L5: Key Distributions

Hui Chen, Ph.D.
Dept. of Engineering & Computer Science
Virginia State University
Petersburg, VA 23806

Acknowledgement

- Many slides are from or are revised from the slides of the author of the textbook
  - Matt Bishop, Introduction to Computer Security, Addison-Wesley Professional, October, 2004, ISBN-13: 978-0-321-24774-5. Introduction to Computer Security @ VSU's Safari Book Online subscription
  - http://nob.cs.ucdavis.edu/book/book-intro/slides/

Outline

- Key exchange: session vs. interchange keys
- Classical cryptographic key exchange and authentication
  - Protocol evolution
    - Needham-Schroeder
    - Otway-Rees
  - Key freshness, authentication, and replay attack
- Public key cryptographic key exchange and authentication
  - Protocol evolution
  - Man-in-the-middle attack

Key Management

- Distributions of cryptographic keys
- Mechanisms used to bind an identity to a key
- Generation, maintenance, and revoking the keys
- Assumption and definition
  - Meaning of *a user's key*
    - e.g., Bob's key: a key bound to the identify "Bob"
  - Assume that authentication has been completed and that identify is assigned
    - Chapter 11 Authentication
    - Chapter 13. Representing Identify

Notation

- $X \rightarrow Y : \{ Z \,||\, W \}_{k_{X,Y}}$
  - 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
- $A \rightarrow T : \{ Z \}_{k_A} \,||\, \{ W \}_{k_{A,T}}$
  - 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, i.e., nonrepeating random numbers
- Alice, Bob: commonly used placeholder names in cryptography and computer security

Session and Interchange Keys

- Interchange key
  - A cryptographic key associated with a principal to a communication
- Session key
  - A cryptographic key associated with the communication itself

Example

- 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*
  - k_s called a ***session key***: may change each communication
- She enciphers k_s with Bob's public key k_B
  - k_B enciphers all session keys Alice uses to communicate with Bob
  - k_B called an ***interchange key***: do not change often
- Alice sends to Bob $\{m\}_{k_s} \mid\mid \{k_s\}_{k_B}$

Session Key: Benefits

- Make cryptanalysis more difficult
  - Limits amount of traffic enciphered with single key
  - Standard practice is to decrease the amount of traffic an attacker can obtain
- Prevents some attacks
  - Replay attack
  - Forward search attack

Forward Searches

- A forward search attack
  - Precomputed ciphertexts
    - The adversary enciphers all plaintexts using the target's public key
  - Intercept and compare
    - The adversary intercepts a ciphertext and compare with the precomputed ciphertexts to quickly obtain the plaintext.
- Effective when the set of plaintext messages is small
  - 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 enciphered message, compares, and gets plaintext at once

Exercise L5-1

- Recap: session key prevents forward search attack
- Question 1 in page 142 of the textbook

Key Exchange

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

Key Exchange

- Classical Cryptographic Key Exchange
  - For classical cryptographic approaches
    - Classical cryptographic approaches rely on a secrete key that shared between the two communicating parties.
    - Require effort to authenticate the origin of the key
- Public Key Cryptographic Key Exchange
  - For public key cryptographic approaches
    - Public key is readily to be shared
    - Require effort to authenticate the origin of the public key

Classical Cryptographic Key Exchange Algorithms

- Goal: let Alice and Bob get their shared key
- The shared key allows the secrete communication between Alice and Bob using a classical cryptographic method
- Key exchange algorithms go through multiple attack & fix cycles
  - Protocol → attack → fix → new protocol → attack → fix …

Recap of Design Criteria

- Key cannot be transmitted in the clear
  - Otherwise, an attacker can listen in
  - Key can be sent enciphered, or derived from exchanged data plus data not known to an eavesdropper
- 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
- Alice and Bob may trust a third party (called "Cathy" here)

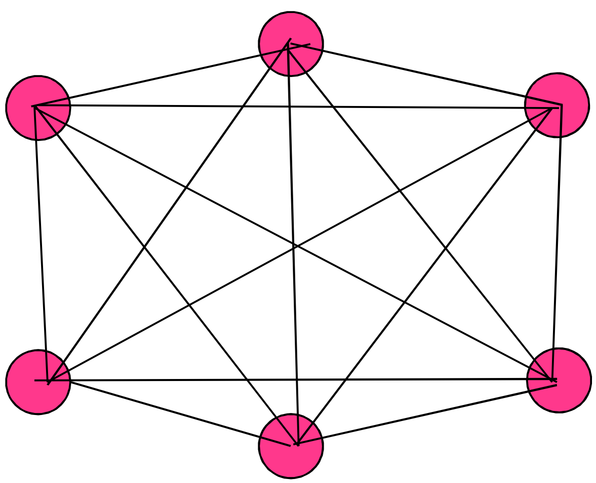
Bootstrap Problem

- Alice cannot transmit the key to Bob in the clear!
- how do Alice and Bob begin?

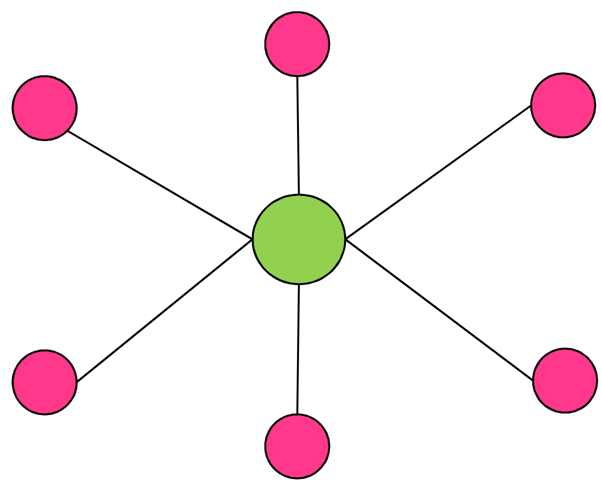
With or Without 3rd Party

- Example: share key via arranged "*physical meetings*"

Without the 3rd party



With the 3rd party

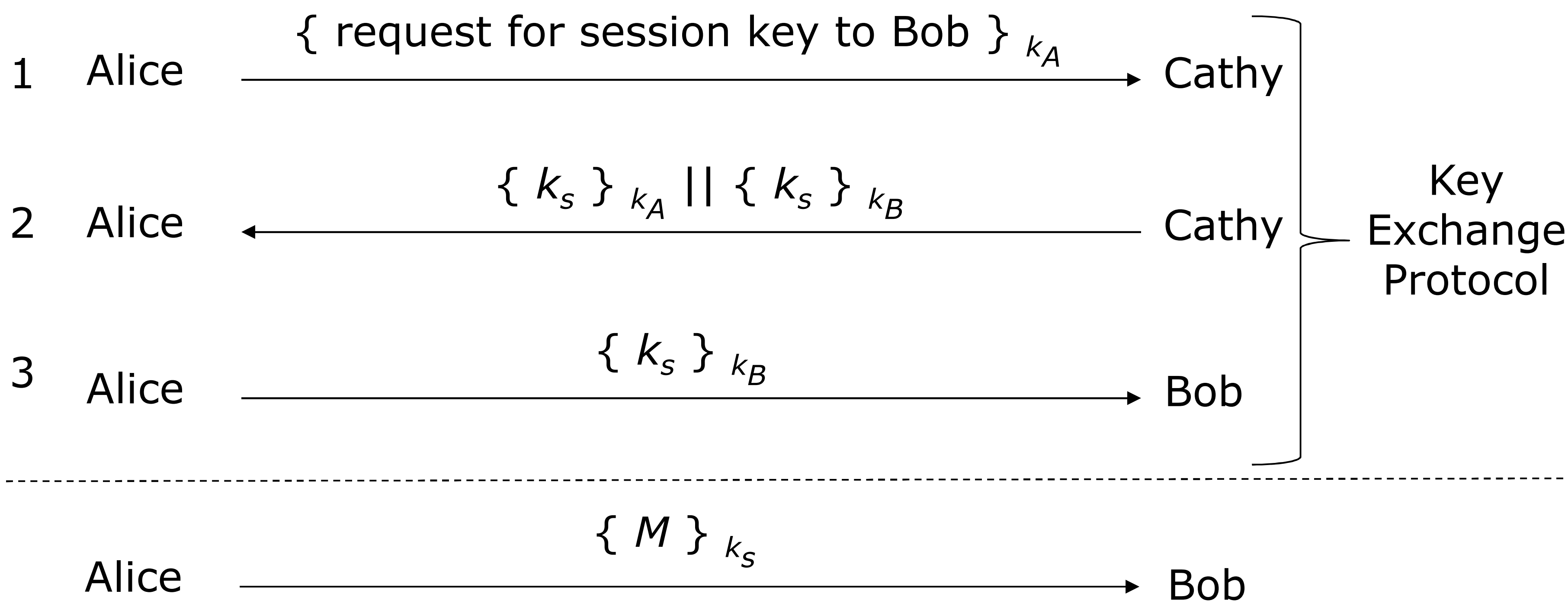


Trusted 3rd Party

- Assume trusted third party, Cathy
  - Alice and Cathy share secret key k_A
  - Bob and Cathy share secret key k_B
- Rely on Cathy to exchange shared *session key* k_s

Simple Protocol

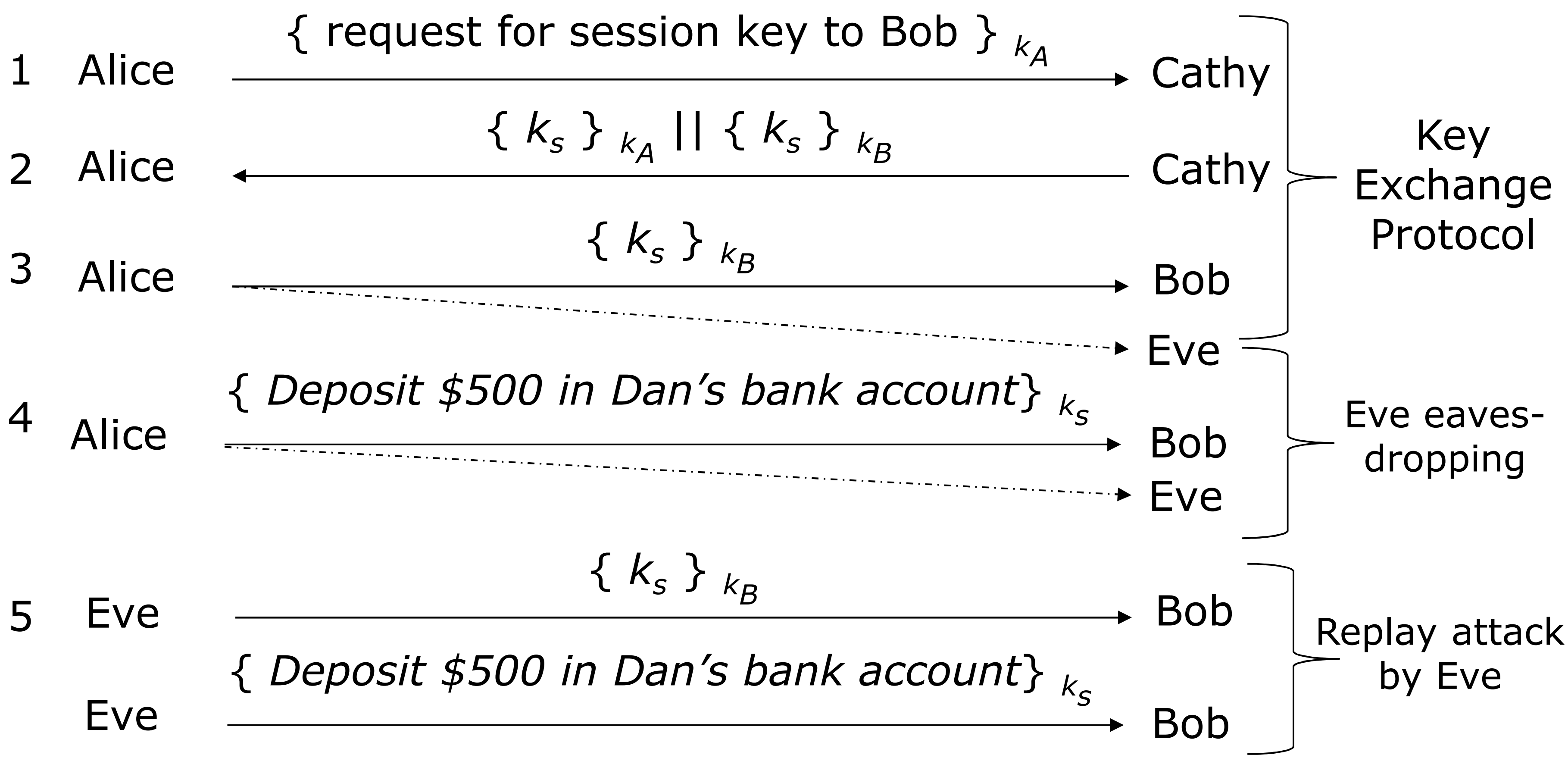
- Alice wants to start a secrete communication with Bob



Simple Protocol: Replay Attack

- Bob does not know to whom he is talking
- Replay attack
  - Alice transmits to Bob an enciphered message, e.g., {"*Deposit $500 in Dan's bank account today*"} $_{k_s}$
  - Eve eavesdrops the communication and records the message and $\{ k_s \}_{k_B}$
  - Eve later replays $\{ k_s \}_{k_B}$ followed by {"*Deposit $500 in Dan's bank account today*"} $_{k_s}$
  - Bob may think he is talking to Alice, but he is not. He is actually talking to Eve

Simple Protocol: Replay Attack



Simple Protocol: Problems

- Replay attack
  - Bob does not know to whom he is talking. Eve can record and replay messages
- Session key reuse
  - When Eve replays message from Alice to Bob, Bob re-uses session key
- Protocols must provide authentication and defense against replay

Needham-Schroeder Protocol

- Adds authentication with random nonces

1	Alice →	Alice || Bob || r_1	Cathy
2	Alice ←	$\{$ Alice || Bob || r_1 || k_s || $\{$ Alice || k_s $\}_{k_B}$ $\}_{k_A}$	Cathy
3	Alice →	$\{$ Alice || k_s $\}_{k_B}$	Bob
4	Alice ←	$\{ r_2 \}_{k_s}$	Bob
5	Alice →	$\{ r_2 - 1 \}_{k_s}$	Bob

Authentications via Key Sharing and Nonces

- Alice needs to know she is talking to Cathy and Bob
- Bob needs to know he is talking to Alice
- How?
  - Nonces: non-repeating random numbers r_1 and r_2
  - Key sharing: shared keys (K_A and K_B) are a secret between the parties who shared the keys
- Assumption: all keys are secure
  - Alice shares K_A with Cathy and nobody else
  - Bob shares K_B with Cathy and nobody else
  - Nonces and session keys are non-repeating

Is it *Alice* that Bob is talking to?

- Third message (Alice → Bob)
  - Bob deciphered the message enciphered using key (K_B) that only he, Bob knows
  - The messages names *Alice* and contains session key K_S
  - Note that Alice does not know K_B. It must have been Cathy that provided session key and named *Alice* is other party

Is it *Alice* that Bob is talking to?

- Note that the third message only provides evidence that Alice at sometime initiated the *communication*. Is the message a replay by Eve?
- Assumption: Cathy does not recycle K_S
- Fourth message (Bob → Alice)
  - Bob initiates a *challenge, i.e.,* uses session key to determine if it is a replay from Eve
  - The challenging message contains a non-repeating random number, nonce r_2, generated by Bob.
    - If not, Alice will respond correctly in fifth message
    - If so, Eve cannot decipher r_2 and so cannot respond, or responds incorrectly

Is it *Alice* that Bob is talking to?

- Fifth message (Alice → Bob)
  - Alice answers the challenge by deciphering the message, obtaining nonce r_2, do a simple agreed computation, and returns the answer.
  - If the answer to the challenge is correct, it is *Alice* who responds the challenge
  - Eve cannot decipher r_2 and so cannot respond, or responds incorrectly
- Bob can determine if it is *Alice* that he is talking to

Is it *Bob* that Alice is talking to?

- Second message (Cathy → Alice)
  - Alice decipher the message.
  - Message enciphered using key K_A that only Cathy knows besides herself. It is Cathy who transmits the message.
  - It is a response to the first message, as r_1 in it matches r_1 in first message. The message is *fresh* and not a replay.

Is it *Bob* that Alice is talking to?

- Third message (Alice → Bob)
  - The message is received from Cathy, the trusted third party. Alice forwards the message to Bob.
  - The message is enciphered using Bob's key K_B.
  - Alice knows only Bob can read it, as only Bob can derive session key from message that is enciphered using K_B
  - Any messages enciphered with that key are from Bob

Denning & Sacco’s Argument

- Assumption of the Needham-Schroeder protocol: all keys are secure
- Question: suppose Eve can obtain session key. How does that affect the Needham-Schroeder protocol?

Denning & Sacco's Argument

- In what follows, Eve knows k_s

1 Alice → Cathy: Alice || Bob || r_1

2 Alice ← Cathy: $\{$ Alice || Bob || r_1 || k_s || $\{$ Alice || k_s $\}_{k_B}$ $\}_{k_A}$

3 Alice → Bob, Eve: $\{$ Alice || k_s $\}_{k_B}$

3 Eve → Bob: $\{$ Alice || k_s $\}_{k_B}$

4 Eve ← Bob: $\{ r_2 \}_{k_s}$

5 Eve → Bob: $\{ r_2 - 1 \}_{k_s}$

Denning-Sacco's Solution

- In protocol above, Eve impersonates Alice
- Problem: Eve replays intercepted third message in third step
- Solution: use time stamp T to detect replay

Needham-Schroeder with Denning-Sacco Modification

- Introduce a time stamp. Reject messages that are too old

1 Alice → Cathy: $\text{Alice} \,||\, \text{Bob} \,||\, r_1$

2 Cathy → Alice: $\{ \text{Alice} \,||\, \text{Bob} \,||\, r_1 \,||\, k_s \,||\, \{ \text{Alice} \,||\, T \,||\, k_s \}_{k_B} \}_{k_A}$

3 Alice → Bob: $\{ \text{Alice} \,||\, T \,||\, k_s \}_{k_B}$

4 Bob → Alice: $\{ r_2 \}_{k_s}$

5 Alice → Bob: $\{ r_2 - 1 \}_{k_s}$

Denning-Sacco's Solution: Weakness

- 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

Otway-Rees Protocol

- Corrects problems with introducing an integer *n* and avoiding using timestamp
  - That is, to detect Eve's 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

Otway-Rees Protocol

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

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

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

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

Is it *Alice* that Bob is talking to?

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

Is it *Bob* that Alice is talking to?

- Fourth message (Bob → Alice)
  - If *n* matches first message, Alice knows it is part of this protocol exchange
  - Cathy generated k_s because only she and Alice know k_A
  - Enciphered part belongs to this protocol exchange as r_1 matches r_1 in encrypted part of first message

Replay Attack

- Eve acquires old k_s, message in third step and attempts to impersonate Bob
  - $n \,||\, \{ r_1 \,||\, k_s \} k_A \,||\, \{ r_2 \,||\, k_s \}_{k_B}$
- 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

Replay Attack

- The only way that Eve can impersonate Bob is that Eve's replay is for the current key exchange
- Eve sent the relevant part *before* Bob did.
- If this is the scenario, Eve could simply listen to traffic
- No replay would be involved

Exercise L5-2

- Question 5 in pages 142-143 of the textbook

Classical Cryptographic Key Exchange in Practice

- Kerberos
  - A client, Alice, wants to use a server S.
  - Kerberos requires her to use two servers to obtain a credential that will authenticate her to S
    - First, she must authenticate herself to the Kerberos System
    - Second, she must obtain a ticket to use S
- Use Classical Cryptographic Key Exchange
  - Requires a trusted third party
- Unix & Unix-like operating systems (e.g., Linux, OS X) and Windows

Kerberos

- Authentication system
  - A client, Alice, wants to use a server *S*. Kerberos requires her to use two servers (*authentication server* and *ticket-granting server*) to obtain a credential that will authenticate her to server *S*.
  - Based on Needham-Schroeder with Denning-Sacco modification
    - Authentication server plays role of trusted third party ("Cathy")
    - Ticket: Issuer vouches for identity of requester of service
    - Authenticator (authentication server): Identifies sender

Main Idea

- User *u* authenticates to Kerberos *authentication server*
- User *u* obtains ticket $T_{u,TGS}$ for Kerberos *ticket-granting service* (TGS)
- User *u* wants to use service *s*:
  - User *u* sends (authenticator A_u, ticket $T_{u,TGS}$) to TGS asking for a *ticket for service*
  - TGS sends ticket $T_{u,s}$ to user *u*
  - User *u* sends (A_u, $T_{u,s}$) to server as a request to use *s*

Ticket

- Credential vouchering issuer has identified ticket requester
- 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
- 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 that user *u* generates for service *s*

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

where:

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

Protocol

- Where "Cathy" is the Kerberos authentication server

1 *user* → Cathy: $user \,||\, TGS$

2 *user* ← Cathy: $\{ k_{u,TGS} \}_{k_u} \,||\, T_{u,TGS}$

3 *user* → TGS: $service \,||\, A_{u,TGS} \,||\, T_{u,TGS}$

4 *user* ← TGS: $user \,||\, \{ k_{u,s} \}_{k_{u,TGS}} \,||\, T_{u,s}$

5 *user* → *service*: $A_{u,s} \,||\, T_{u,s}$

6 *user* ← *service*: $\{ t + 1 \}_{k_{u,s}}$

Analysis: Steps 1 - 2

- First two steps get user ticket to use TGS
  - User *u* can obtain session key only if *u* knows key shared with Cathy (K_u)

Analysis: Steps 3 - 6

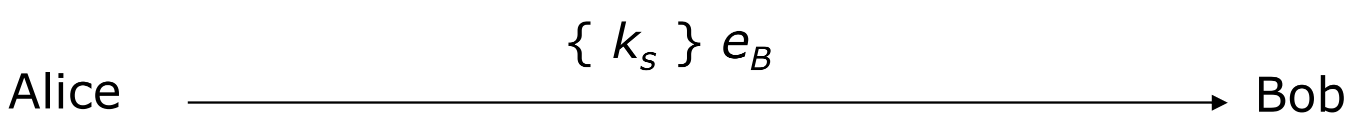
- Next four steps show how *u* gets and uses ticket for service *s*
  - Service *s* validates request by checking sender (using $A_{u,s}$) is same as entity ticket issued to
  - Step 6 optional; used when *u* requests confirmation

Problems

- Relies on synchronized clocks
  - If not synchronized and old tickets, authenticators not cached, replay is possible (Bellovin & Merritt, 1991)
- Tickets have some fixed fields
  - Dictionary attacks possible
  - Weakness in Kerberos 4 (Dole, Lodin, and Spafford, 1997)
    - Session keys weak (had much less than 56 bits of randomness);
    - Researchers at Purdue found them from tickets in minutes
- Kerberos 5
  - Improvements (e.g., adopted AES)
  - Authenticators are valid for 5 minutes

Public Key Cryptographic Key Exchange

- Public key cryptographic makes exchanging keys very easy
  - 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

- Similar flaw to the original classical key exchange protocol
- Vulnerable to forgery or replay
  - Because e_B known to anyone, Bob has no assurance that Alice sent message
  - Eve can forge such a message

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

Solution

- Authenticate Sender, i.e., Alice
  - Simple fix: Alice signs the session key K_s using her private key d_A

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

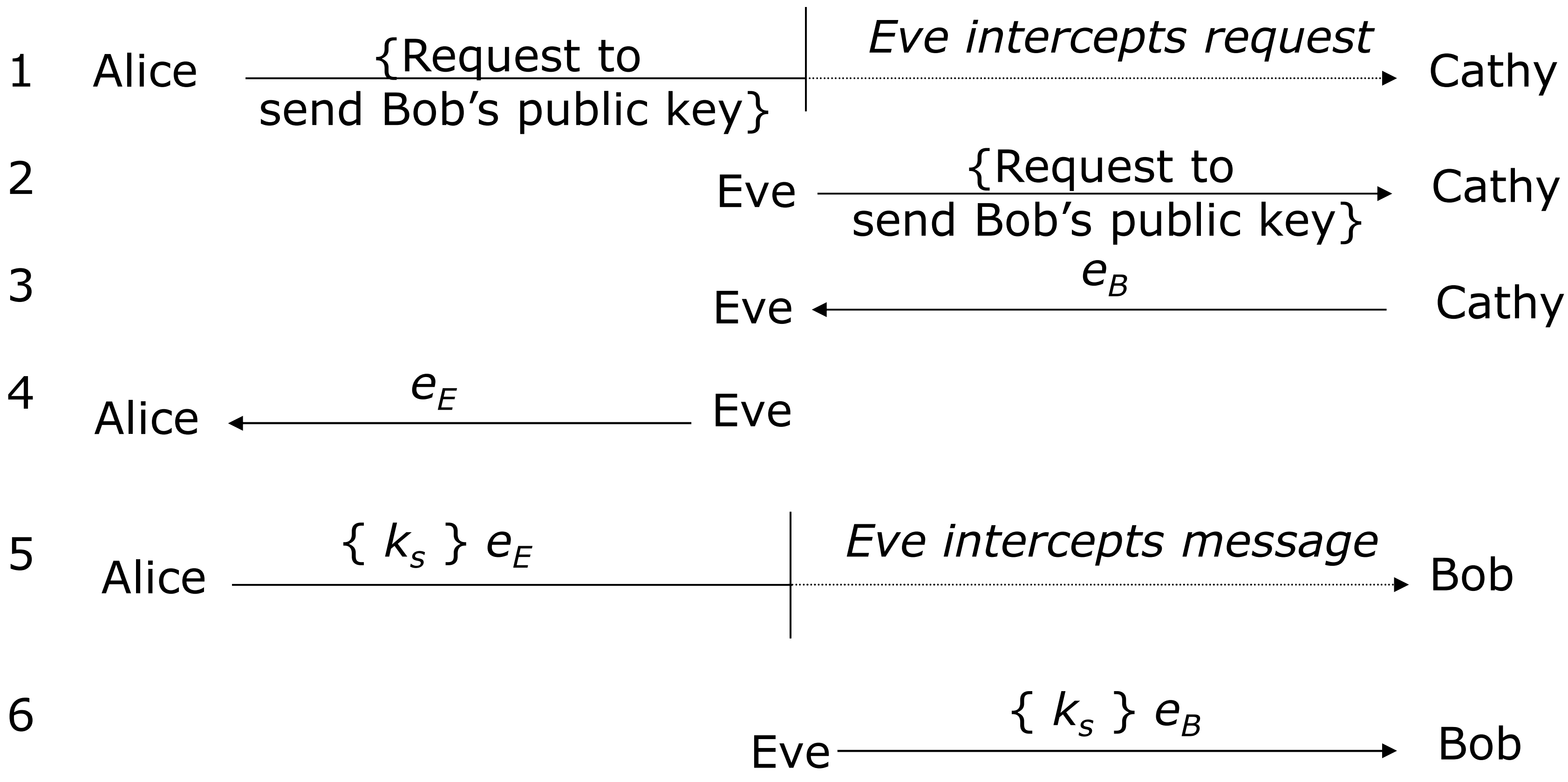
  - Bob deciphers the message using his *private key* (d_B) to obtain $\{k_s\}_{d_A}$
  - Bob deciphers $\{k_s\}_{d_A}$ using Alice *public key* and thereby *authenticates* Alice

Discussion

- Can also include message enciphered with k_s (Schneier, 1996)
- Man-in-the-middle attack
  - The above 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

Man-in-the-Middle Attack

- Cathy is public server providing public keys



Man-in-the-Middle Attack

- When presented with a public key purportedly belonging to Bob, Alice has no way to verify that the public key in fact belongs to Bob
- Solution
  - binding identity to keys
  - Discussed later as public key infrastructure (PKI)

Summary

- Key management critical to effective use of cryptosystems
  - Different levels of keys (session *vs*. interchange)
- Key Exchange for Classical Cryptography
- Key Exchange for Public Key Cryptography
- Lessons learned from attack and fix cycles