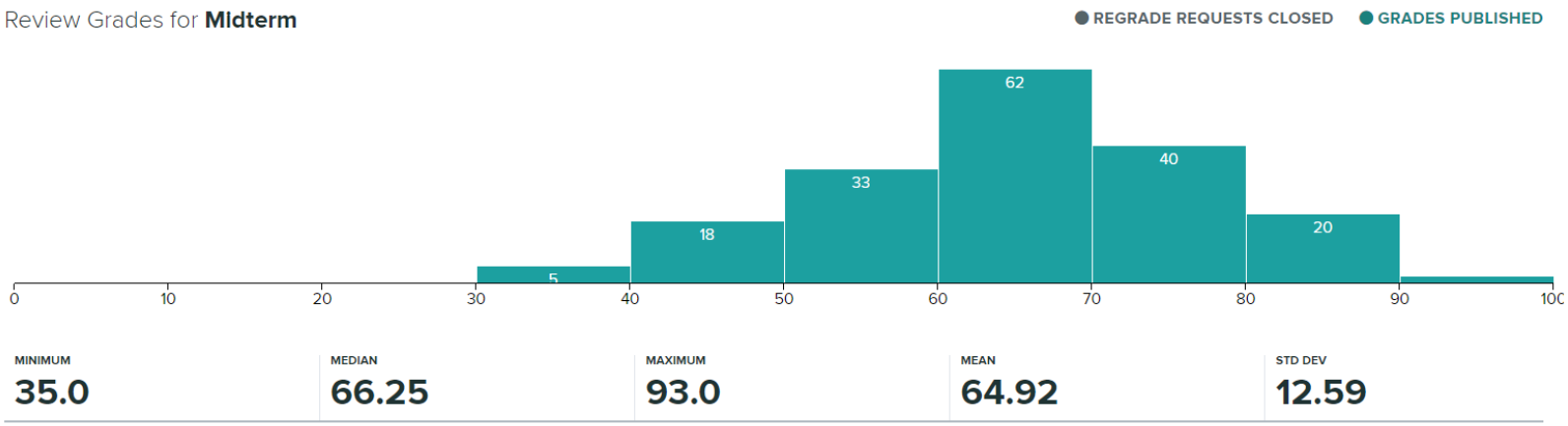
# Introduction to Information Security
# 14-741/18-631 Fall 2018
# Unit 4: Lecture 1 & 2
# Security Protocols

**Section A/J: Limin Jia** liminjia@andrew

**Section B/SV: Marty Carlisle** mcarlisl@andrew

# Midterm




- Midterm grades have been posted to S3.
- Exam results are available at gradescope.com using your Andrew ID
- Watch Piazza for information on regrade requests

# This lecture's agenda

- **Outline (the return of Alice and Bob)**
  - Engineering principles for cryptographic protocols
    - Naming
    - Which primitives (schemes) to use
    - Timeliness
  - Kerberos example
- **Objective**
  - Expose you to the difficulties of secure communication protocol design
  - Convince you that crypto is a powerful tool, but it is easy to make design errors that render it useless
  - Give practical example of a secure communication protocol that you actually use every day

# Security protocols

- **Entity authentication**
  - Proving identity to each other
- **Key exchange, establishment or agreement**
  - Establish a trusted session between two entities
  - Usually used to set up trusted communication channel providing secrecy and authenticity
- **Basis for**
  - Secure electronic commerce
  - Electronic voting
  - Time synchronization
- **We use the basic cryptographic primitives discussed before to design higher-level security properties**

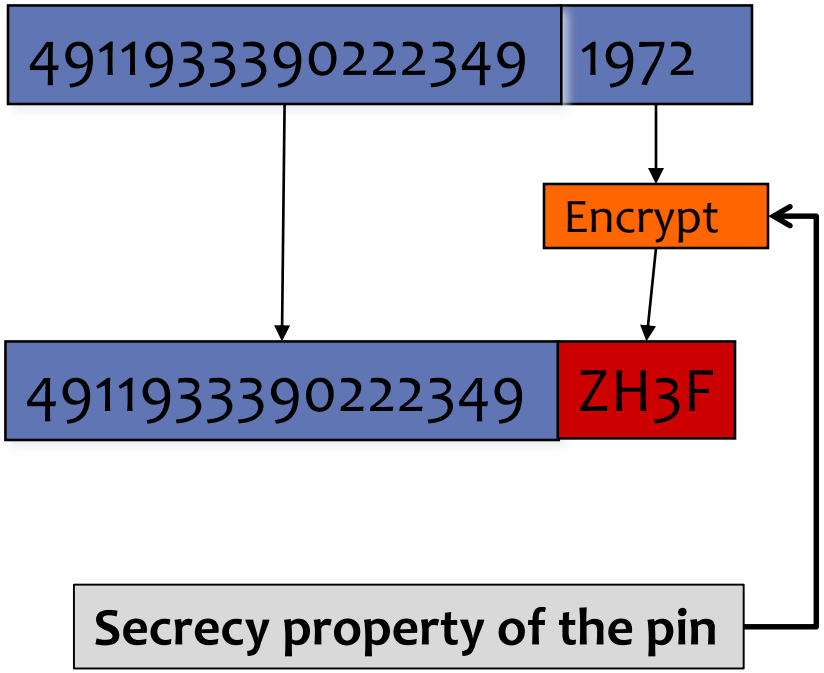
# Difficulties with security protocols

- **Combine a number of basic primitives**
  - Cryptography
  - Network communication
- **Individual primitives are generally working as expected, but interaction between primitives is generally Achilles' heel**

# Example of basic security failure

- **Bank encrypts very carefully PIN on credit cards**
  - Encryption is "standalone" - doesn't need the account number
- **Does not see the need for encrypting the account number**
  - after all it's printed on the front of the card

**The pin on the card is the pin set up by the user of the account, whose number is on the card.**

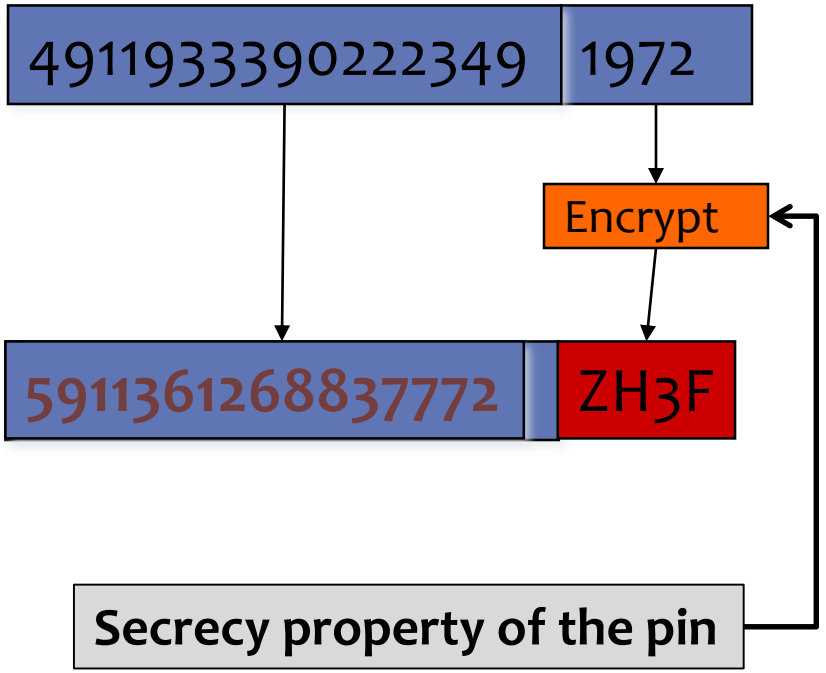



**Is this the only property?**

# Example of basic security failure

- **Bank encrypts very carefully PIN on credit cards**
  - Encryption is "standalone" - doesn't need the account number
- **Does not see the need for encrypting the account number**
  - after all it's printed on the front of the card
- **Possible to change the account number to someone else's while still using your PIN!**

**The pin on the card is the pin set up by the user of the account, whose number is on the card.**

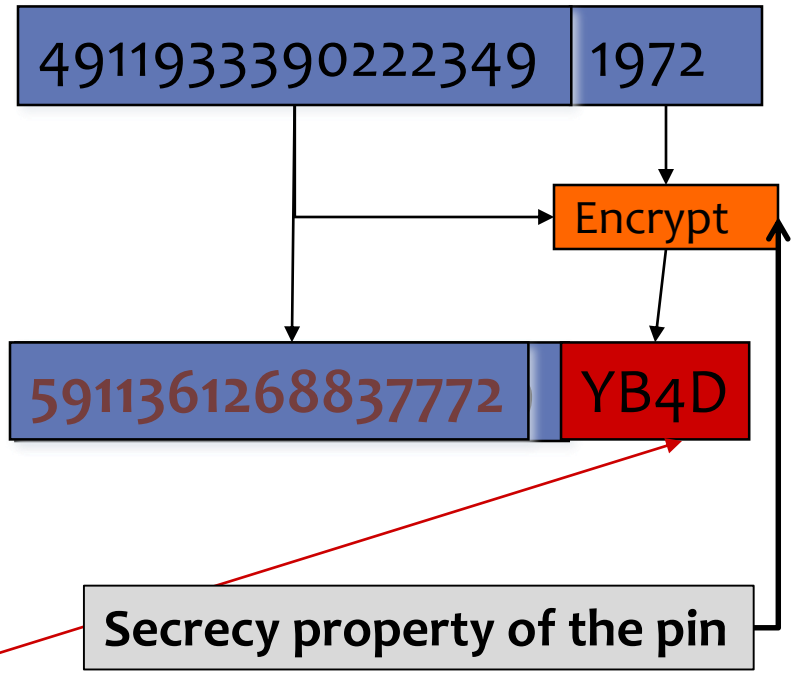



**Is this the only property?**

# Example of basic security failure

- **Bank encrypts very carefully PIN on credit cards**
  - Encryption is "standalone" - doesn't need the account number
- **Does not see the need for encrypting the account number**
  - after all it's printed on the front of the card
- **Possible to change the account number to someone else's while still using your PIN!**
- **Solution:**
  - Encrypt PIN using acct number

**The pin on the card is the pin set up by the user of the account, whose number is on the card.**




**Gives an incorrect PIN when decrypted using fake account number**

# Protocol design basics

- **Protocols involve principals (hosts, users, services, processes)**
  - e.g., a car lock and a car owner
- **Secrets**
  - e.g., symmetric keys
- **Authenticated information**
  - e.g., public keys
- **Basic crypto primitives**
  - Block/stream cipher
  - Diffie-Hellman
  - RSA …
  - Hash function, MAC
- **Trusted entities**
- **Timeliness proofs**
  - Nonces and timestamps

# Nonces

- **NONCE = Number used only ONCE**
- **Can be**
  - Counter
    - Unique (non-repeating) but predictable
    - May use a timestamp for this purpose
  - Random number
    - Unique and (hopefully) unpredictable
- **Timestamps can be nonces, but nonces don't have to be timestamps**

# Preparing for the worst

- **Always assume that the attacker can control at will the network where you want to deploy your secure communication protocol**

# Active attackers

- **Or, what can Mallory do?**
  - Can eavesdrop on all protocol runs
  - Can **replay** messages at will
  - Can **inject** fabricated messages in the network
    - For instance fabricated from pieces of old messages
  - Can **modify** a principal's message
  - Can **initiate multiple parallel protocol sessions**
  - Can perform **dictionary attack** on passwords
  - Can perform **exhaustive attack** on non-random (or poorly random) nonce

# “Ideal” protocol wishlist

- **Efficient protocol**
  - Low computational overhead
    - Don’t encrypt what you don’t need to encrypt
  - Low communication overhead
    - Don’t send unnecessary messages
- **As little trust as necessary**
- **As few assumptions as necessary**
  - Synchronized clocks?
  - Synchronized sequence numbers?
  - Randomly selected nonces and initialization vectors?
  - Security of crypto primitives?
  - Authenticity or secrecy of keys
- **Little client/server state**

**Ensure necessary security properties**

# Sample authentication protocol

- **Goal: Alice wants to authenticate Bob, Bob wants to authenticate Alice**
  - Sample setting: Alice is a car, Bob is a keychain, open car door only if Bob is authenticated
- **Assumption: Alice and Bob share secret key $K_{AB}$**

# Sample authentication protocol


Alice

$N_A$

Bob

# Sample authentication protocol


Alice

$N_A$

$\{N_A, N_B, A\}_{K_{AB}}$

Bob

# Sample authentication protocol


Alice

$N_A$

$\{N_A, N_B, A\}_{K_{AB}}$

$N_B$

Bob

Is this protocol secure?

# Design principles for protocols

- Abadi and Needham: Prudent Engineering Practice for Cryptographic Protocols
- Following slides based on a lecture by Abadi, modified us

# Principle 1: Explicit communication

- Every message should say what it means: the interpretation of the message should depend only on its content
- It should be possible to write down an English sentence describing the content though if there is a suitable formalism available that is good too.
- This principle counteracts that messages are used out of context, prevent replay attacks, and intermixing of messages from concurrent sessions

# The Denning-Sacco protocol

- Remember that for public key algorithms, it is indispensable to ensure the authenticity of a public key
- One way is to go through a trusted server (Trent) which will act as a certification authority
- The Denning-Sacco protocol (1982) is such an instance

# The Denning-Sacco protocol


Alice

*A,B*

Trent (trusted server)

Bob

# The Denning-Sacco protocol


Alice

*A,B*

*CA,CB*

Trent (trusted server)

Bob

# The Denning-Sacco protocol


Alice

$A, B$

$CA, CB$

Trent (trusted server)

$CA, CB, \{\{T_A, K_{AB}\}_{KR_A}\}_{KU_B}$

Bob

# Problem with Denning-Sacco

Trent (trusted server)

**Bob's reasoning:**
**CA: Alice wants to share a key K with me**
**After 1st decryption: K has been kept secret in transit**
**After 2st decryption: K is computed by Alice**
**Alice must want to share this key K with me**


Alice

$CA, CB, \{\{T, K\}_{KR_A}\}_{KU_B}$

Bob

# Problem with Denning-Sacco

Trent (trusted server)

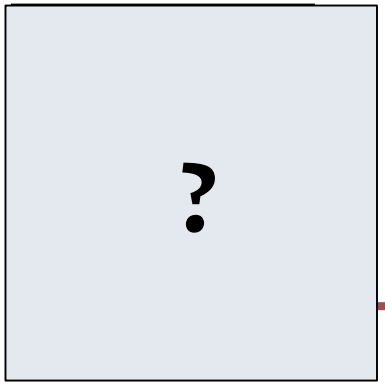
**Bob's reasoning:**

**CA: Alice wants to share a key K with me**

**After 1st decryption: K has been kept secret in transit**

**After 2st decryption: K is computed by Alice**

**Alice must want to share this key K with me**




$CA, CB, \{\{T, K\}_{KR_A}\}_{KU_B}$

Bob

# Problem with Denning-Sacco

Charlie

Trent (trusted server)

**Bob's reasoning:**

**CA: Alice wants to share a key K with me**

**After 1st decryption: K has been kept secret in transit**

**After 2st decryption: K is computed by Alice**

**Alice must want to share this key K with me**


Alice

$CA, CB, \{\{T, K\}_{KR_A}\}_{KU_B}$

Bob

**I can pretend to be Alice!**

# Problem with Denning-Sacco

Charlie

Trent (trusted server)

**Charlie's reasoning:**

**CA: Alice wants to share a key K with me**

**After 1st decryption: K has been kept secret in transit**

**After 2st decryption: K is computed by Alice**

**Alice must want to share this key K with me**

$CA, CC, \{\{T, K\}_{KR_A}\}_{KU_C}$


Alice

$CA, CB, \{\{T, K\}_{KR_A}\}_{KU_B}$

Bob

**I can pretend to be Alice!**

# Problem with Denning-Sacco

Charlie

Trent (trusted server)

**Charlie's reasoning:**

**CA: Alice wants to share a key K with me**

**After 1st decryption: K has been kept secret in transit**

**After 2st decryption: K is computed by Alice**

**Alice must want to share this key K with me**

$B, C$

$CB, CC$

$CA, CC, \{\{T, K\}_{KR_A}\}_{KU_C}$


Alice

$CA, CB, \{\{T, K\}_{KR_A}\}_{KU_B}$

Bob

**I can pretend to be Alice!**

# Problem with Denning-Sacco

- **Bob receives *CA*, *CB*, $\{\{T_A, K_{AB}\}_{KR_A}\}_{KU_B}$ from Alice**
- **With $KR_B$, which he has, he can extract $\{T_A, K_{AB}\}_{KR_A}$**
  - That is, the only thing that is used to prove Alice's identity!!!
- **And now Bob can pose as Alice to anyone else (Charlie in our example below) as long as $T_A$ is valid**
- **May look obvious, but it took 12 years to notice**

# Failure diagnosis

- **Optimistic use of encryption**
- **Names are missing**
- **It is not possible to parse the message into the statement that represents its meaning**
- **Solution**
  - $A \rightarrow B$: $CA, CB, \{\{A, B, T_A, K_{AB}\}_{KR_A}\}_{KU_B}$
  - or any other unambiguous encoding of the meaning of the message

# Principle 2: Appropriate action

- **The conditions for a message to be acted upon should be clearly set out so that someone reviewing a design may see whether they are acceptable or not.**
- **Said differently: Clearly state your assumptions!**
  - Be clear on how encryption is used, and the meaning of encryption
  - Be clear on how the timeliness of messages is proved, and on the meaning of temporal information in messages

# Principle 3: Naming

- **If the identity of a principal is important for the meaning of a message, it is prudent to mention the principal's name explicitly in the message**

# The Woo-Lam protocol

- Alice wants to prove her presence to Bob
- Alice shares a key with Trent
- Alice doesn’t share a key with Bob

# The Woo-Lam protocol

Trent (trusted server)


Alice

*A*

Bob

# The Woo-Lam protocol

Trent (trusted server)


Alice

$A$

$N_B$

Bob

# The Woo-Lam protocol

Trent (trusted server)


Alice

$A$

$N_B$

$\{N_B\}_{K_{AT}}$

Bob

# The Woo-Lam protocol

Trent (trusted server)

$\{A,\{N_B\}_{K_{AT}}\}_{K_{BT}}$


Alice

$A$

$N_B$

$\{N_B\}_{K_{AT}}$

Bob

# The Woo-Lam protocol

Trent (trusted server)

$\{N_B\}_{K_{BT}}$

$\{A,\{N_B\}_{K_{AT}}\}_{K_{BT}}$


Alice

$A$

$N_B$

$\{N_B\}_{K_{AT}}$

Bob

# The Woo-Lam protocol

Trent (trusted server)

**Bob's reasoning:**
**A: Alice wants to talk to me**
**$N_B$: Challenge for Alice**
**After decryption: Blob = $\{N_B\}_{K_{AT}}$**
**Alice has encrypted $N_B$**
**I am talking to Alice**

$\{N_B\}_{K_{BT}}$

$\{A,\{N_B\}_{K_{AT}}\}_{K_{BT}}$


Alice

$A$

$N_B$

**Blob = $\{N_B\}_{K_{AT}}$**

Bob

# The Woo-Lam protocol

Trent (trusted server)

**Bob's reasoning:**
**A: Alice wants to talk to me**
**$N_B$: Challenge for Alice**
**After decryption: Blob = $\{N_B\}_{K_{AT}}$**
**Alice has encrypted $N_B$**
**I am talking to Alice**

$\{N_B\}_{K_{BT}}$

$\{A,\{N_B\}_{K_{AT}}\}_{K_{BT}}$


Alice

$A$

$N_B$

**Blob = $\{N_B\}_{K_{AT}}$**

Bob

# The Woo-Lam protocol

Trent (trusted server)

**Bob's reasoning:**
**A: Alice wants to talk to me**
**$N_B$: Challenge for Alice**
**After decryption: Blob = $\{N_B\}_{K_{AT}}$**
**Alice has encrypted $N_B$**
**I am talking to Alice**

$\{N_B\}_{K_{BT}}$

$\{M,\{N_B\}_{K_{MT}}\}_{K_{BT}}$

$\{A,\{N_B\}_{K_{AT}}\}_{K_{BT}}$

?

$A$

$N_B$

**Blob = $\{N_B\}_{K_{AT}}$**

Bob

# The Woo-Lam protocol

Trent (trusted server)


*A*

*M*

Bob

# The Woo-Lam protocol

Trent (trusted server)


$A$

$M$

$N_B$

$N_B'$

Bob

# The Woo-Lam protocol

Trent (trusted server)


$A$

$M$

$N_B$

$N_B'$

$\{N_B\}_{K_{MT}}$

$\{N_B\}_{K_{MT}}$

Bob

# The Woo-Lam protocol

Trent (trusted server)

$\{M,\{N_B\}_{K_{MT}}\}_{K_{BT}}$

$\{A,\{N_B\}_{K_{MT}}\}_{K_{BT}}$


$A$

$M$

$N_B$

$N_B'$

$\{N_B\}_{K_{MT}}$

$\{N_B\}_{K_{MT}}$

Bob

# The Woo-Lam protocol

Trent (trusted server)

$\{M,\{N_B\}_{K_{MT}}\}_{K_{BT}}$

$\{N_B''\}_{K_{BT}}$

$\{A,\{N_B\}_{K_{MT}}\}_{K_{BT}}$

$\{N_B\}_{K_{BT}}$


$A$

$M$

$N_B$

$N_B'$

$\{N_B\}_{K_{MT}}$

$\{N_B\}_{K_{MT}}$

Bob

# Diagnosis

- **Mallory can convince Bob that Alice is present**
- **This is because nothing ties the identity to the nonce!**
  - Trent's response doesn't mention Alice by name
  - Nonce are good for ensuring freshness but not always for association
  - Double encryption is no cause for optimism

# Solution


Alice

Trent (trusted server)

Bob

$\{A, N_B\}_{K_{BT}}$

$\{A,\{N_B\}_{K_{AT}}\}_{K_{BT}}$

$A$

$N_B$

$\{N_B\}_{K_{AT}}$

# Optimization

Trent (trusted server)

$\{A, N_B\}_{K_{BT}}$


Alice

$A$

$\{A, B, \{N_B\}_{K_{AT}}\}_{K_{BT}}$

$N_B$

$\{N_B\}_{K_{AT}}$

Bob

# Optimization

Trent (trusted server)

$A, B, \{N_B\}_{K_{AT}}$

$\{A, N_B\}_{K_{BT}}$


Alice

$A$

$N_B$

$\{N_B\}_{K_{AT}}$

Bob

# Other principles

- **Other principles concern**
  - encryption
  - timeliness
  - trust
  - secrecy
- **The principles serve to**
  - simplify protocols
  - simplify formal analysis
  - avoid many mistakes

# Be careful with encryption

- **Principle 4: Be clear as to why encryption is being done. Encryption is not synonymous with security, and its improper use can lead to errors.**

- **Principle 5: When signing already encrypted data, it should not be inferred that the principal knows the content of the message. On the other hand, it is proper to infer that the principal that signs a message and then encrypts it for secrecy knows the content of the message.**

# Uses of encryption

- **Secrecy**
- **Authenticity: a principal proves ownership of a key by encrypting a known message with that key**
- **Bind together parts of a message**
  - $\{N_A, N_B\}_{K_{AT}}$ is different from $\{N_A\}_{K_{AT}}\{N_B\}_{K_{AT}}$
  - Signature is sufficient for binding purpose
- **Produce random numbers**

# Kerberos protocol

- **Famous authentication protocol using trusted server and "tickets"**
- **Used when logging into andrew**

  (outside of WebISO…)

# Kerberos protocol


Alice

*A*, *B*

Trent (trusted server)

Bob

# Kerberos protocol

Trent (trusted server)

$A, B$

$\{T_T, L, K_{AB}, B, \{T_T, L, K_{AB}, A\}_{K_{BT}}\}_{K_{AT}}$


Alice

Bob

# Kerberos protocol

Trent (trusted server)

$A, B$


Alice

$\{T_T, L, K_{AB}, B, \{T_T, L, K_{AB}, A\}_{K_{BT}}\}_{K_{AT}}$

$\{T_T, L, K_{AB}, A\}_{K_{BT}}, \{A, T_A\}_{K_{AB}}$

Bob

# Kerberos protocol

Trent (trusted server)

$A, B$

$\{T_T, L, K_{AB}, B, \{T_T, L, K_{AB}, A\}_{K_{BT}}\}_{K_{AT}}$


Alice

$\{T_T, L, K_{AB}, A\}_{K_{BT}}, \{A, T_A\}_{K_{AB}}$

$\{T_A+1\}_{K_{AB}}$

Bob

# Kerberos protocol

1. $A \rightarrow T: A, B$
2. $T \rightarrow A: \{T_T, L, K_{AB}, B, \{T_T, L, K_{AB}, A\}_{K_{BT}}\}_{K_{AT}}$
3. $A \rightarrow B: \{T_T, L, K_{AB}, A\}_{K_{BT}}, \{A, T_A\}_{K_{AB}}$
4. $B \rightarrow A: \{T_A+1\}_{K_{AB}}$

- Encryption is not necessary for message 1, but DoS attack possible without encryption (?)
- Message 2 requires encryption, $K_{AB}$ needs to remain secret, T should sign message as a proof of authenticity
- Double encryption does not add any secrecy or authenticity, but it proves to B in message 3 that A must have successfully decrypted message 2
- 2nd encryption in message 3 proves knowledge of $K_{AB}$

# Timeliness/Freshness (1/3)

- **Principle 6: Be clear as to what properties you assume of nonces.**
  - What may do for ensuring timeliness may not do for ensuring association
  - Perhaps association is best established by other means

# The Wide-Mouthed Frog protocol

- **Revocation is difficult, much easier to do if key has limited lifetime**
  - Expires automatically, has to be explicitly renewed
- **Protocol used to deliver a key with an "expiry date"**

(illustration by Jonathan Lambert)

# The Wide-Mouthed Frog protocol

$\{T_A, B, K_{AB}\}_{K_{AT}}$

Trent (trusted server)


Alice

Bob

# The Wide-Mouthed Frog protocol

$\{T_A, B, K_{AB}\}_{K_{AT}}$

Trent (trusted server)

$\{T_T, A, K_{AB}\}_{K_{BT}}$


Alice

Bob

**$T_T$ is not associated with Alice**

- Trent only puts $T_T$ in the message if $K_{AB}$ is fresh
- If $T_T$ is fresh, Alice must have shared $K_{AB}$ recently

# Attacking the frog

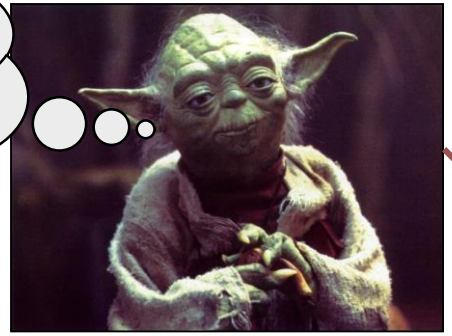
Trent (trusted server)

$\{T_T, A, K_{AB}\}_{K_{BT}}$

$\{T_T, A, K_{AB}\}_{K_{BT}}$

Bob

# Attacking the frog

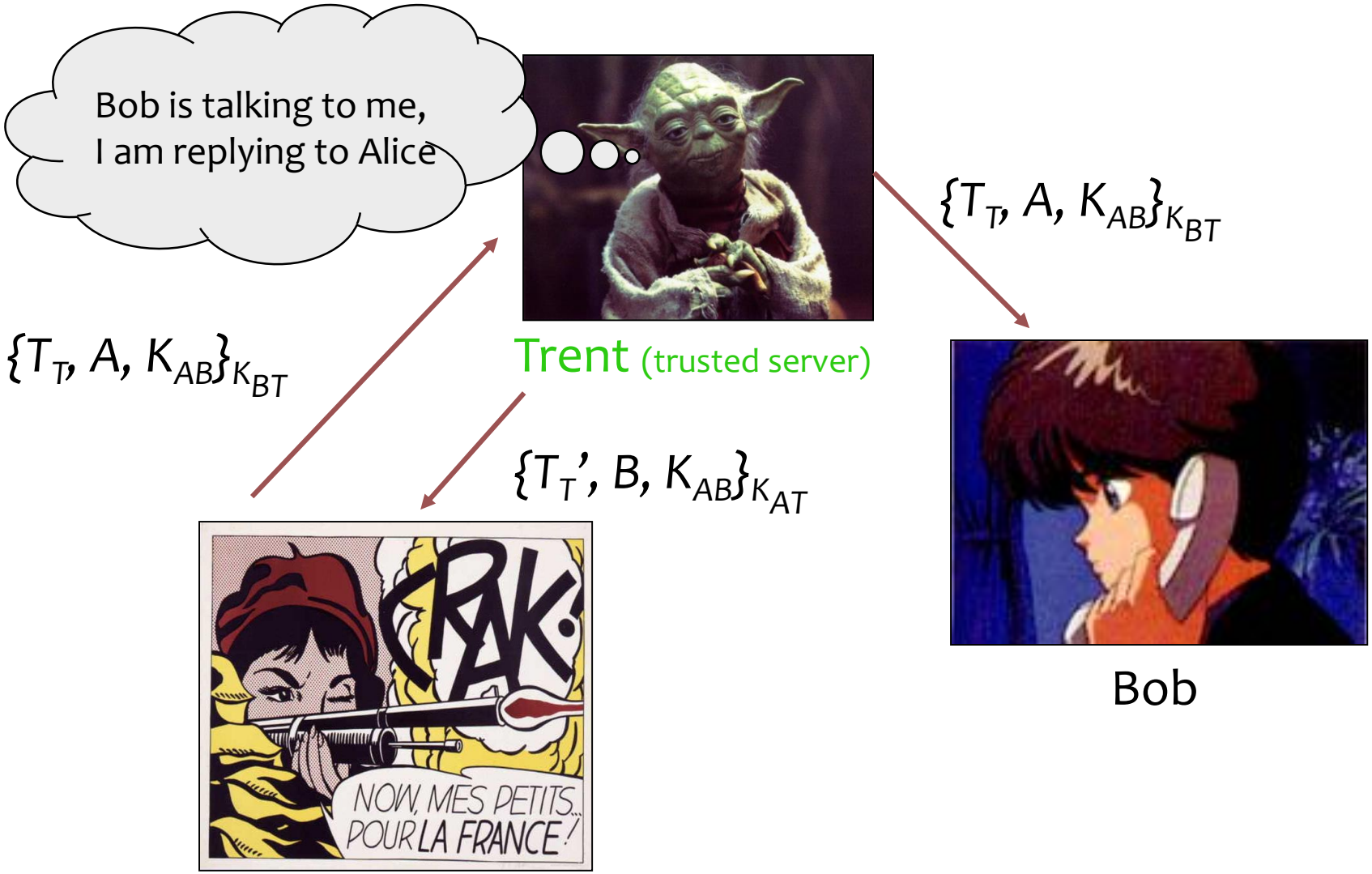
Trent (trusted server)

$\{T_T, A, K_{AB}\}_{K_{BT}}$

$\{T_T, A, K_{AB}\}_{K_{BT}}$

Bob

# Attacking the frog



# Attacking the frog

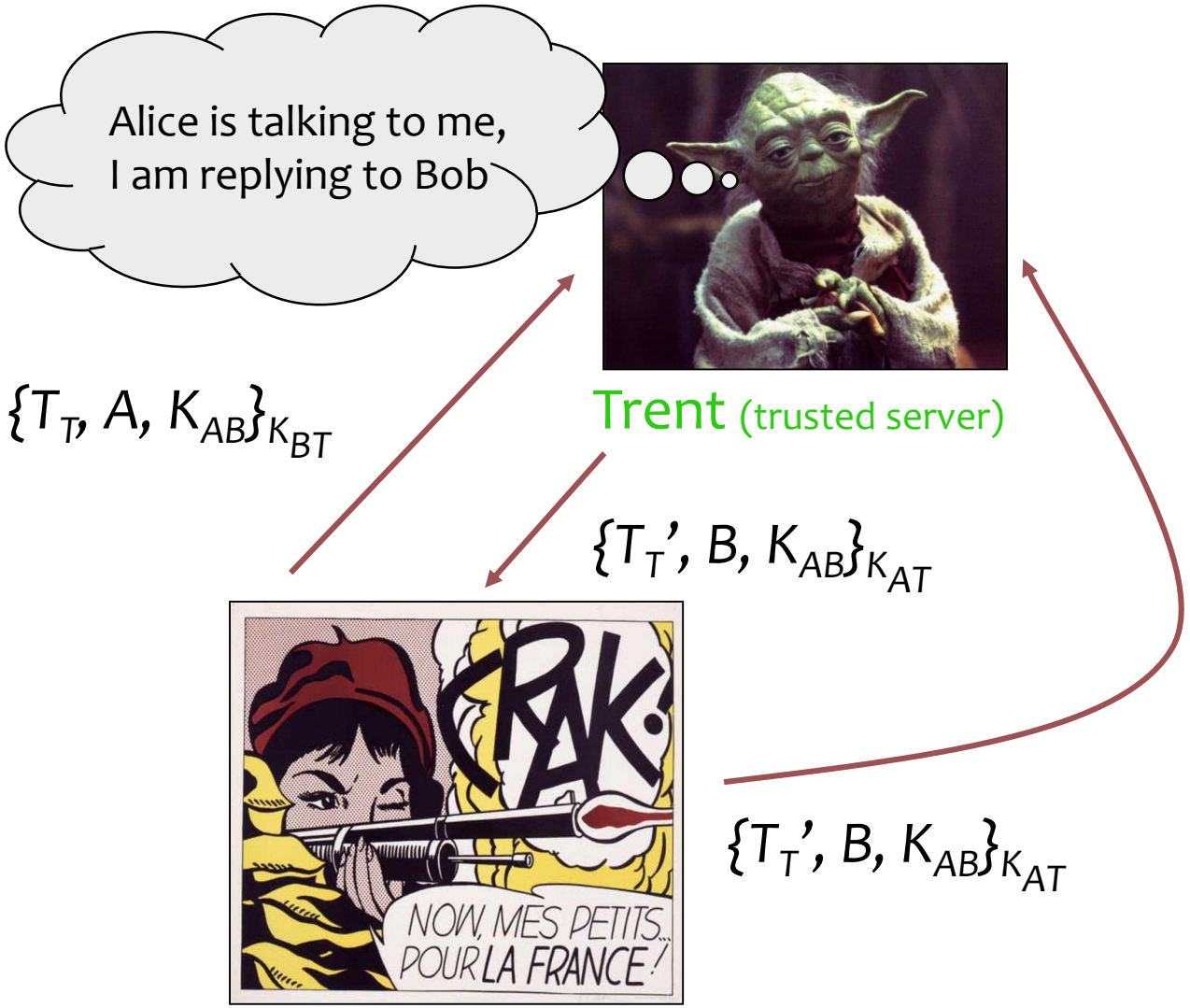
Trent (trusted server)

$\{T_T, A, K_{AB}\}_{K_{BT}}$

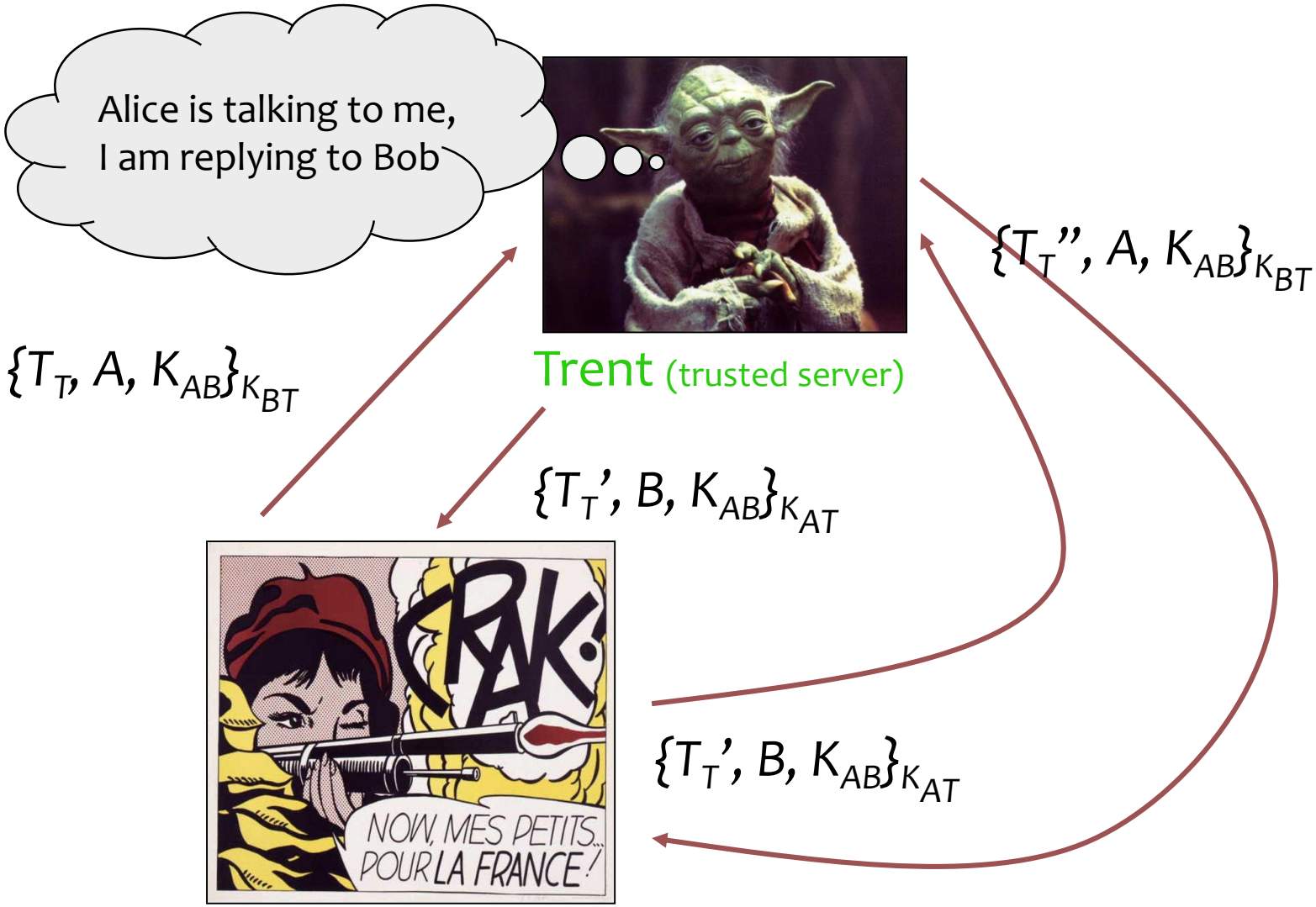
$\{T_T', B, K_{AB}\}_{K_{AT}}$


$\{T_T', B, K_{AB}\}_{K_{AT}}$

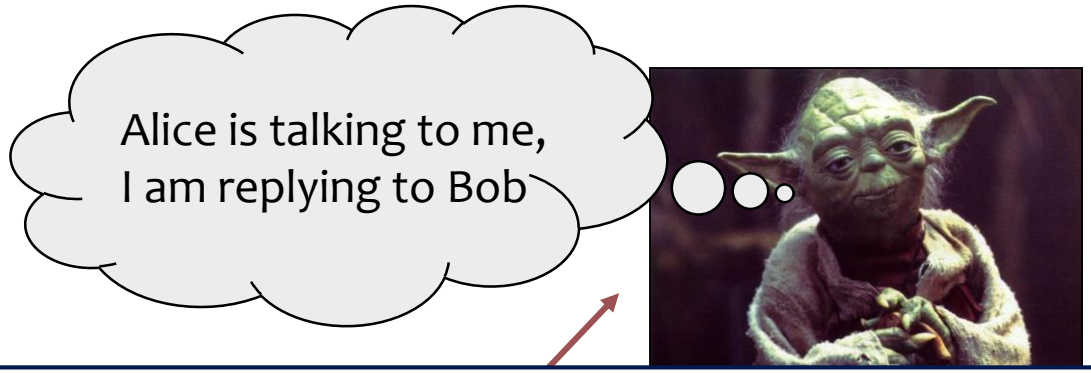
# Attacking the frog



# Attacking the frog



# Attacking the frog




$\{T_T'', A, K_{AB}\}_{K_{BT}}$

**Mallory can keep the key $K_{AB}$ valid for as long as she wants… Which eventually can be useful if she manages to steal the key!**


$\{T_T', B, K_{AB}\}_{K_{AT}}$

# Failure diagnosis

- **Need to be careful about association!**
- **Possible solutions**
  - Have Trent keep a list of current/recent working keys and timestamps
  - Do not change $T_A$ into $T_T$?
    - Lead to other problems?

# Timeliness/Freshness (2/3)

- **Principle 7: The use of a predictable quantity (such as time or the value of a counter) can serve in guaranteeing freshness. But if a predictable quantity is to be effective, it should be protected so that an intruder cannot simulate a challenge and later replay a response.**

# Predictability of nonces

- **Simple clock synchronization protocol**


Alice

$A, N_A$

$\{T_T, N_A\}_{K_{AT}}$

Trent (trusted server)

# Predictability of nonces


$A, N_A$

$\{T_T, N_A\}_{K_{AT}}$

Trent (trusted server)


Alice

$A, N_A$

$\{T_T, N_A\}_{K_{AT}}$

# Diagnosis

- **If $N_A$ is predictable**
  - Possible for Mallory to set the value of Alice's clock back
- **Solutions**
  - Make $N_A$ random
  - Encrypt $N_A$ with $K_{AT}$
    - Which essentially makes it unpredictable by anyone but Alice and Trent

# Predictability of nonces

- **Simple clock synchronization protocol**


Alice

$A, \{N_A\}_{K_{AT}}$

$\{\{T_T, \{N_A\}_{K_{AT}}\}_{K_{AT}}$

Trent (trusted server)

# Timeliness/Freshness (3/3)

- Freshness: use vs. generation
- Principle 9: A key may have been used recently for example to encrypt a nonce, yet be quite old, and be possibly compromised. Recent use does not make the key look any better than it would otherwise

# The Needham-Schroeder protocol

1. $A \rightarrow T: A, B, N_A$
2. $T \rightarrow A: \{N_A, B, K_{AB}, \{K_{AB}, A\}_{K_{BT}}\}_{K_{AT}}$
3. $A \rightarrow B: \{K_{AB}, A\}_{K_{BT}}$
4. $B \rightarrow A: \{N_B\}_{K_{AB}}$
5. $A \rightarrow B: \{N_B+1\}_{K_{AB}}$

1. $M \rightarrow B: \{K_{AB}, A\}_{K_{BT}}$
2. $B \rightarrow M: \{N_B\}_{K_{AB}}$
3. $M \rightarrow B: \{N_B+1\}_{K_{AB}}$

- Does A obtain freshness for key $K_{AB}$?
- Does B obtain freshness for key $K_{AB}$?

# How to stay out of trouble

- **Keep your protocols simple (KISS)**
  - Remember the Saltzer Schroeder principles for access control… they sometimes apply to other fields!
- **Be suspicious of clever optimizations**
- **Be explicit**
  - include sufficient proofs of freshness
  - include sufficient names
  - do not count on context
  - use evident classifications
- **Interpreting a message should be a simple matter of parsing (no context should be needed)**
- **Cryptography helps, but is not the whole story**

# Take away slide

- **Designing correct security protocols is extremely challenging**
- **Subtle flaws can result in a vulnerable protocol**
- **Often unsatisfied assumption results in vulnerability**
- **Promising research direction**
  - automated protocol verification