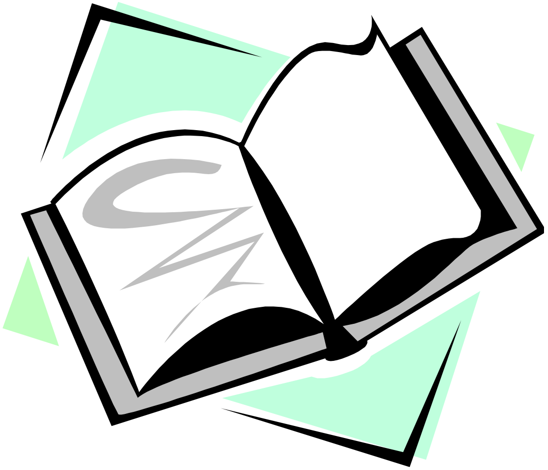
Cryptography
CS 555

Topic 16: Key Management and The Need for Public Key Cryptography

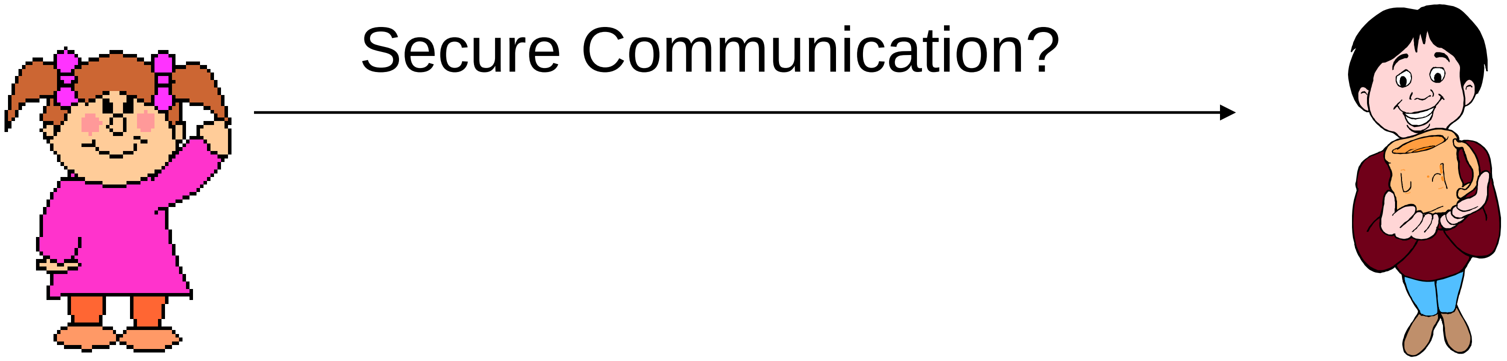
Outline and Readings

- Outline
  - Private key management between two parties
  - Key management with multiple parties
  - Public key cryptography



- Readings:
  - Katz and Lindell: Chapter 9

Need for Key Establishment



- When Alice and Bob share secret keys, they can communicate securely.
- **How to establish the shared key?**
- **How to refresh it (not a good idea to encrypt a lot of data with the same key)**

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

Long-Term Key vs. Session Key

- **Session key**: temporary key, used for a short time period.
  - Assumed to be compromisible after some time
- **Long-term key**: used for a long term period, public/private keys are typically long-term.
- 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

Basic Key Transport Protocol

- Assumes a long term symmetric key K shared between A and B
- Basic: A chooses a random r_A and sends it encrypted to B; A and B use it for the next session

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

  – Subject to replay attack: when attacker replays the message to B, B will be using an old session key r_A ; defeating the purpose of using session keys
- Enhancements to prevent replay: uses time t_A new key is r_A

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

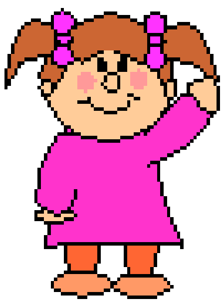
- Key transport with challenge/response:

$$A \leftarrow B: n_B$$

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

Authenticated Key Exchange Protocol 2 (AKEP2)

- Setup: A and B share long-term keys K and K'
- MAC_K is a MAC
- $F_{K'}$ is a pseudo-random permutation (a block cipher)
- Both A and B compute session key = $F_{K'}(r_B)$



A → B: r_A

B → A: $(B, A, r_A, r_B), MAC_K(B, A, r_A, r_B)$

A → B: $(A, r_B), MAC_K(A, r_B)$

Protocol ensures that r_B is a fresh random number chosen by B, intended to use with A for this session.

Key Agreement among Multiple Parties

- For a group of N parties, every pair needs to share a different key
  - Needs to establish N(N-1)/2 keys, which are too many
- Solution: Uses a Key Distribution Center (KDC), which is a central authority, a.k.a., Trusted Third Party (TTP)
  - Every party shares a key with a central server.
  - In an organization with many users, often times already every user shares a secret with a central TTP, e.g., password for an organization-wide account

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: Mutual entity authentication between A and B; key establishment
- 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)

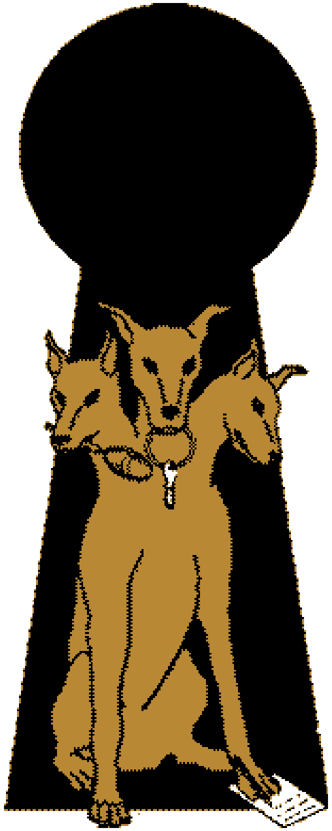
$A \rightarrow B$: E[k] (N_B-1) (5)

What bad things can happen if there is no N_A?

Another subtle flaw in Step 3.

Kerberos

- Implement 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: two versions, Version 4 and Version 5 (specified as RFC1510)
- http://web.mit.edu/kerberos/www
- Used in many systems, e.g., Windows 2000 and later as default authentication protocol

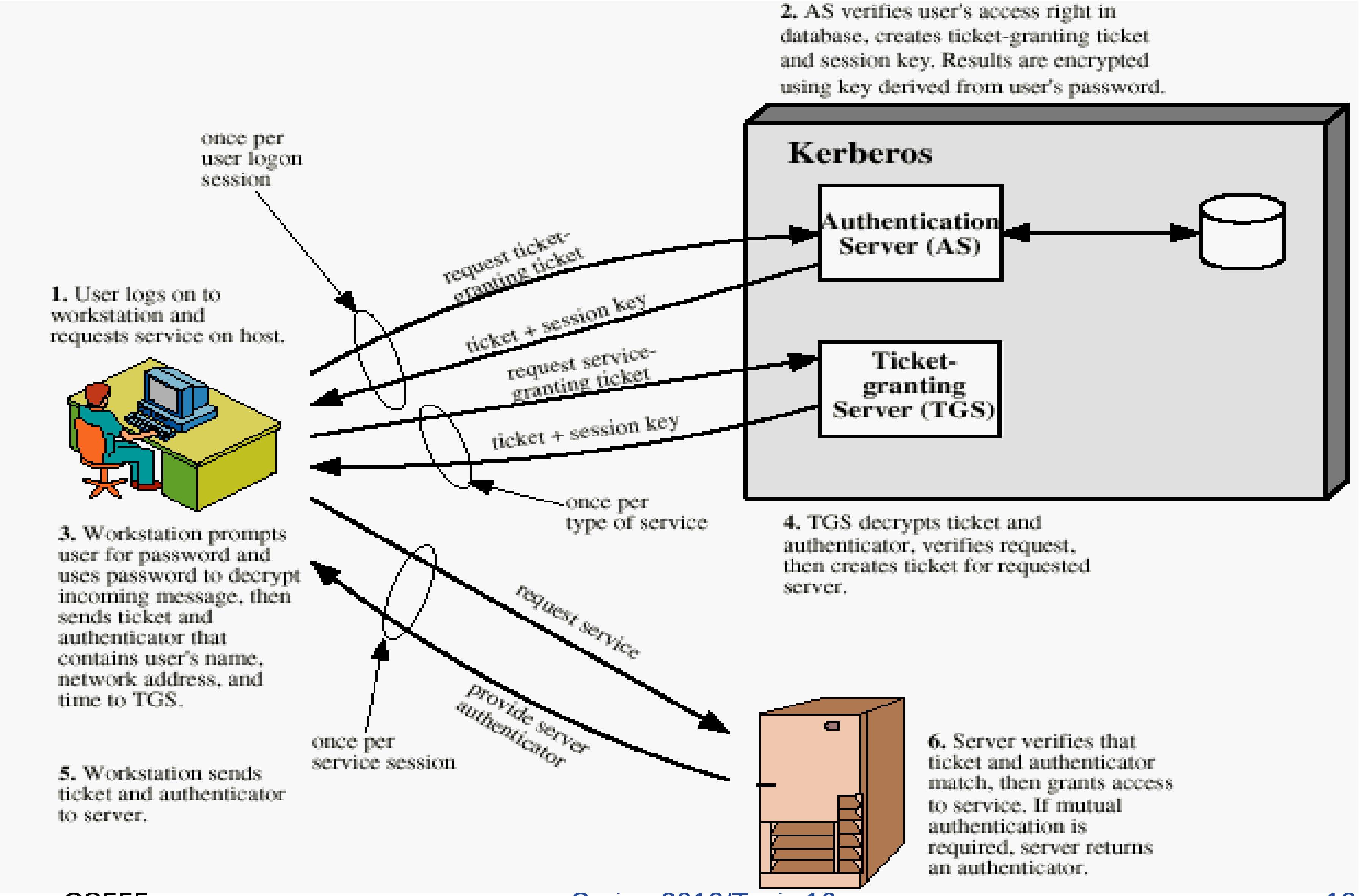


Kerberos Overview

- One issue of Needham-Schroeder
  - Needs the key each time a client talks with a service
  - Either needs to store the secret, or ask user every time
- Solution: Separates TTP into an AS and a TGS.
- The client authenticates to AS using a long-term *shared secret* and receives a TGT.
  - supports single sign-on
- Later the client can use this TGS to get additional tickets from TGS without resorting to using the shared secret. These tickets can be used to prove authentication to SS.

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

Overview of Kerberos



Kerberos Drawback

- Single point of failure:
  - Requires online Trusted Third Party: Kerberos server
- Security partially depends on tight clock synchronization. Convenience requires loose clock synchronization
  - Use timestamp in the protocol
  - Hosts typically run Network Time Protocol to synchronize clocks
- Useful primarily inside an organization
  - Does it scale to Internet? What is the main difficulty?

Concept of Public Key Encryption

- Each party has a pair (K, K^{-1}) of keys:
  - K is the **public** key, and used for encryption
  - K^{-1} is the **private** key, and used for decryption
  - Satisfies $\mathbf{D}_{K^{-1}}[\mathbf{E}_K[M]] = M$
- Knowing the public-key K, it is computationally infeasible to compute the private key K^{-1}
  - How to check (K,K^{-1}) is a pair?
  - Offers only computational security. Secure PK Encryption impossible when P=NP, as deriving K^{-1} from K is in NP.
- The public-key K may be made publicly available, e.g., in a publicly available directory
  - Many can encrypt, only one can decrypt
- Public-key systems aka *asymmetric* crypto systems

Public Key Cryptography Early History

- Proposed by Diffie and Hellman, documented in "New Directions in Cryptography" (1976)
  1. Public-key encryption schemes
  2. Key distribution systems
     - Diffie-Hellman key agreement protocol
  3. Digital signature

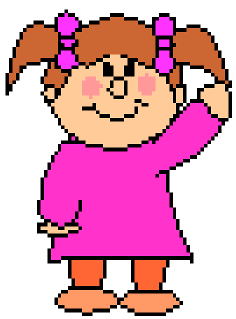
- Public-key encryption was proposed in 1970 in a classified paper by James Ellis
  - paper made public in 1997 by the British Governmental Communications Headquarters

- Concept of digital signature is still originally due to Diffie & Hellman

Diffie-Hellman Key Agreement Protocol

Not a Public Key Encryption system, but can allow A and B to agree on a shared secret in a public channel (with passive, i.e., eavesdropping adversaries)

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 $g^b \bmod p$

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

Diffie-Hellman

- Example: Let p=11, g=2, then

a	1	2	3	4	5	6	7	8	9	10	11
g^a	2	4	8	16	32	64	128	256	512	1024	2048
g^a mod p	2	4	8	5	10	9	7	3	6	1	2

A chooses 4, B chooses 3, then shared secret is

$$(2^3)^4 = (2^4)^3 = 2^{12} = 4 \pmod{11}$$

Adversaries sees $2^3=8$ and $2^4=5$, needs to solve one of $2^x=8$ and $2^y=5$ to figure out the shared secret.

Three Problems Believed to be Hard to Solve

- Discrete Log (DLG) Problem: Given <g, h, p>, computes a such that $g^a = h \bmod p$.
- Computational Diffie Hellman (CDH) Problem: Given <g, $g^a \bmod p$, $g^b \bmod p$> (without a, b) compute $g^{ab} \bmod p$.
- Decision Diffie Hellman (DDH) Problem: distinguish (g^a,g^b,g^{ab}) from (g^a,g^b,g^c), where a,b,c are randomly and independently chosen

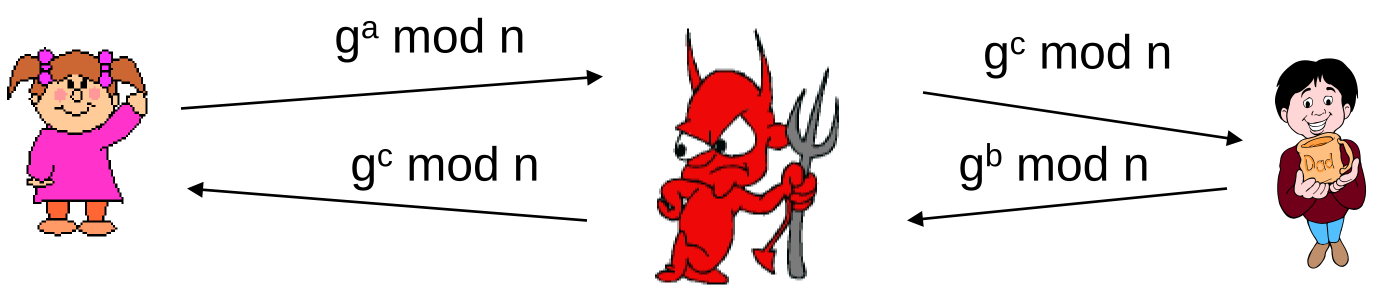
- If one can solve the DL problem, one can solve the CDH problem. If one can solve CDH, one can solve DDH.

Assumptions

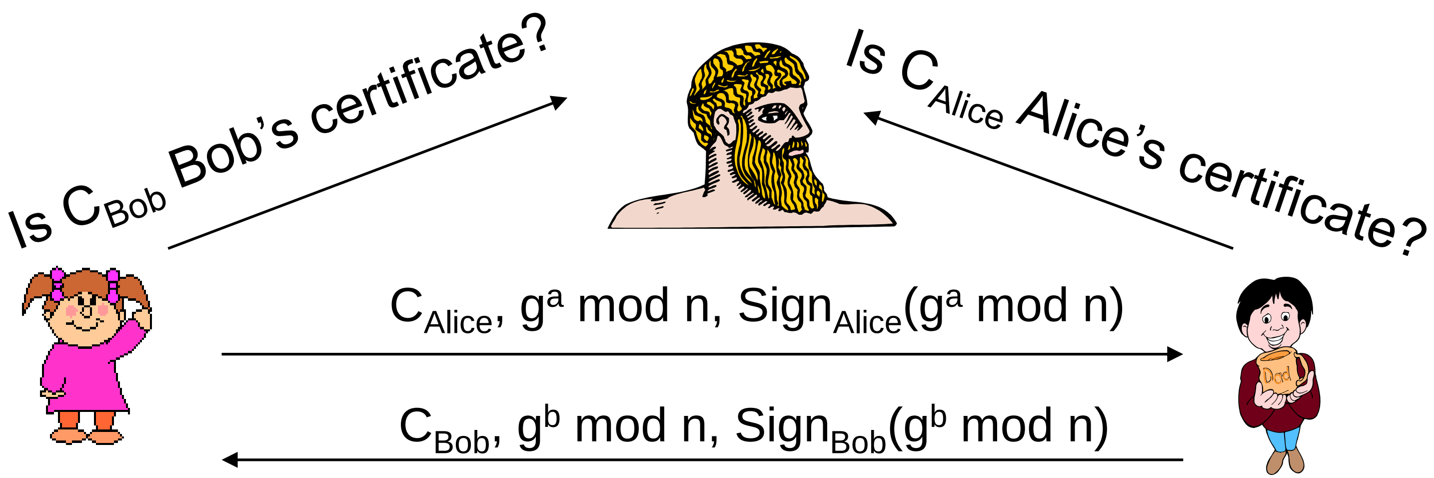
- DDH Assumption: DDH is hard to solve.
- CDH Assumption: CDH is hard to solve.
- DLG Assumption: DLG is hard to solve

- DDH assumed difficult to solve for large p (e.g., at least 1024 bits).

Authenticated Diffie-Hellman



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



Coming Attractions …

- Textbook RSA encryption, number theory
- Reading: Katz & Lindell: 7.2