

# Security Protocols

## Computer Security

Peter Reiher

April 17, 2014

# Outline

- Designing secure protocols
- Key exchange protocols
- Common security problems in protocols



# Basics of Security Protocols

- Assume (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
- Many of the concepts are important in many areas of computer/network security

# 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



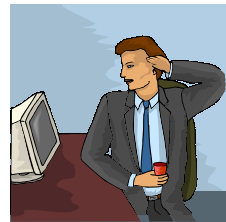
Bob

# 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 and may limit applicability

# Goals of Security Protocols

- Each protocol is intended to achieve some very particular goal
  - Like setting up a key between two parties
- Protocols may only be suitable for that particular purpose
- Important secondary goal is minimalism
  - Fewest possible messages
  - Least possible data
  - Least possible encryption

# 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



$K_A$

Alice

*Alice  
Requests  
Session  
Key for  
Bob*



$K_B$

Bob

Who knows  
what at this  
point?



$K_A$

Trent

$K_B$

## Step Two



$K_A$

Alice

$E_{K_A}(K_S),$   
 $E_{K_B}(K_S)$



$K_B$

Bob

Who knows  
what at this  
point?

$E_{K_A}(K_S),$   
 $E_{K_B}(K_S)$

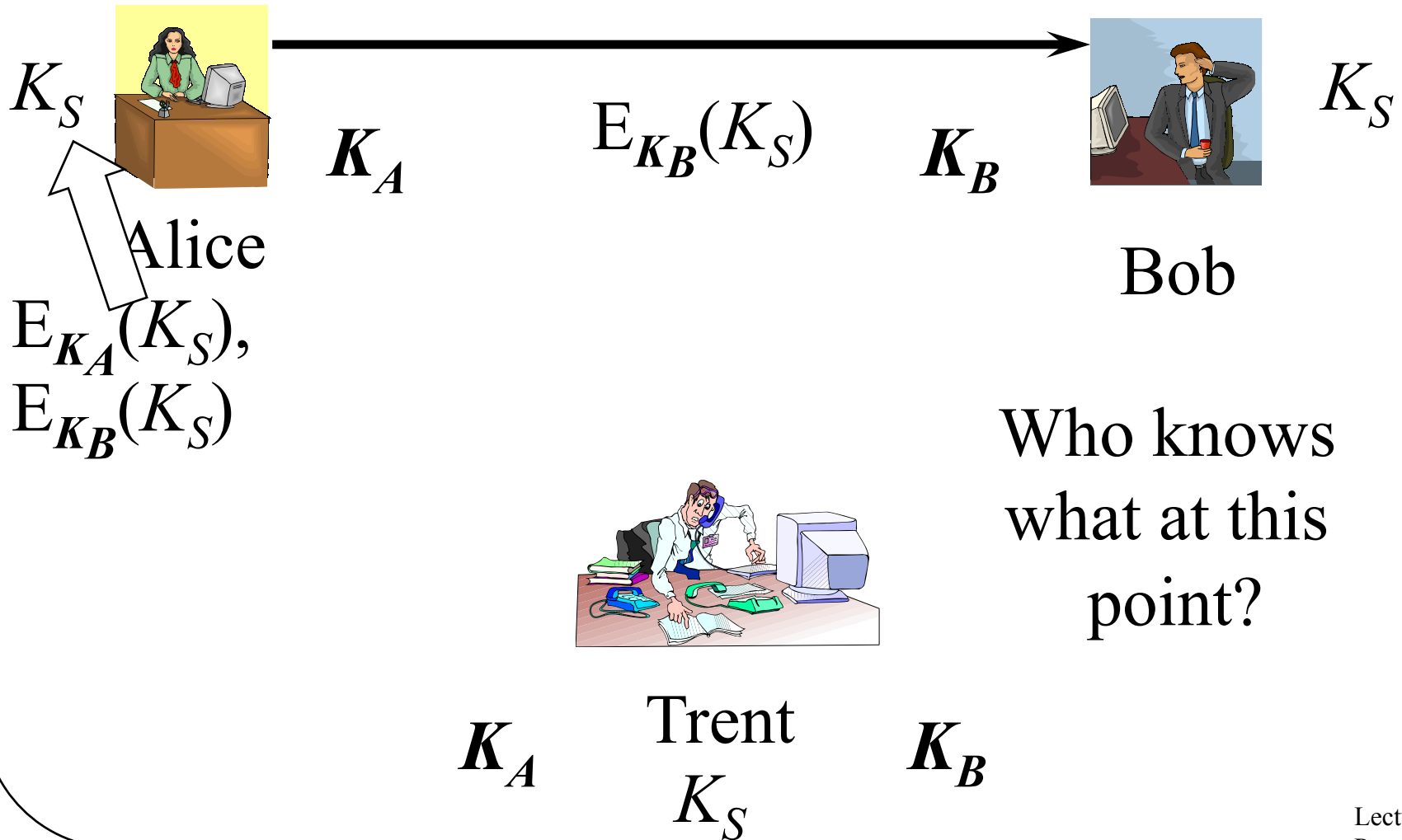


$K_A$

Trent  
 $K_S$

$K_B$

# 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

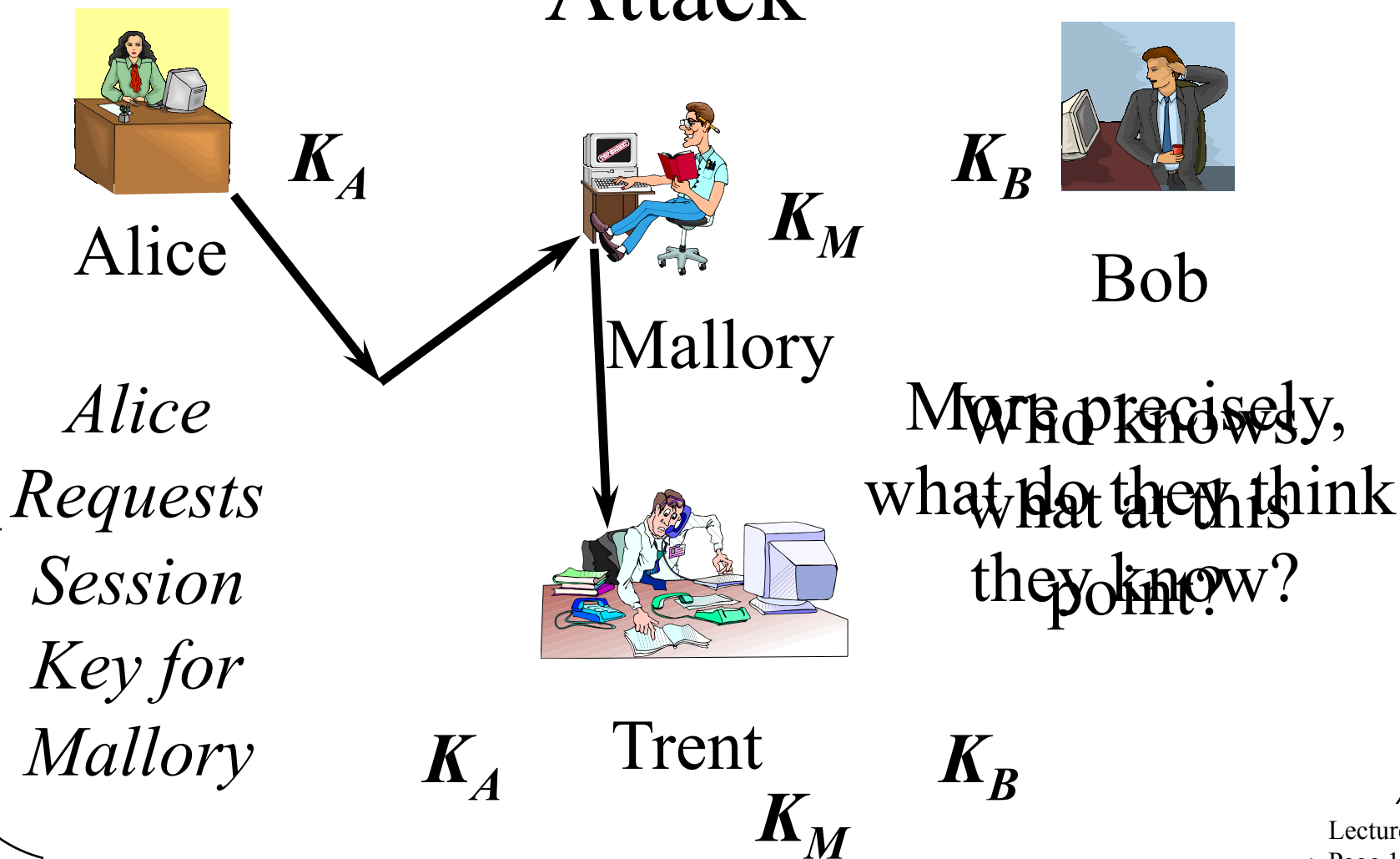
# Problems With the Protocol

- 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

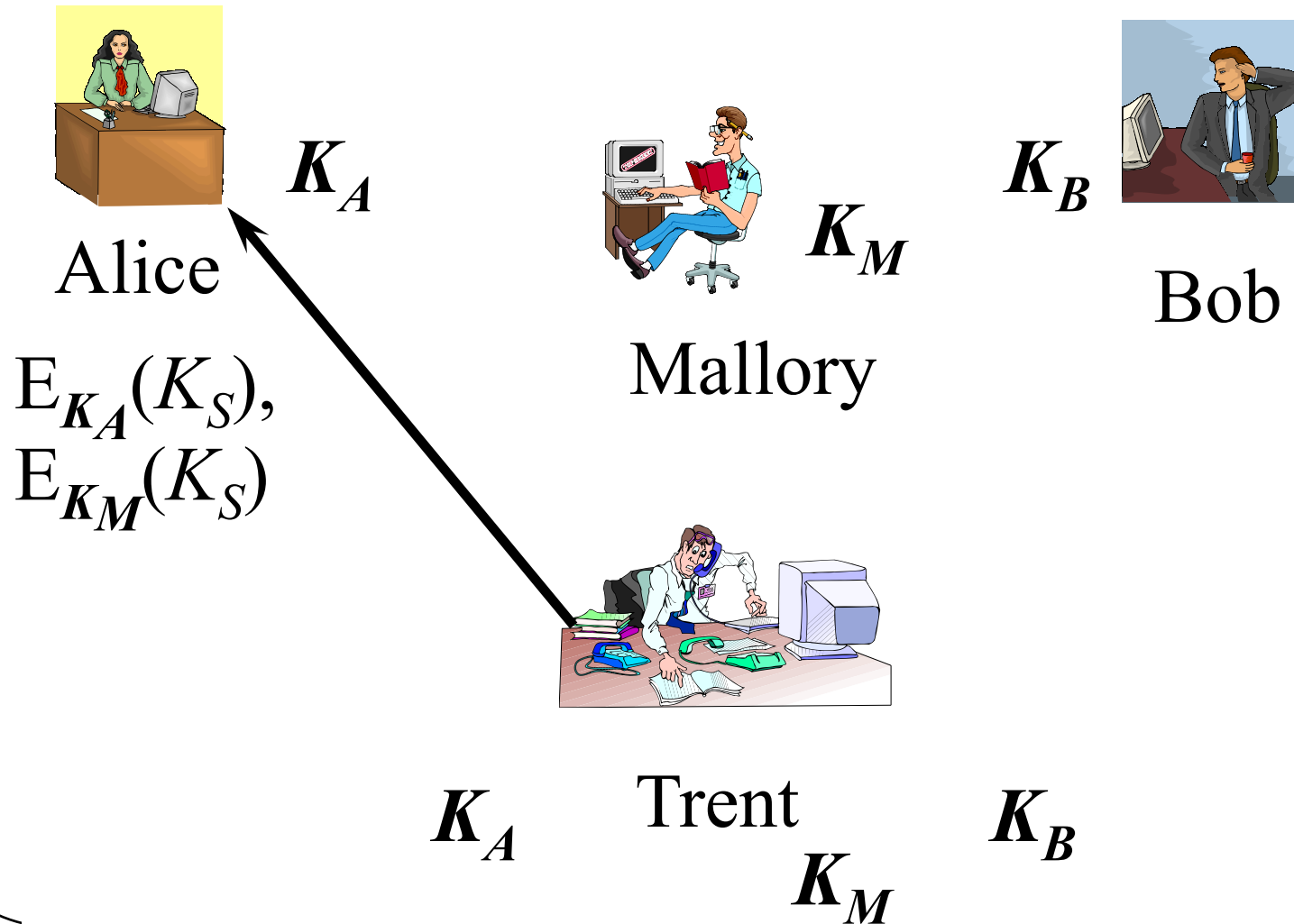
- A class of attacks where an active attacker interposes himself secretly in a protocol
- Allowing alteration of the effects of the protocol
- Without necessarily attacking the encryption

# Applying the Man-in-the-Middle Attack

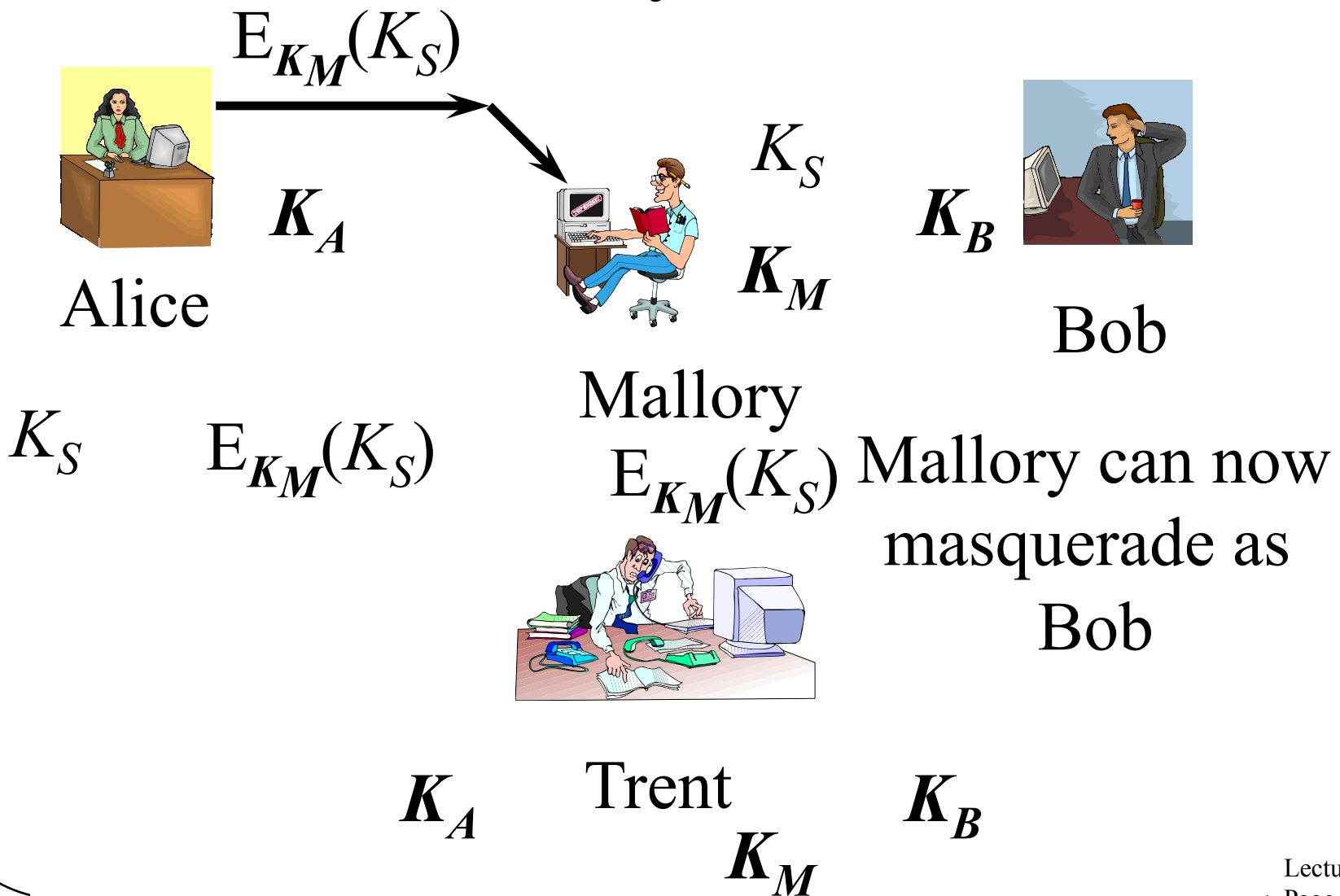




# Trent Does His Job



# Alice Gets Ready to Talk to Bob



# Really Getting in the Middle



$K_A$

Alice

$K_S$



$K_M$

Mallory

$E_{K_M}(K_{S1})$   
 $E_{K_B}(K_{S1})$

$K_B$



Bob

$K_{S1}$

$E_{K_B}(K_{S1})$



Trent

$K_M$

$K_A$

$K_B$

Mallory can also  
ask Trent for a  
key to talk to  
Bob

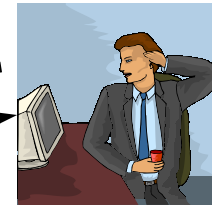
# Mallory Plays Man-in-the-Middle



Alice



Mallory



Bob

$K_{S1}$

$K_S$

Alice's big secret

$E_{K_S}(\text{Alice's big secret})$

$E_{K_S}(\text{Bob's big secret})$

Bob's big secret

$K_S$   
 $K_{S1}$

$E_{K_S}(\text{Alice's big secret})$

$E_{K_{S1}}(\text{Alice's big secret})$

$E_{K_{S1}}(\text{Bob's big secret})$

$E_{K_S}(\text{Bob's big secret})$

Alice's big secret

Bob's big secret

Alice's big secret

Bob's big secret

$E_{K_{S1}}(\text{Bob's big secret})$

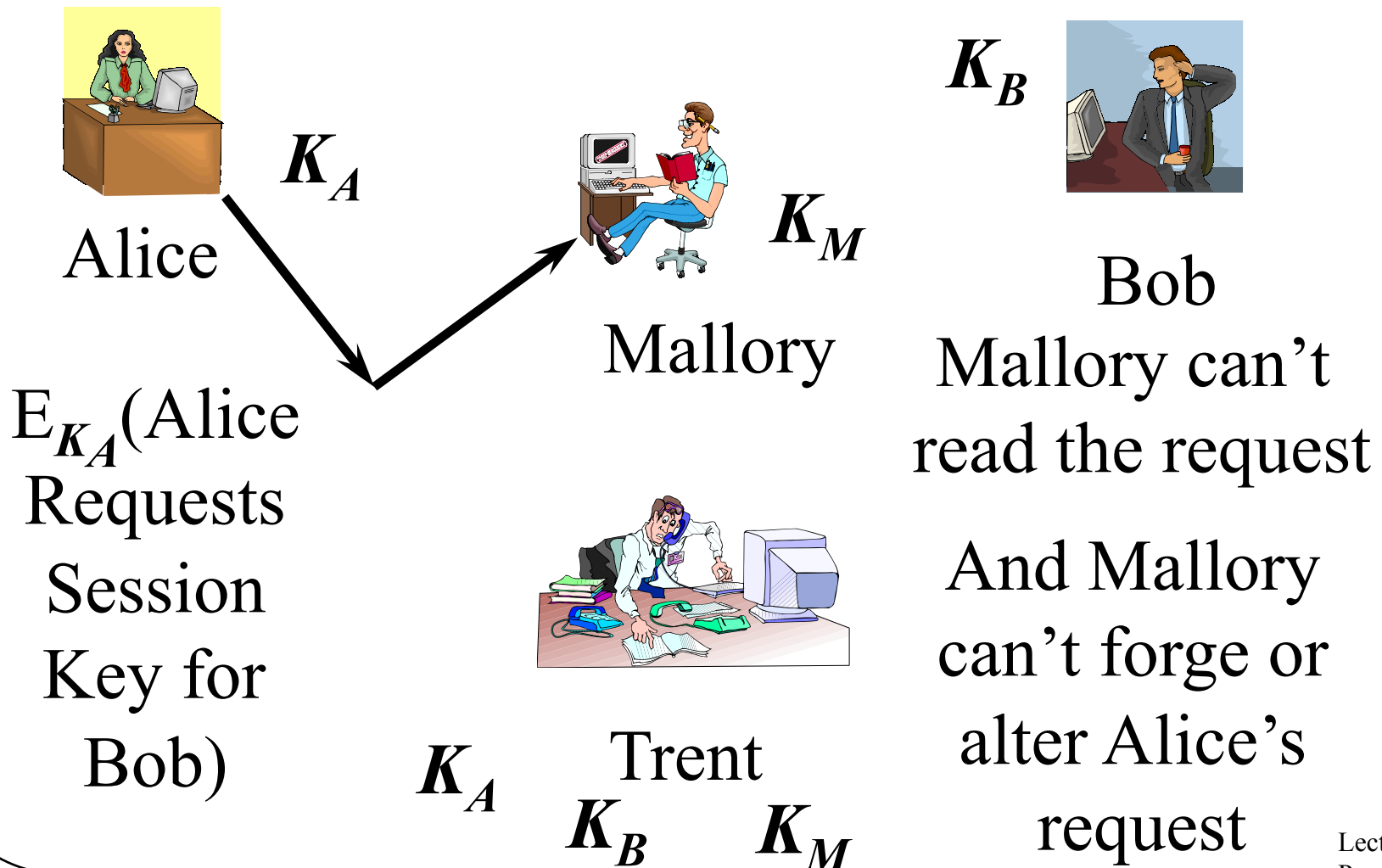
$E_{K_S}(\text{Bob's big secret})$

$E_{K_S}(\text{Bob's big secret})$

# 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 -  $E_{K_A}(K_S, \text{Bob})$

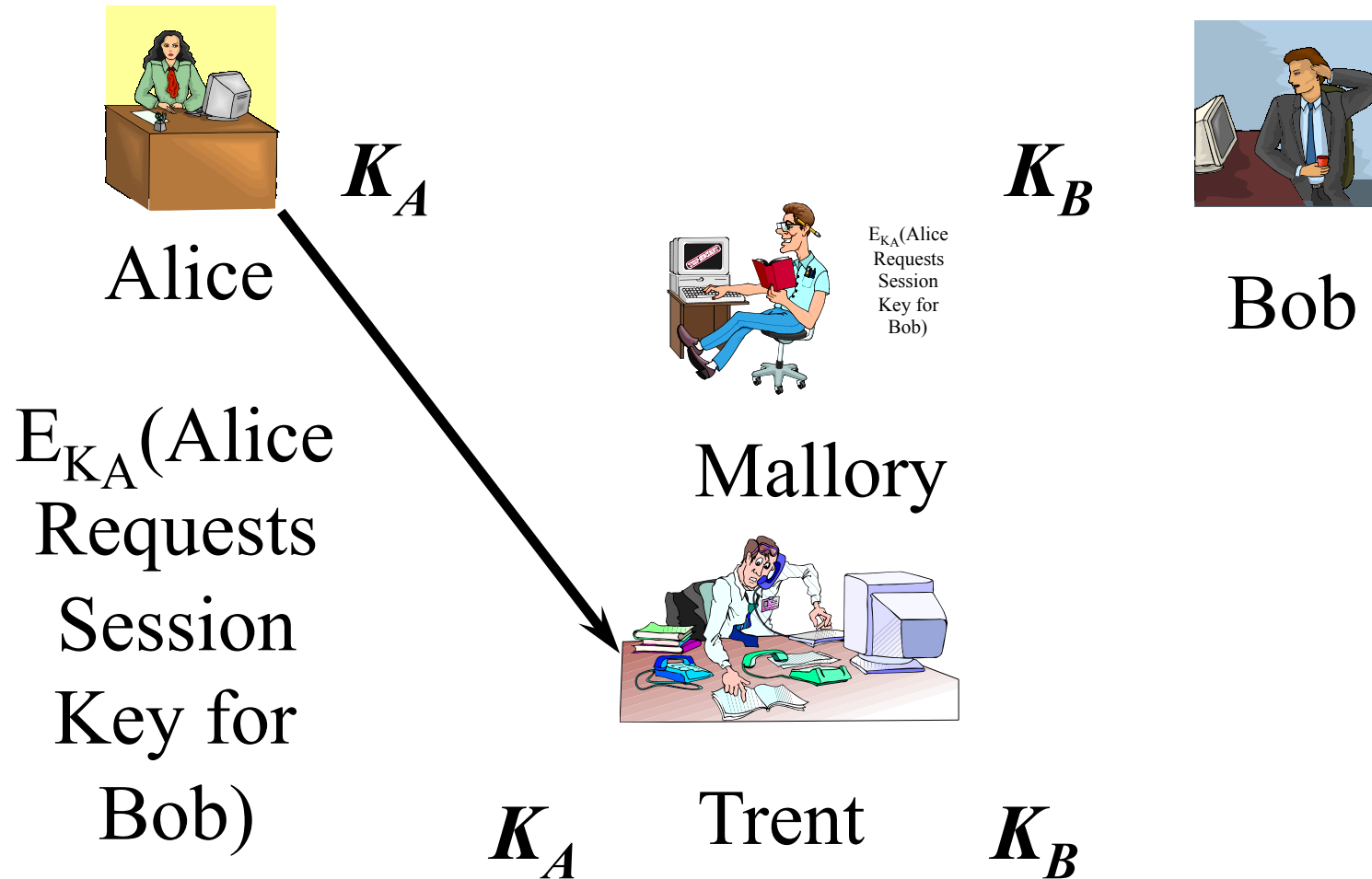
# Applying the First Fix



## 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



Alice

$K_A$

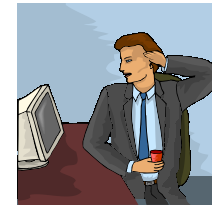
$E_{K_A}(K_S),$   
 $E_{K_B}(K_S)$



Mallory

$E_{K_A}(\text{Alice Requests Session Key for Bob})$

$E_{K_A}(K_S),$   
 $E_{K_B}(K_S)$



Bob

$K_B$

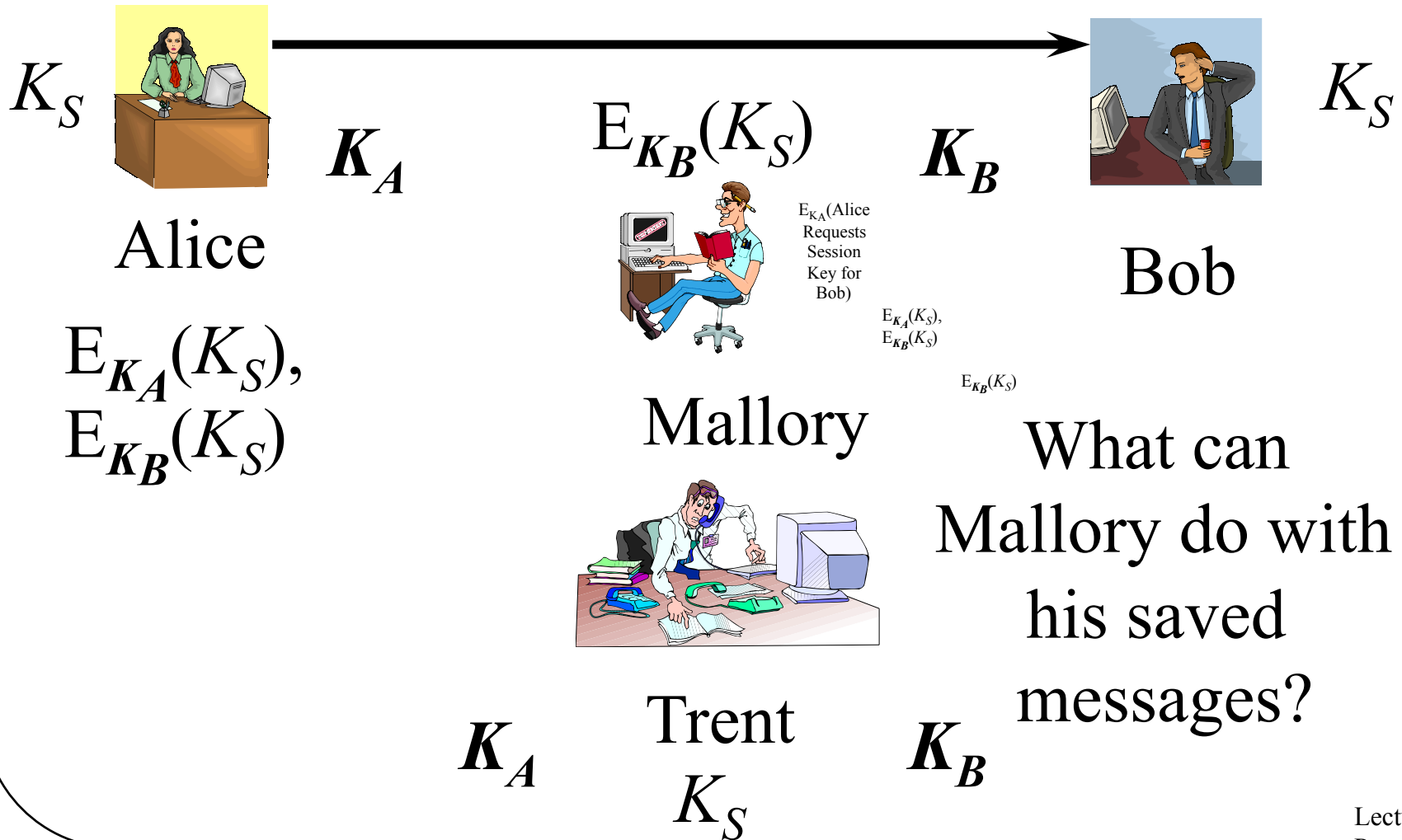


Trent  
 $K_S$

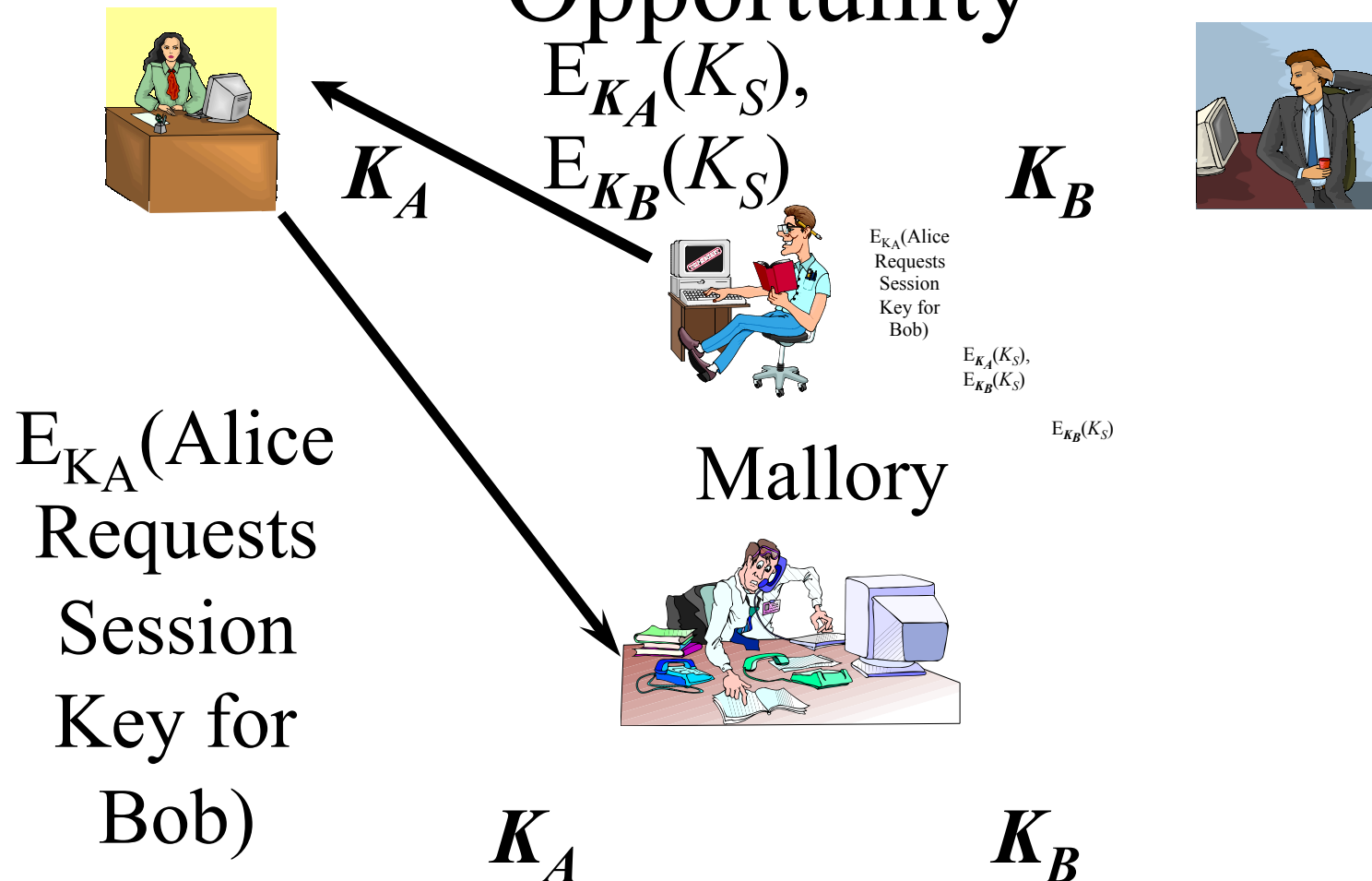
$K_A$

$K_B$

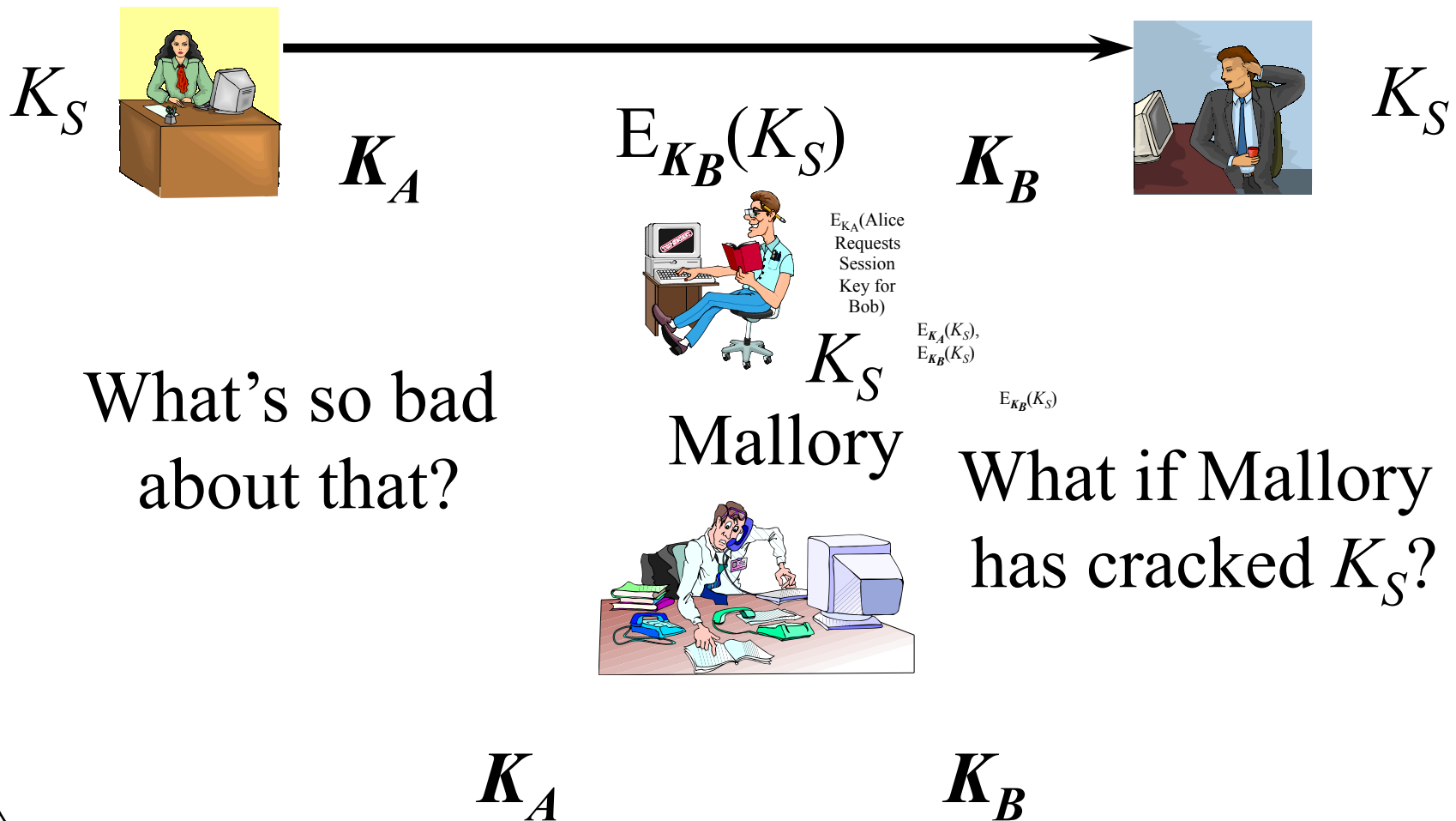
# Step Three



# Mallory Waits for His Opportunity



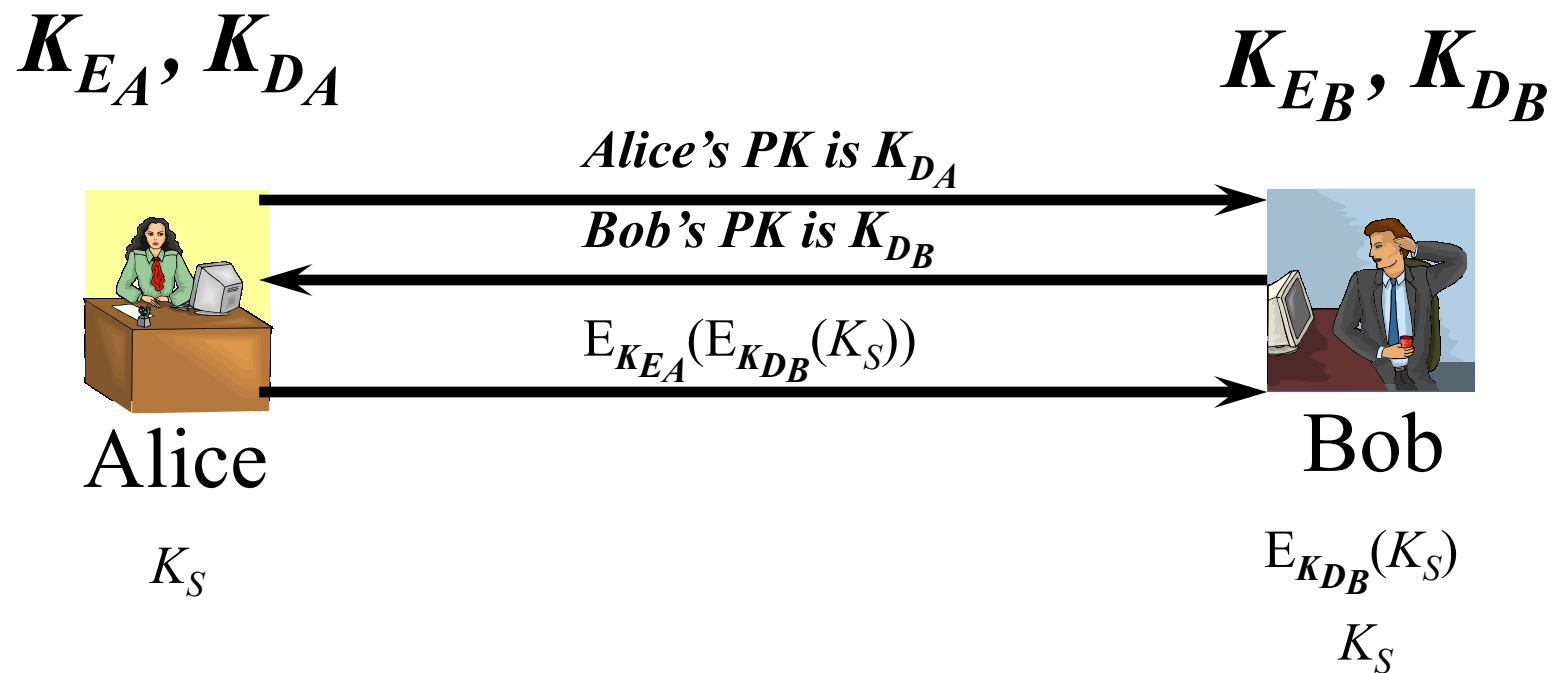
# What Will Happen Next?



# 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

$K_{E_A}, K_{D_A}$

$K_{E_M}, K_{D_M}$

$K_{E_B}, K_{D_B}$



Alice

*Alice's PK is  $K_{D_A}$*



Mallory

*Alice's PK is  $K_{D_M}$*



Bob

Now Mallory can pose as Alice to Bob

# And Bob Sends His Public Key

$K_{E_A}, K_{D_A}$

$K_{E_M}, K_{D_M}$

$K_{E_B}, K_{D_B}$



Alice

*Bob's PK is  $K_{D_M}$*



Mallory

*Bob's PK is  $K_{D_B}$*

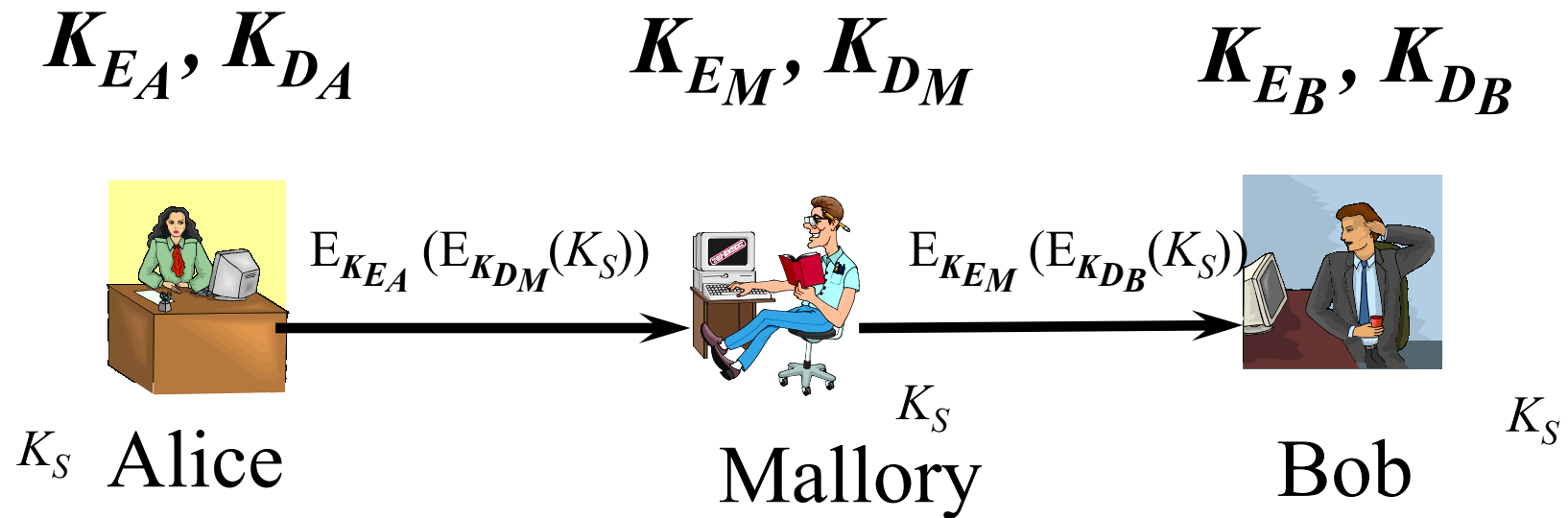


Bob

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

# 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

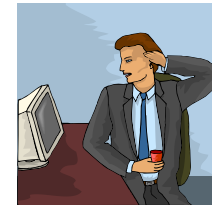
- Uses symmetric cryptography
- Requires a trusted authority
  - Who takes care of generating the new key
- More complicated than some protocols we've seen

# Needham-Schroeder, Step 1



$K_A$

$R_A$  Alice



$K_B$

Bob

Alice, Bob,  $R_A$



Trent

$K_A$   $K_B$

## 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



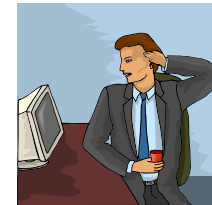
$K_A$

$R_A$  Alice

Including  $R_A$  prevents replay

Including Bob prevents

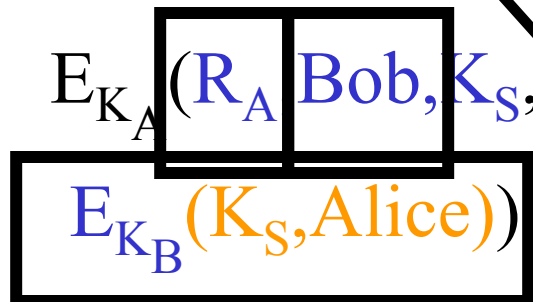
attacker from replacing Bob's identity



$K_B$

Bob

Including the encrypted message for Bob ensures Bob's message can't be replaced



Trent

$R_A$

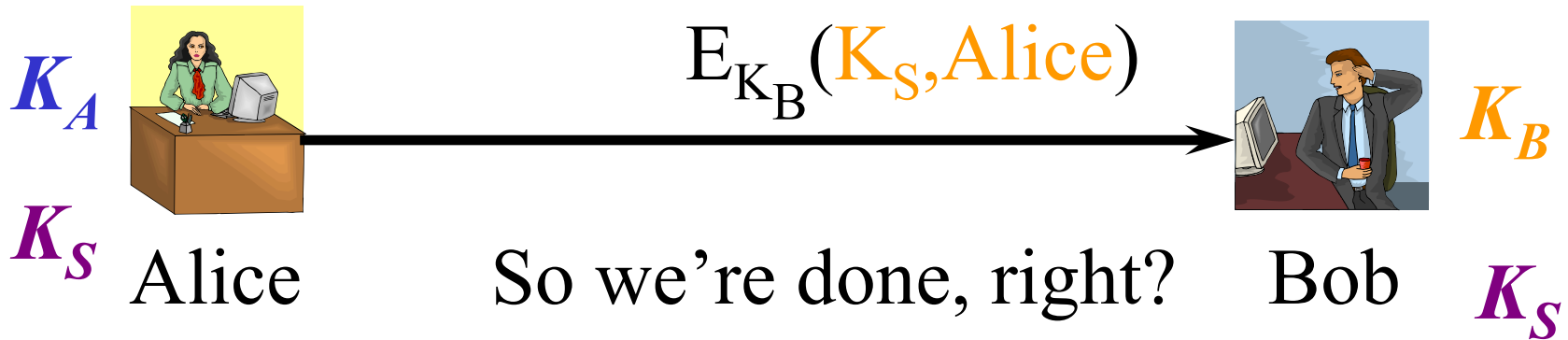
$K_S$

$K_A$

$K_B$

What's all this stuff for?

# Needham-Schroeder, Step 3



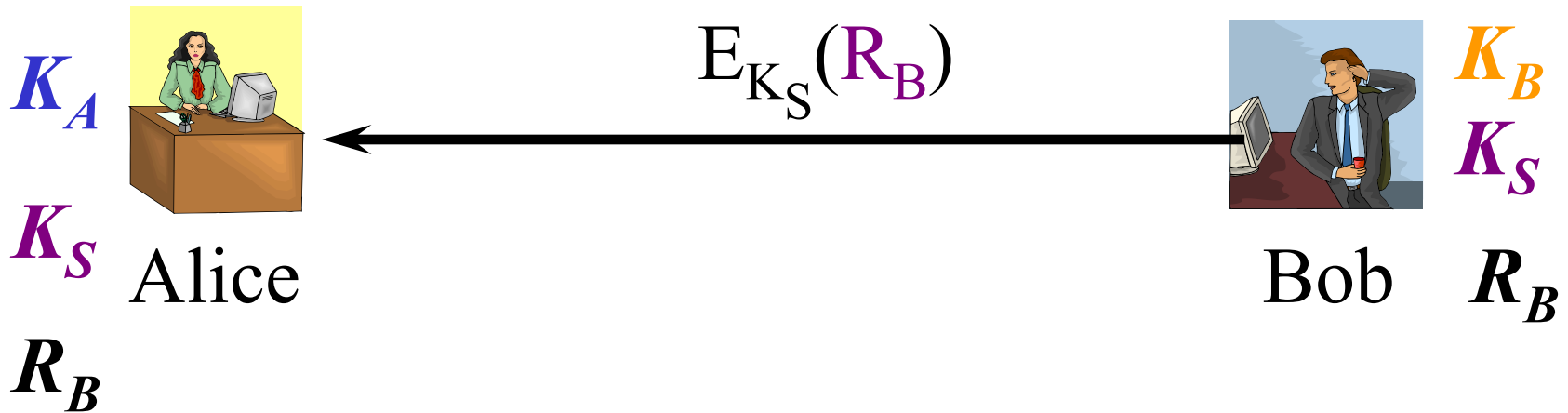
Wrong!



Trent

$K_A$   $K_B$

# Needham-Schroeder, Step 4

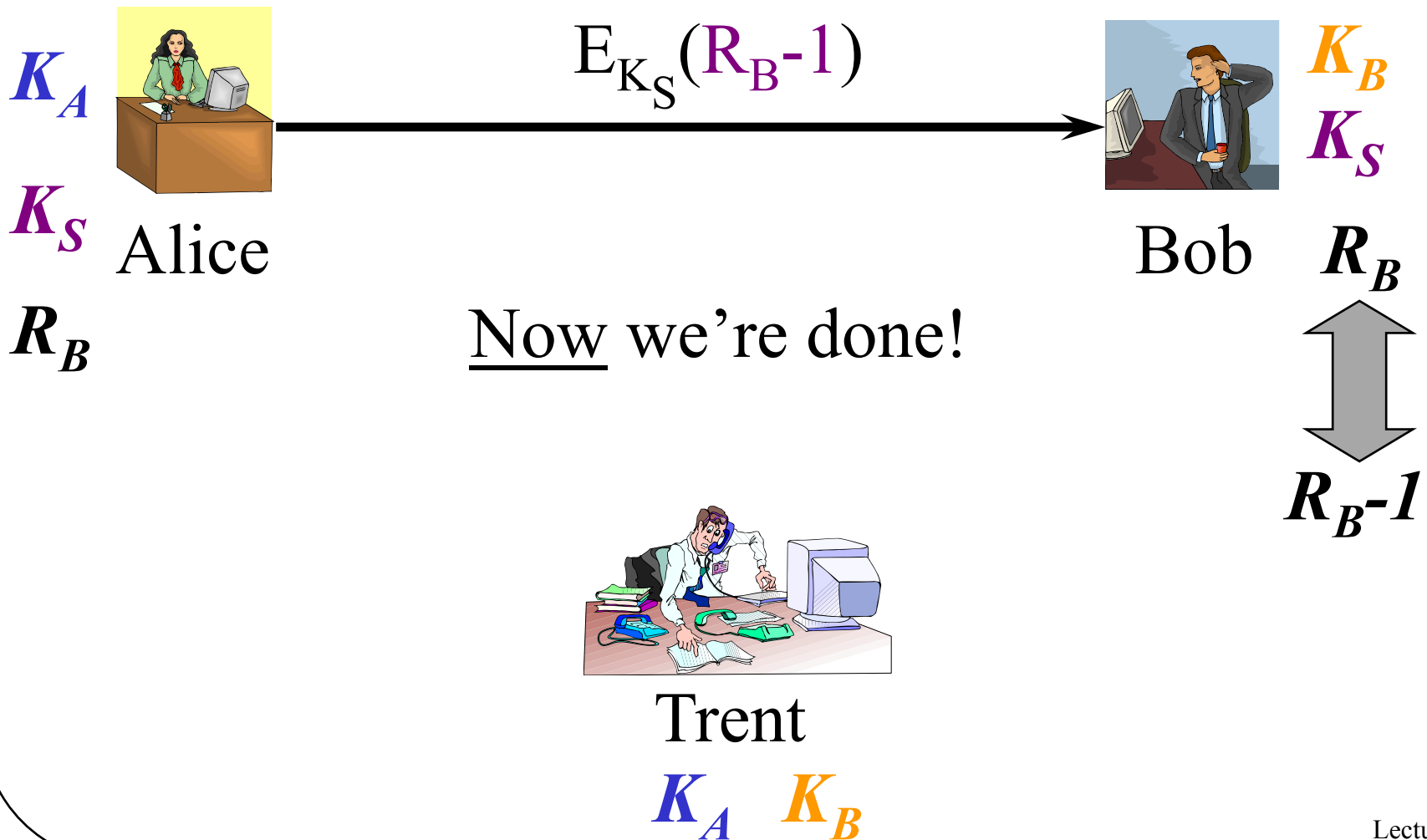


Trent

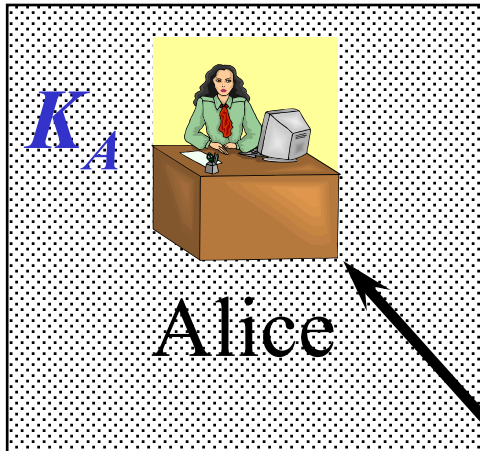
$K_A$   $K_B$



# Needham-Schroeder, Step 5



# What's All This Extra Stuff For?



Alice knows she's  
talking to Bob



Trent said she was

Can Mallory  
jump in later?

$E_{K_A}(R_A, \text{Bob}, K_S,$

$E_{K_B}(K_S, \text{Alice}))$



No, only Bob  
could read the  
key package  
Trent created

# What's All This Extra Stuff For?

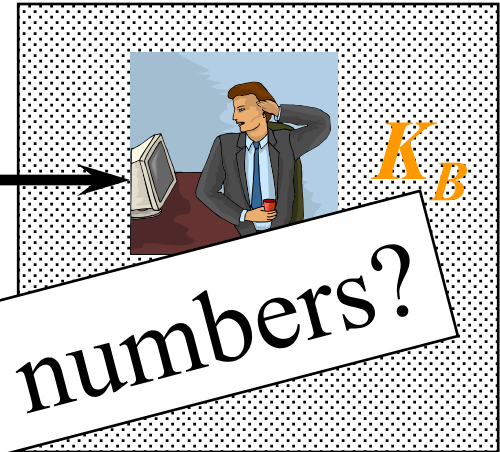


$K_A$

$K_S$

Alice

$E_{K_B}(K_S, \text{Alice})$



$K_B$

What about those random numbers?

Can Mallory  
jump in?

Bob knows  
he's talking  
to Alice

Can later  
messages will use  
 $K_S$ , which Mallory  
doesn't know



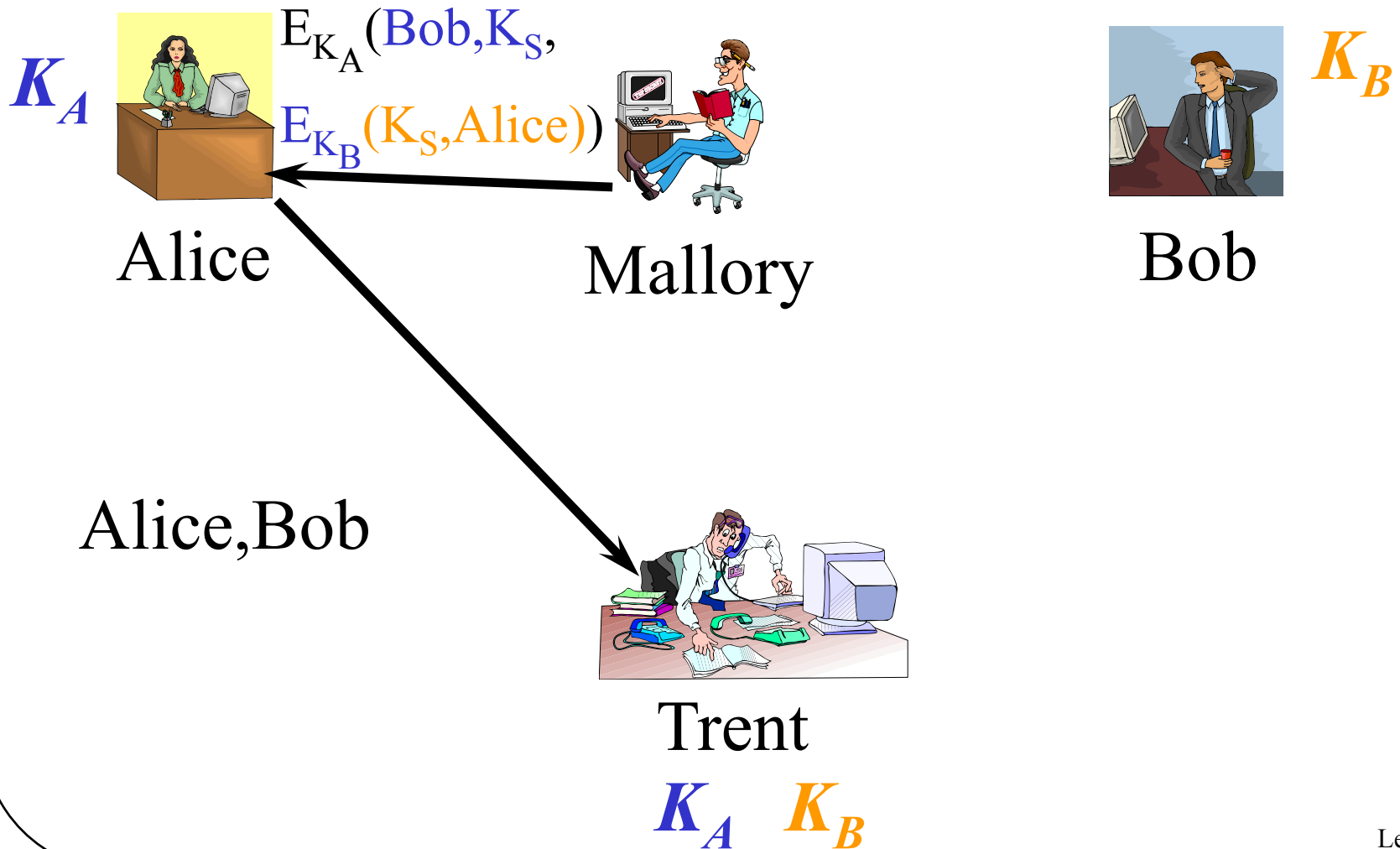
Trent

$K_A$   $K_B$

# 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

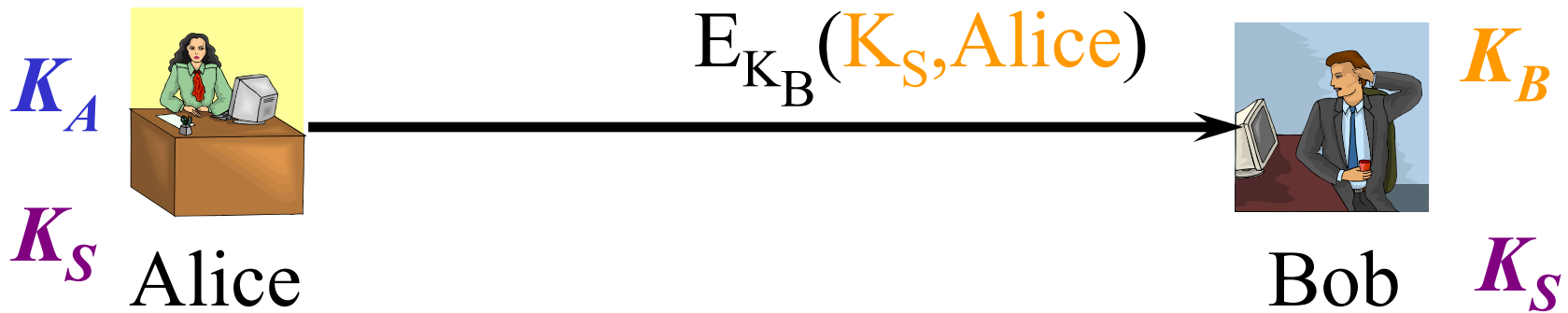
# Mallory Waits For His Chance



# 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



Mallory steps  
aside for a bit



Trent

$K_A$   $K_B$

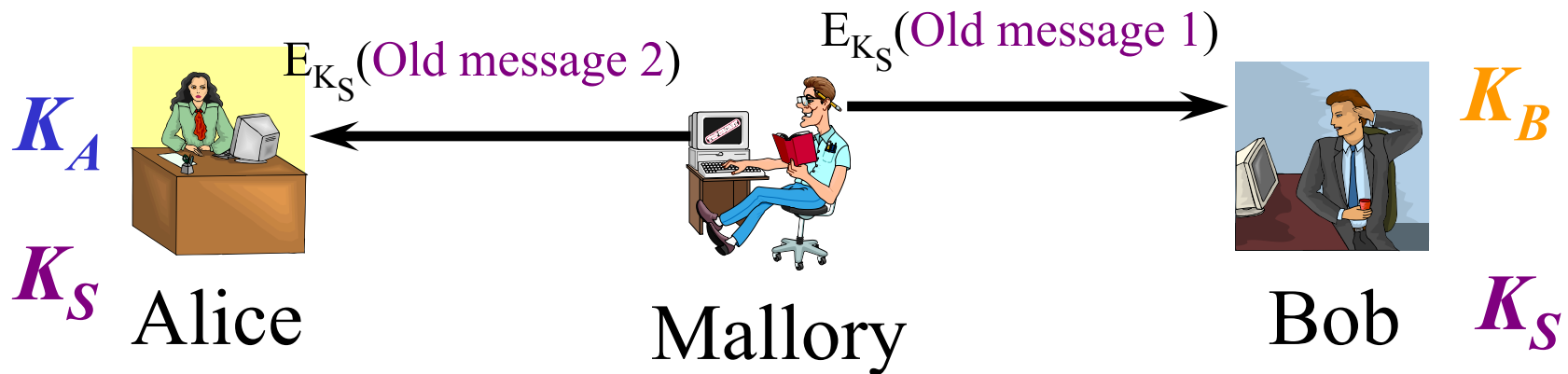
With no nonces,  
we're done

## 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



Mallory can  
replay Alice and  
Bob's old  
conversation



Trent

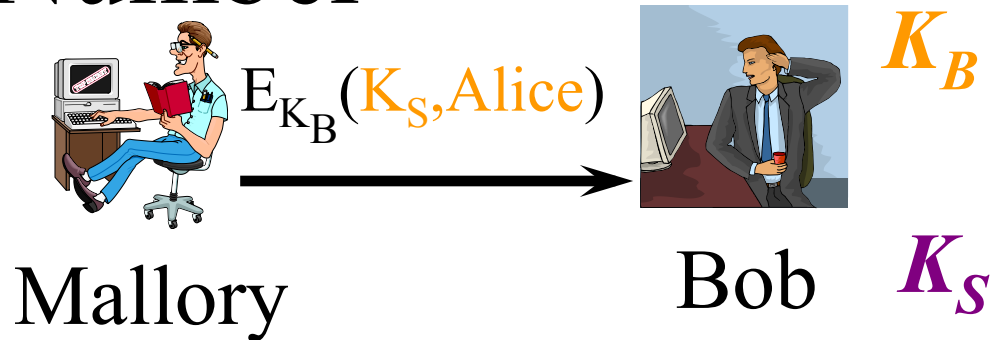
$K_A$   $K_B$

It's using the  
current key, so  
Alice and Bob  
will accept it

# 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



Trent

$K_A$   $K_B$

But Mallory wants Bob to think Alice wants to talk

# So What?



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

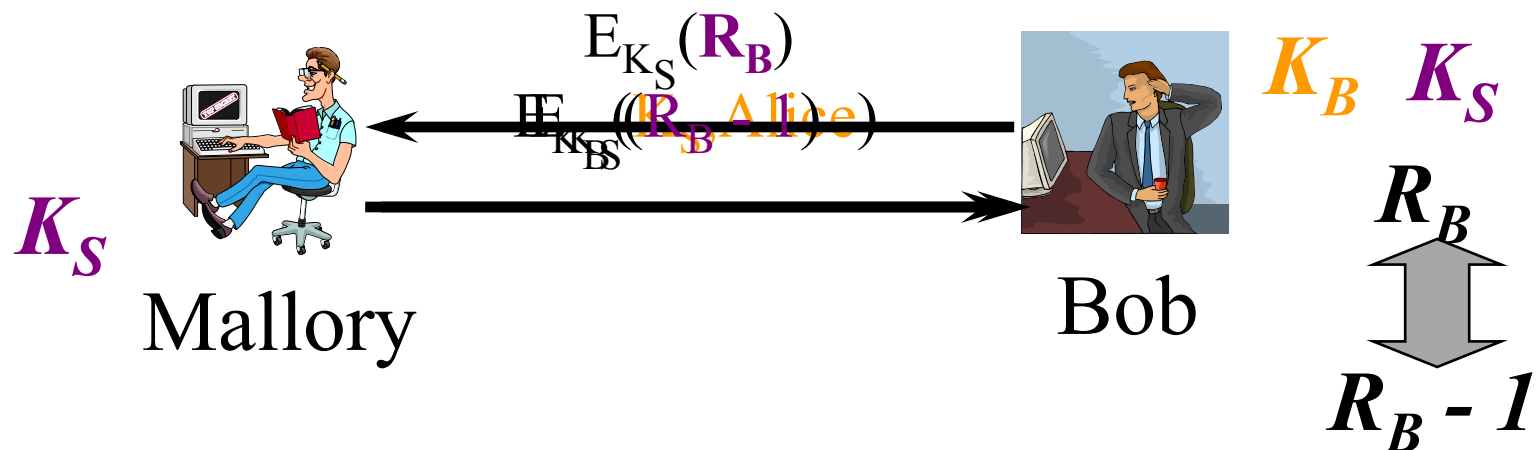
And Bob will have no reason to be suspicious

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 compromises 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

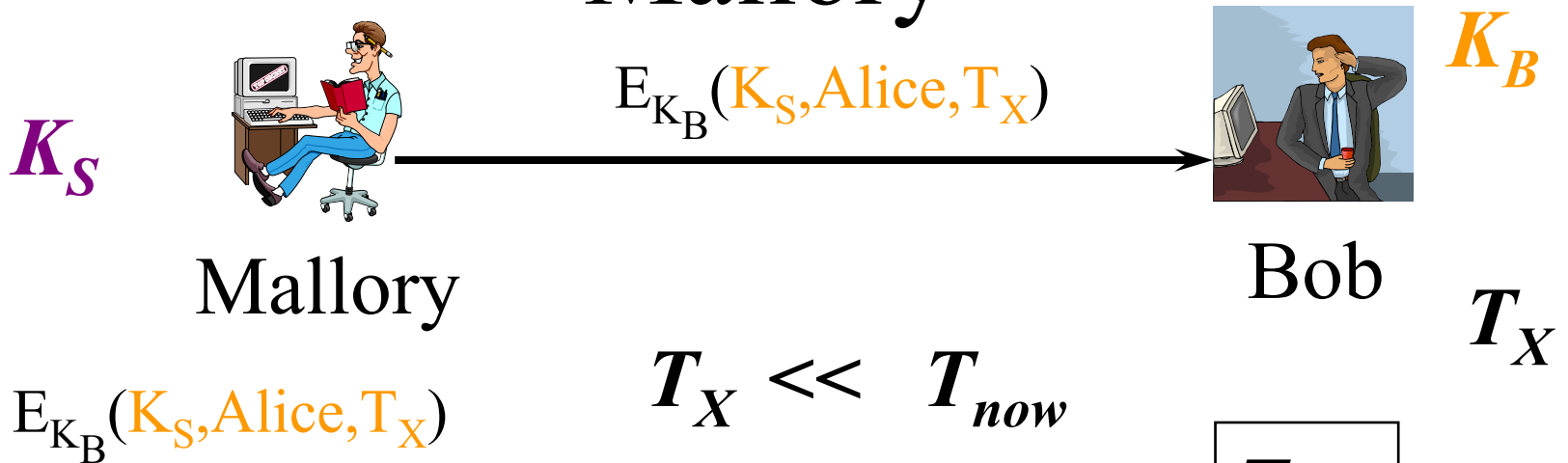
- One method of handling this kind of problem is timestamps
- Proper use of timestamps can limit the time during which an exposed key is dangerous
- But timestamps have their own problems

# 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



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

## Is This Overkill?

- Some of these attacks are pretty specialized
  - Requiring special access or information
- Some can only achieve certain limited effects
- Do we really care?

# Why Should We Care?

- Bad guys are very clever
- Apparently irrelevant vulnerabilities give them room to show that
- Changes in how you use protocols can make vulnerabilities more relevant
- A protocol without a vulnerability is always better
  - Even if you currently don't care

# Something to Bear in Mind

- These vulnerabilities aren't specific to just these protocols
- They are common and pop up all over
  - Even in cases where you aren't thinking about a “protocol”
- Important to understand them at a high conceptual level