Security Protocols
CS 239
Computer Security
February 12, 2007

Outline

- Designing secure protocols
- Basic protocols
  – Key exchange

Basics of Security Protocols

- Work from the assumption (usually) that your encryption is sufficiently strong
- Given that, how do you design a message exchange to achieve a given result securely?
- Not nearly as easy as you probably think

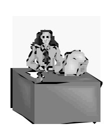
Security Protocols

- A series of steps involving two or more parties designed to accomplish a task with suitable security
- Sequence is important
- Cryptographic protocols use cryptography
- Different protocols assume different levels of trust between participants

Types of Security Protocols

- Arbitrated protocols
  – Involving a trusted third party
- Adjudicated protocols
  – Trusted third party, after the fact
- Self-enforcing protocols
  – No trusted third party

Participants in Security Protocols


Alice


Bob

Carol

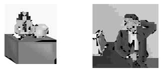


David

And the Bad Guys

Eve

Who only listens passively



And sometimes Alice or Bob might cheat

Mallory

Who is actively malicious

Trusted Arbitrator

Trent

A disinterested third party trusted by all legitimate participants

Arbitrators often simplify protocols, but add overhead

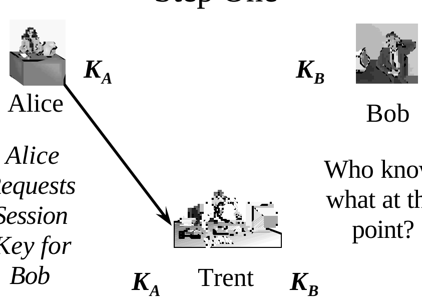
Key Exchange Protocols

- Often we want a different encryption key for each communication session
- How do we get those keys to the participants?
  - Securely
  - Quickly
  - Even if they've never communicated before

Key Exchange With Symmetric Encryption and an Arbitrator

- Alice and Bob want to talk securely with a new key
- They both trust Trent
  - Assume Alice & Bob each share a key with Trent
- How do Alice and Bob get a shared key?

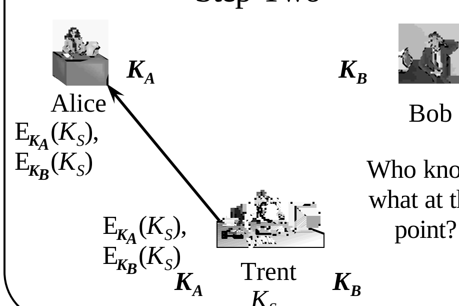
Step One



Alice Requests Session Key for Bob

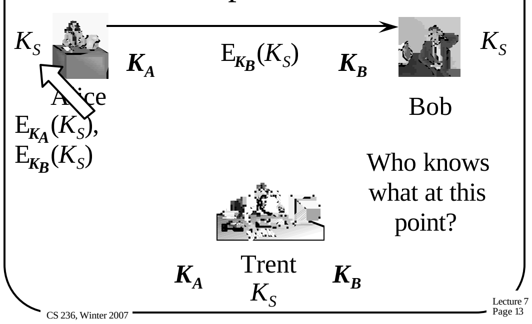
Who knows what at this point?

Step Two



Who knows what at this point?

Step Three



What Has the Protocol Achieved?

- Alice and Bob both have a new session key
- The session key was transmitted using keys known only to Alice and Bob
- Both Alice and Bob know that Trent participated
- But there are vulnerabilities

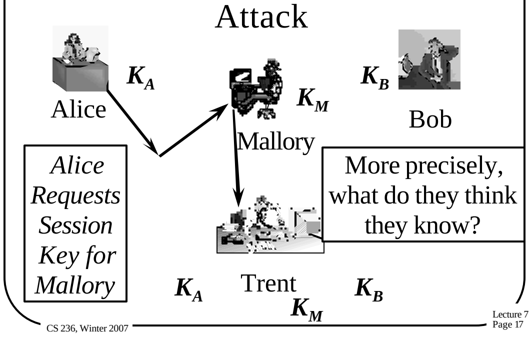
Problems With the Protocol

- What if the initial request was grabbed by Mallory?
- Could he do something bad that ends up causing us problems?
- Yes!

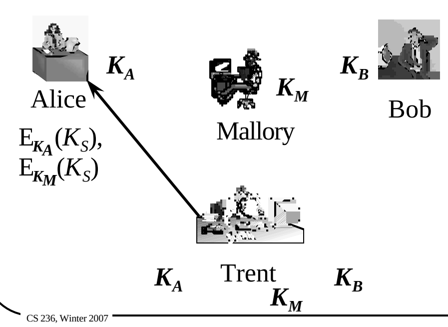
The Man-in-the-Middle Attack

- A class of attacks where an active attacker interposes himself secretly in a protocol
- Allowing alteration of the effects of the protocol
- Without necessarily attacking the encryption

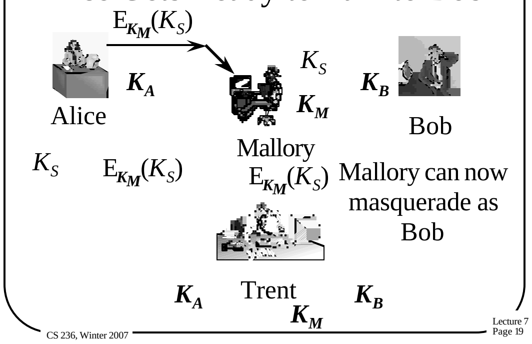
Applying the Man-in-the-Middle Attack



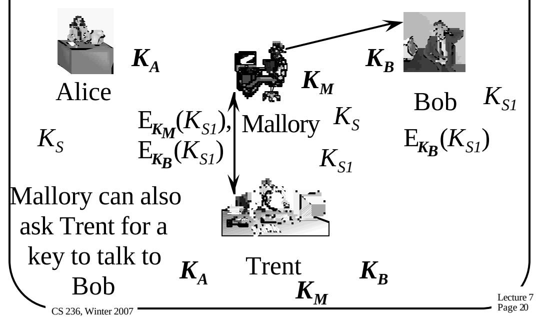
Trent Does His Job



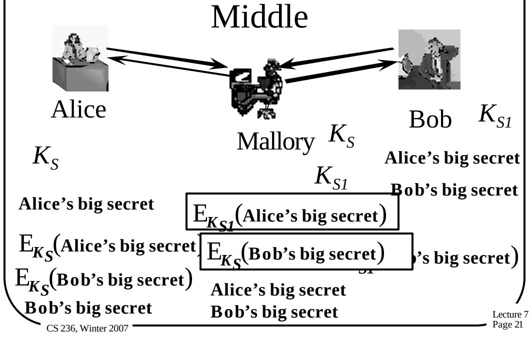
Alice Gets Ready to Talk to Bob



Really Getting in the Middle



Mallory Plays Man-in-the-Middle



Defeating the Man In the Middle

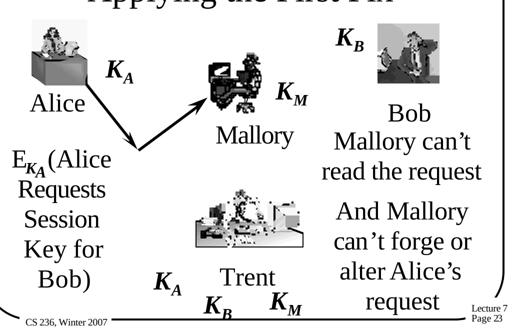
- Problems:

1). Trent doesn't really know what he's supposed to do

2). Alice doesn't verify he did the right thing

- Minor changes can fix that
  1). Encrypt request with K_A
  2). Include identity of other participant in response - $E_{K_A}(K_S$, Bob)

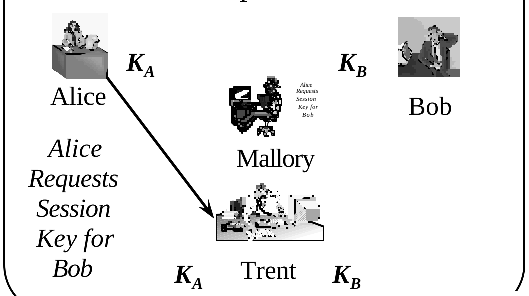
Applying the First Fix



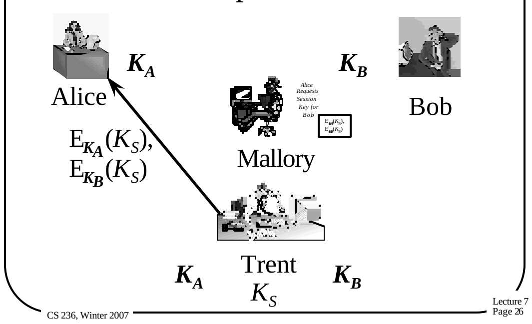
But There's Another Problem

- A replay attack
- Replay attacks occur when Mallory copies down a bunch of protocol messages
- And then plays them again
- In some cases, this can wreak havoc
- Why does it here?

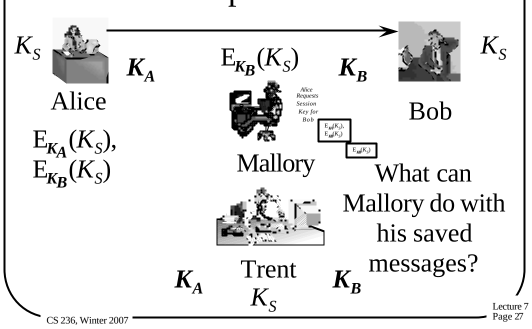
Step One



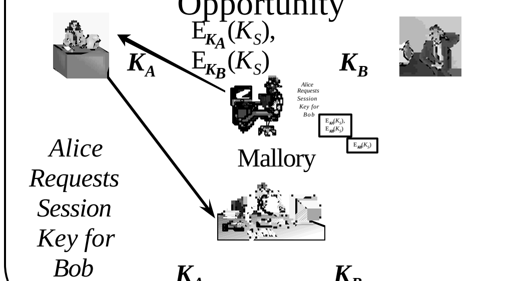
Step Two



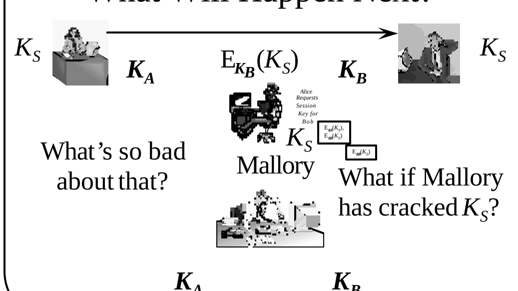
Step Three



Mallory Waits for His Opportunity



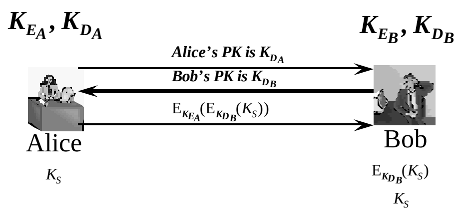
What Will Happen Next?



Key Exchange With Public Key Cryptography

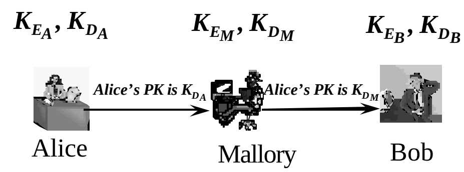
- With no trusted arbitrator
- Alice sends Bob her public key
- Bob sends Alice his public key
- Alice generates a session key and sends it to Bob encrypted with his public key, signed with her private key
- Bob decrypts Alice's message with his private key
- Encrypt session with shared session key

Basic Key Exchange Using PK



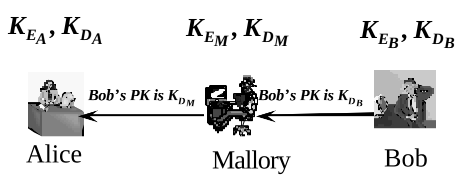
Bob verifies the message came from Alice
Bob extracts the key from the message

Man-in-the-Middle With Public Keys



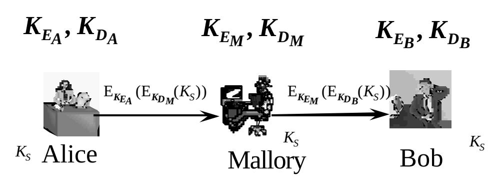
Now Mallory can pose as Alice to Bob

And Bob Sends His Public Key



Now Mallory can pose as Bob to Alice

Alice Chooses a Session Key



Bob and Alice are sharing a session key

Unfortunately, they're also sharing it with Mallory

Diffie/Hellman Key Exchange

- Securely exchange a key
  - Without previously sharing any secrets
- Alice and Bob agree on a large prime n and a number g
  - g should be primitive mod n
- n and g don't need to be secrets

Exchanging a Key in Diffie/Hellman

- Alice and Bob want to set up a session key
  - How can they learn the key without anyone else knowing it?
- Protocol assumes authentication
- Alice chooses a large random integer x and sends Bob $X = g^x \bmod n$

Exchanging the Key, Con't

- Bob chooses a random large integer *y* and sends Alice $Y = g^y \bmod n$
- Alice computes $k = Y^x \bmod n$
- Bob computes $k' = X^y \bmod n$
- k and k' are both equal to $g^{xy} \bmod n$
- But nobody else can compute *k* or *k'*

Why Can't Others Get the Secret?

- What do they know?
  - *n*, *g*, *X*, and *Y*
  - Not *x* or *y*
- Knowing *X* and *y* gets you *k*
- Knowing *Y* and *x* gets you *k'*
- Knowing *X* and *Y* gets you nothing
  - Unless you compute the discrete logarithm to obtain *x* or *y*

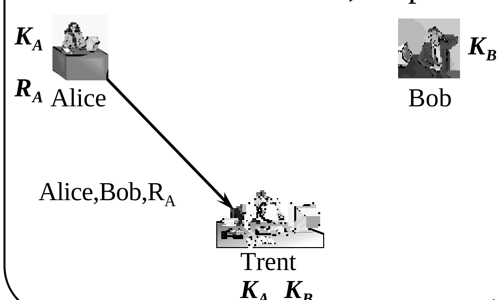
Combined Key Distribution and Authentication

- Usually the first requires the second
  - Not much good to be sure the key is a secret if you don't know who you're sharing it with
- How can we achieve both goals?
  - In a single protocol
  - With relatively few messages

Needham-Schroeder Key Exchange

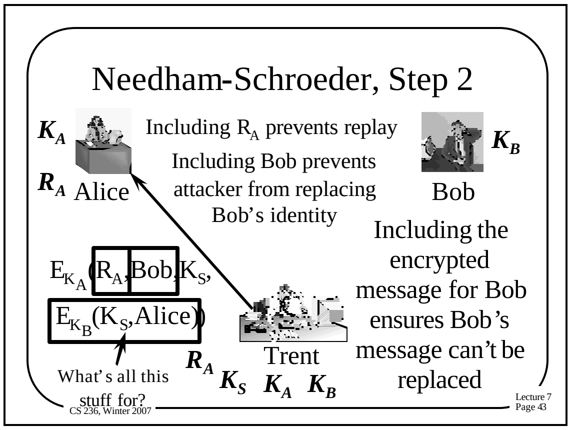
- Uses symmetric cryptography
- Requires a trusted authority
  - Who takes care of generating the new key
- More complicated than some protocols we've seen

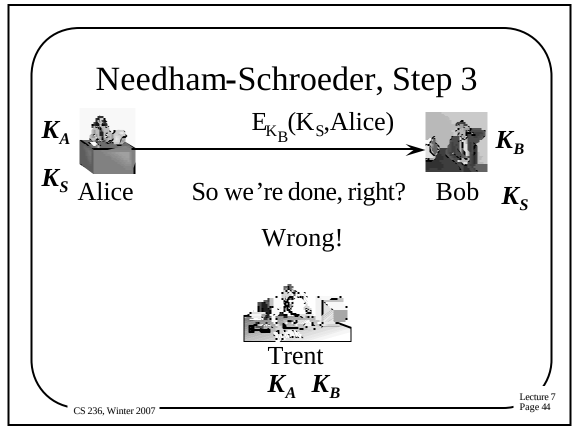
Needham-Schroeder, Step 1

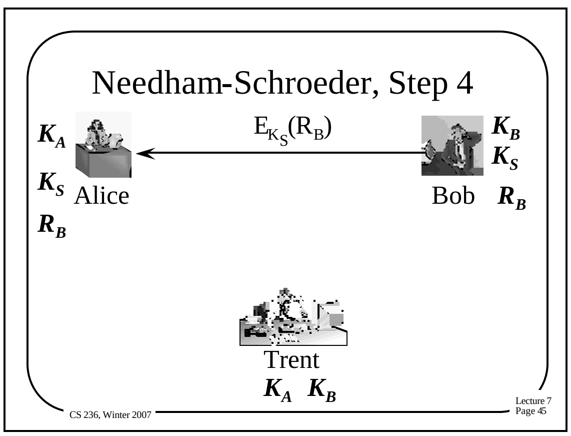


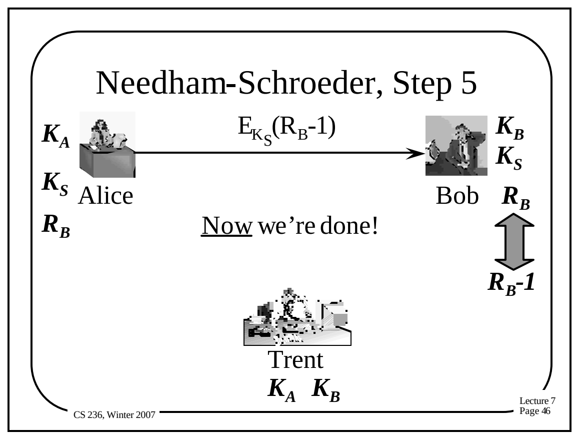
What's the Point of R_A?

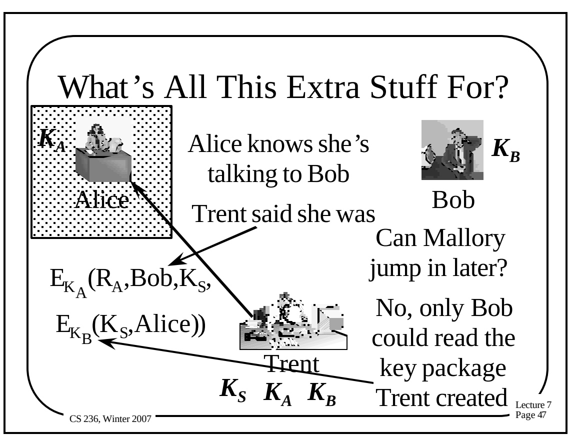
- R_A is random number chosen by Alice for this invocation of the protocol
  - Not used as a key, so quality of Alice's random number generator not too important
- Helps defend against replay attacks
- This kind of random number is sometimes called a *nonce*

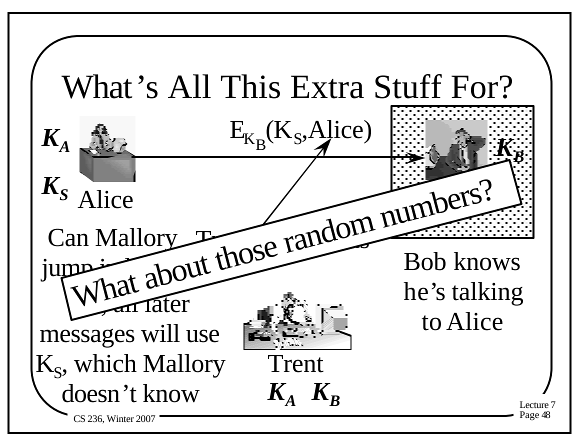

Needham-Schroeder, Step 2
K_A
R_A
Alice
Including R_A prevents replay
Including Bob prevents attacker from replacing Bob's identity
Bob
K_B
Including the encrypted message for Bob ensures Bob's message can't be replaced
E_KA(R_A,Bob,K_S,
E_KB(K_S,Alice))
What's all this stuff for?
R_A
Trent
K_S K_A K_B
CS 236, Winter 2007
Lecture 7
Page 43


Needham-Schroeder, Step 3
K_A
K_S
Alice
E_KB(K_S,Alice)
Bob
K_B
K_S
So we're done, right?
Wrong!
Trent
K_A K_B
CS 236, Winter 2007
Lecture 7
Page 44


Needham-Schroeder, Step 4
K_A
K_S
R_B
Alice
E_KS(R_B)
Bob
K_B
K_S
R_B
Trent
K_A K_B
CS 236, Winter 2007
Lecture 7
Page 45


Needham-Schroeder, Step 5
K_A
K_S
R_B
Alice
E_KS(R_B-1)
Bob
K_B
K_S
R_B
Now we're done!
R_B-1
Trent
K_A K_B
CS 236, Winter 2007
Lecture 7
Page 46

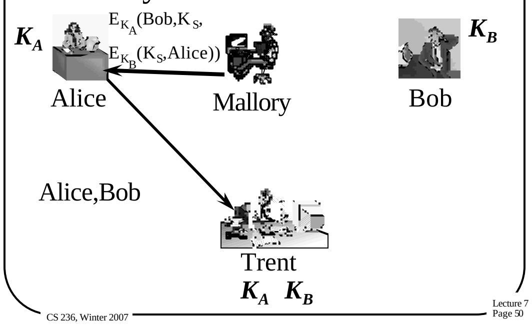

What's All This Extra Stuff For?
K_A
Alice
Alice knows she's talking to Bob
Trent said she was
Bob
K_B
Can Mallory jump in later?
No, only Bob could read the key package Trent created
E_KA(R_A,Bob,K_S,
E_KB(K_S,Alice))
Trent
K_S K_A K_B
CS 236, Winter 2007
Lecture 7
Page 47


What's All This Extra Stuff For?
K_A
K_S
Alice
E_KB(K_S,Alice)
K_B
What about those random numbers?
Bob knows he's talking to Alice
messages will use K_S, which Mallory doesn't know
Trent
K_A K_B
CS 236, Winter 2007
Lecture 7
Page 48

Mallory Causes Problems

- Alice and Bob do something Mallory likes
- Mallory watches the messages they send to do so
- Mallory wants to make them do it again
- Can Mallory replay the conversation?
  - Let's try it without the random numbers

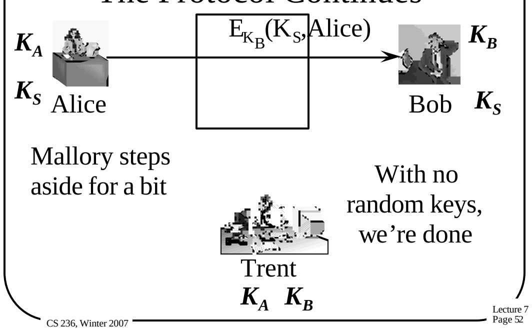
Mallory Waits For His Chance



What Will Alice Do Now?

- The message could only have been created by Trent
- It properly indicates she wants to talk to Bob
- It contains a perfectly plausible key
- Alice will probably go ahead with the protocol

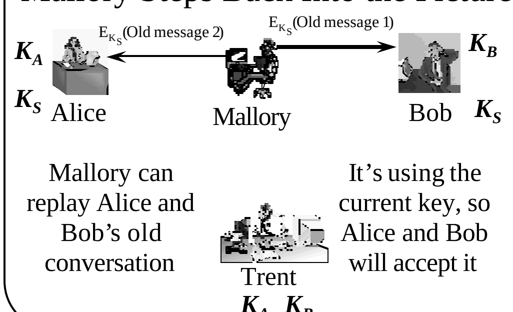
The Protocol Continues



So What's the Problem

- Alice and Bob agree K_S is their key
  - They both know the key
  - Trent definitely created the key for them
  - Nobody else has the key
- But . . .

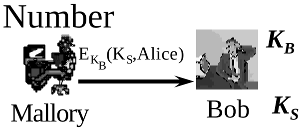
Mallory Steps Back Into the Picture



How Do the Random Numbers Help?

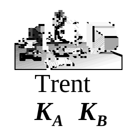
- Alice's random number assures her that the reply from Trent is fresh
- But why does Bob need another random number?

Why Bob Also Needs a Random Number

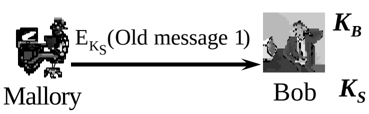


Let's say Alice doesn't want to talk to Bob

But Mallory wants Bob to think Alice wants to talk



So What?



Mallory can now play back an old message from Alice to Bob

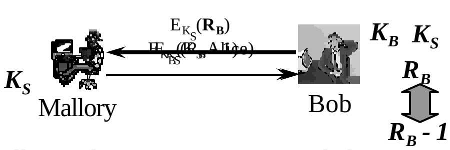
And Bob will have no reason to be suspicious

Bob's random number exchange assures him that Alice really wanted to talk

So, Everything's Fine, Right?

- Not if any key K_S ever gets divulged
- Once K_S is divulged, Mallory can forge Alice's response to Bob's challenge
- And convince Bob that he's talking to Alice when he's really talking to Mallory

Mallory Cracks an Old Key



Mallory enlists 10,000 computers belonging to 10,000 grandmothers to crack K_S

Unfortunately, Mallory knows K_S

So Mallory can answer Bob's challenge

Timestamps in Security Protocols

- One method of handling this kind of problem is timestamps
- Proper use of timestamps can limit the time during which an exposed key is dangerous
- But timestamps have their own problems

Using Timestamps in the Needham-Schroeder Protocol

- The trusted authority includes timestamps in his encrypted messages to Alice and Bob
- Based on a global clock
- When Alice or Bob decrypts, if the timestamp is too old, abort the protocol

Using Timestamps to Defeat Mallory

K_S

Mallory

$E_{K_B}(K_S, Alice, T_X)$

$E_{K_B}(K_S, Alice, T_X)$

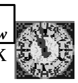
Bob

K_B

T_X

$T_X << T_{now}$

T_{now}



Now Bob checks T_X against his clock

So Bob, fearing replay, discards ***K_S***

And Mallory's attack is foiled

Problems With Using Timestamps

- They require a globally synchronized set of clocks
  - Hard to obtain, often
  - Attacks on clocks become important
- They leave a window of vulnerability

The Suppress-Replay Attack

- Assume two participants in a security protocol
  - Using timestamps to avoid replay problems
- If the sender's clock is ahead of the receiver's, attacker can intercept message
  - And replay later, when receiver's clock still allows it

Handling Clock Problems

1). Rely on clocks that are fairly synchronized and hard to tamper
- Perhaps GPS signals

2). Make all comparisons against the same clock
- So no two clocks need to be synchronized

Neuman-Stubblebine Protocol, Step 1

K_A

R_A Alice

Alice, R_A

R_A

K_B

Bob

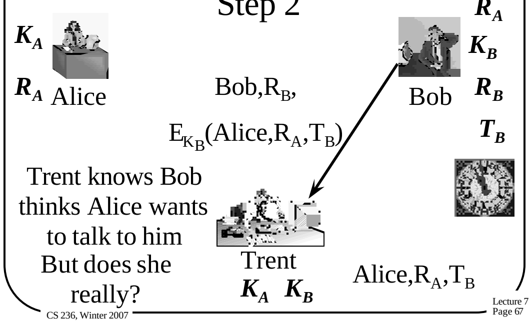
What does Bob know?

Trent

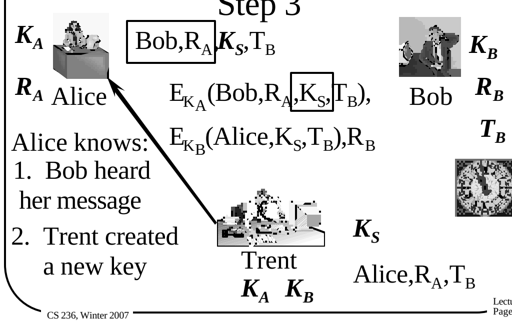
K_A K_B

Someone claiming to be Alice wants to talk securely

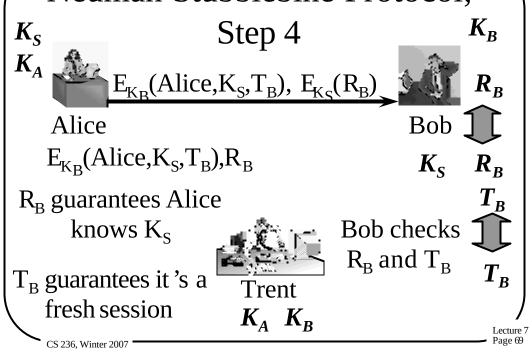
Neuman-Stubblebine Protocol, Step 2



Neuman-Stubblebine Protocol, Step 3



Neuman-Stubblebine Protocol, Step 4



What Has the Protocol Achieved?

- Alice and Bob share a key
- They know the key was generated by Trent
- Alice knows this key matches her recent request for a key
- Bob knows this key matches Alice's recent request and Bob's agreement

What Has the Timestamp Done For Bob and Alice?

- Bob knows that the whole agreement is timely
- Since the only timestamp originated with his clock, no danger of suppress-replay attacks

What Else Can You Do With Security Protocols?

- Secret sharing
- Fair coin flips and other games
- Simultaneous contract signing
- Secure elections
- Lots of other neat stuff