Key Management and Identity

CS 458: Information Security

Kevin Jin

Administrivia

- Lab 1 due on 9/27
- Homework 3 solution and Homework 4 released

Reading Material

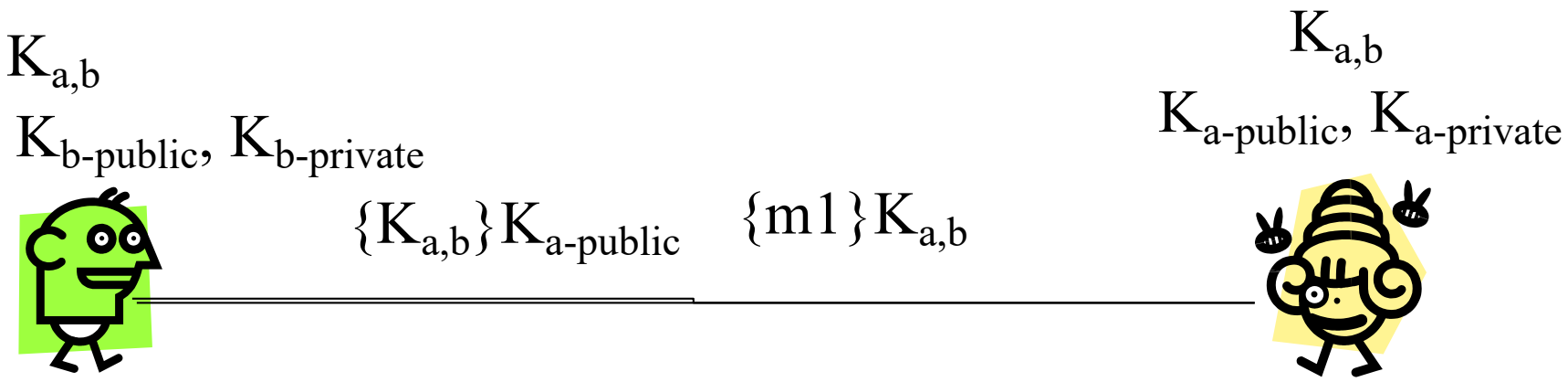
- Chapters 2.4, 2.5 and 23.1 - 3 from text
- Handbook of Applied Cryptography
  - http://www.cacr.math.uwaterloo.ca/hac/
  - Chapter 5 for information on Pseudorandom sequences
  - Chapter 12 for Needham-Schroeder protocol
- The Gnu Privacy Handbook
  - http://www.gnupg.org/gph/en/manual.html

Overview

- Key Exchange & Random Numbers
- Key/Identity Management
  - Kerberos
  - Digital Signatures & PKI
    - Hierarchical – X.509
    - Web of Trust

Session and Interchange Keys

- Long lived Interchange Keys only exist to boot strap
- Short lived session keys used for bulk encryption



Picking Good Session Keys

- Goal: generate keys that are difficult to guess
- Problem statement: given a set of *K* potential keys, choose one randomly
  - Equivalent to selecting a random number between 0 and *K*–1 inclusive
  - Uniform distribution (use entire space)
  - Independence (should not be able to predict next selection)
- Why is this hard?
  - Actually, numbers are usually *pseudo-random*, that is, generated by an algorithm

What is "Random"?

- *Sequence of cryptographically random numbers*: a sequence of numbers n_1, n_2, ... such that for any integer $k > 0$, an observer cannot predict n_k even if all of n_1, ..., n_{k-1} are known
  - Best: physical source of randomness
    - Random pulses
    - Electromagnetic phenomena
    - Characteristics of computing environment such as disk latency
    - Ambient background noise

What is "Pseudorandom"?

- *Sequence of cryptographically pseudorandom numbers*: sequence of numbers intended to simulate a sequence of cryptographically random numbers but **generated by an algorithm**
  - Very difficult to do this well
    - Linear congruential generators [$n_k = (an_{k-1} + b) \bmod n$] broken
    - Polynomial congruential generators [$n_k = (a_j n_{k-1}^{\ j} + \ldots + a_1 n_{k-1}\ a_0) \bmod n$] broken too
    - Here, "broken" means next number in sequence can be determined

Best Pseudorandom Numbers

- *Strong mixing function*: function of 2 or more inputs with each bit of output depending on some nonlinear function of all input bits
  - Examples: DES, MD5, SHA-1, avalanche effect
  - Use on UNIX-based systems:

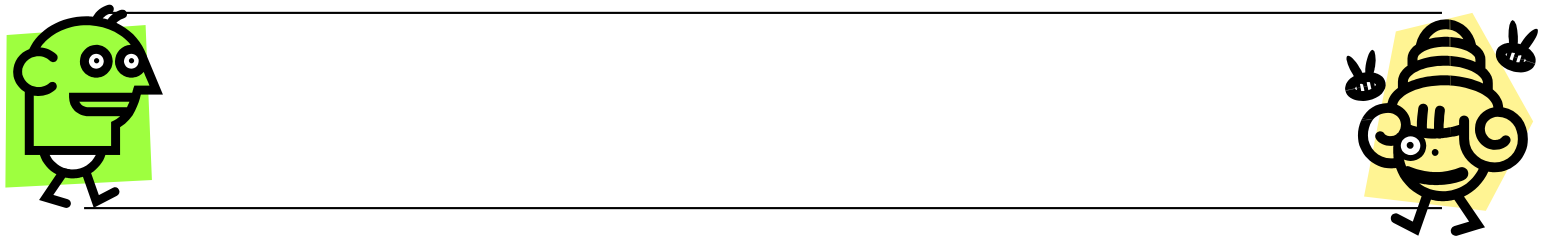
```
(date; ps gaux) | md5
```

  where "ps gaux" lists all information about all processes on system

Separate Channel

- Ideally you have separate secure channel for exchanging Interchange keys
  - Direct secret sharing grows at N^2

Telephone, separate data network, ESP, sneaker net

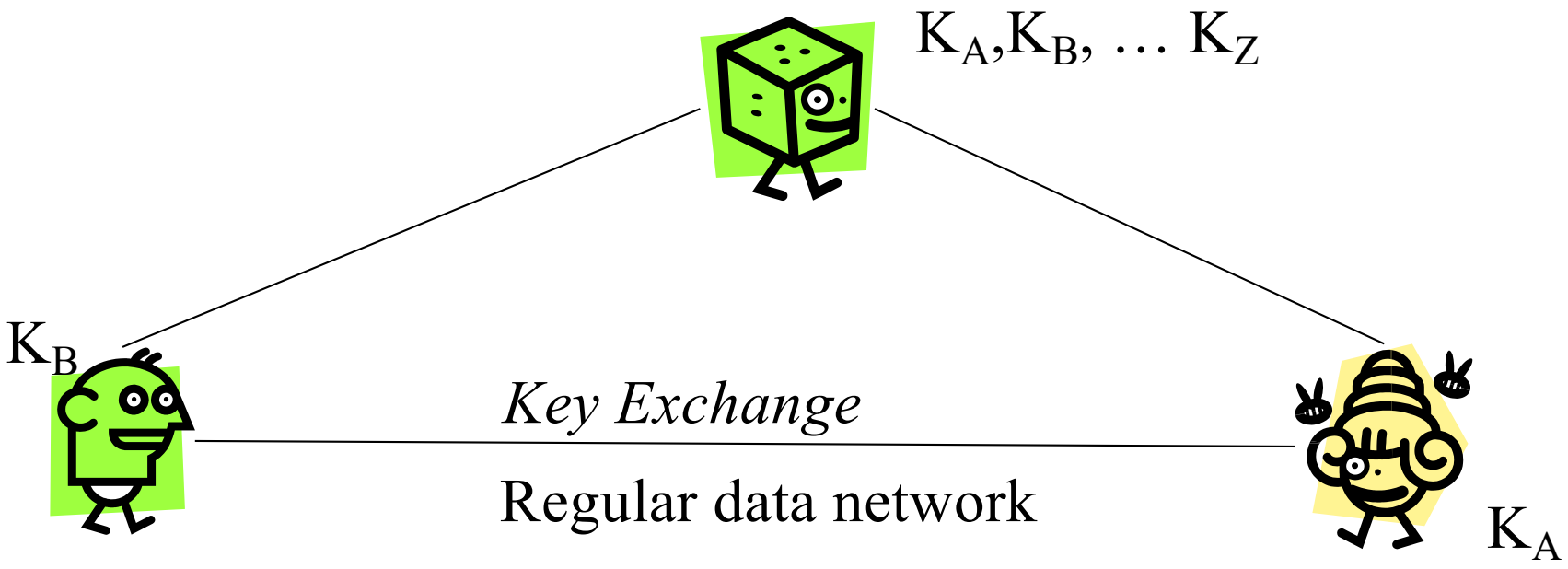


Regular data network

So how can we solve that?

Shared Channel: Trusted Third Party

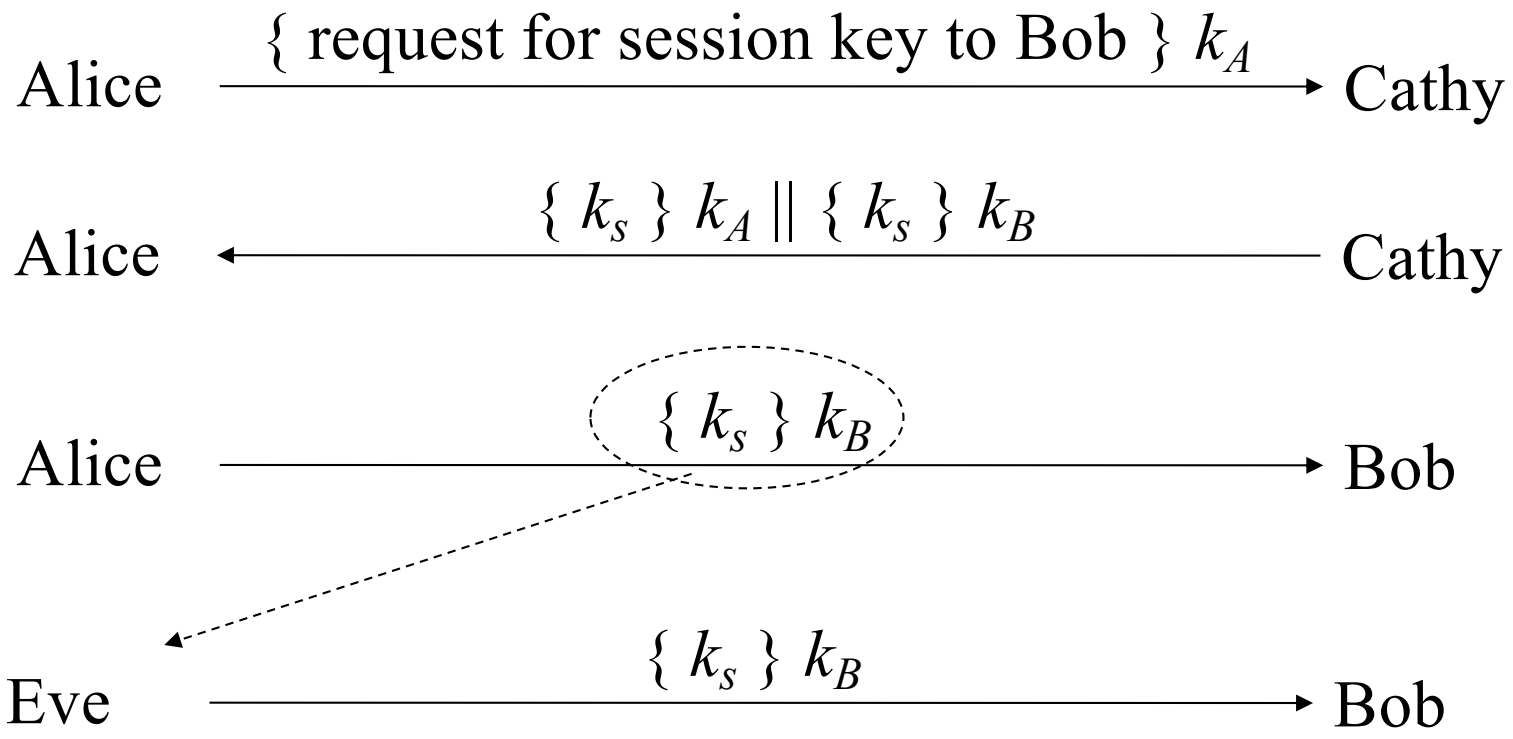
- Generally separate channel is not practical
  - No trustworthy separate channel
  - Want to scale linearly with additional users



Classical Key Exchange

- Bootstrap problem: how do Alice, Bob begin?
  - Alice can't send it to Bob in the clear!
- Assume trusted third party, Cathy
  - Alice and Cathy share secret key k_A
  - Bob and Cathy share secret key k_B
- Use this to exchange shared key k_s

Simple Protocol

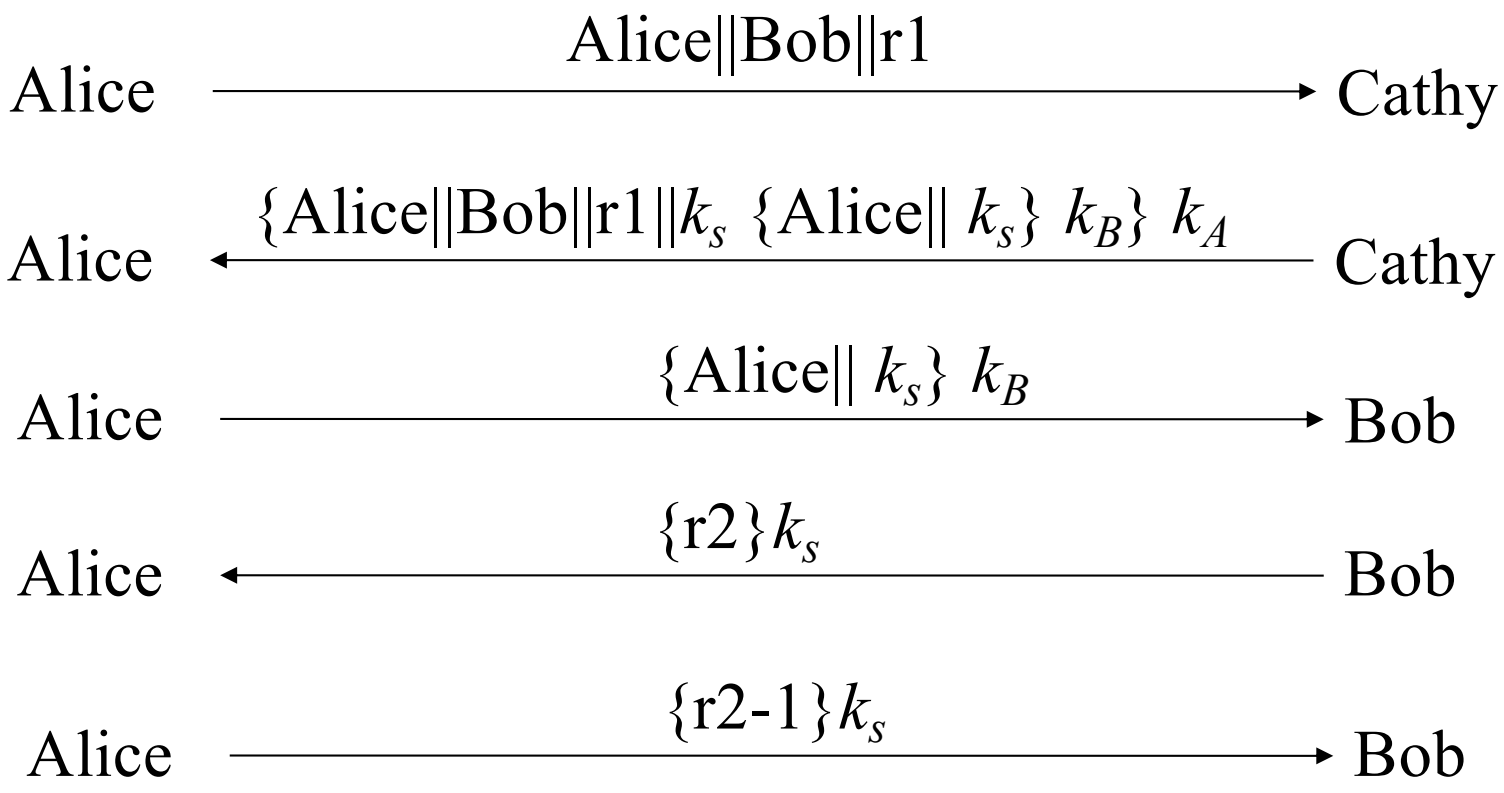


What is the problem?

Problems

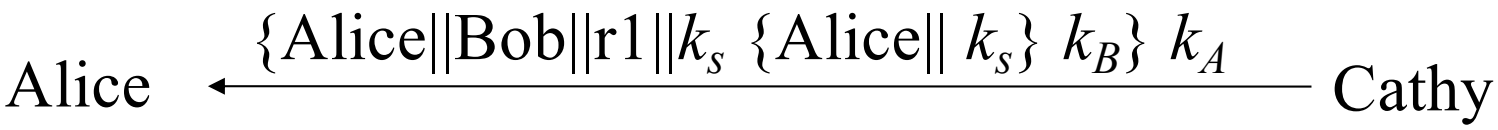
- How does Bob know he is talking to Alice?
  - Replay attack
    - Eve records message from Alice to Bob, later replays it
    - Bob may think he's talking to Alice, but he isn't
  - Session key reuse
    - Eve replays message from Alice to Bob, so Bob re-uses session key
- Protocols must provide authentication and defence against replay

Needham-Schroeder



Argument: Alice talking to Bob

- Second message
  - Encrypted using key only Alice and Cathy know
    - So Cathy encrypted it
  - Response to first message
    - As r_1 in it matches r_1 in first message

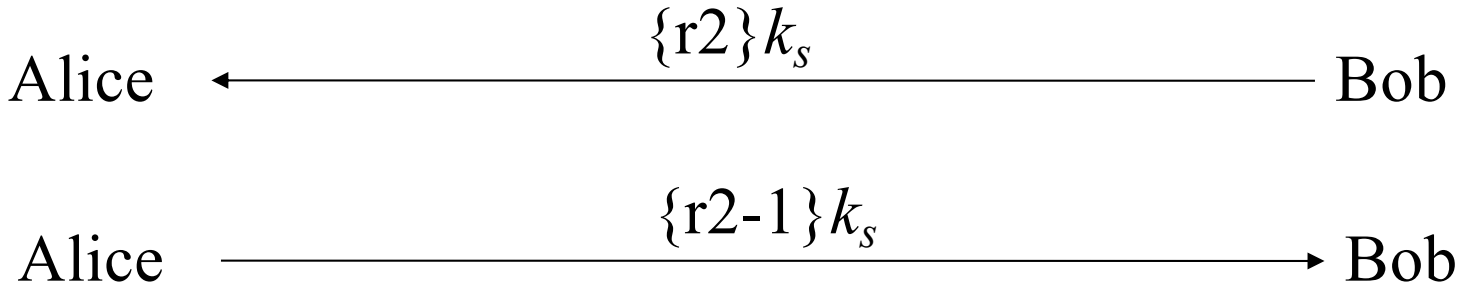


Argument: Alice talking to Bob

- Third message
  - Alice knows only Bob can read it
    - As only Bob can derive session key from message
  - Encrypted using key only Bob and Cathy know
    - So Cathy encrypted it
  - Names Alice, session key
    - Cathy provided session key, says Alice is other party

Argument: Bob talking to Alice

- Fourth and fifth messages
  - Uses session key to determine if it is replay from Eve
    - If not, Alice will respond correctly in fifth message
    - If so, Eve can't decrypt r_2 and so can't respond, or responds incorrectly



Kerberos

Kerberos

- In Greek mythology, Kerberos is a three-headed dog that guards entrance to Hades
  - "Wouldn't it make more sense to guard the exit?"
- In security, Kerberos is an authentication protocol based on symmetric key crypto
  - Originated at MIT
  - Based on work by Needham and Schroeder
  - Relies on a **Trusted Third Party (TTP)**

Motivation for Kerberos

- Authentication using public keys
  - N users -> N key pairs
- Authentication using symmetric keys
  - N users requires (on the order of) N^2 keys
- Symmetric key case **does not scale**
- Kerberos based on symmetric keys but only requires N keys for N users
  - \- Security depends on TTP
  - \+ No PKI is needed

Kerberos KDC

- Kerberos **Key Distribution Center** or **KDC**
  - KDC acts as the TTP
  - TTP is trusted, so it must not be compromised
- KDC shares symmetric key K_A with Alice, key K_B with Bob, key K_C with Carol, etc.
- A master key K_{KDC} known ***only*** to KDC
- KDC enables authentication, session keys
  - Session key for confidentiality and integrity
- In practice, crypto algorithm is DES

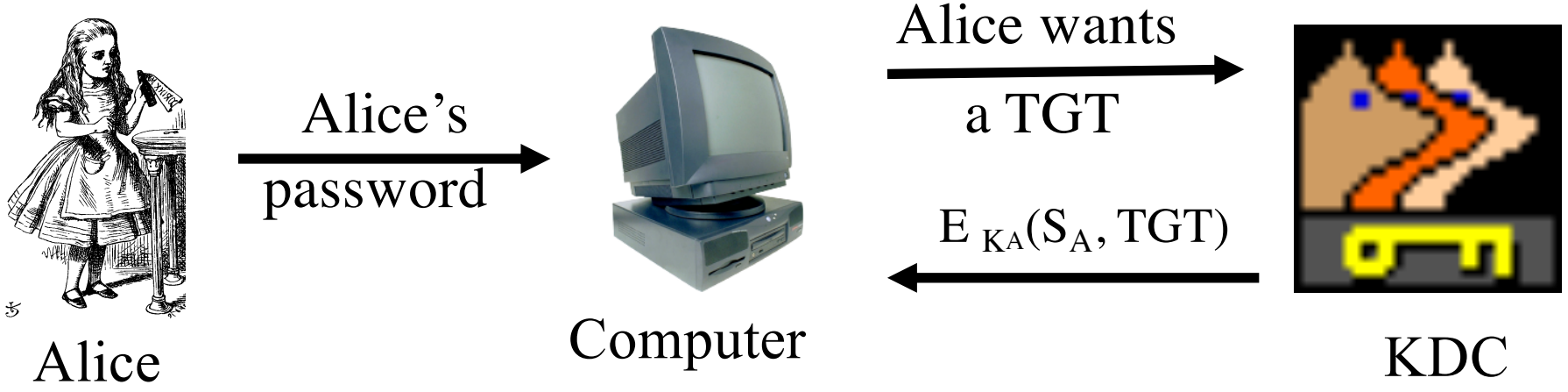
Kerberos Tickets

- KDC issue **tickets** containing info needed to access network resources
- KDC also issues **Ticket-Granting Tickets** or **TGTs** that are used to obtain tickets
- Each TGT contains
  - Session key
  - User's ID
  - Expiration time
- Every TGT is encrypted with K_{KDC}
  - So, TGT can only be read by the KDC

Kerberized Login

- Alice enters her password
- Then Alice's computer does following:
  - Derives K_A from Alice's password
  - Uses K_A to get TGT for Alice from KDC
- Alice then uses her TGT (credentials) to securely access network resources
- **Plus:** Security is transparent to Alice
- **Minus:** KDC ***must*** be secure - it's trusted!

Kerberized Login



- Key $K_A = h(\text{Alice's password})$
- KDC creates session key S_A
- Alice's computer decrypts S_A and TGT
  - Then it forgets K_A
- $TGT = E_{K_{KDC}}(\text{"Alice"}, S_A)$

Alice Requests "Ticket to Bob"

Alice

Talk to Bob

Computer

I want to talk to Bob

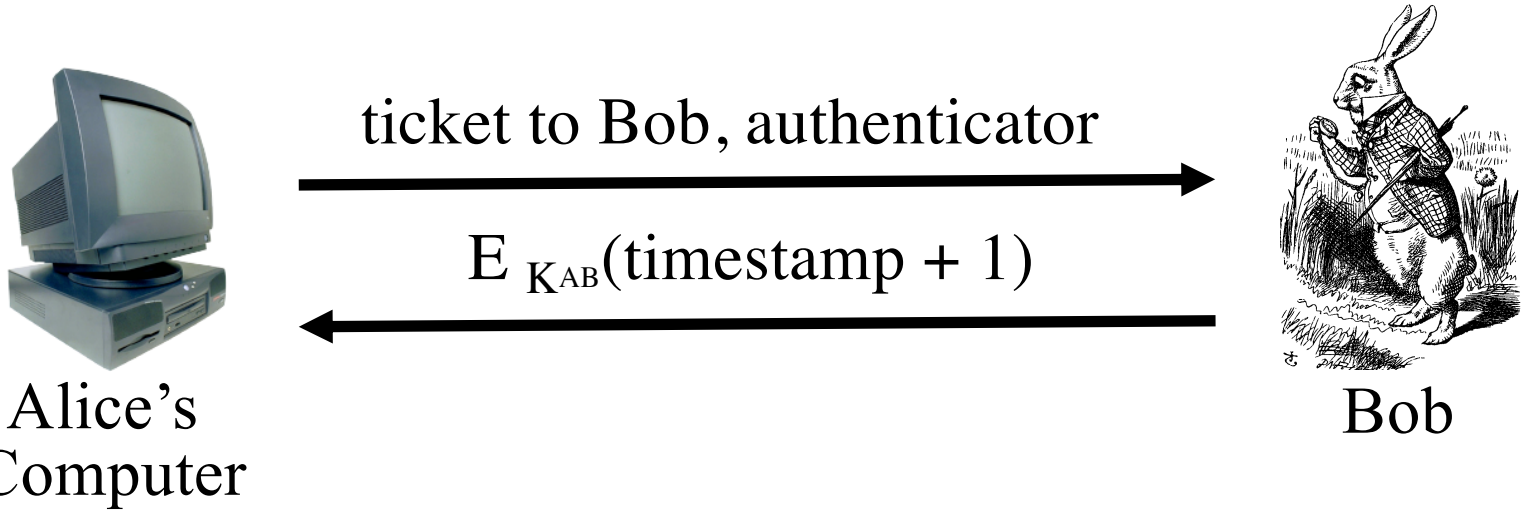
REQUEST

REPLY

KDC

- REQUEST = (TGT, authenticator)
  - authenticator = E_{S_A}(timestamp)
- REPLY = E_{S_A}("Bob", K_{AB}, ticket to Bob)
  - ticket to Bob = E_{K_B}("Alice", K_{AB})
- KDC gets S_A from TGT to verify timestamp

Alice Uses Ticket to Bob



- ticket to Bob = E_{K_B}("Alice", K_{AB})
- authenticator = $E_{K_{AB}}$(timestamp)
- Bob decrypts "ticket to Bob" to get K_{AB} which he then uses to verify timestamp

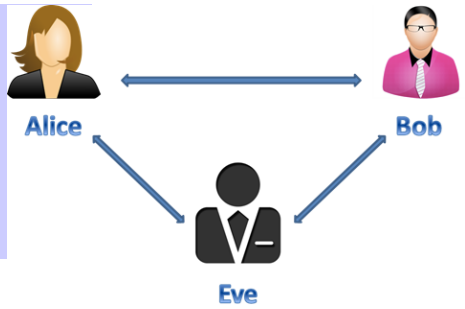
Kerberos

- Key S_A used in authentication
  - For confidentiality/integrity
- Timestamps for authentication and replay protection
- Recall, that timestamps…
  - Reduce the number of messages
    - like a nonce that is known in advance
  - But, "time" is a security-critical parameter

Kerberos Alternatives

- Could have Alice's computer remember password and use that for authentication
  - Then no KDC required
  - But hard to protect passwords
  - Also, does not scale
- Could have KDC remember session key instead of putting it in a TGT
  - Then no need for TGT
  - But **stateless** KDC is major feature of Kerberos

Man-in-the-Middle Attack



Alice —Send Bob's public key— *Eve intercepts request* → Cathy

Eve → send Bob's public key → Cathy

Eve ← e_B ← Cathy

Alice ← e_E ← Eve

Alice —$\{ k_s \} e_E$— *Eve intercepts message* → Bob

Eve → $\{ k_s \} e_B$ → Bob

Digital Signatures

- Alice wants Bob to know that she sent message, m
  - Sends digital signature along with message
  - m || $\{h(m)\}d_A$
- How would Bob verify signature?
- Could Eve intercept and change message?
- How does Bob know that Alice is the sender?

Cryptographic Key Infrastructure

- Goal: bind identity to key
- Classical: not possible as all keys are shared
  - Use protocols to agree on a shared key (see earlier)
- Public key: bind identity to public key
  - Crucial as people will use key to communicate with principal whose identity is bound to key
  - Erroneous binding means no secrecy between principals
  - Assume principal identified by an acceptable name

Certificates

- Create token (message) containing
  - Identity of principal (here, Alice)
  - Corresponding public key
  - Timestamp (when issued)
  - Other information (perhaps identity of signer)
  - Compute hash (message digest) of token

Hash encrypted by trusted authority (here, Cathy) using private key: called a "signature"

$$C_A = e_A \,||\, \text{Alice} \,||\, T \,||\, \{h(e_A \,||\, \text{Alice} \,||\, T)\}\, d_C$$

X.509 Certificates

- Some certificate components in X.509v3:
  - Version
  - Serial number
  - Signature algorithm identifier: hash algorithm
  - Issuer's name; uniquely identifies issuer
  - Interval of validity
  - Subject's name; uniquely identifies subject
  - Subject's public key
  - Signature: encrypted hash

Use

- Bob gets Alice's certificate
  - If he knows Cathy's public key, he can validate the certificate
    - Decrypt the encrypted hash using Cathy's public key
    - Re-compute hash from certificate and compare
    - Check validity
    - Is the principal Alice?
  - Now Bob has Alice's public key
- Problem: Bob needs Cathy's public key to validate certificate
  - That is, **secure distribution of public keys**
  - Solution: Public Key Infrastructure (PKI) using trust anchors called Certificate Authorities (CAs) that issue certificates

PKI Trust Models

- Single Global CA
  - Unmanageable
  - Who is the universally trusted root?
- Hierarchical CA
  - Tree of CA's
  - But still need to be rooted somewhere

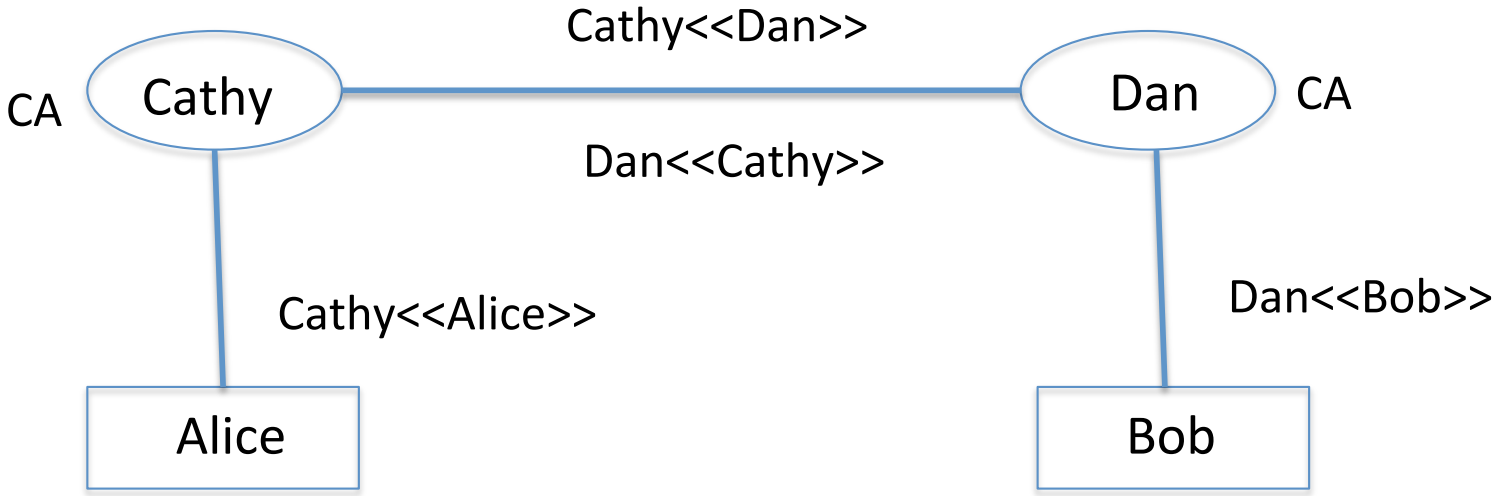
CA: Certificate Authorities

PKI Trust Models

- Hierarchical CAs with cross-certification
  - Multiple root CAs that are cross-certified
  - Cross-certification at lower levels for efficiency
- Web Model
  - Browsers come pre-configured with multiple trust anchor certificates
  - New certificates can be added
- Distributed (e.g., PGP)
  - No CA; instead, users certify each other to build a "web of trust"

Validation and Cross-Certifying

- Alice gets something signed by Bob, needs his public key
  - Certificate signed by Dan, Dan<<Bob>>
- Alice trusts Cathy who trusts Dan (Cathy<<Dan>>)
- Alice uses (known) public key of Cathy to validate Cathy<<Dan>>
- Alice uses validated public key of Dan to validate Dan<<Bob>>



Recent Root Certificate Issues

- Vast numbers of root certifiers in web browsers
- How strenuous is the background check of the certificate providers?
- How strong is the internal security of the certificate providers?
- What goes wrong with bad root certificates appear?

Key Points

- Tracking identity is important
  - Key negotiation
  - Digital signatures
- Managing the root of trust is a very hard practical problem