

KEY DISTRIBUTION

The public key setting

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

Bob can:

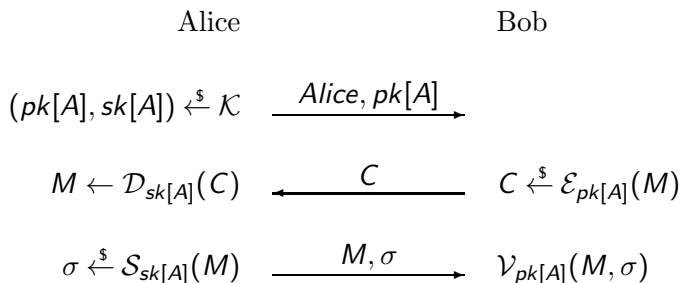
- send encrypted data to Alice
- verify her signatures

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

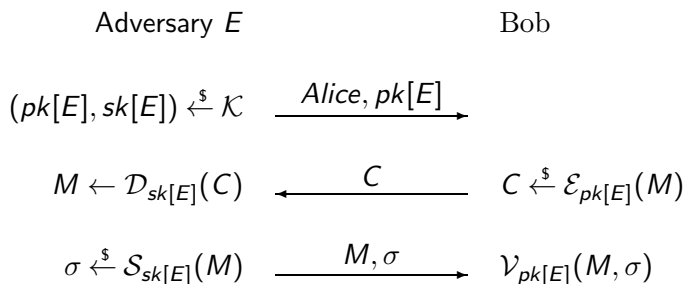
But how does he get $pk[A]$?

Distributing public keys

How about:



Man-in-the-middle attack



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.

Alice can send $CERT_A$ to Bob who will:

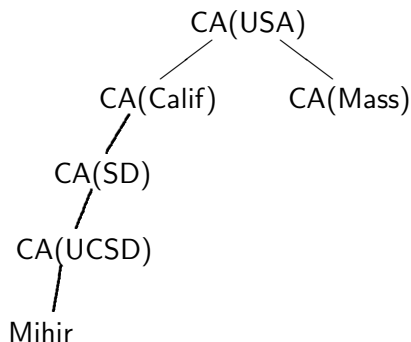
- $(CERTDATA, \sigma) \leftarrow CERT_A$
- Check $\mathcal{V}_{pk[CA]}(CERTDATA, \sigma) = 1$ where $pk[CA]$ is CA's public key
- $(pk, Alice, expiry, \dots) \leftarrow CERTDATA$
- Check certificate has not expired
- ...

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



$CERT_{Mihir}$

$CERT[CA(USA) : CA(Calif)]$

$CERT[CA(Calif) : CA(SD)]$

$CERT[CA(SD) : CA(UCSD)]$

$CERT[CA(UCSD) : Mihir]$

$CERT[X : Y] = (pk[Y], Y, \dots, S_{sk[X]}(pk[Y], Y, \dots))$

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.)

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*, Revocation date)

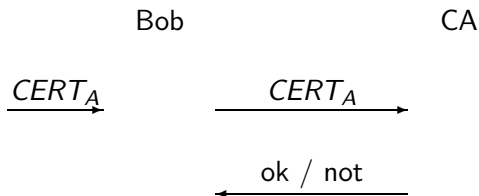
This list is disseminated.

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

- 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.



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

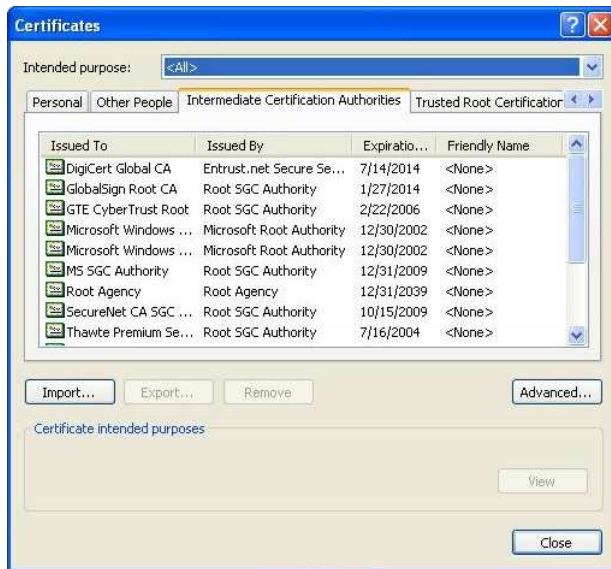
Revocation in practice

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

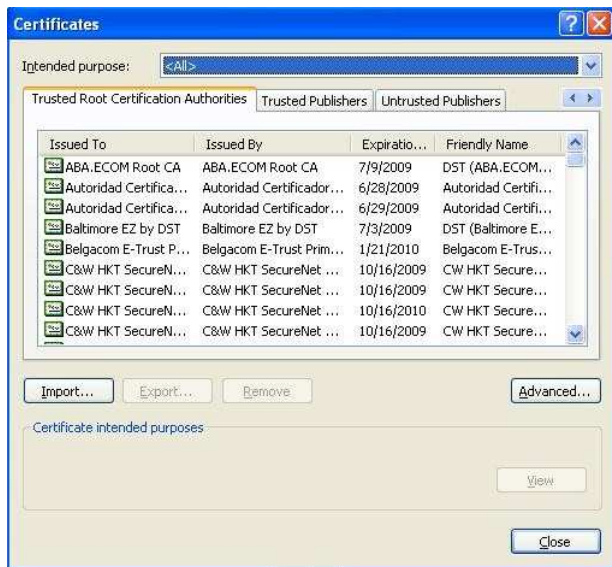
Revocation is a big problem and one of the things that is holding up widespread deployment of a PKI and use of public-key cryptography.

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



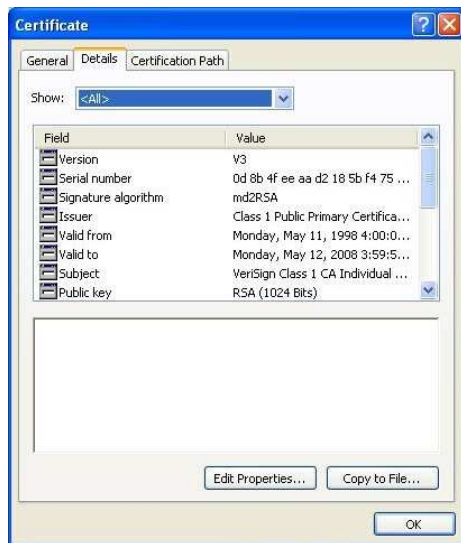
Certificate Examples



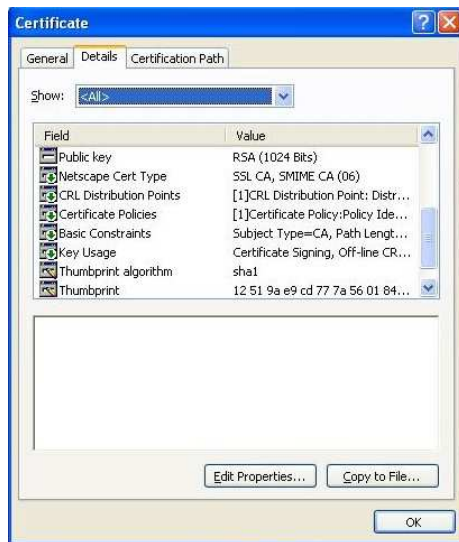
Certificate Examples



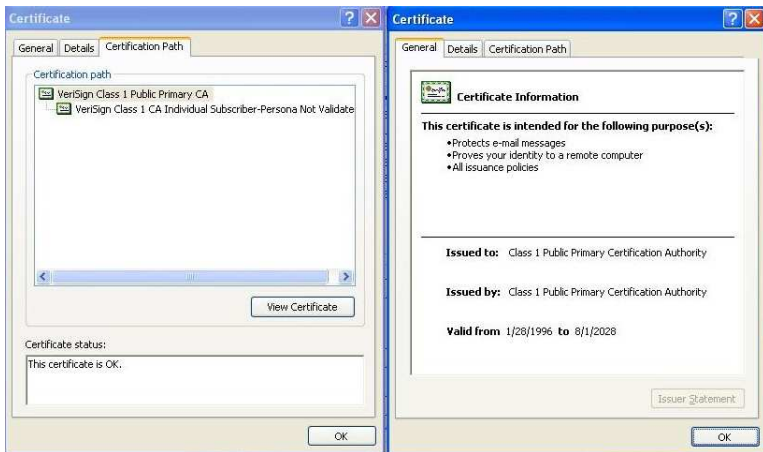
Certificate Examples



Certificate Examples



Certificate Examples



Shared key setting

$$\begin{array}{ccc} \text{Alice}^K & & \text{Bob}^K \\ M \leftarrow \mathcal{D}_K(C) & \xleftarrow{C} & C \xleftarrow{\$} \mathcal{E}_K(M) \\ \sigma \xleftarrow{\$} \mathcal{T}_K(M) & \xrightarrow{M, \sigma} & \mathcal{V}_K(M, \sigma) \end{array}$$

Alice and Bob can

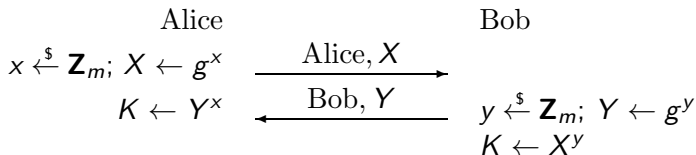
- send each other encrypted data
- verify each other's MACs

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

But how do Alice and Bob get a shared key?

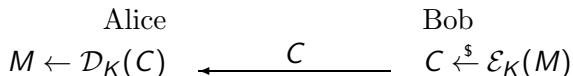
Diffie-Hellman Key Exchange

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

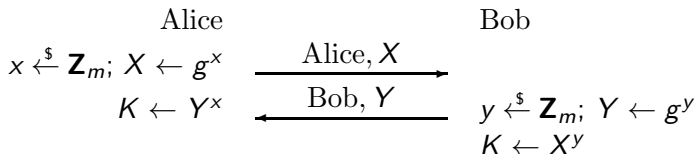


$$Y^x = (g^y)^x = \underbrace{g^{xy}}_K = (g^x)^y = X^y$$

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



Security of DH Key Exchange under Passive Attack

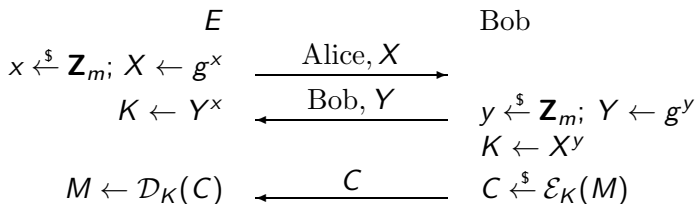


Eavesdropping adversary gets $X = g^x$ and $Y = g^y$ and wants to compute g^{xy} . But this is the (presumed hard) CDH problem.

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.

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

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

Why session keys? Example

Alice and Bob share long-lived key K .

App1 uses CBC encryption for privacy.

App2 uses CBC MAC for integrity.

What happens if they both use the same overlying key K ?

Why session keys? Example

$E : \{0, 1\}^k \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ a block cipher.

In App1, Alice lets $C_0 \xleftarrow{\$} \{0, 1\}^n$; $C_1 \leftarrow E_K(C_0 \oplus M_1)$ and sends (C_0, C_1) to Bob.

Assume adversary sees C_0, C_1 and knows M_1 . Then $T = C_1$ is the CBC MAC of $M_2 = C_0 \oplus M_1$, so adversary can forge a MAC in App2 by sending (M_2, T) .

Another example

Suppose an application needs to encrypt just one, short message, and does it using a one-time pad.

$$\underline{C = K \oplus M} \rightarrow$$

If M later becomes known, adversary gets K .

If latter is the long-lived key, other applications are compromised.

The fault (for the above attacks) is not with the application(s) or the cryptography (CBC encryption, CBC MAC, or OTP) they use. An application has a right to think it has exclusive use of its key.

We need to design the system so that applications can use their keys as they wish, yet there are no “bad interactions” between them.

The solution is to give each “instance” of each application its own session key.

Basic setting and requirements

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 α 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 $N_B - 1$.

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 α .

What E does:

$E \rightarrow B : \{\alpha, A\}_{K[B]}$

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 α .

What E does:

$E \rightarrow B : \{\alpha, A\}_{K[B]}$

$B \rightarrow A$

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 α .

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 α .

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 α .

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 α . But E knows α and any use of it by B is insecure.

Time stamps

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.

$A \rightarrow S : A, B$

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

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

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

Session key α 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?

$A \rightarrow S : A, B$

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

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

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

Assume E obtains all of the above flows and also learns α .

- $E \rightarrow B : t_B, \{A, T\}_\alpha$

Known-key attack?

$A \rightarrow S : A, B$

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

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

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

Assume E obtains all of the above flows and also learns α .

- $E \rightarrow B : t_B, \{A, T\}_\alpha$; B will reject because T is not current

Known-key attack?

$A \rightarrow S : A, B$

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

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

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

Assume E obtains all of the above flows and also learns α .

- $E \rightarrow B : t_B, \{A, T\}_\alpha$; B will reject because T is not current
- $E \rightarrow B : t_B, \{A, T'\}_\alpha$

Known-key attack?

$A \rightarrow S : A, B$

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

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

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

Assume E obtains all of the above flows and also learns α .

- $E \rightarrow B : t_B, \{A, T\}_\alpha$; 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?

$A \rightarrow S : A, B$

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

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

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

Assume E obtains all of the above flows and also learns α .

- $E \rightarrow B : t_B, \{A, T\}_\alpha$; 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?

$A \rightarrow S : A, B$

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

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

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

Assume E obtains all of the above flows and also learns α .

- $E \rightarrow B : t_B, \{A, T\}_\alpha$; 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?

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?

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: α only (T, B, A are known!)

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: α only (T, B, A are known!)
- Question: Then why are T, B, A encrypted?

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: α only (T, B, A are known!)
- Question: Then why are T, B, A encrypted?
- 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: α 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: α 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 is a good match

Provides:

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

Sender

- $C \xleftarrow{\$} \mathcal{E}_K(N, AD, M)$
- 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

$$S \rightarrow A : N, \underbrace{N', T, C_B}_{t_B}, C_A$$

where

- $C_B \stackrel{\$}{\leftarrow} \mathcal{E}_{K[B]}(N, T \| A, \alpha)$
- $C_A \stackrel{\$}{\leftarrow} \mathcal{E}_{K[A]}(N', T \| B \| C_B, \alpha)$

Alternative implementation

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

can be implemented as

$$S \rightarrow A : T, C_A, \underbrace{C_B, \tau_B}_{t_B}, C_A, \tau_A$$

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

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

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:

$$A \rightarrow B : t_B, \text{MAC}_\alpha(A, T)$$

$$B \rightarrow A : \text{MAC}_\alpha(T + 1)$$

Security of Session Key

Question: What is desired security attribute of session key?

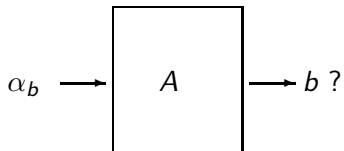
Security of Session Key

Question: What is desired security attribute of session key?

Answer: It should be indistinguishable from random to adversary

At end of protocol:

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



Session key security under key confirmation

$$\begin{aligned} A \rightarrow B &: t_B, \overbrace{\{A, T\}}^C_\alpha \\ B \rightarrow A &: \{T + 1\}_\alpha \end{aligned}$$

Session key is not indistinguishable from random. Adversary given challenge α_b can decrypt C under α_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!

$A \rightarrow B : R_A$

$B \rightarrow S : R_A, R_B$

$S \rightarrow A : C_A, \text{MAC}_{K[A]}(A, B, R_A, C_A)$

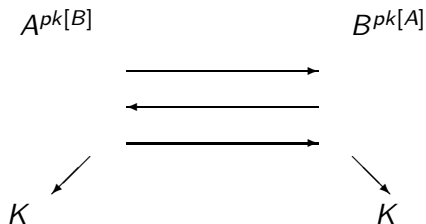
$S \rightarrow B : C_B, \text{MAC}_{K[B]}(A, B, R_B, C_B)$

where $C_A \stackrel{\$}{\leftarrow} \mathcal{E}_{K[A]}(\alpha)$; $C_B \stackrel{\$}{\leftarrow} \mathcal{E}_{K[B]}(\alpha)$

NO key confirmation: α never used!

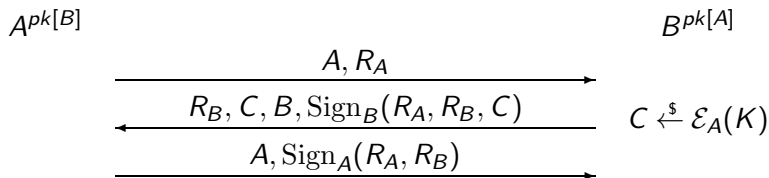
This protocol can be proven to satisfy a strong, formal notion of session key distribution security assuming standard properties of \mathcal{E} , MAC [BR95].

Session key exchange in public key setting



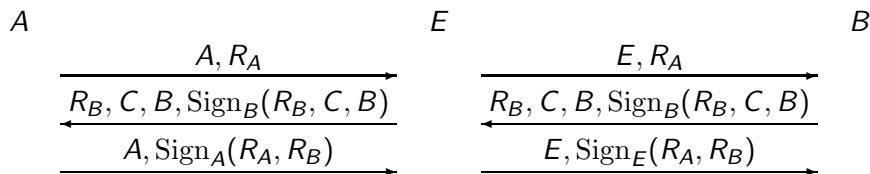
Most important type of session key exchange in practice, used in all communication security protocols: SSL, SSH, TLS, IPSEC, 802.11, ...

Protocol KE1



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

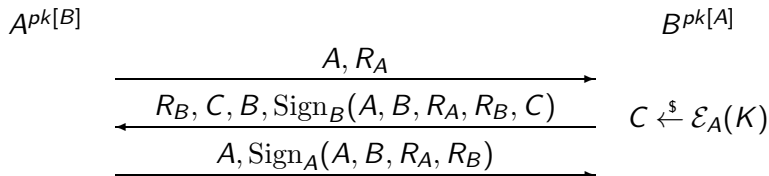
Binding attack



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 .

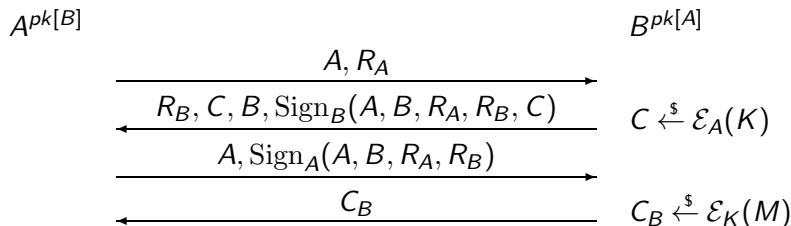
A good example of exactly why this is a problem is, however, lacking.

Protocol KE2



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



Nov. 20: Adversary E records above flows.

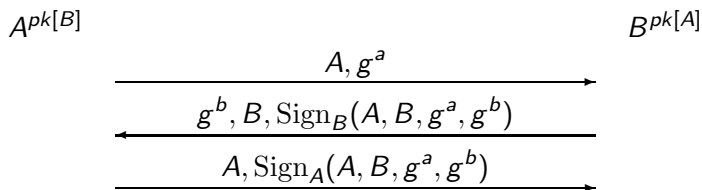
Dec. 18: A's, system compromised and $sk[A]$ exposed.

Dec. 19: A revokes $pk[A]$ so that no further damage is done but cannot prevent E from

$$K \leftarrow \mathcal{D}_{sk[A]}(C); M \leftarrow \mathcal{D}_K(C_B)$$

Can we achieve forward secrecy: Privacy of communication done prior to exposure of $sk[A]$ is not compromised?

KE3: Forward secrecy



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

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.

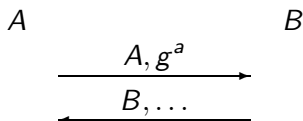
All standard protocols use DH to get forward security.

Anonymity

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

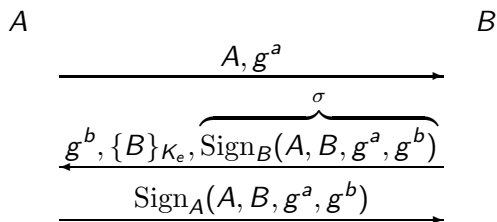
This might be desirable when B is a mobile client, communicating with base station A ; B does not want her location known to E .

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



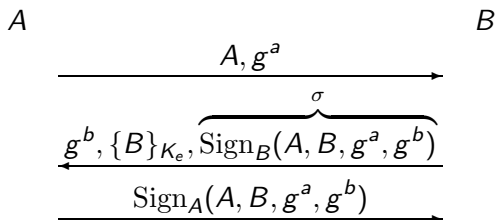
Such protocols do not provide anonymity.

KE4: Targeting anonymity for B



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 $B \in \{B_1, \dots, B_n\}$ then E can identify B via:

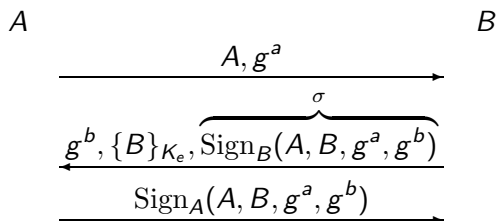
For $i = 1, \dots, n$ **do**:

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 $B \in \{B_1, \dots, B_n\}$ then E can identify B via:

For $i = 1, \dots, n$ **do**:

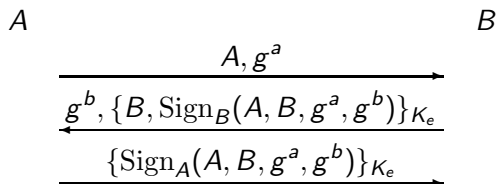
if $\mathcal{V}_{pk[B_i]}((A, B_i, g^a, g^b), \sigma) = 1$ **then return** B_i

Signatures reveal identity!

Question: so why don't we send g^b encrypted too?

Answer: How would A decrypt?

KE5: Anonymity for B



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

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

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.

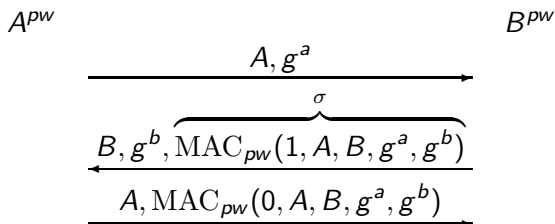
Password-based session-key exchange

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.

Protocol KE6



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

Dictionary attack is possible: Let f be defined by

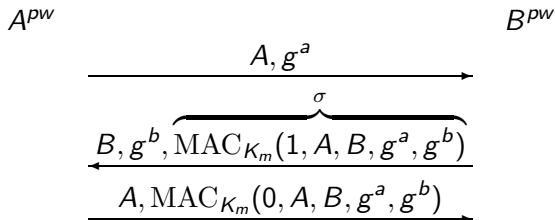
$$f(x) = MAC_x(1, A, B, g^a, g^b)$$

Then get pw via

for all $pw' \in D$ do

if $f(pw') = \sigma$ then return pw'

Protocol KE7



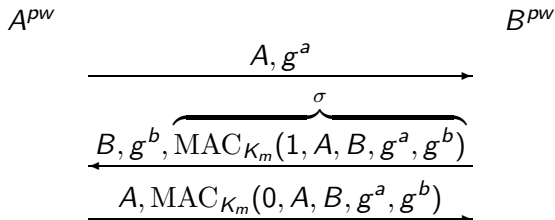
where $K_m = H(1, A, B, g^a, g^b, g^{ab}, pw)$ and the session key is $K = H(0, A, B, g^a, g^b, g^{ab})$.

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



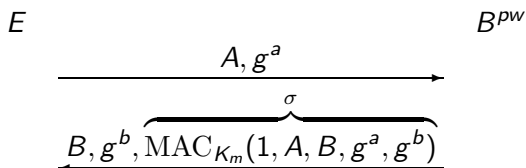
where $K_m = H(1, A, B, g^a, g^b, g^{ab}, pw)$ and the session key is $K = H(0, A, B, g^a, g^b, g^{ab})$.

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 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



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

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

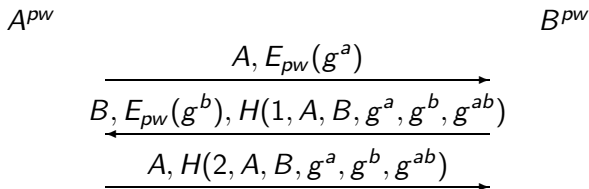
becomes public and a dictionary attack is possible.

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]



$E : PW \times G \rightarrow G$ 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].