

Fall 2008: CS442

Introduction to Computer Security

Vinod Ganapathy
Lecture 7

- Topic: Key exchange protocols
- Material:
 - Class handout
 - Chapter 2 in Anderson's book.

Today's agenda

- Key exchange basics
- Simple protocols and replay attacks
- Needham Schroeder and refinements
- Public-key exchange protocols
- Man-in-the-middle attacks
- Certificates

Notation

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

Session, 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 encipher m
 - To be used for this message *only*
 - Called a *session key*
 - She enciphers k_s with Bob's public key k_B
 - k_B enciphers 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 enciphered 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 $\{ \text{“BUY”} \} k_B$ and $\{ \text{“SELL”} \} k_B$. Eve intercepts enciphered message, compares, and gets plaintext at once

Key Exchange Algorithms

- Goal: Alice, Bob get shared key
 - Key cannot be sent in clear
 - Attacker can listen in
 - Key can be sent enciphered, or derived from exchanged data plus data not known to an eavesdropper
 - Alice, Bob may trust third party
 - All cryptosystems, protocols publicly known
 - Only secret data is the keys, ancillary information known only to Alice and Bob needed to derive keys
 - Anything transmitted is assumed known to attacker

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

Alice $\xrightarrow{\{ \text{request for session key to Bob} \} k_A}$ Cathy

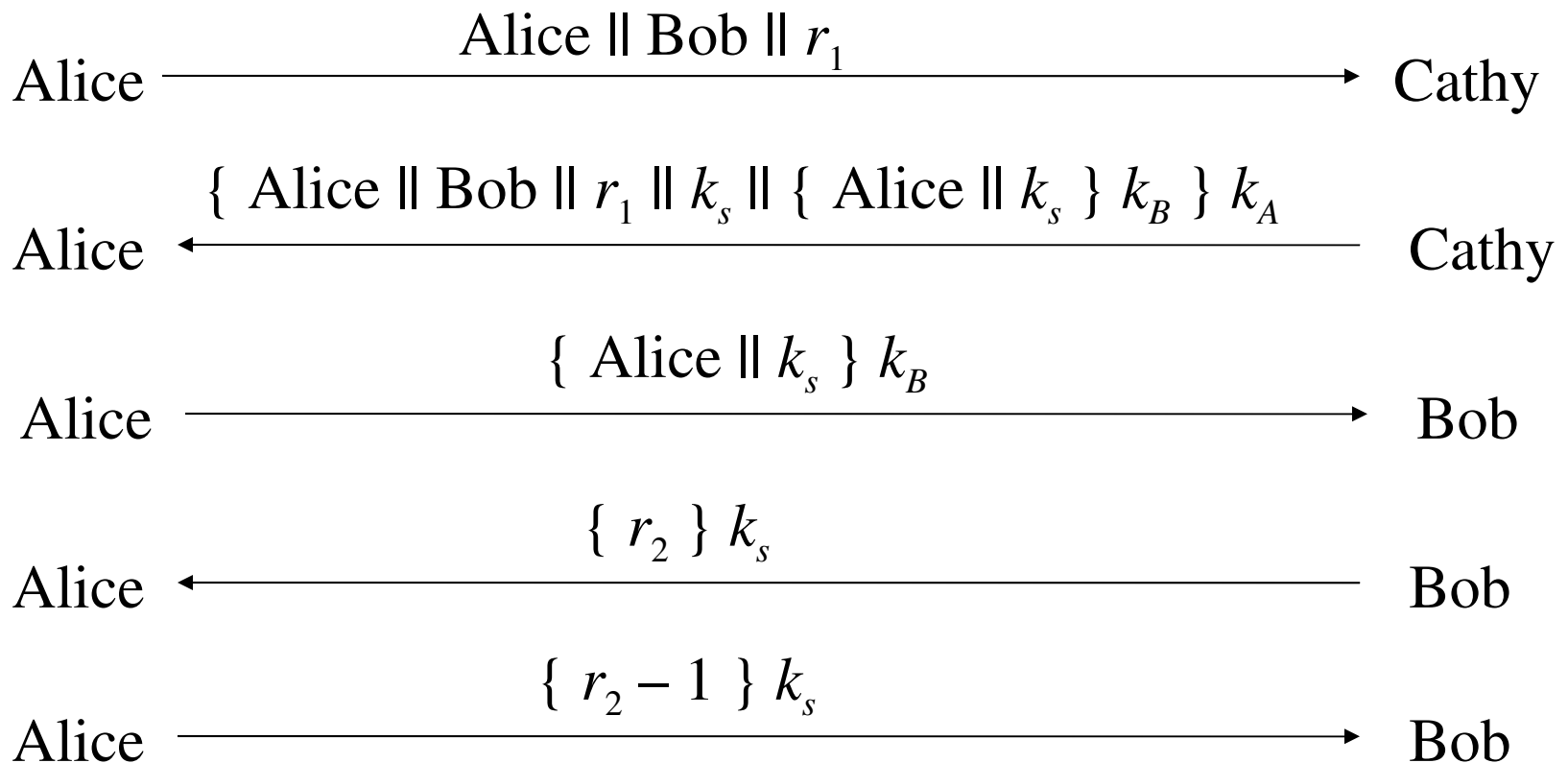
Alice $\xleftarrow{\{ k_s \} k_A \parallel \{ k_s \} k_B}$ Cathy

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

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

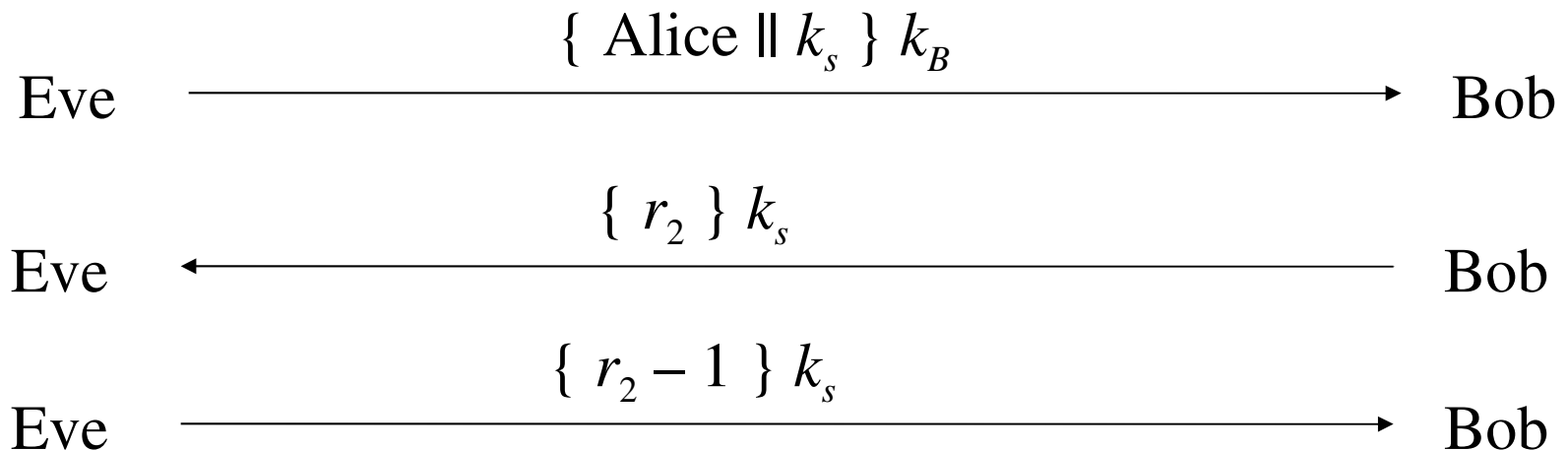
- Second message
 - Enciphered using key only she, Cathy knows
 - So Cathy enciphered it
 - Response to first message
 - As r_1 in it matches r_1 in first message
- Third message
 - Alice knows only Bob can read it
 - As only Bob can derive session key from message
 - Any messages enciphered with that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

Argument: Bob talking to Alice

- Third message
 - Enciphered using key only he, Cathy know
 - So Cathy enciphered 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 decipher r_2 and so can't respond, or responds incorrectly

Denning-Sacco Modification

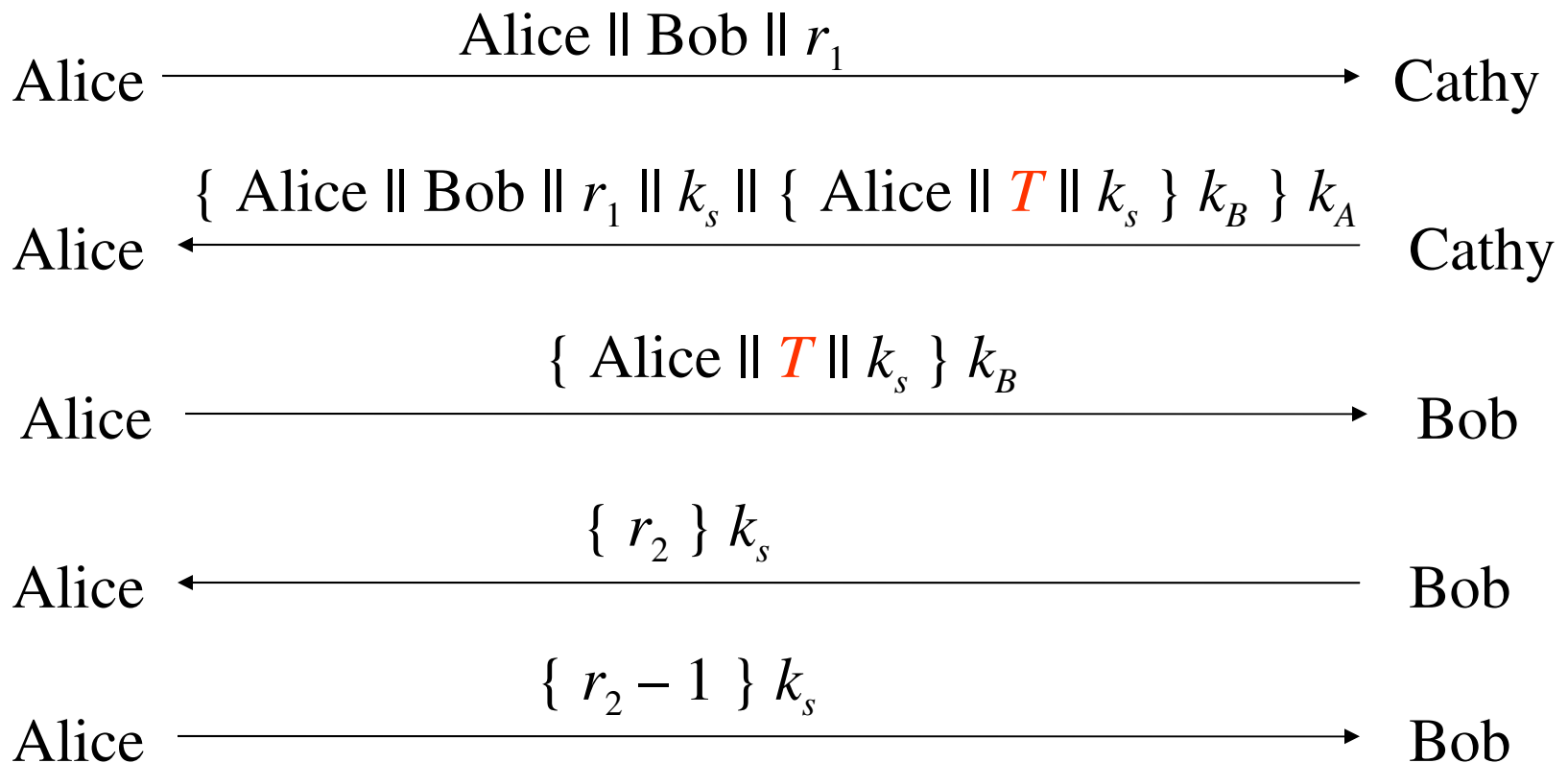
- Assumption: all keys are secret
- Question: suppose Eve can obtain session key. How does that affect protocol?
 - In what follows, Eve knows k_s



Solution

- In protocol above, Eve impersonates Alice
- Problem: replay in third step
 - First in previous slide
- Solution: use time stamp T to detect replay
- Proposed by Denning and Sacco

Needham-Schroeder with Denning-Sacco Modification



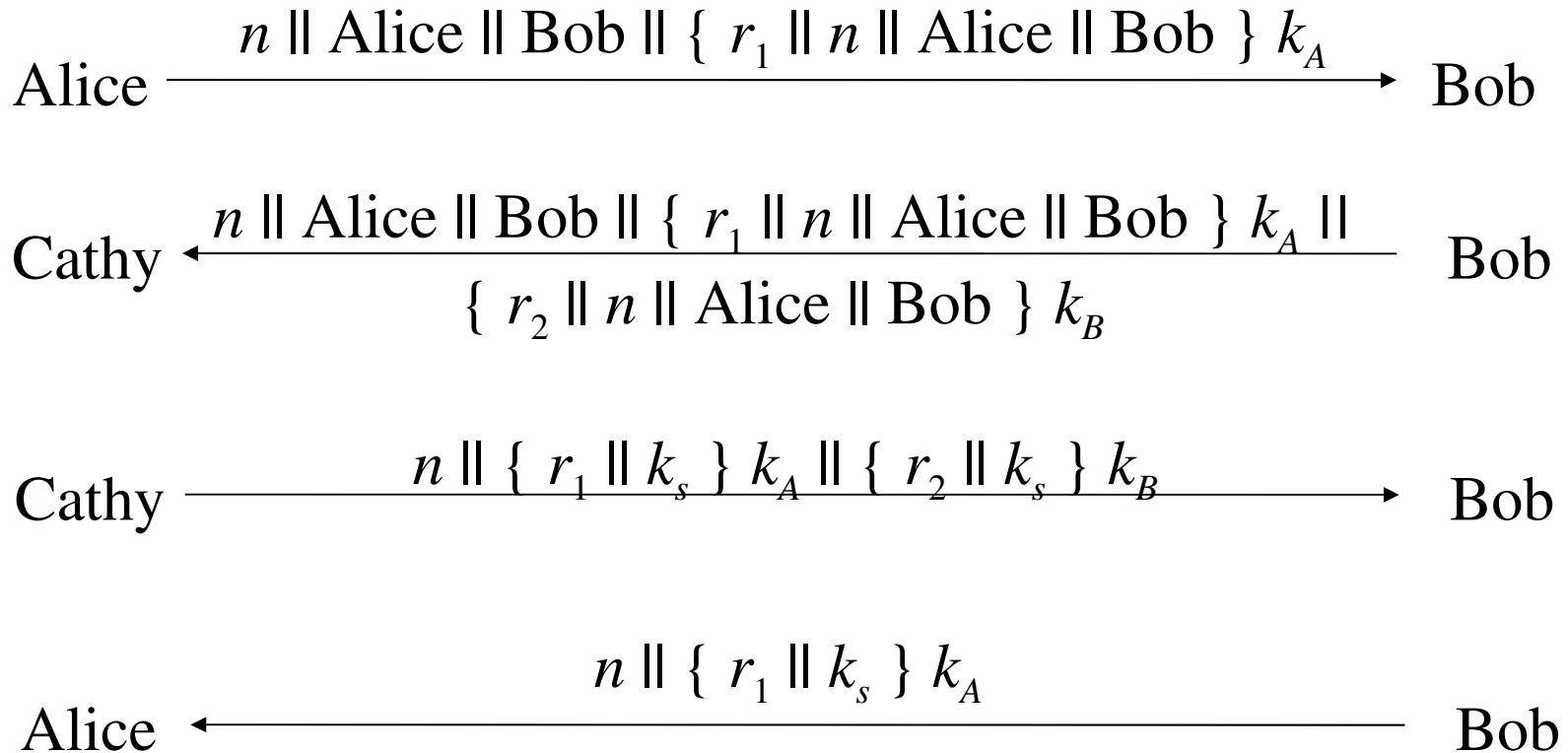
Problem with timestamps?

- If clocks not synchronized, may either reject valid messages or accept replays
 - Parties with either slow or fast clocks vulnerable to replay
 - Resetting clock does *not* eliminate vulnerability
- We'll see timestamps used in Kerberos as well.

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
- Uses integer n to associate all messages with particular exchange

The Protocol



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
 - Enciphered part belongs to exchange as r_1 matches r_1 in encrypted part of first message

Argument: Bob talking to Alice

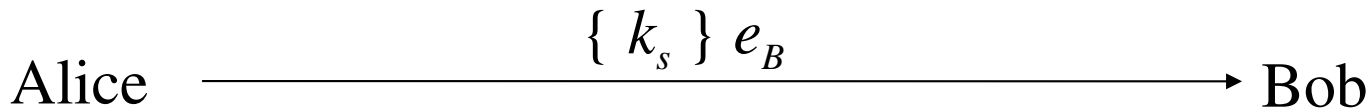
- Third message
 - If n matches second message, Bob knows it is part of this protocol exchange
 - Cathy generated k_s because only she, Bob know k_B
 - Enciphered part belongs to exchange as r_2 matches r_2 in encrypted part of second message

Replay Attack

- Eve acquires old k_s , message in third step
 - $n \parallel \{ r_1 \parallel k_s \} k_A \parallel \{ r_2 \parallel k_s \} k_B$
- Eve forwards appropriate part to Alice
 - Alice has no ongoing key exchange with Bob: n matches nothing, so is rejected
 - Alice has ongoing key exchange with Bob: n does not match, so is again rejected
 - If replay is for the current key exchange, *and* Eve sent the relevant part *before* Bob did, Eve could simply listen to traffic; no replay involved

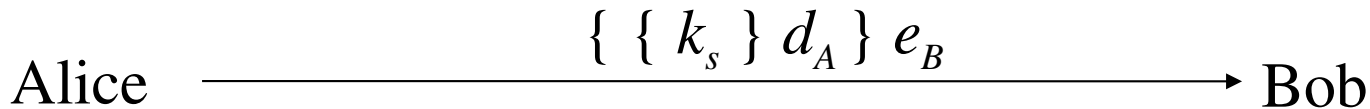
Public-Key Key Exchange

- 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



Problem and Solution

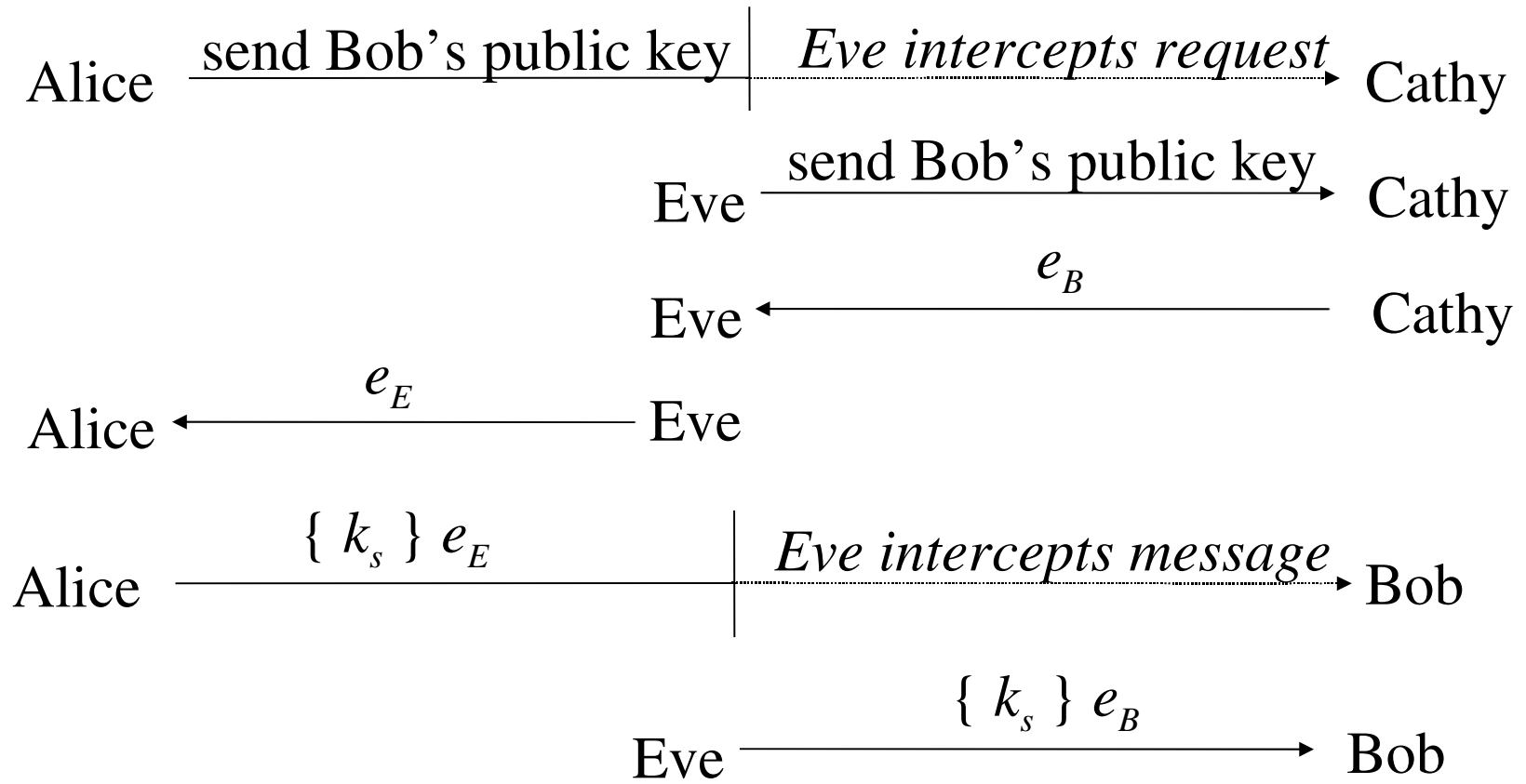
- Vulnerable to forgery or replay
 - Because e_B known to anyone, Bob has no assurance that Alice sent message
- Simple fix uses Alice's private key
 - k_s is desired session key



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

Certificates

- Create token (message) containing
 - Identity of principal (here, Alice)
 - Corresponding public key
 - Timestamp (when issued)
 - Other information (perhaps identity of signer)signed by trusted authority (here, Cathy)

$$C_A = \{ e_A \parallel \text{Alice} \parallel T \} d_C$$

Use

- Bob gets Alice's certificate
 - If he knows Cathy's public key, he can decipher the certificate
 - When was certificate issued?
 - Is the principal Alice?
 - Now Bob has Alice's public key
- Problem: Bob needs Cathy's public key to validate certificate
 - Problem pushed "up" a level
 - Two approaches: Merkle's tree, signature chains

Certificate Signature Chains

- Create certificate
 - Generate hash of certificate
 - Encipher hash with issuer's private key
- Validate
 - Obtain issuer's public key
 - Decipher enciphered hash
 - Recompute hash from certificate and compare
- Problem: getting issuer's public key

X.509 Chains

- 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

X.509 Certificate Validation

- Obtain issuer's public key
 - The one for the particular signature algorithm
- Decipher signature
 - Gives hash of certificate
- Recompute hash from certificate and compare
 - If they differ, there's a problem
- Check interval of validity

Issuers

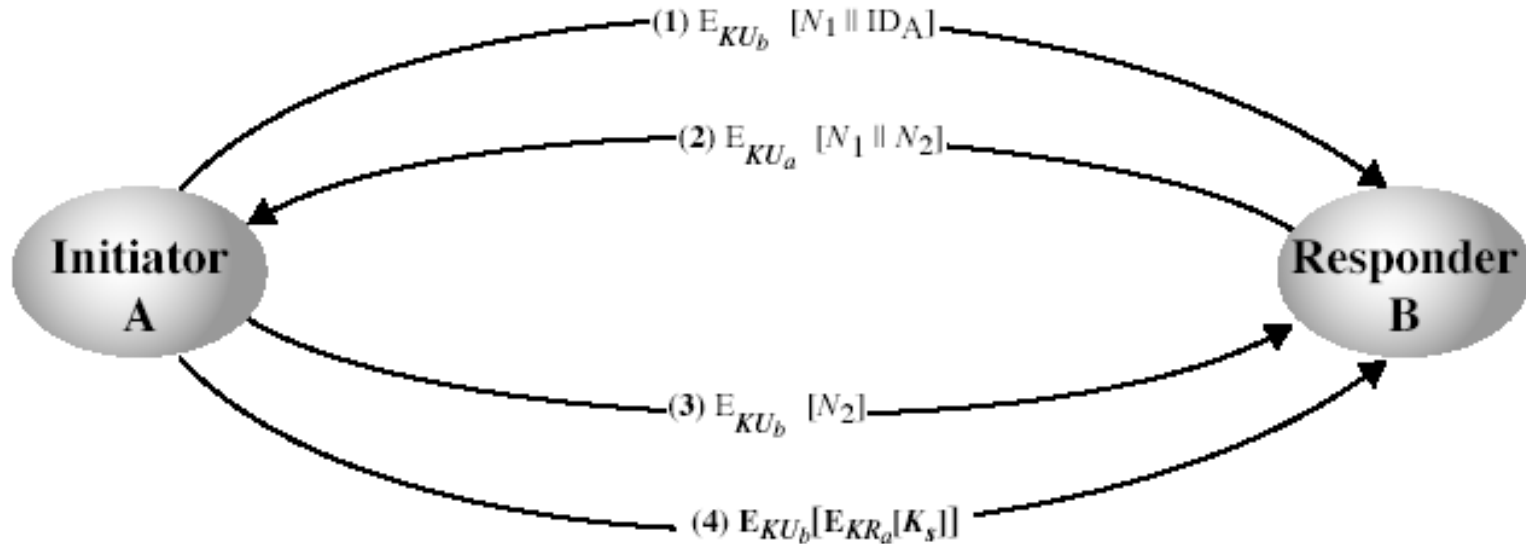
- *Certification Authority (CA)*: entity that issues certificates
 - Multiple issuers pose validation problem
 - Alice's CA is Cathy; Bob's CA is Don; how can Alice validate Bob's certificate?
 - Have Cathy and Don cross-certify
 - Each issues certificate for the other

Validation and Cross-Certifying

- 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>>

Public-Key Distribution of Secret Keys

- if have securely exchanged public-keys:



Diffie-Hellman Key Exchange

- first public-key type scheme proposed
- by Diffie & Hellman in 1976 along with the exposition of public key concepts
 - note: now know that James Ellis (UK CESG) secretly proposed the concept in 1970
- is a practical method for public exchange of a secret key
- used in a number of commercial products

Diffie-Hellman Key Exchange

- a public-key distribution scheme
 - cannot be used to exchange an arbitrary message
 - rather it can establish a common key
 - known only to the two participants
- value of key depends on the participants (and their private and public key information)
- based on exponentiation in a finite (Galois) field (modulo a prime or a polynomial) - easy
- security relies on the difficulty of computing discrete logarithms (similar to factoring) – hard

Next class

- Certificate chains
- Kerberos

Diffie-Hellman Setup

- all users agree on global parameters:
 - large prime integer or polynomial q
 - α a primitive root mod q
- each user (eg. A) generates their key
 - chooses a secret key (number): $x_A < q$
 - compute their **public key**: $Y_A = \alpha^{x_A} \bmod q$
- each user makes public that key Y_A

Diffie-Hellman Key Exchange

- shared session key for users A & B is K_{AB} :

$$K_{AB} = \alpha^{x_A \cdot x_B} \text{ mod } q$$

$$= Y_A^{x_B} \text{ mod } q \quad (\text{which } \mathbf{B} \text{ can compute})$$

$$= Y_B^{x_A} \text{ mod } q \quad (\text{which } \mathbf{A} \text{ can compute})$$

- K_{AB} is used as session key in private-key encryption scheme between Alice and Bob
- if Alice and Bob subsequently communicate, they will have the **same** key as before, unless they choose new public-keys
- attacker needs an x , must solve discrete log

Diffie-Hellman Example

- users Alice & Bob who wish to swap keys:
- agree on prime $q=353$ and $\alpha=3$
- select random secret keys:
 - A chooses $x_A=97$, B chooses $x_B=233$
- compute public keys:
 - $y_A=3^{97} \bmod 353 = 40$ (Alice)
 - $y_B=3^{233} \bmod 353 = 248$ (Bob)
- compute shared session key as:
 - $K_{AB} = y_B^{x_A} \bmod 353 = 248^{97} = 160$ (Alice)
 - $K_{AB} = y_A^{x_B} \bmod 353 = 40^{233} = 160$ (Bob)