

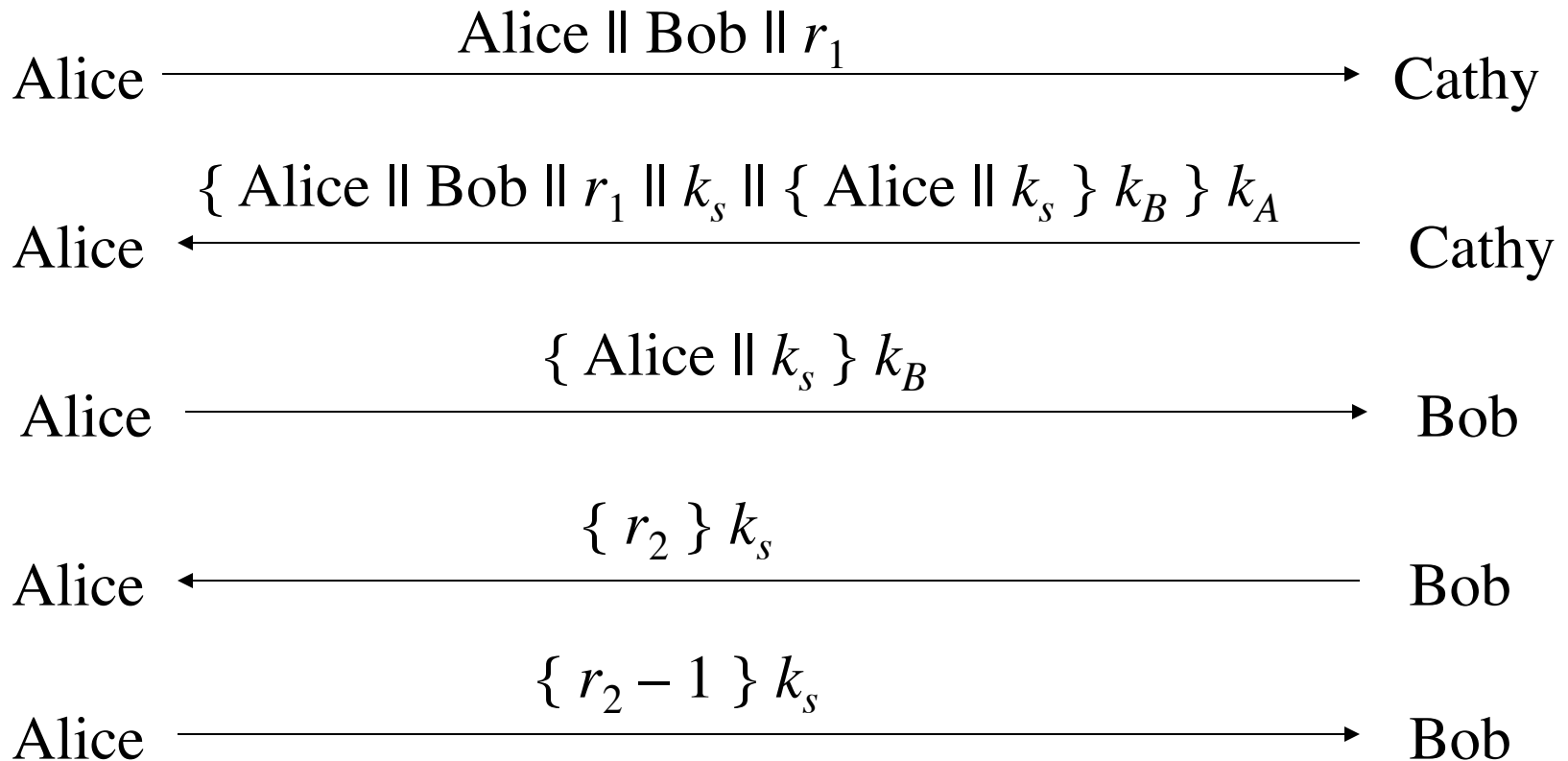
Lecture for February 5, 2016

ECS 235A

UC Davis

Matt Bishop

Needham-Schroeder



Argument: Alice talking to Bob

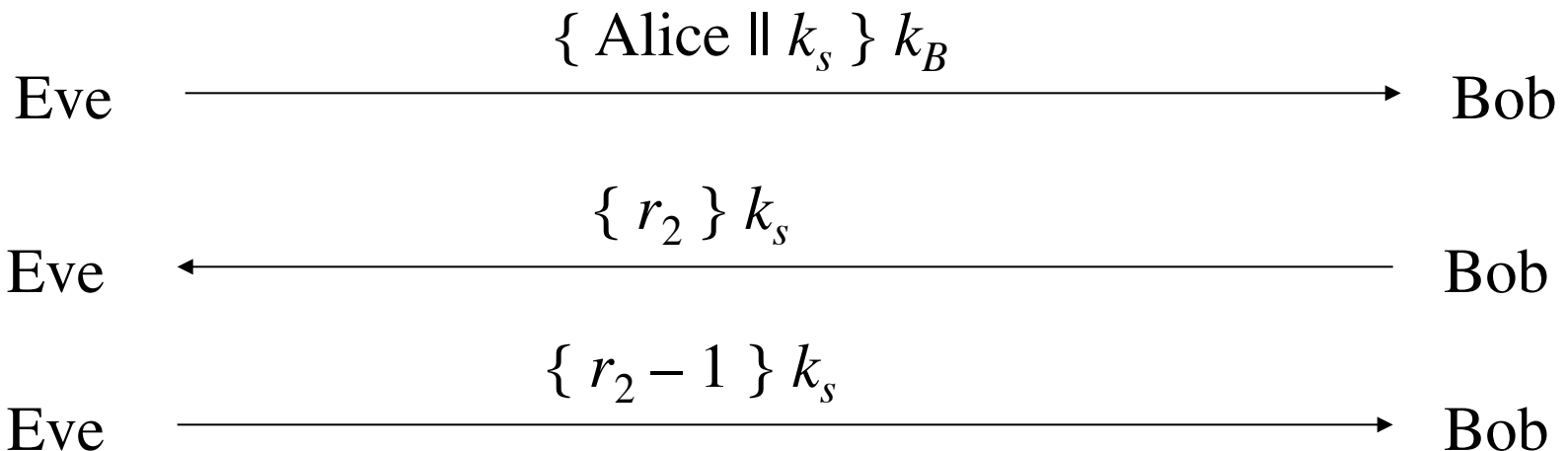
- Second message
 - Enciphered using key only she, Cathy knows
 - So Cathy enciphered it
 - Response to first message
 - As r_1 in it matches r_1 in first message
- Third message
 - Alice knows only Bob can read it
 - As only Bob can derive session key from message
 - Any messages enciphered with that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

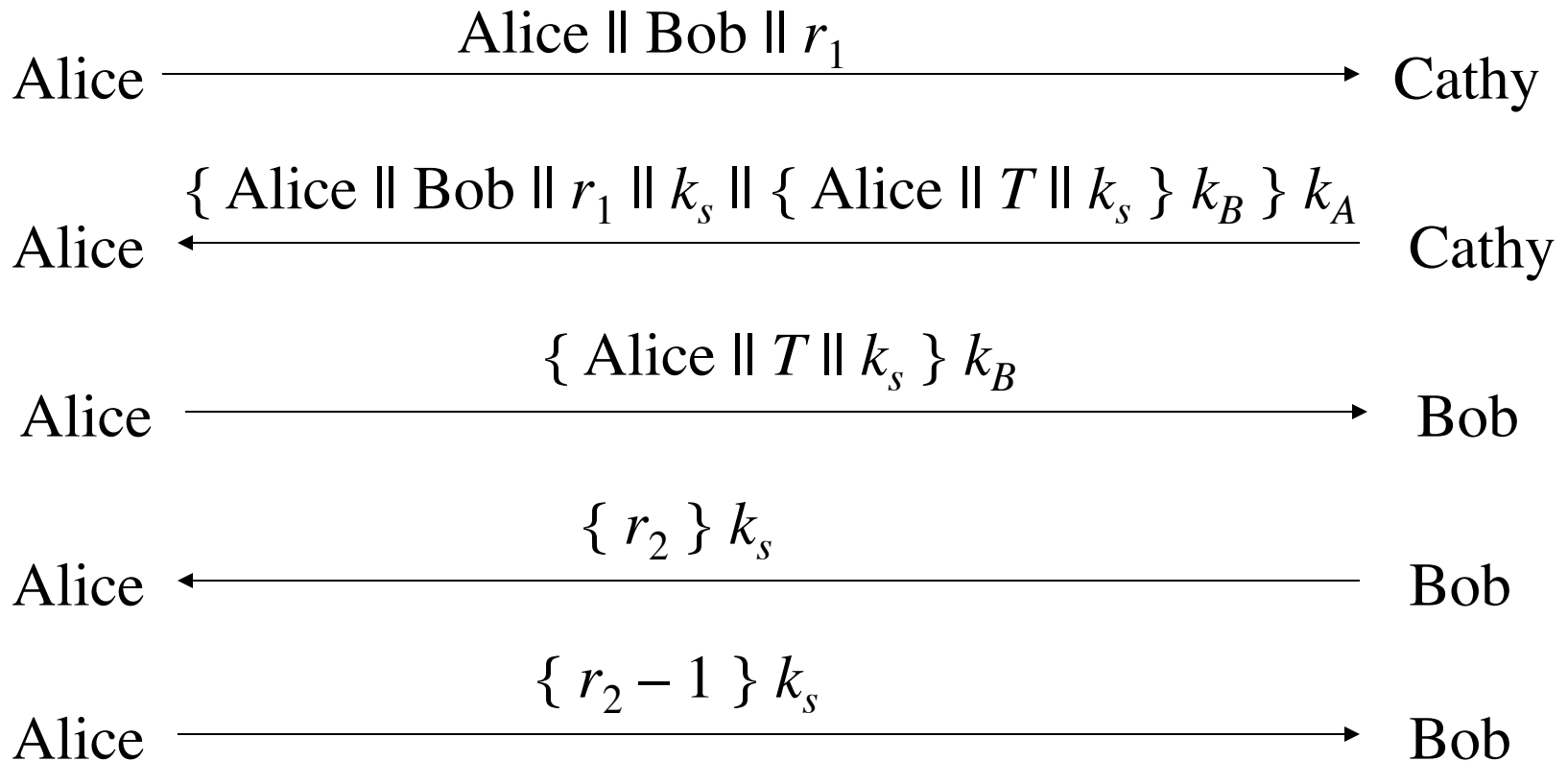
- Assumption: all keys are secret
- Question: suppose Eve can obtain session key.
How does that affect protocol?
 - In what follows, Eve knows k_s



Solution

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

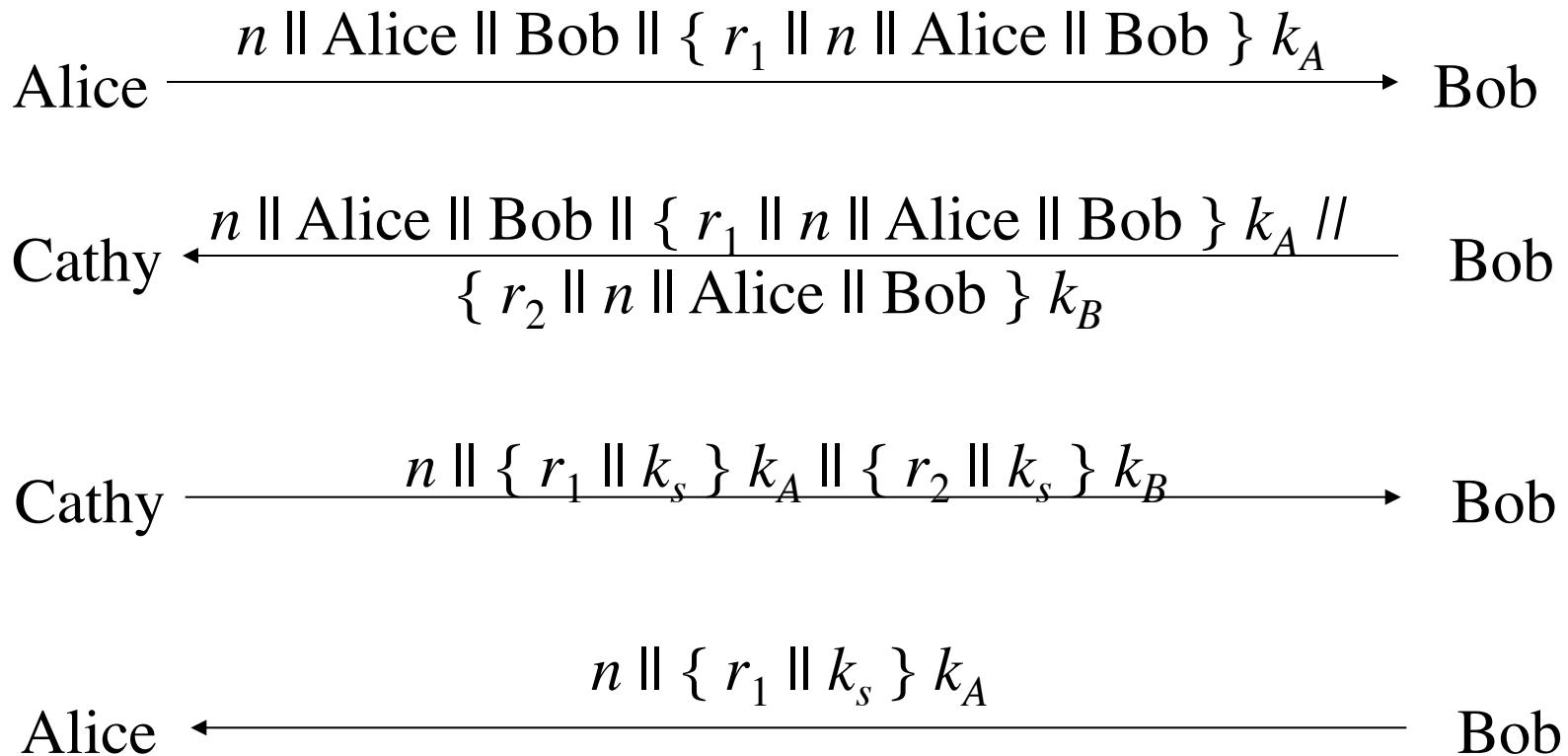
Needham-Schroeder with Denning-Sacco Modification



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

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

Idea

- User u authenticates to Kerberos server
 - Obtains ticket $T_{u,TGS}$ 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 service s

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

where:

- $k_{u,s}$ is session key for user and service
- Valid time is interval for which ticket valid
- u' s address may be IP address or something else
 - Note: more fields, but not relevant here

Authenticator

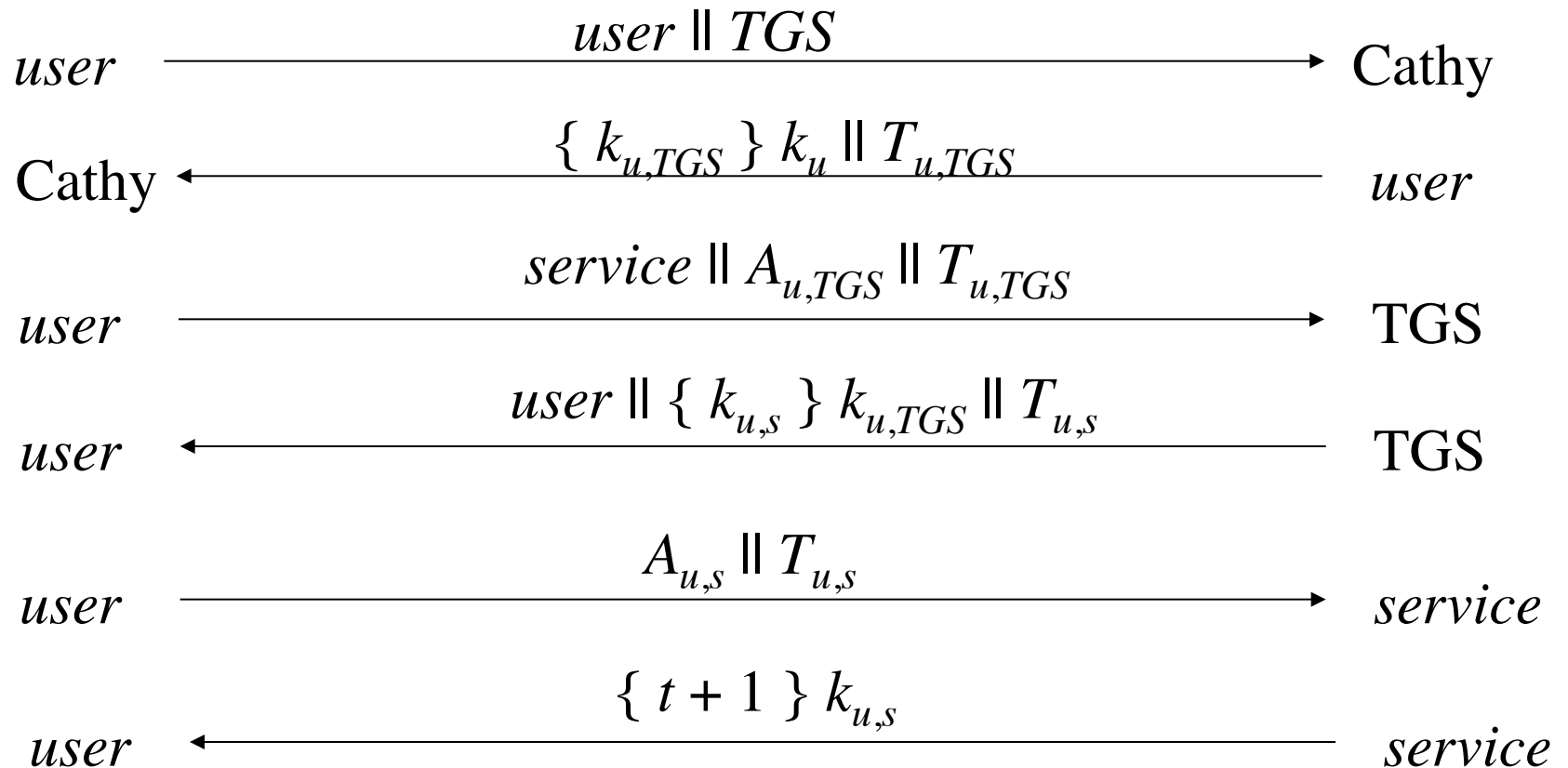
- Credential containing identity of sender of ticket
 - Used to confirm sender is entity to which ticket was issued
- Example: authenticator user u generates for service s

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

where:

- k_t is alternate session key
- Generation time is when authenticator generated
 - Note: more fields, not relevant here

Protocol



Analysis

- First two steps get user ticket to use TGS
 - User u can obtain session key only if u knows key shared with Cathy
- Next four steps 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
 - 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

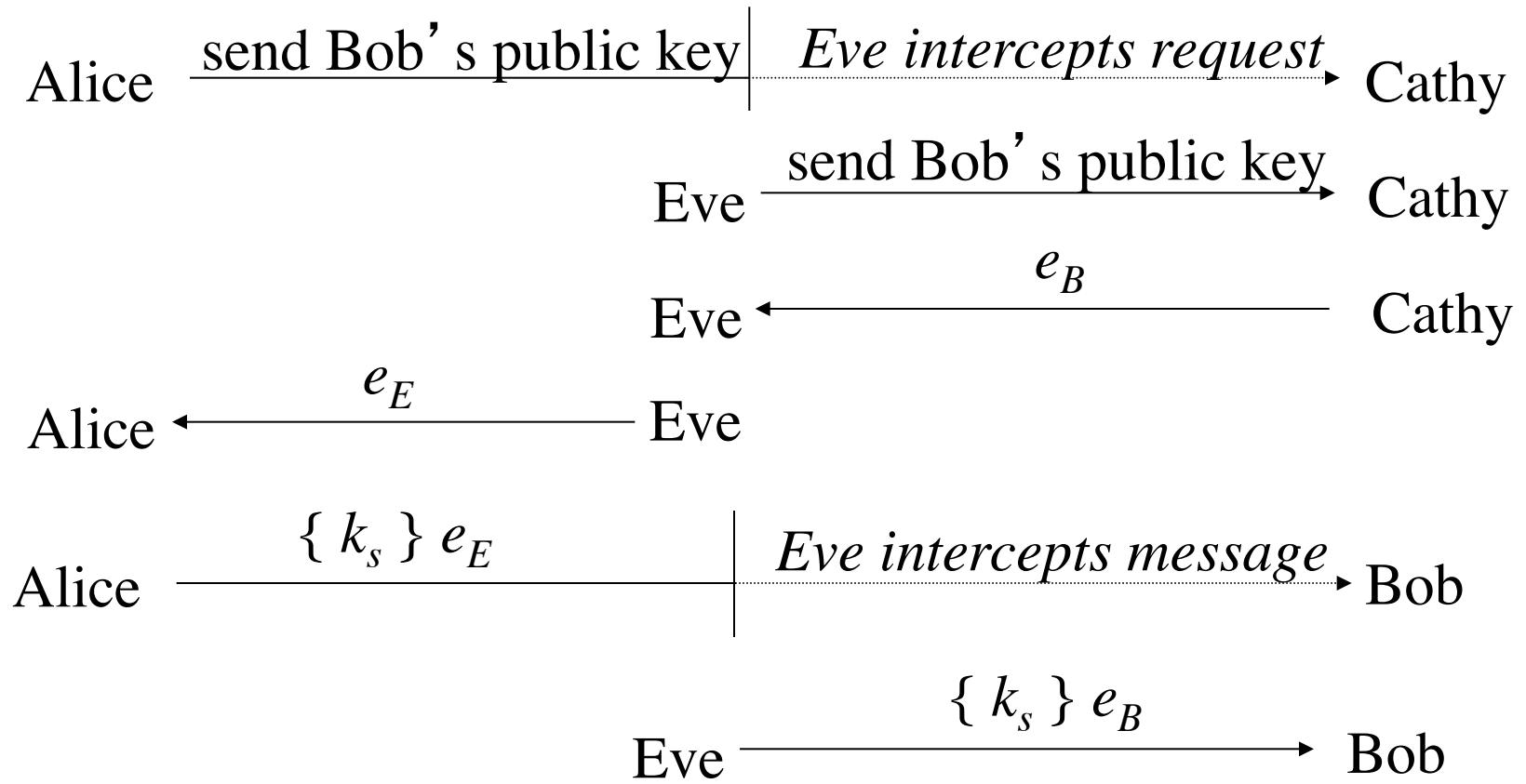
- Vulnerable to forgery or replay
 - Because e_B known to anyone, Bob has no assurance that Alice sent message
- Simple fix uses Alice's private key
 - k_s is desired session key

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



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, \dots such that for any integer $k > 0$, an observer cannot predict n_k even if all of n_1, \dots, 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 [$n_k = (a_j n_{k-1}^j + \dots + a_1 n_{k-1} + a_0) \bmod n$] 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
 - Use on UNIX-based systems:
`(date; ps gaux) | md5`
where “ps gaux” lists all information about all processes on system

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