Data Security and Privacy

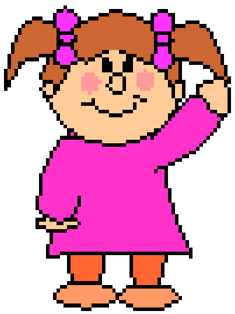
Topic 14: Authentication and Key Establishment

Announcements

- Mid-term Exam
  - Tuesday March 6, during class

Need for Key Establishment

$\text{Encrypt}_K(M)$



$C = \text{Encrypt}_K(M)$

$M = \text{Decrypt}_K(C)$

- Alice and Bob share a secret key K
- **How to establish the shared key?**
- **How to refresh it (not a good idea to encrypt a lot of data with the same key)**

Long-Term Key vs. Session Key

- **Session key**: temporary key, used for a short time period.
- **Long-term key**: used for a long term period, sometimes public and secret key pairs used to sign messages.
- Using session keys to:
  - limit available cipher-text encrypted with the same key
  - limit exposure in the event of key compromise
  - avoid long-term storage of a large number of distinct secret keys
  - create independence across communications sessions or applications

Key Transport vs. Key Agreement

- **Key establishment**: process to establish a shared secret key available to two or more parties;
  - key transport: one party creates, and securely transfers it to the other(s).
  - key agreement: key establishment technique in which a shared secret is derived by two (or more) parties

Key Pre-distribution vs. Dynamic Key Establishment

- **Key establishment**
  - **Key pre-distribution**: established keys are completely determined a priori by initial keying material
    - generally in the form of key agreement
  - **Dynamic shared key establishment**: protocols that keys established between a fixed group of users varies in different sessions
    - also known as session key establishment
    - could be key transport or key agreement

Assumptions and Adversaries

- Assumption: Protocol messages are transmitted over open networks
- An adversary may
  - Eavesdrop messages.
  - Altering messages to help recover the key
  - Inject messages into existing sessions
  - Initiate one or more protocol execution (possibly simultaneously) and combine messages from one with another)

Effects of Key Compromise

- Perfect forward secrecy: compromise of long-term key does not compromise past session keys.

- Known-key attack: compromise of past session keys allows either a passive adversary to compromise future session keys, or impersonation by an active adversary in the future.

Basic Key Transport Protocol

- Assumes a long term symmetric key K shared between A and B
- Basic: new key is r_A

$$A \rightarrow B: E_K(r_A)$$

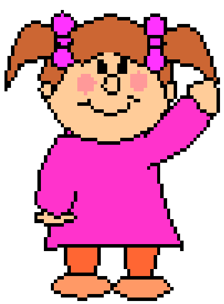
- Prevents replay: new key is r_A

$$A \rightarrow B: E_K(r_A, t_A, B),$$

Where t_A is a time-stamp

Basic Key Transport Protocol (cont.)

- Provides mutual entity authentication and key authentication
- Jointly control the key
- Does not provide perfect forward secrecy



n_B (←)

$E_K(r_A, n_A, n_B, B)$ (→)

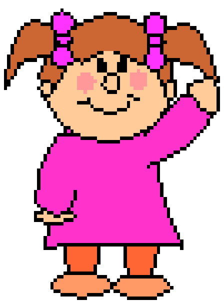
$E_K(r_B, n_B, n_A, A)$ (←)

n_A, n_B are nounces, newly generated random numbers

Key derived from r_A and r_B

Authenticated Key Exchange Protocol 2 (AKEP2)

- Setup: A and B share long-term keys K and K'
- h_K is a MAC (keyed hash function)
- $h'_{K'}$ is a pseudo-random permutation (a block cipher)
- establish key $W = h'_{K'}(r_B)$



r_A →

← $(B, A, r_A, r_B), h_K(B, A, r_A, r_B)$

$(A, r_B), h_K(A, r_B)$ →

Properties Associated with Authentication Protocols

- Entity authentication: identity of a party, and aliveness at a given instant
- Data origin authentication: identity of the source of the data
- Implicit key authentication: one party is assured that no other party aside from a specifically identified second party may gain access to a particular secret key.
- Key confirmation: one party is assured that a second party actually has possession of a particular secret key.
- Explicit key authentication: both (implicit) key authentication and key confirmation hold.

Other Issues in Key Establishment

- Type of the authentication: unilateral vs. mutual
- Key freshness: whether the established key could be one used in previous sessions
- Key control: key distribution vs. key agreement
- Efficiency: communication (number of message and communication rounds) and computation (exponentiations and digital signatures) costs
- Use of trusted third party (TTP):
  - on-line/off-line/no third party
  - degree of trust required in a third party

Key Agreement among Multiple Parties

- For a group of N parties, every pair needs to share a different symmetric key
  - What is the number of keys?
  - What secure channel to use to establish the keys?

- How to establish such keys
  - Symmetric Encryption - Use a central authority, a.k.a. (TTP).
  - Asymmetric Encryption – PKI.

Key Establishment by Means of Symmetric Encryption

- Every pair shares one long-term key
- Use TTP
  - Each entity maintains long-term keys with TTP
  - Easy to add and remove entities
  - Each entity needs to store only one long-term secret key
  - Trust in TTP, it can read all messages.
  - Compromise of TTP leads to compromise of all communication channels.

Needham-Schroeder Shared-Key Protocol

- Parties: A, B, and trusted server T
- Setup: A and T share K_{AT}, B and T share K_{BT}
- Goal:
  - Security: Mutual entity authentication between A and B; key establishment, Secure against active attacker & replay
  - Efficiency: Minimize the involvement of T; T can be stateless
- Messages:

$A \rightarrow T$: A, B, N_A (1)

$A \leftarrow T$: $E[K_{AT}]$ (N_A, B, k, $E[K_{BT}]$(k,A)) (2)

$A \rightarrow B$: $E[K_{BT}]$ (k, A) (3)

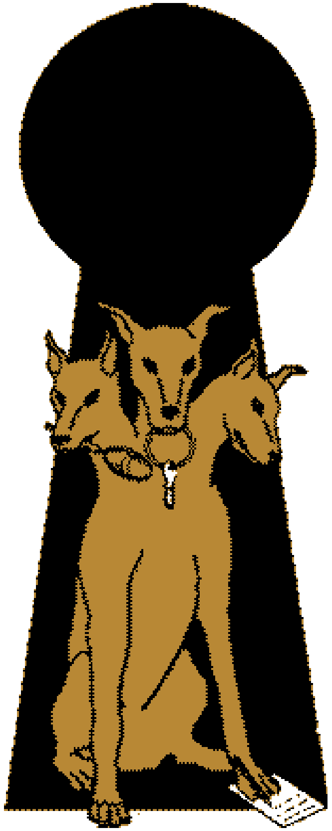
$A \leftarrow B$: E[k] (N_B) (4)

$A \rightarrow B$: E[k] (N_B-1) (5)

What bad things can happen if there is no N_A?
Another subtle flaw in Step 3.

Kerberos

- Implements the idea of Needham-Schroeder protocol
- Kerberos is a **network authentication protocol**
- Provides authentication and secure communication
- Relies entirely on **symmetric cryptography**
- Developed at MIT: http://web.mit.edu/kerberos/www
- Used in many systems, e.g., Windows 2000 and later as default authentication protocol

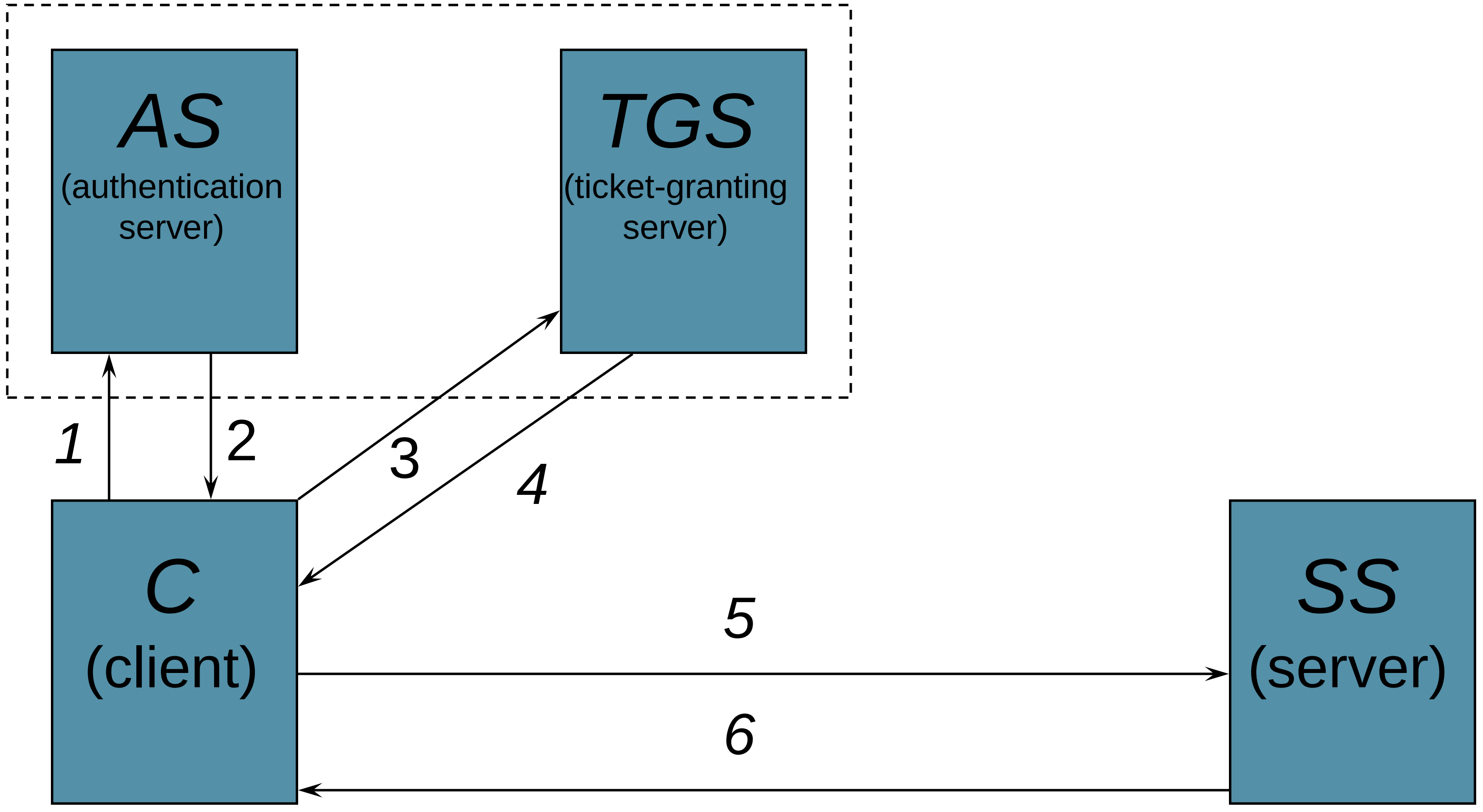


Kerberos Overview

- One issue of Needham-Schroeder – Needs [K_{AT}] to talk to any new service.
  - Think about a login session with K_{AT} derived from a password; either the password needs to be stored, or user needs to enter
- Kerberos solution:
  - Separates TTP into an AS and a TGS.
- The client authenticates to AS using a long-term *shared secret* and receives a TGT [SSO].
- Use this TGT to get additional tickets from TGS without resorting to using the shared secret.

AS = Authentication Server
TGS = Ticket Granting Server
SS = Service Server
TGT = Ticket Granting Ticket

Kerberos Protocol - 1



Kerberos Protocol – 2 (Simplified)

1. $C \rightarrow AS$: $TGS \| N_C$
2. $AS \rightarrow C$: $\{K_{C,TGS} \| C\}_{K_{AS,TGS}} \| \{K_{C,TGS} \| N_C \| TGS\}_{K_{AS,C}}$
   (Note that the **first** part of message 2 is the **ticket granting ticket (TGT)** for the TGS)
3. $C \rightarrow TGS$: $SS \| N'_C \| \{K_{C,TGS} \| C\}_{K_{AS,TGS}} \| \{C \| T_1\}_{K_{C,TGS}}$
4. $TGS \rightarrow C$: $\{K_{C,SS} \| C\}_{K_{TGS,SS}} \| \{K_{C,SS} \| N'_C \| SS\}_{K_{C,TGS}}$
   (Note that the **first** part in message 4 is the **ticket** for the server S).
5. $C \rightarrow SS$: $\{K_{C,SS} \| C\}_{K_{TGS,SS}} \| \{C \| T_2\}_{K_{C,SS}}$
6. $SS \rightarrow C$: $\{T_3\}_{K_{C,SS}}$

Kerberos Drawback

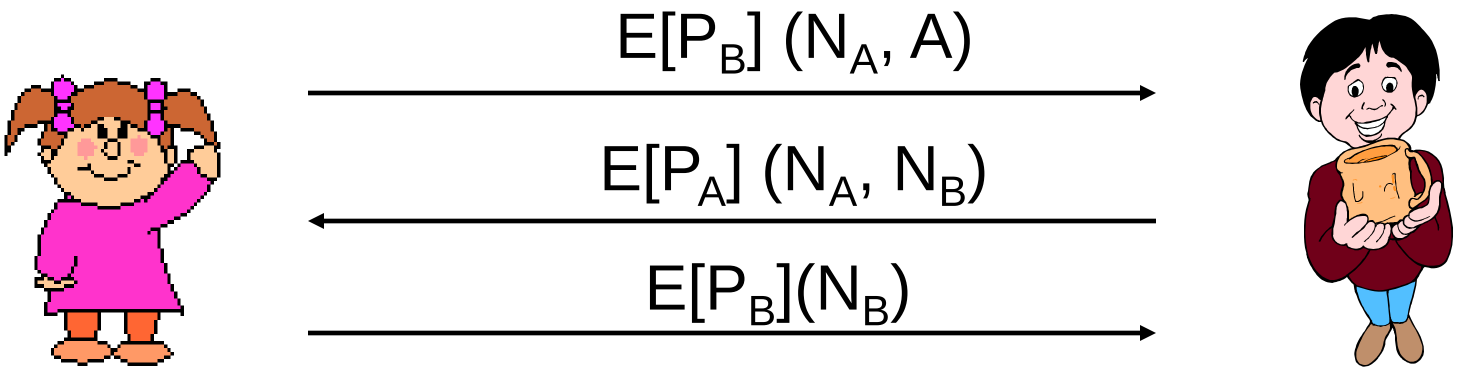
- Highly trusted TTP: KS
  - Malicious KS can silently eavesdrop in any communication
- Single point of failure:
- Security partially depends on tight clock synchronization.
- Useful primarily inside an organization
  - Does it scale to Internet? What is the main difficulty?

Key Establishment by Means of Public Key Encryption

- Often use public-key certificates
- Require off-line Trusted Third Party in the form of CA

Needham-Schroeder Public Key Protocol

- Setup: A and B both have each other's public key
- Goal: mutual entity authentication and authenticated key establishment



P_A and P_B are public keys, N_A and N_B are nounces that can be used to derive a session key.

Lowe's Attack on Needham-Schroeder Public-key Protocol [95]

The intruder can convince B that it is A, when A communicates with I.

$A \rightarrow I$: $E[P_I]$ (N_A, A)

$I \rightarrow B$: $E[P_B]$ (N_A, A)

$I \leftarrow B$: $E[P_A]$ (N_A, N_B)

$A \leftarrow I$: $E[P_A]$ (N_A, N_B)

$A \rightarrow I$: $E[P_I]$ (N_B)

$I \rightarrow B$: $E[P_B]$ (N_B)

How to fix this?

Fix: add B's name the second message

Public Keys and Trust

Public Key: P_A
Secret key: S_A

Public Key: P_B
Secret key: S_B

How are public keys stored?

How to obtain the public key?

How does Bob know or 'trusts' that P_A is Alice's public key?

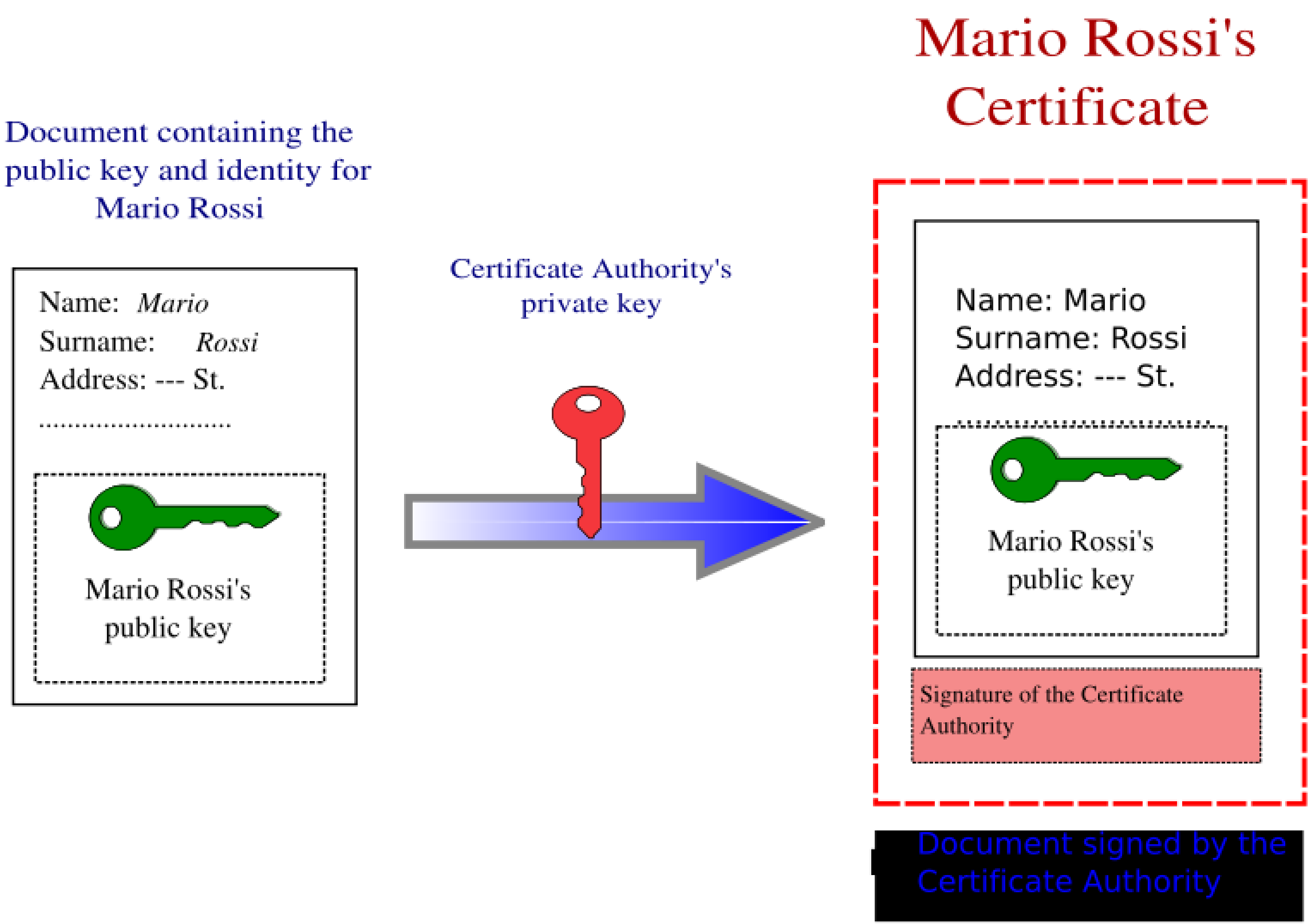
Distribution of Public Keys

- **Public announcement**: users distribute public keys to recipients or broadcast to community at large.
- **Publicly available directory**: can obtain greater security by registering keys with a public directory.
- Both approaches have problems, and are vulnerable to forgeries

Public-Key Certificates

- A certificate binds identity (or other information) to public key
- Contents digitally signed by a trusted Public-Key or Certificate Authority (CA)
  - Can be verified by anyone who knows the public-key authority's public-key.
- For Alice to send an encrypted message to Bob, obtains a certificate of Bob's public key

Public Key Certificates

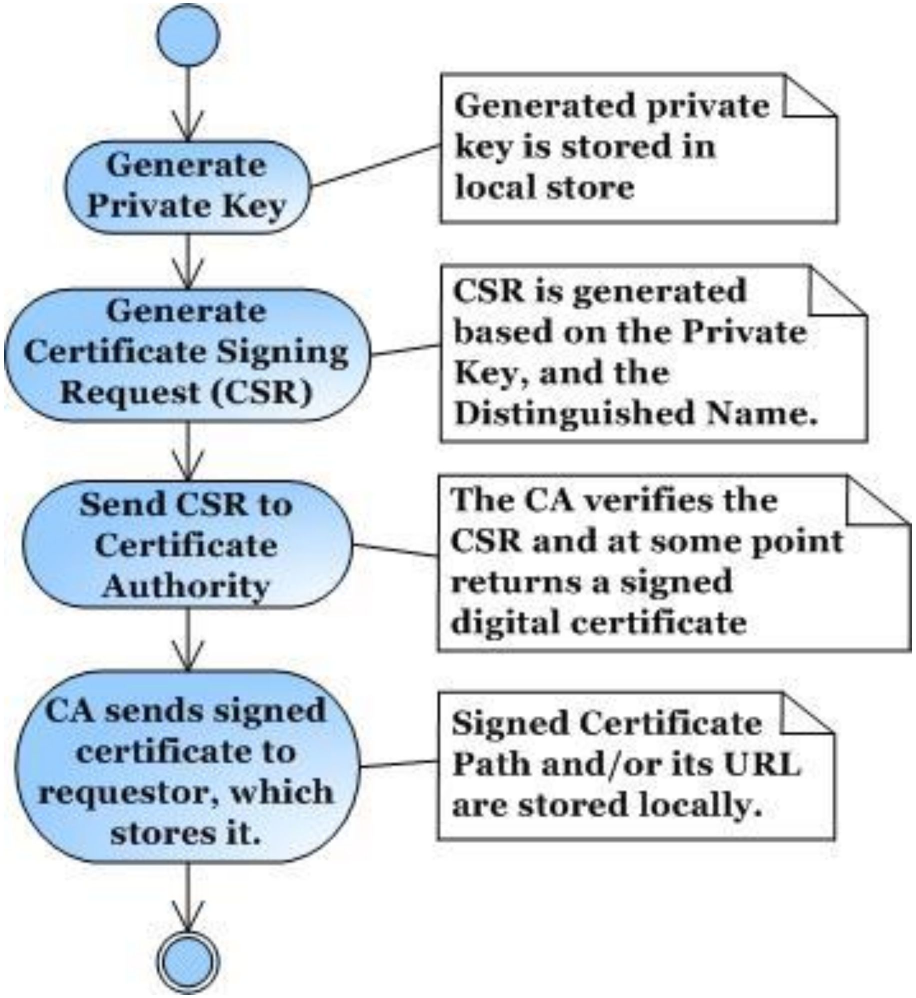


X.509 Certificates

- Part of X.500 directory service standards.
  - Started in 1988
- Defines framework for authentication services:
  - Defines that public keys stored as **certificates** in a public directory.
  - Certificates are **issued and signed** by an entity called **certification authority (CA).**
- Used by numerous applications: SSL, IPSec, SET
- Example: see certificates accepted by your browser

How to Obtain a Certificate?

- Define your own CA (use openssl or Java Keytool)
  - Certificates unlikely to be accepted by others
- Obtain certificates from one of the vendors: VeriSign, Thawte, and many others



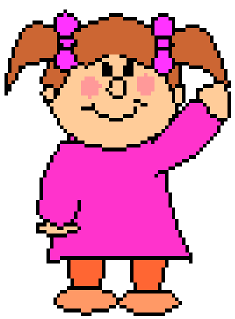
CAs and Trust

- Certificates are trusted if signature of CA verifies
- Chain of CA's can be formed, head CA is called root CA
- In order to verify the signature, the public key of the root CA should be obtain.
- TRUST is centralized (to root CA's) and hierarchical
- What bad things can happen if the root CA system is compromised?
- How does this compare with the TTP in Needham/Schroeder protocol?

Key Agreement: Diffie-Hellman Protocol

Key agreement protocol, both A and B contribute to the key

Setup: p prime and g generator of Z_p^*, p and g public.



$g^a \bmod p$ →

← $g^b \bmod p$

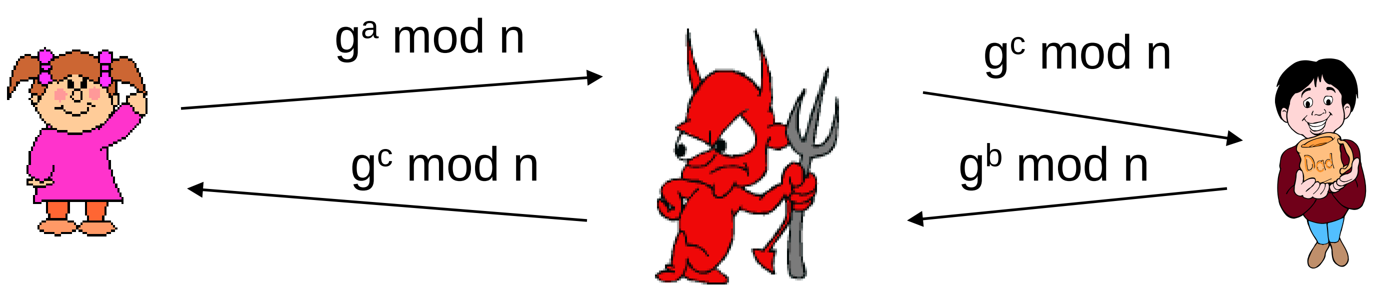
Pick random, secret (a)
Compute and send $g^a \bmod p$

$K = (g^b \bmod p)^a = g^{ab} \bmod p$

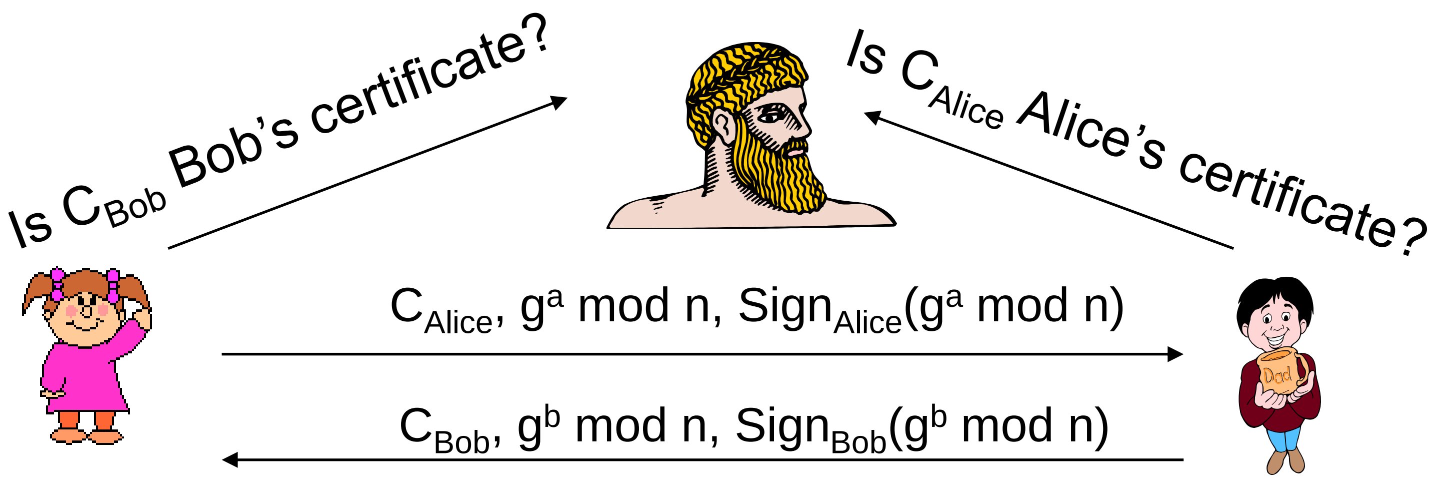
Pick random, secret (b)
Compute and send $g^b \bmod p$

$K = (g^a \bmod p)^b = g^{ab} \bmod p$

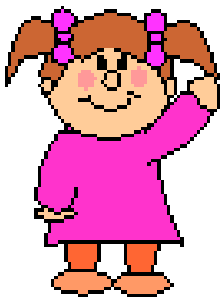
Authenticated Diffie-Hellman



Alice computes g^{ac} mod n and Bob computes g^{bc} mod n !!!



MTI



gx mod p →

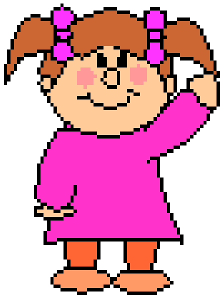
← gy mod p

$k = (gy)aPK_bx = gya\ gbx = gya+bx$

$k = (gx)bPK_ay$

- a and b are the private keys of A and B
- g^a and g^b are public keys of A and B
- Secure against passive attacks only
- Provides mutual (implicit) key authentication but neither key confirmation nor entity authentication

Station-to-Station (STS)



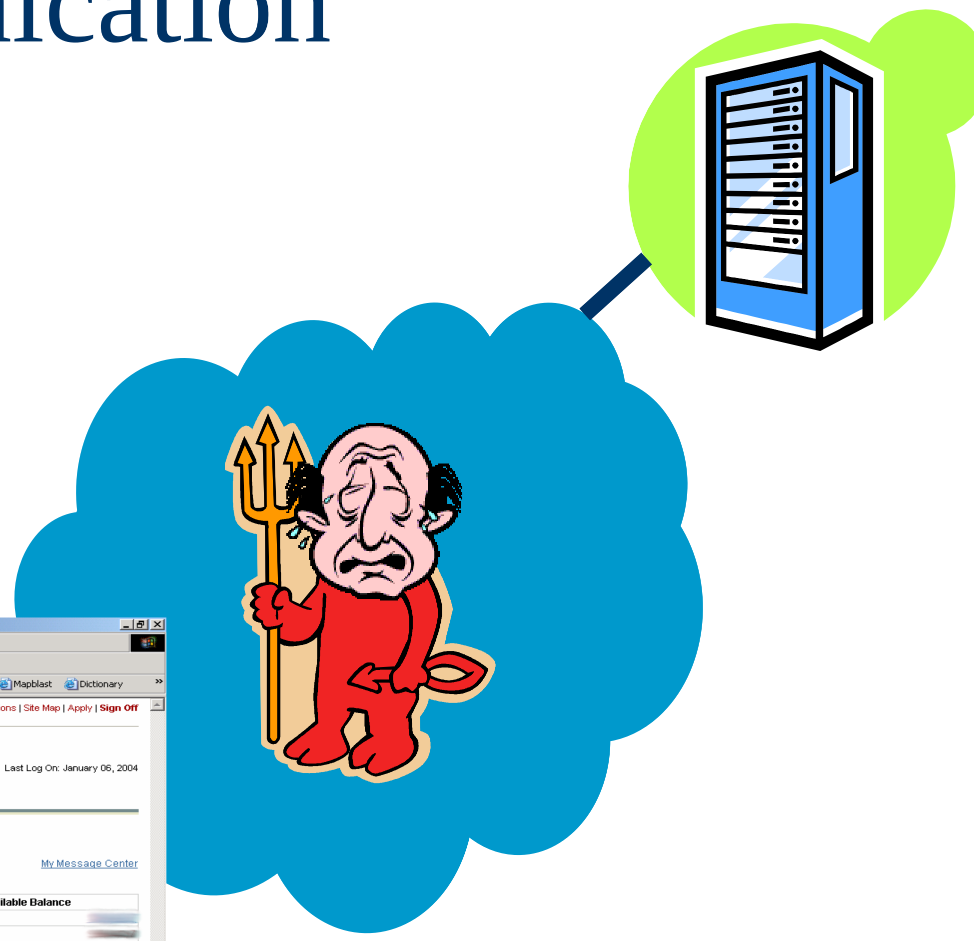
$g^x \bmod p$ →

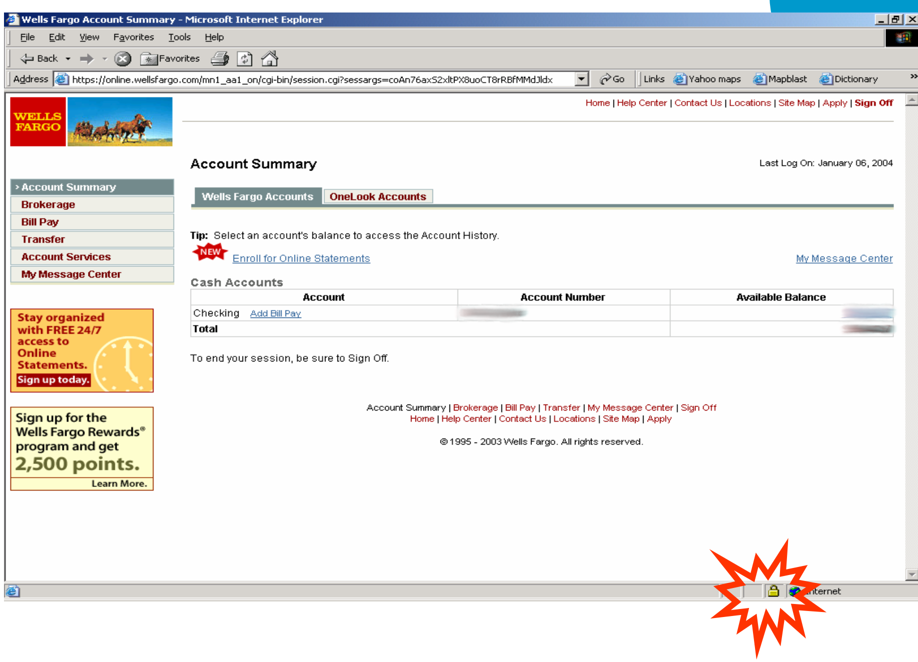
← $g^y \bmod p$, $E_k(Sign_B(g^y, g^x))$

$E_k(Sign_A(g^x, g^y))$ →

- where $k=(g^x)^y \bmod p$
- Provides mutual entity authentication

Secure communication





Transport Layer Security (TLS)

- Predecessors: Secure socket layer (SSL): Versions 1.0, 2.0, 3.0
- TLS 1.0 (SSL 3.1); Jan 1999
- TLS 1.1 (SSL 3.2); Apr 2006
- TLS 1.2 (SSL 3.3); Aug 2008
- Standard for Internet security
  - Originally designed by Netscape
  - Goal: "... provide privacy and reliability between two communicating applications"
- Two main parts
  - Handshake Protocol
    - Establish shared secret key using public-key cryptography
    - Signed certificates for authentication
  - Record Layer
    - Transmit data using negotiated key, encryption function

Usage of SSL/TLS

- Applied on top of transport layer (typically TCP)
- Used to secure HTTP (HTTPS), SMTP, etc.
- One or both ends can be authenticated using public key and certificates
  - Typically only the server is authenticated

- Client & server negotiate a cipher suite, which includes
  - A key exchange algorithm, e.g., RSA, Diffie-Hellman, SRP, etc.
  - An encryption algorithm, e.g., RC4, Triple DES, AES, etc.
  - A MAC algorithm, e.g., HMAC-MD5, HMC-SHA1, etc.

Viewing HTTPS web sites

- Browser needs to communicate to the user the fact that HTTPS is used
  - E.g., a golden lock indicator on the bottom or on the address bar
  - Check some common websites
  - When users correctly process this information, can defeat phishing attacks
  - Security problems exist
    - People don't know about the security indicator
    - People forgot to check the indicator
    - Browser vulnerabilities enable incorrect indicator to be shown
    - Use confusing URLs, e.g.,
      - https:// homebanking.purdueefcu.com@host.evil.com/
    - Stored certificate authority info may be changed

Next Lecture

- How Crypto Fails in Real World