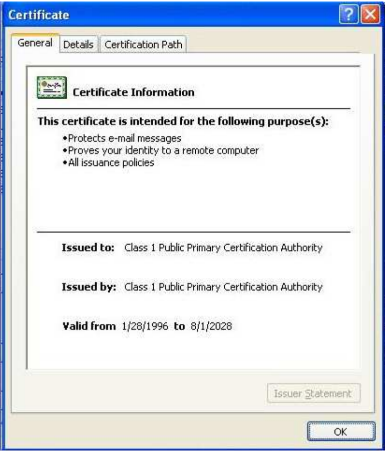


Certification path

VeriSign Class 1 Public Primary CA
VeriSign Class 1 CA Individual Subscriber-Persona Not Validate

Certificate status:

This certificate is OK.



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

Shared key setting

$$\begin{tikzcd}
\text{Alice}^K & \text{Bob}^K \\
M \leftarrow \mathcal{D}_K(C) & C \xleftarrow{\$} \mathcal{E}_K(M) \arrow[l, "C"'] \\
\sigma \xleftarrow{\$} \mathcal{T}_K(M) \arrow[r, "{M, \sigma}"] & \mathcal{V}_K(M, \sigma)
\end{tikzcd}$$

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.

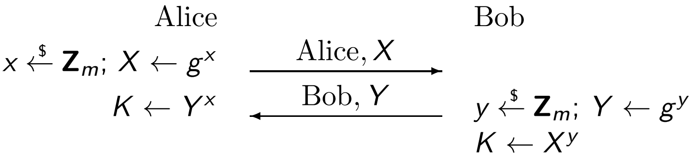
$$\begin{array}{rcl} \text{Alice} & & \text{Bob} \\ x \overset{\$}{\leftarrow} \mathbf{Z}_m;\ X \leftarrow g^x & \xrightarrow{\quad \text{Alice}, X \quad} & \\ K \leftarrow Y^x & \xleftarrow{\quad \text{Bob}, Y \quad} & y \overset{\$}{\leftarrow} \mathbf{Z}_m;\ Y \leftarrow g^y \\ & & K \leftarrow X^y \end{array}$$

$$Y^x = (g^y)^x = \underbrace{g^{xy}}_{K} = (g^x)^y = X^y$$

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

$$\begin{array}{rcl} \text{Alice} & & \text{Bob} \\ M \leftarrow \mathcal{D}_K(C) & \xleftarrow{\qquad C \qquad} & C \overset{\$}{\leftarrow} \mathcal{E}_K(M) \end{array}$$

Security of DH Key Exchange under Passive Attack



Alice

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

$K \leftarrow Y^x$

$\xrightarrow{\text{Alice}, X}$

$\xleftarrow{\text{Bob}, Y}$

Bob

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

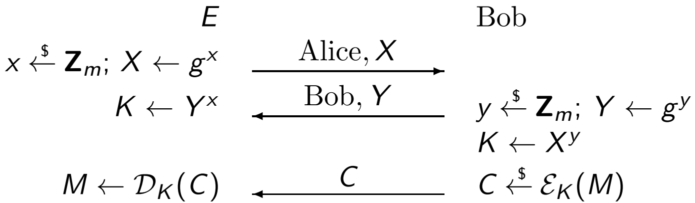
$K \leftarrow X^y$

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:



$$\begin{array}{rcl}
E & & \text{Bob} \\
x \overset{\$}{\leftarrow} \mathbf{Z}_m;\ X \leftarrow g^x & \xrightarrow{\text{Alice}, X} & \\
K \leftarrow Y^x & \xleftarrow{\text{Bob}, Y} & y \overset{\$}{\leftarrow} \mathbf{Z}_m;\ Y \leftarrow g^y \\
 & & K \leftarrow X^y \\
M \leftarrow \mathcal{D}_K(C) & \xleftarrow{C} & C \overset{\$}{\leftarrow} \mathcal{E}_K(M)
\end{array}$$

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.

Session keys: The “real” key distribution problem

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

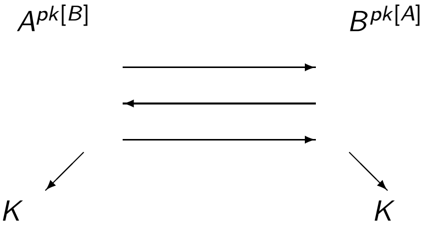
Why session keys?

- 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

Session key distribution

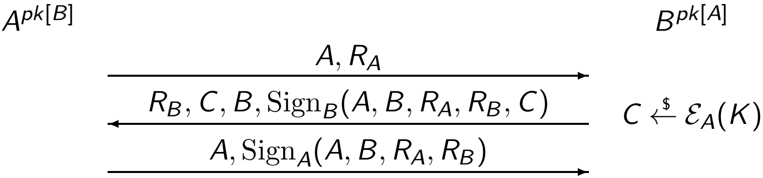
- Hundreds of protocols
- Dozens of security requirements
- Lots of broken protocols
- Protocols easy to specify and hard to get right
- Used ubiquitously: SSL, TLS, SSH, ...

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: Basic Exchange



$A^{pk[B]}$

$B^{pk[A]}$

A, R_A $\longrightarrow$

$\longleftarrow$ $R_B, C, B, \mathrm{Sign}_B(A, B, R_A, R_B, C)$

$C \xleftarrow{\$} \mathcal{E}_A(K)$

$A, \mathrm{Sign}_A(A, B, R_A, R_B)$ $\longrightarrow$

Session key K is chosen by B.

R_A, R_B are random nonces.

$\mathrm{Sign}_P(M)$ is P's signature of M under $sk[P]$. It is verifiable given $pk[P]$.

$\mathcal{E}_A(\cdot)$ is encryption under A's public key $pk[A]$, decryptable using $sk[A]$.

Beyond KE2

Modern session key exchange protocols enhance KE2 to incorporate many other security features including

- **Forward security:** Exposure of long-lived secret keys should not compromise PRIOR session keys
- **Anonymity:** Eavesdropper cannot figure out identity (public key) of client authenticating to some server.

Passwords

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

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

This is called a dictionary attack.

Password usage

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.

Preventing dictionary attacks: avoiding image revelation

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.