# Key Management

CS461/ECE422

Fall 2010

# Reading

- Handbook of Applied Cryptography
  http://www.cacr.math.uwaterloo.ca/hac/
  - Section 11.3.2 attack on RSA signature
  - Section 13.8.3 Key Escrow
- Chapter 10 in Computer Security: Art and Science

# Key Management Motivation

- Cryptographic security depends on keys
  - Size
  - Generation
  - Retrieval and Storage
- Example
  - House security system no good if key or code is under the mat

# Overview

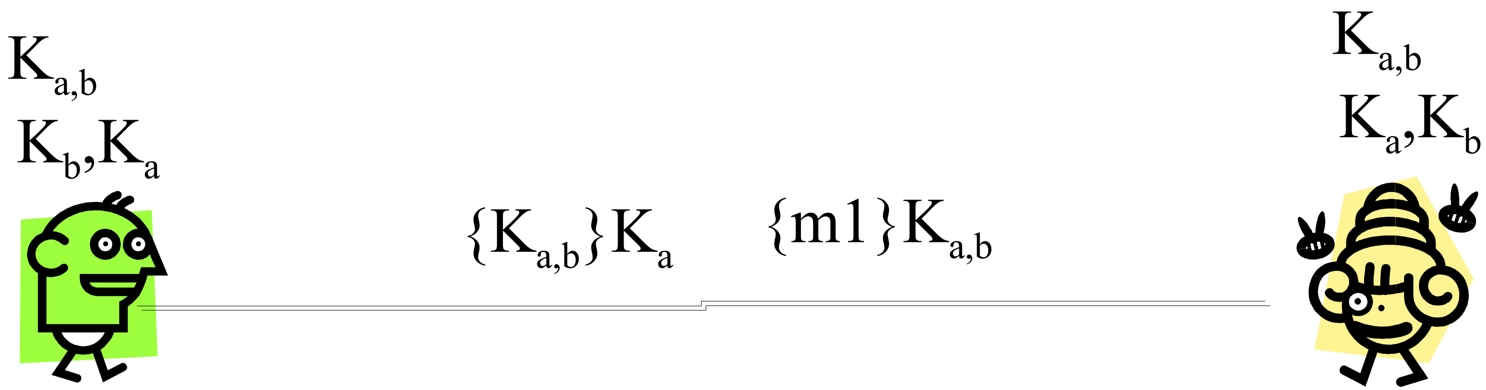
- Key Generation
- Key Exchange and management
  - Classical (symmetric)
  - Public/private
- Digital Signatures
- Key Storage

# Notation

- $X \rightarrow Y : \{ Z \| W \} k_{X,Y}$
  - $X$ sends $Y$ the message produced by concatenating $Z$ and $W$ encrypted by key $k_{X,Y}$, which is shared by users $X$ and $Y$
- $A \rightarrow T : \{ Z \} k_A \| \{ W \} k_{A,T}$
  - $A$ sends $T$ a message consisting of the concatenation of $Z$ encrypted using $k_A$, $A$'s key, and $W$ encrypted using $k_{A,T}$, the key shared by $A$ and $T$
- $r_1$, $r_2$ nonces (nonrepeating random numbers)

# Session and Interchange Keys

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



# Session and Interchange Keys

- Alice wants to send a message *m* to Bob
  - Assume public key encryption
  - Alice generates a random cryptographic key $k_s$ and uses it to encrypt *m*
    - To be used for this message *only*
    - Called a *session key*
  - She encrypts $k_s$ with Bob's public key $k_B$
    - $k_B$ encrypts all session keys Alice uses to communicate with Bob
    - Called an interchange *key*
  - Alice sends $\{ m \} k_s \| \{ k_s \} k_B$

# Benefits

- Limits amount of traffic encrypt with single key
  – Standard practice, to decrease the amount of traffic an attacker can obtain
- Prevents some attacks
  – Example: Alice will send Bob message that is either "BUY" or "SELL". Eve computes possible ciphertexts { "BUY" } $k_B$ and { "SELL" } $k_B$. Eve intercepts encrypted message, compares, and gets plaintext at once

# Key Generation

- 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
- Why is this hard: generating random numbers
  - 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:

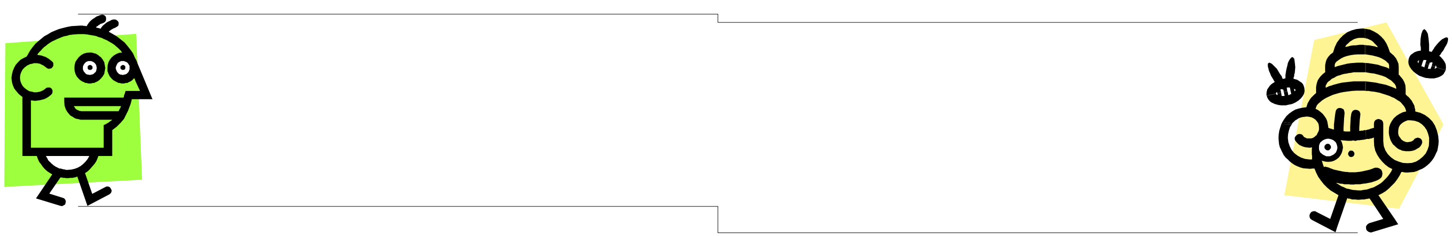
```
(date; ps gaux) | md5
```

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

# Separate Channel

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

Telephone, separate data network, ESP, sneaker net



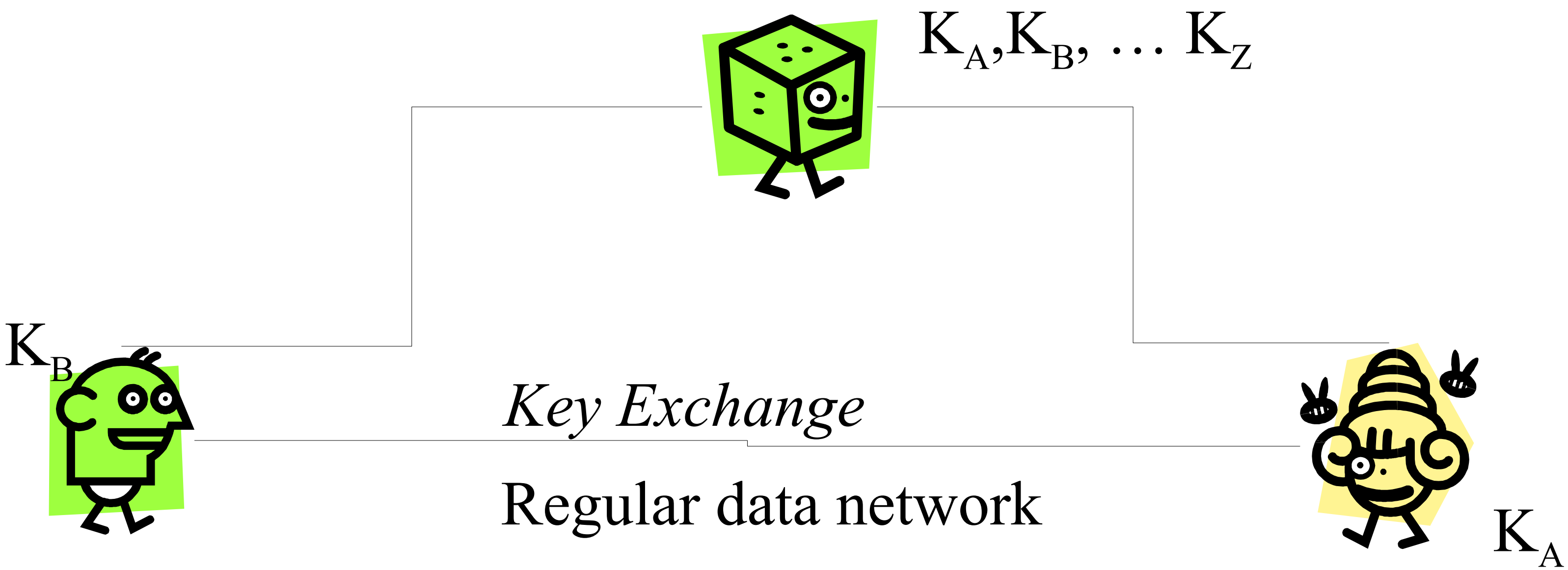
Regular data network

# Key Exchange Algorithms

- Goal: Alice, Bob get shared key
  - All cryptosystems, protocols publicly known
    - Only secret data is the keys
  - Anything transmitted is assumed known to attacker
    - Key cannot be sent in clear as attacker can listen in
  - Options
    - Key can be sent encrypted, or derived from exchanged data plus data not known to an eavesdropper (Diffie-Hellman)
    - Alice, Bob may trust third party

# Shared Channel: Trusted Third Party

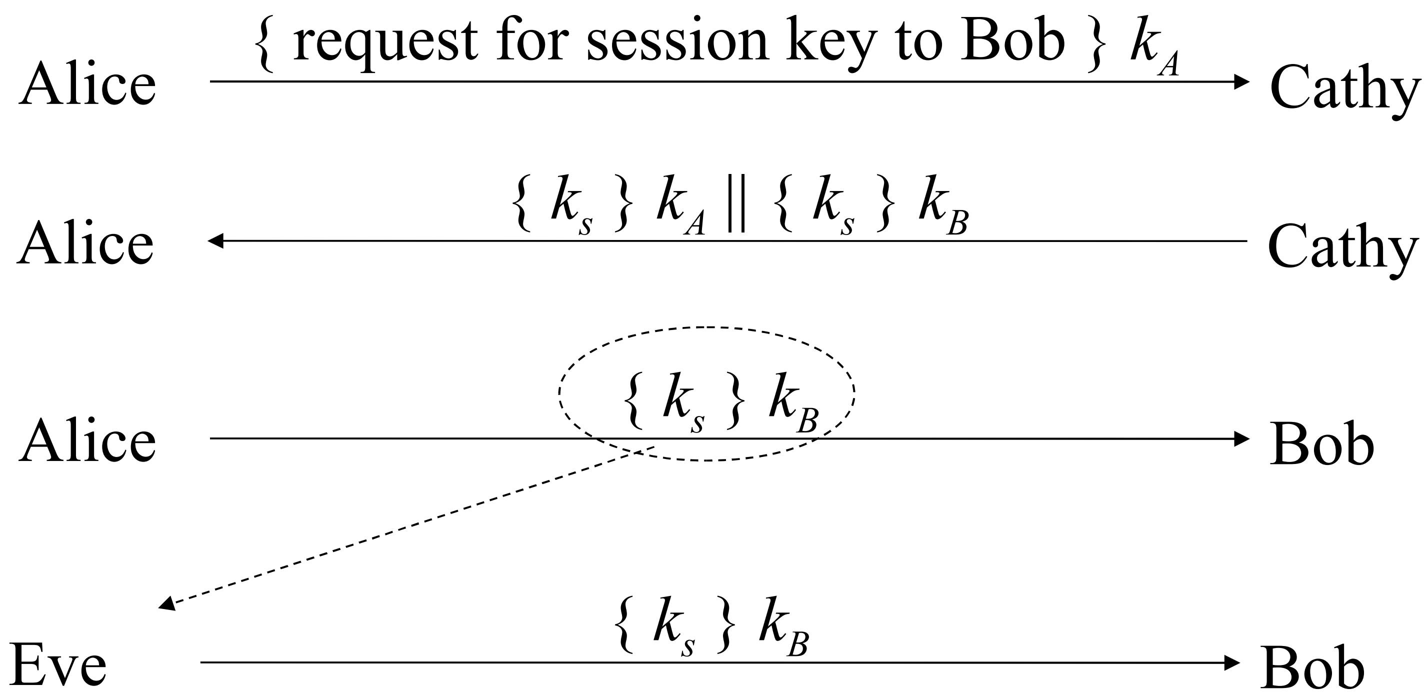
- Generally separate channel is not practical
  - No trustworthy separate channel
  - Want to scale linearly with additional users



# 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 $k_s$

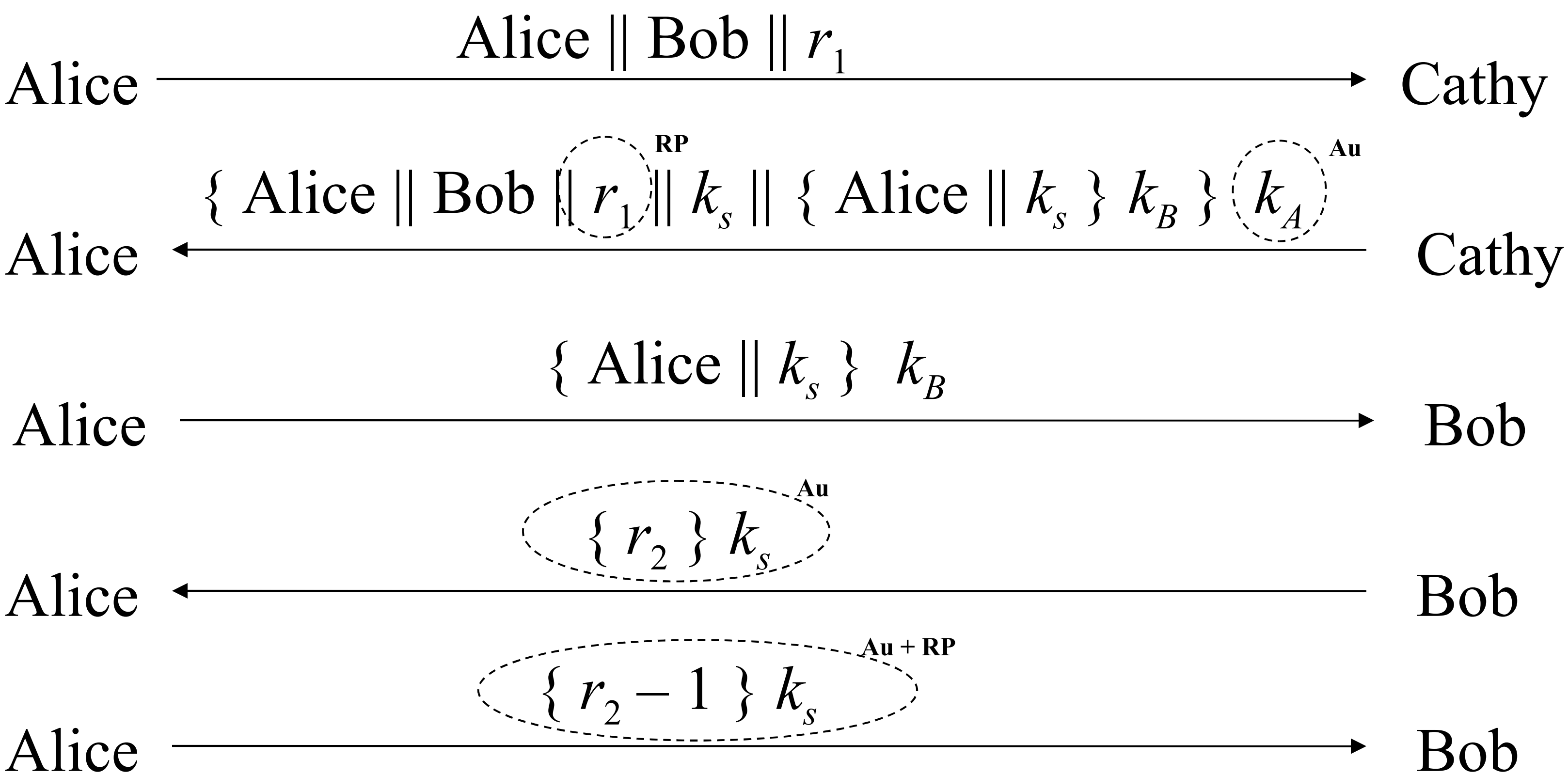
# Simple Protocol



# 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 defense against replay

# Needham-Schroeder



# Argument: Alice talking to Bob

- Second message
  - Encrypted using key only she, Cathy knows
    - So Cathy encrypted it
  - Response to first message
    - As $r_1$ in it matches $r_1$ in first message
- Third message
  - Alice knows only Bob can read it
    - As only Bob can derive session key from message
  - Any messages encrypted with that key are from Bob

# Argument: Bob talking to Alice

- Third message
  - Encrypted using key only he, Cathy know
    - So Cathy encrypted it
  - Names Alice, session key
    - Cathy provided session key, says Alice is other party
- Fourth message
  - 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

# Denning-Sacco Modification

- Needham-Schroeder Assumption: all keys are secret
- Question: suppose Eve can obtain session key. How does that affect protocol?
  - In what follows, Eve knows $k_s$

Eve → Bob: $\{ \text{Alice} \parallel k_s \}\ k_B$

Eve ← Bob: $\{ r_2 \}\ k_s$

Eve → Bob: $\{ r_2 - 1 \}\ k_s$

# Solution

- In protocol above, Eve impersonates Alice
- Problem: replay in third step
  - First in previous slide
- Solution: use time stamp *T* to detect replay
- Weakness: if clocks not synchronized, may either reject valid messages or accept replays
  - Parties with either slow or fast clocks vulnerable to replay

# Needham-Schroeder with Denning-Sacco Modification

Alice → Cathy: Alice || Bob || $r_1$

Alice ← Cathy: { Alice || Bob || $r_1$ || $k_s$ || { Alice || $T$ || $k_s$ } $k_B$ } $k_A$

Alice → Bob: { Alice || $T$ || $k_s$ } $k_B$

Alice ← Bob: { $r_2$ } $k_s$

Alice → Bob: { $r_2 - 1$ } $k_s$

# Otway-Rees Protocol

- Corrects problem
  - That is, Eve replaying the third message in the protocol
- Does not use timestamps
  - Not vulnerable to the problems that Denning-Sacco modification has

# The Protocol

Alice → Bob: $n \,\|\, \text{Alice} \,\|\, \text{Bob} \,\|\, \{ r_1 \,\|\, n \,\|\, \text{Alice} \,\|\, \text{Bob} \}\, k_A$

Cathy ← Bob: $n \,\|\, \text{Alice} \,\|\, \text{Bob} \,\|\, \{ r_1 \,\|\, n \,\|\, \text{Alice} \,\|\, \text{Bob} \}\, k_A \,\|\, \{ r_2 \,\|\, n \,\|\, \text{Alice} \,\|\, \text{Bob} \}\, k_B$

Cathy → Bob: $n \,\|\, \{ r_1 \,\|\, k_s \}\, k_A \,\|\, \{ r_2 \,\|\, k_s \}\, k_B$

Alice ← Bob: $n \,\|\, \{ r_1 \,\|\, k_s \}\, k_A$

# Argument: Alice talking to Bob

- Fourth message
  - If $n$ matches first message, Alice knows it is part of this protocol exchange
  - Cathy generated $k_s$ because only she, Alice know $k_A$
  - Encrypted part belongs to exchange as $r_1$ matches $r_1$ in encrypted part of first message

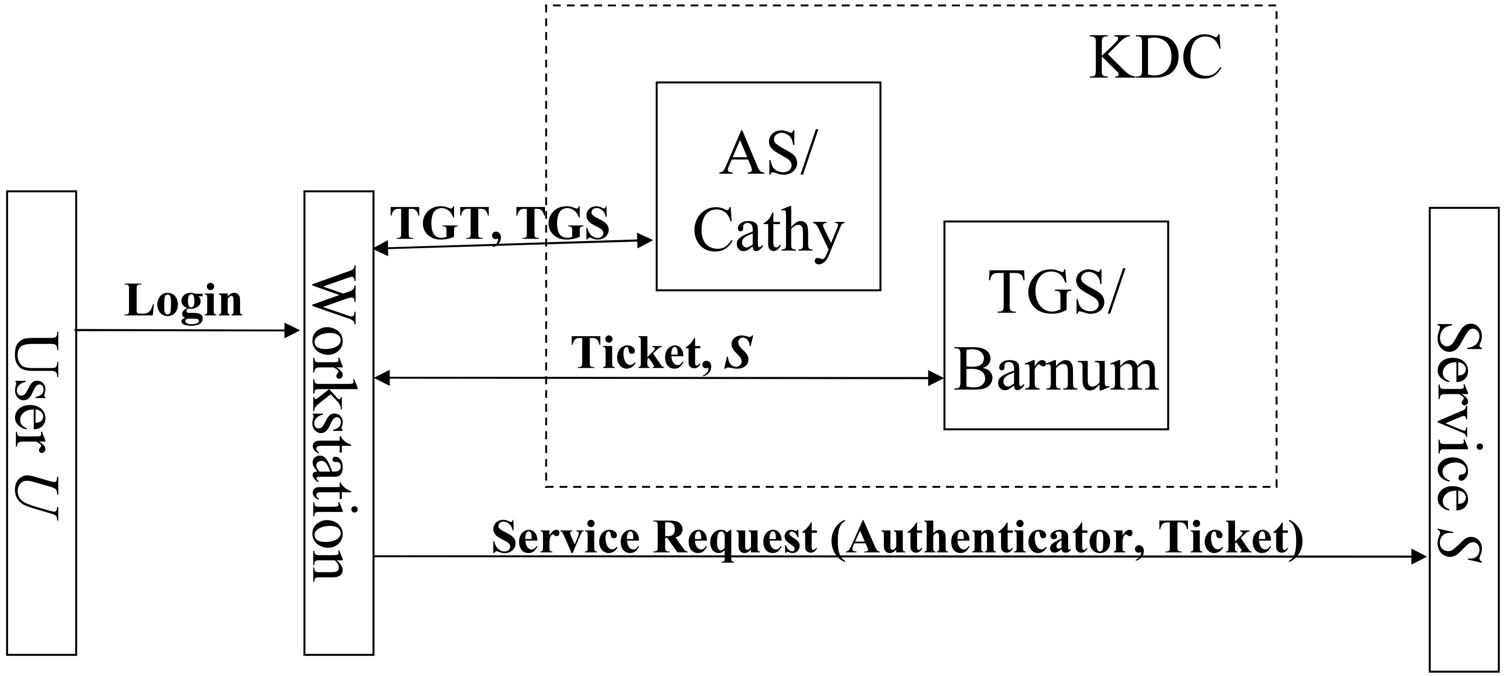
# Argument: Bob talking to Alice

- Third message
  - If $n$ matches second message, Bob knows it is part of this protocol exchange
  - Cathy generated $k_s$ because only she, Bob know $k_B$
  - Encrypted part belongs to exchange as $r_2$ matches $r_2$ in encrypted part of second message

# Replay Attack

- Eve acquires old $k_s$, message in third step
  - $n \| \{ r_1 \| k_s \} k_A \| \{ r_2 \| k_s \} k_B$
- Eve forwards appropriate part to Alice
  - Nonce $r_1$ matches nothing, so is rejected

# Network Authentication with Kerberos




Legend: AS = Authentication Server; TGS = Ticket Granting Server
KDC = Key Distribution Center; TGT = Ticket Granting Ticket;

# Kerberos

- Authentication system
  - Based on Needham-Schroeder with Denning-Sacco modification
  - Central server plays role of trusted third party ("Cathy")
- Ticket
  - Issuer vouches for identity of requester of service
- Authenticator
  - Identifies sender
- Two Competing Versions: 4 and 5
  - Version 4 discussed here

# Idea

- User $u$ authenticates to Kerberos AS
  - Obtains ticket (TGT) $T_{u,TGS}$ for ticket granting service (TGS)
- User $u$ wants to use service $s$:
  - User sends authenticator $A_u$, ticket $T_{u,TGS}$ to TGS asking for ticket for service
  - TGS sends ticket $T_{u,s}$ to user
  - User sends $A_u$, $T_{u,s}$ to server as request to use $s$
- Details follow

# Ticket

- Credential saying issuer has identified ticket requester
- Example ticket issued to user $u$ for TGS

$$T_{u,\text{TGS}} = \text{TGS} \parallel \{ u \parallel u\text{'s address} \parallel \text{valid time} \parallel k_{u,\text{TGS}} \}\, k_{\text{AS,TGS}}$$

where:

- $k_{u,\text{TGS}}$ is session key for user and TGS
- $k_{\text{AS,TGS}}$ is long-term key shared between AS and TGS
- Valid time is interval for which ticket valid; e.g., a day
- $u$'s address may be IP address or something else
  - Note: more fields, but not relevant here

# Ticket

- Example ticket issued to user *u* for service *s*

$$T_{u,s} = s \,||\, \{ u \,||\, u\text{'s address} \,||\, \text{valid time} \,||\, k_{u,s} \}\, k_s$$

where:

- – $k_{u,s}$ is session key for user and service
- – $k_s$ is long-term key shared between TGS and S
- – Valid time is interval for which ticket valid; e.g., hours/days
- – *u*'s address may be IP address or something else
  - Note: more fields, but not relevant here

# Authenticator

- Credential containing identity of sender of ticket
  - Used to confirm sender is entity to which ticket was issued
- Example: authenticator user *u* generates for service *s*

$$A_{u,s} = \{ u \,\|\, \text{generation time} \} \, k_{u,s}$$

where:

  - Generation time is when authenticator generated
    - Note: more fields, not relevant here

# Protocol

* Initially, user $u$ registers with KDC and establishes a password
    - used to derive long-term key $k_u$

* User U logs into workstation (WS) using password

M1: *user/ws* —— [AS_REQ]: *user* || *TGS* ——→ AS

M2: *user/ws* ←—— [AS_REP]: $\{ k_{u,TGS} \} k_u \; || \; T_{u,TGS}$ —— AS

* WS decrypts session key $k_{u,TGS}$ using supplied password

# Protocol

M3: *user/ws* → TGS
[TGS_REQ]: $service \,\|\, A_{u,TGS} \,\|\, T_{u,TGS}$

* TGS decrypts ticket using long-term key $k_{AS,TGS}$

M4: *user/ws* ← TGS
[TGS_REP]: $user \,\|\, \{ k_{u,s} \}\, k_{u,TGS} \,\|\, T_{u,s}$

M5: *user/ws* → *service*
[AP_REQ]: $A_{u,s} \,\|\, T_{u,s}$

* Service decrypts ticket using long-term key $k_{TGS,s}$

M6: *user/ws* ← *service*
[AP_REP]: $\{ t + 1 \}\, k_{u,s}$

# Summary of Messages

- First two messages get user ticket to use TGS
  - User *u* can obtain session key only if *u* knows key shared with AS
- Next four messages show how *u* gets and uses ticket for service *s*
  - Service *s* validates request by checking sender (using $A_{u,s}$) is same as entity ticket issued to
  - Step 6 optional; used when *u* requests confirmation

# Problems

- Relies on synchronized clocks
  - Typical clock skew allowed is 5 minutes
  - If not synchronized and old tickets, authenticators not cached, replay is possible
- Tickets have some fixed fields
  - Dictionary attacks possible
  - Kerberos 4 session keys weak (had much less than 56 bits of randomness); researchers at Purdue found them from tickets in minutes

# Public Key Key Exchange

- Here interchange keys known
  - $e_A$, $e_B$ Alice and Bob's public keys known to all
  - $d_A$, $d_B$ Alice and Bob's private keys known only to owner
- Simple protocol
  - $k_s$ is desired session key

Alice $\xrightarrow{\{ k_s \} e_B}$ Bob

# Problem and Solution

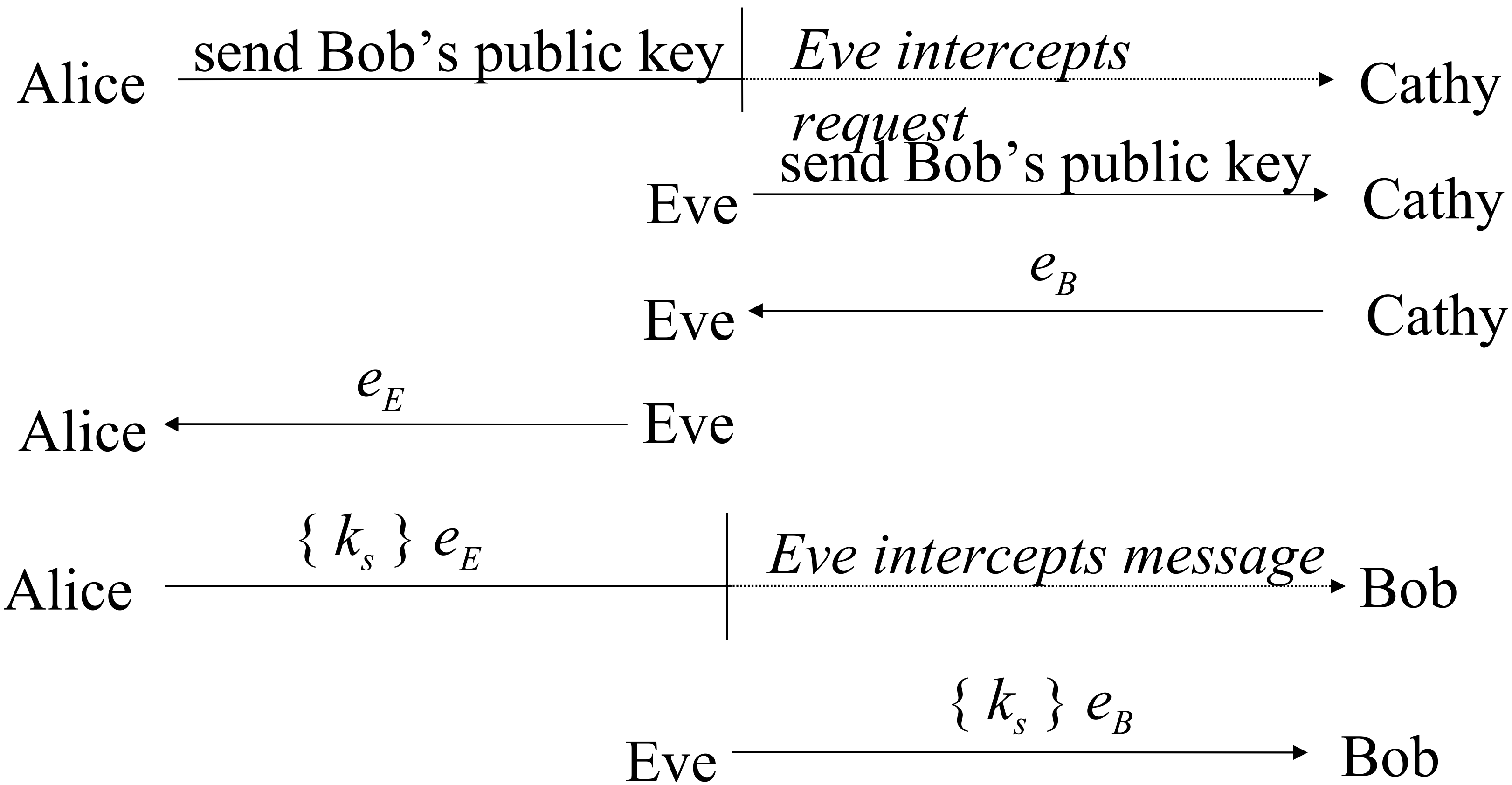
- Vulnerable to forgery or replay
  - Because $e_B$ known to anyone, Bob has no assurance that Alice sent message
- Simple fix uses Alice's private key
  - $k_s$ is desired session key

Alice $\xrightarrow{\{\ \{\ k_s\ \}\ d_A\ \}\ e_B}$ Bob

# Notes

- Can include message enciphered with $k_s$
- Assumes Bob has Alice's public key, and *vice versa*
  - If not, each must get it from public server
  - If keys not bound to identity of owner, attacker Eve can launch a *man-in-the-middle* attack (next slide; Cathy is public server providing public keys)
    - Solution to this (binding identity to keys) discussed later as public key infrastructure (PKI)

# Man-in-the-Middle Attack



# 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

# 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"

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

# X.509 Certificates

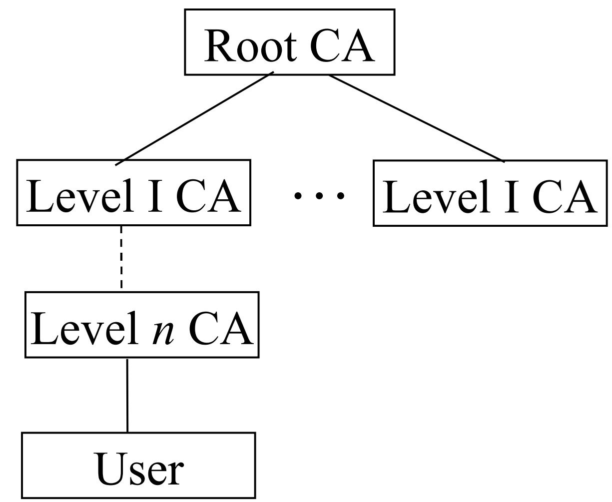
- Some certificate components in X.509v3:
  - Version
  - Serial number
  - Signature algorithm identifier: hash algorithm
  - Issuer’s name; uniquely identifies issuer
  - Interval of validity
  - Subject’s name; uniquely identifies subject
  - Subject’s public key
  - Signature: encrypted hash

# Use

- Bob gets Alice's certificate
  - If he knows Cathy's public key, he can validate the certificate
    - Decrypt 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

- **A Single Global CA**
  - Unmanageable, inflexible
  - There is no universally trusted organization
- **Hierarchical CAs (Tree)**




  - Offloads burden on multiple CAs
  - Need to verify a chain of certificates
  - Still depends on a single trusted root CA

# 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's CA is Cathy; Bob's CA is Don; how can Alice validate Bob's certificate?
  - Have Cathy and Don cross-certify
  - Each issues certificate for the other
- Certificates:
  - Cathy<<Alice>>
  - Dan<<Bob>
  - Cathy<<Dan>>
  - Dan<<Cathy>>
- Alice validates Bob's certificate
  - Alice obtains Cathy<<Dan>>
  - Alice uses (known) public key of Cathy to validate Cathy<<Dan>>
  - Alice uses Cathy<<Dan>> to validate Dan<<Bob>>

# PGP Chains

- OpenPGP certificates structured into packets
  - One public key packet
  - Zero or more signature packets
- Public key packet:
  - Version (3 or 4; 3 compatible with all versions of PGP, 4 not compatible with older versions of PGP)
  - Creation time
  - Validity period (not present in version 3)
  - Public key algorithm, associated parameters
  - Public key

# OpenPGP Signature Packet

- Version 3 signature packet
  - Version (3)
  - Signature type (level of trust)
  - Creation time (when next fields hashed)
  - Signer's key identifier (identifies key to encrypt hash)
  - Public key algorithm (used to encrypt hash)
  - Hash algorithm
  - Part of signed hash (used for quick check)
  - Signature (encrypted hash)
- Version 4 packet more complex

# Signing

- Single certificate may have multiple signatures
- Notion of "trust" embedded in each signature
  - Range from "untrusted" to "ultimate trust"
  - Signer defines meaning of trust level (no standards!)
  - Few implementations support this
- All version 4 keys signed by subject
  - Called "self-signing"

# Trust in GPG

## Owner Trust

GPG enables assignment of trust in entity

- Unknown, Not Trusted, Marginal, Full
- How much do you trusted signatures by that entity

## Key Validity

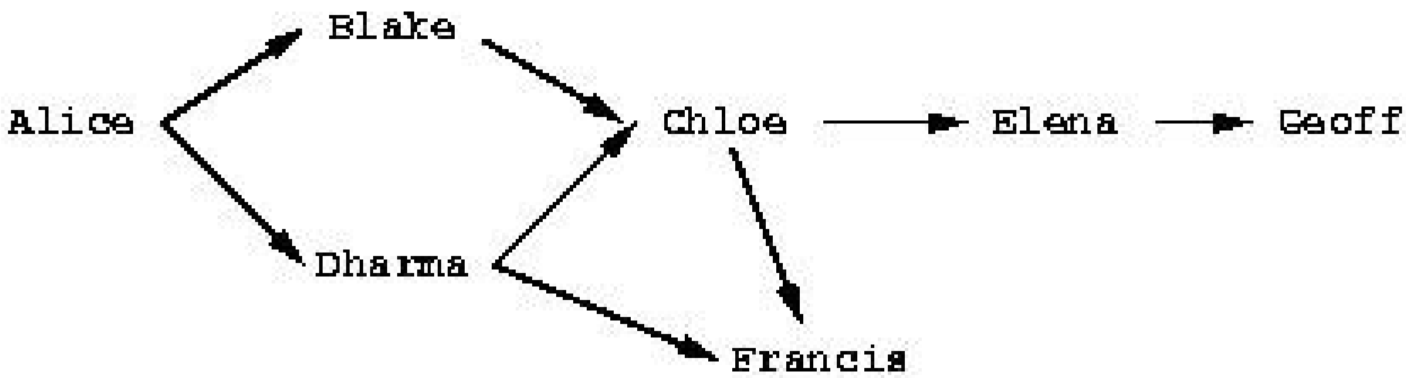
How much do you trust that the key corresponds to the owner?

- Unknown, Not trusted, Marginal Full

Configurable

- X full paths or Y marginal paths

# Web of Trust Example




From GPG manual, arrows show key signatures

# Web of Trust Scenarios

- In all cases the key is valid given 1 full or 2 marginal paths. Always fully trust directly signed keys
  - Fully Trust Dharma
  - Marginally Trust Dharma and Blake
  - Marginally Trust Dharma and Chloe
  - Marginally Trust Dharma, Blake, and Chloe
  - Fully Trust Blake, Chloe, and Elena

# Key Revocation

- Certificates invalidated *before* expiration
  - Usually due to compromised key
  - May be due to change in circumstance (*e.g.,* someone leaving company)
- Problems
  - Verify that entity revoking certificate authorized to do so
  - Revocation information circulates to everyone fast enough
    - Network delays, infrastructure problems may delay information

# GPG: Revocation Certificate

- Use the –gen-revoke option to create a revocation certification.
- When you suspect that your certificate has been compromised
  - Send revocation certificate to all your friends
  - Or send the revocation certificate to a GPG key server

# CRLs

- *Certificate revocation list* lists certificates that are revoked
- X.509: only certificate issuer can revoke certificate
  - Added to CRL

# Digital Signature

- Construct that authenticated origin, contents of message in a manner provable to a disinterested third party ("judge")
- Sender cannot deny having sent message (service is "nonrepudiation")
  - Limited to *technical* proofs
    - Inability to deny one's cryptographic key was used to sign
  - One could claim the cryptographic key was stolen or compromised
    - Legal proofs, *etc.,* probably required; not dealt with here

# Simple Approach

- Classical: Alice, Bob share key $k$
  - Alice sends $m \parallel \{ m \} k$ to Bob

  This is a digital signature

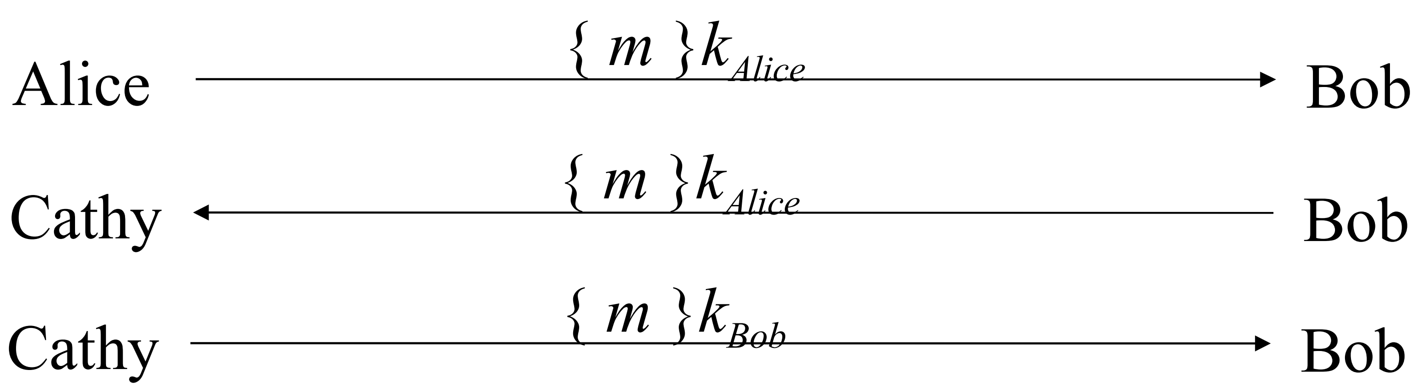
***<u>WRONG</u>***

**This is not a digital signature**

  - Why? Third party cannot determine whether Alice or Bob generated message

# Classical Digital Signatures

- Require trusted third party
  - Alice, Bob each share keys with trusted party Cathy
- To resolve dispute, judge gets $\{ m \} k_{Alice}$, $\{ m \} k_{Bob}$, and has Cathy decipher them; if messages matched, contract was signed



# Public Key Digital Signatures

- Alice's keys are $d_{Alice}$, $e_{Alice}$
- Alice sends Bob

$$m \,\|\, \{ m \} \, d_{Alice}$$

- In case of dispute, judge computes

$$\{ \{ m \} \, d_{Alice} \} \, e_{Alice}$$

- and if it is *m*, Alice signed message
  - She's the only one who knows $d_{Alice}$!

# RSA Digital Signatures

- Use private key to encrypt message
- Key points:
  - Never sign random documents, and when signing, always sign hash and never document
    - Mathematical properties can be turned against signer
  - Sign message first, then encrypt
    - Changing public keys causes forgery

# Attack #1

- $m_1 \times m_2 \bmod n_b = m$
- Get Bob to sign $m_1$ and $m_2$
- $m_1^d \bmod n_b \times m_2^d \bmod n_b =$
- $(m_1^d \times m_2^d) \bmod n_b =$
- $(m_1 \times m_2)^d \bmod n_b = m^d \bmod n_b$

# Attack #1 example

- Example: Alice, Bob communicating
  - $n_A = 95$, $e_A = 59$, $d_A = 11$
  - $n_B = 77$, $e_B = 53$, $d_B = 17$
- 26 contracts, numbered 00 to 25
  - Alice has Bob sign 05 and 17:
    - $c = m^{d_B} \bmod n_B = 05^{17} \bmod 77 = 3$
    - $c = m^{d_B} \bmod n_B = 17^{17} \bmod 77 = 19$
  - Alice computes $05 \times 17 \bmod 77 = 08$; corresponding signature is $03 \times 19 \bmod 77 = 57$; claims Bob signed 08
  - Judge computes $c^{e_B} \bmod n_B = 57^{53} \bmod 77 = 08$
    - Signature validated; Bob is toast

# Attack #2: Bob's Revenge

- Bob, Alice agree to sign contract m but Bob wants it to appear that she signed contract M
  - Alice encrypts, then signs:

$(m^{eB} \bmod n_B)^{dA} \bmod n_A$

- Bob now changes his public key
  - Computes $r$ such that $M^r \bmod n_B = m$
  - Creates new public key $e'_B = re_B$ and computes a new matching private key $d'_B$
- Bob claims contract was M. Judge computes:
  - $(c^{eA} \bmod n_A)^{d'B} \bmod n_B = M$

# Attack #2 Example

- Bob, Alice agree to sign contract 06
- Alice encrypts, then signs:

$(m^{e_B} \bmod 77)^{d_A} \bmod n_A = (06^{53} \bmod 77)^{11} \bmod 95 = 63$

- Bob now changes his public key
  - Computes $r$ such that $13^r \bmod 77 = 6$; say, $r = 59$
  - Computes $re_B \bmod \phi(n_B) = 59 \times 53 \bmod 60 = 7$
  - Replace public key $e_B$ with 7, private key $d_B = 43$
- Bob claims contract was 13. Judge computes:
  - $(63^{59} \bmod 95)^{43} \bmod 77 = 13$
  - Verified; now Alice is toast

# El Gamal Digital Signature

- Relies on discrete log problem
- Choose $p$ prime, $g$, $d < p$; compute $y = g^d \bmod p$
- Public key: $(y, g, p)$; private key: $d$
- To sign contract m:
  - Choose $k$ relatively prime to $p-1$, and not yet used
  - Compute $a = g^k \bmod p$
  - Find $b$ such that $m = (da + kb) \bmod p-1$
  - Signature is $(a, b)$
- To validate, check that
  - $y^a a^b \bmod p = g^m \bmod p$

# Example

- Alice chooses $p = 29$, $g = 3$, $d = 6$

  $y = 3^6 \bmod 29 = 4$
- Alice wants to send Bob signed contract 23
  - Chooses $k = 5$ (relatively prime to 28)
  - This gives $a = g^k \bmod p = 3^5 \bmod 29 = 11$
  - Then solving $23 = (6\times11 + 5b) \bmod 28$ gives $b = 25$
  - Alice sends message 23 and signature (11, 25)
- Bob verifies signature: $g^m \bmod p = 3^{23} \bmod 29 = 8$ and $y^a a^b \bmod p = 4^{11} 11^{25} \bmod 29 = 8$
  - They match, so Alice signed

# Attack

- Eve learns $k$, corresponding message $m$, and signature $(a, b)$
  - Extended Euclidean Algorithm gives $d$, the private key
- Example from above: Eve learned Alice signed last message with $k = 5$

$$m = (da + kb) \bmod p{-}1 = (11d + 5 \times 25) \bmod 28$$

so Alice's private key is $d = 6$

# Storing Keys

- Multi-user or networked systems: attackers may defeat access control mechanisms
  - Encrypt file containing key
    - Attacker can monitor keystrokes to decrypt files
    - Key will be resident in memory that attacker may be able to read
      - Cold Boot attack
  - Use physical devices like “smart card” or TPM
    - Key never enters system
    - Card can be stolen, so have 2 devices combine bits to make single key

# Key Recovery

Key recovery quite a practical need for Enterprise/Organizational operations

- Long term storage of encrypted data
- What if key employee gets hit by a bus?

Ideas for key recovery?

Concerns?

# Key Escrow

- *Key escrow system* allows authorized third party to recover key
  - Useful when keys belong to roles, such as system operator, rather than individuals
  - Business: recovery of backup keys
  - Law enforcement: recovery of keys that authorized parties require access to
- Goal: provide this without weakening cryptosystem
- Very controversial

  Peter Neumann's congressional testamony

  http://www.csl.sri.com/users/neumann/judiciary.html

# Desirable Properties

- Escrow system should not depend on encryption algorithm
- Privacy protection mechanisms must work from end to end and be part of user interface
- Requirements must map to key exchange protocol
- System supporting key escrow must require all parties to authenticate themselves
- If message to be observable for limited time, key escrow system must ensure keys valid for that period of time only



Beth, Knobloch, Otten, Simmons, Wichmann 94

# Components

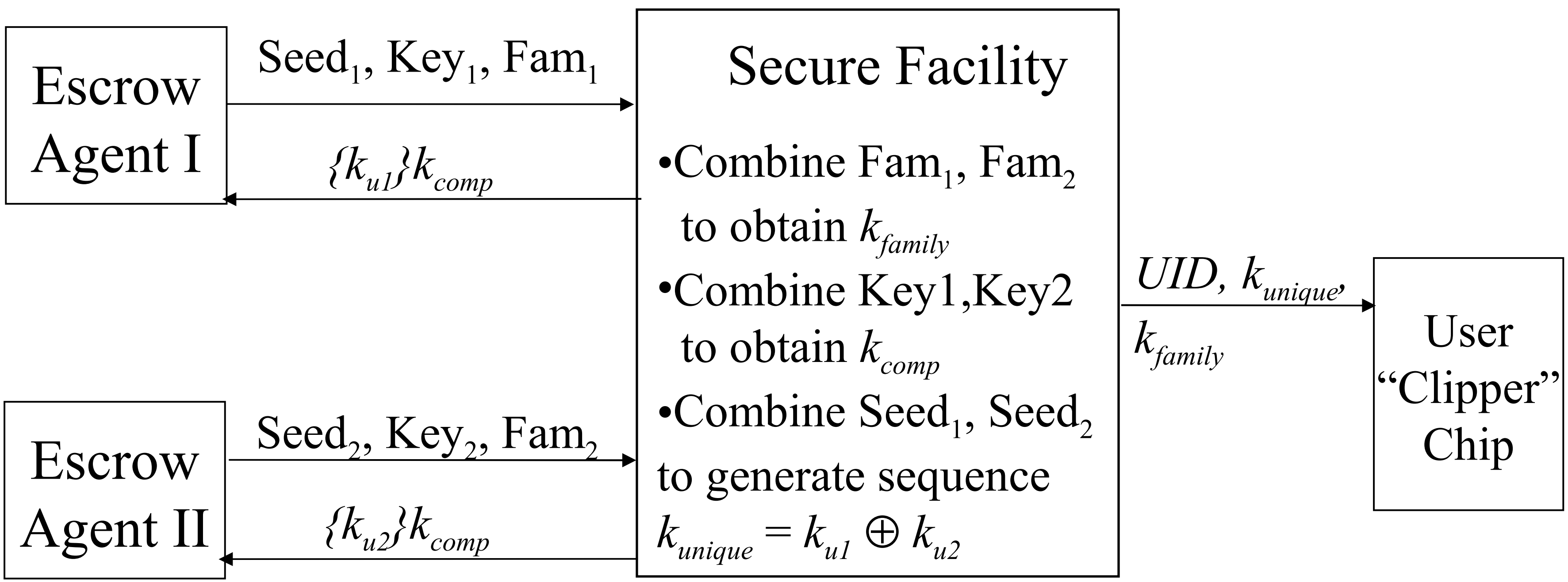
- User security component
  - Does the encryption, decryption
  - Supports the key escrow component
- Key escrow component
  - Manages storage, use of data recovery keys
- Data recovery component
  - Does key recovery

# Example: EES, Clipper Chip

- Escrow Encryption Standard
  - Set of interlocking components
  - Designed to balance need for law enforcement access to enciphered traffic with citizens' right to privacy
- Clipper chip given to users prepares per-message escrow information
  - Each chip numbered uniquely by UID
  - Special facility programs chip
- Key Escrow Decrypt Processor (KEDP)
  - Available to agencies authorized to read messages
- Details in Handbook of Applied Cryptography
  - http://www.cacr.math.uwaterloo.ca/hac/about/chap13.pdf

NIST 94

# Initialization of User Security Component



# User Security Component

- Unique device key $k_{unique}$
- Non-unique family key $k_{family}$
- Cipher is Skipjack
  – Classical cipher: 80 bit key, 64 bit input, output blocks
- Generates Law Enforcement Access Field (LEAF) of 128 bits:
  – $\{ \text{UID} \| \{ k_{session} \} k_{unique} \| hash \} k_{family}$
  – *hash*: 16 bit authenticator from session key and initialization vector

# Obtaining Access

- Alice obtains legal authorization to read message
- She runs message LEAF through KEDP
  - LEAF is { UID || { $k_{session}$ } $k_{unique}$ || *hash* } $k_{family}$
- KEDP uses (known) $k_{family}$ to validate LEAF, obtain sending device's UID
- Authorization, LEAF taken to escrow agencies

# Agencies' Role

- Each validates authorization
- Each supplies $\{ k_{ui} \} k_{comp}$, corresponding key number
- KEDP takes these and LEAF: $\{ \text{UID} \parallel \{ k_{session} \} k_{unique} \parallel hash \} k_{family}$
  - Key numbers produce $k_{comp}$
  - $k_{comp}$ produces $k_{u1}$ and $k_{u2}$
  - $k_{u1}$ and $k_{u2}$ produce $k_{unique}$
  - $k_{unique}$ and LEAF produce $k_{session}$

# Problems

- *hash* too short
  - LEAF 128 bits, so given a hash:
    - $2^{112}$ LEAFs show this as a valid hash
    - 1 has actual session key, UID
    - Takes about 42 minutes to generate a LEAF with a valid hash but meaningless session key and UID
      - Turns out deployed devices would prevent this attack
  - Scheme does not meet temporal requirement
    - As $k_{unique}$ fixed for each unit, once message is read, any future messages can be read

# Yaksha Security System

- Key escrow system meeting all 5 criteria
- Based on RSA, central server
  - Central server (Yaksha server) generates session key
- Each user has 2 private keys
  - Alice's modulus $n_A$, public key $e_A$
  - First private key $d_{AA}$ known only to Alice
  - Second private key $d_{AY}$ known only to Yaksha central server
  - $d_{AA} d_{AY} = d_A \bmod \Phi( n_{A)}$



Ganesan 96

# Alice and Bob

- Alice wants to send message to Bob
  - Alice asks Yaksha server for session key
  - Yaksha server generates $k_{session}$
  - Yaksha server sends Alice the key as:

$$C_A = (k_{session})^{d_{AY} e_A} \bmod n_A$$

  - Alice computes

$$(C_A)^{d_{AA}} \bmod n_A = k_{session}$$

# Analysis

- Authority can read only one message per escrowed key
  - Meets requirement 5 (temporal one), because "time" interpreted as "session"
- Independent of message enciphering key
  - Meets requirement 1
  - Interchange algorithm, keys fixed
- Others met by supporting infrastructure

# Alternate Approaches

- Tie to time
  - Session key not given as escrow key, but related key is
  - To derive session key, must solve instance of discrete log problem
- Tie to probability
  - Oblivious transfer: message received with specified probability
  - Idea: *translucent cryptography* allows fraction *f* of messages to be read by third party
  - Not key escrow, but similar in spirit

# Key Points

- Key management critical to effective use of cryptosystems
  - Different levels of keys (session *vs*. interchange)
- Exchange algorithms can be vulnerable to attacks
  - Replay
  - Identity integrity
- Digital signatures provide integrity of origin and content

Much easier with public key cryptosystems than with classical cryptosystems

- Keys need infrastructure to identify holders, allow revoking and possible escrow