# Protocol

Shuai Wang

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

# Protocol

- Human protocols — the rules followed in human interactions
  - Example: Asking a question in class
- Networking protocols — rules followed in networked communication systems
  - Examples: HTTP, FTP, etc.
- Security protocol — the (communication) rules followed in a security application
  - Examples: SSH, SCP, SSL, IPSec, Kerberos, etc.

# Protocols

- Protocol flaws can be very ***subtle***
- Several well-known security protocols have significant flaws
  - Including WEP, GSM, and IPSec
- Implementation errors can also occur
  - Heartbleed
- Not easy to get protocols right…

# Ideal Security Protocol

- Must satisfy security requirements
  - Requirements need to be precise
- Efficient
  - Minimize computational requirement
  - Minimize bandwidth usage, delays…
- Robust
  - Works when attacker tries to break it
  - Works if environment changes (slightly)
- Easy to implement, easy to use, flexible…
- Difficult to satisfy all of these!

# Authentication Protocols

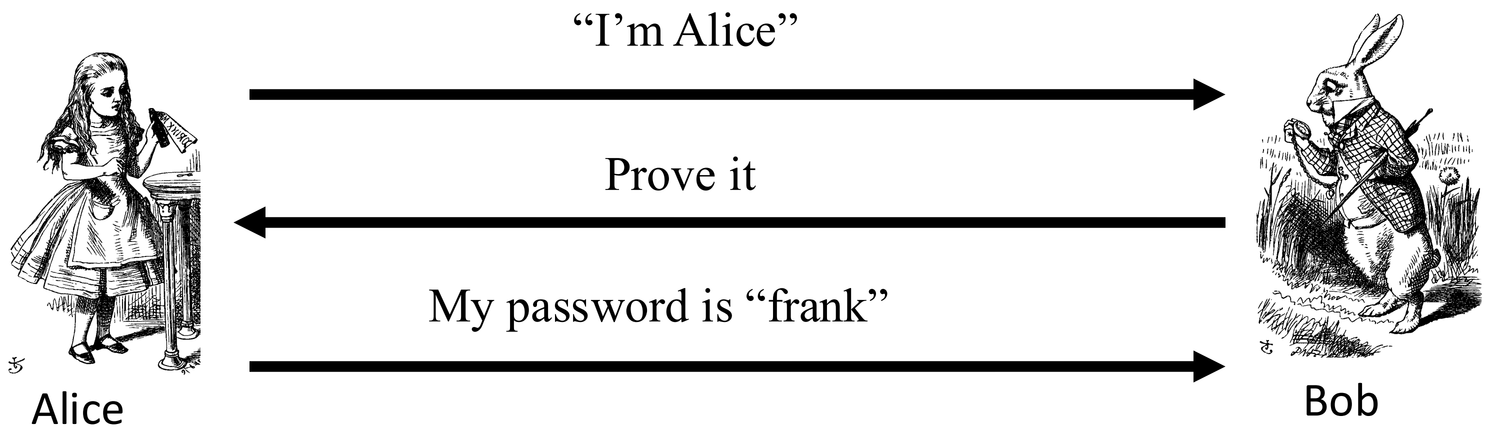
# Authentication

- We are talking about remote authentication via untrusted channel!
  - Those “3 factors” authentication itself is just a “local” setting.
- Alice must prove her identity to Bob
  - Alice and Bob can be humans or **computers**
- May also require Bob to prove he’s Bob (mutual authentication)
- Probably need to establish a **session key**
- May have other requirements, such as
  - Public keys, symmetric keys, hash functions, …
  - Anonymity, plausible deniability, perfect forward secrecy, etc.

# Authentication

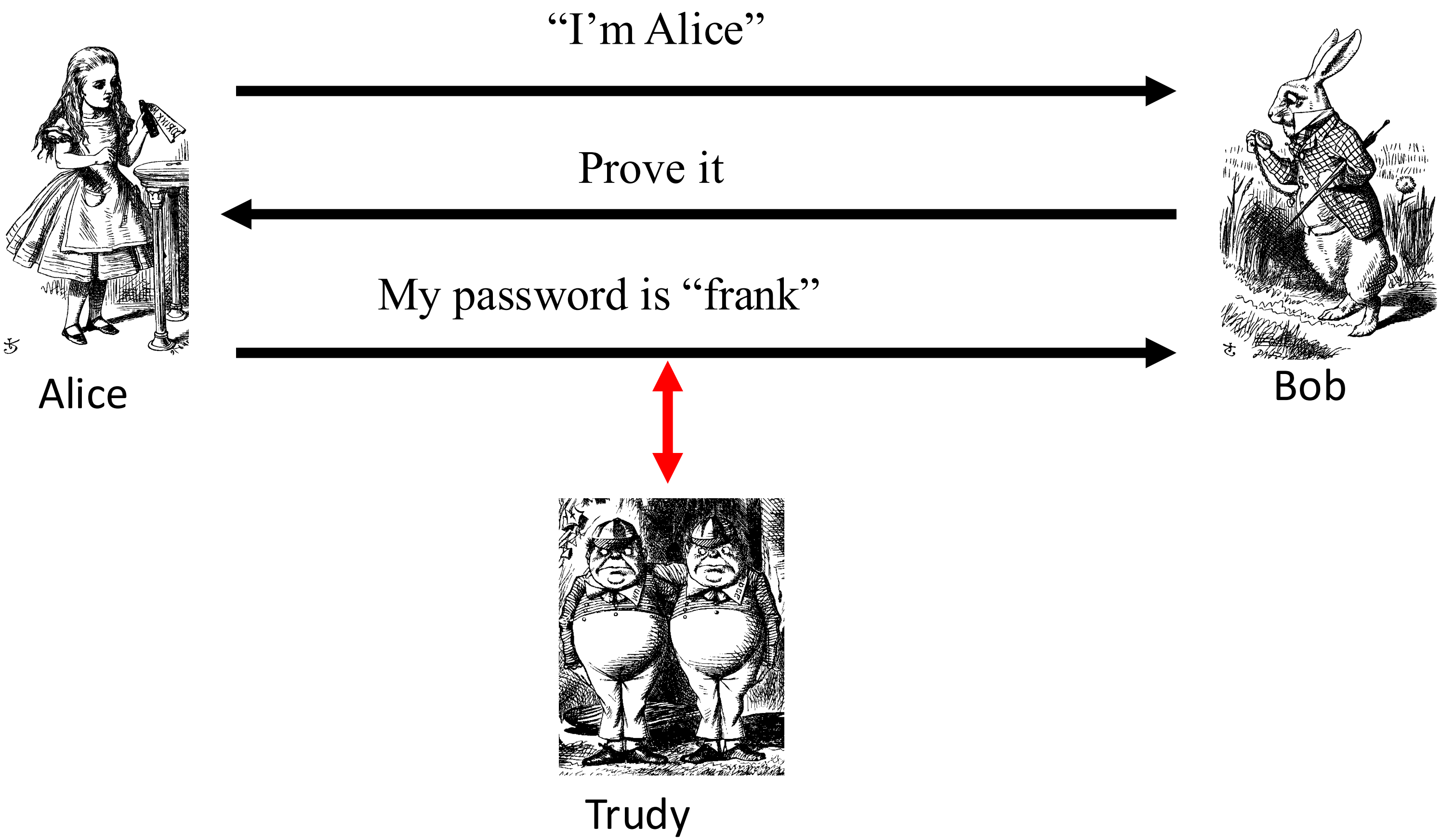
- Authentication on a stand-alone computer is relatively simple
  - For example, hash a password with a salt
- Authentication over a network is challenging
  - Attacker can passively observe messages
  - Attacker can replay messages
  - Active attacks possible (insert, delete, change)

# Simple Authentication

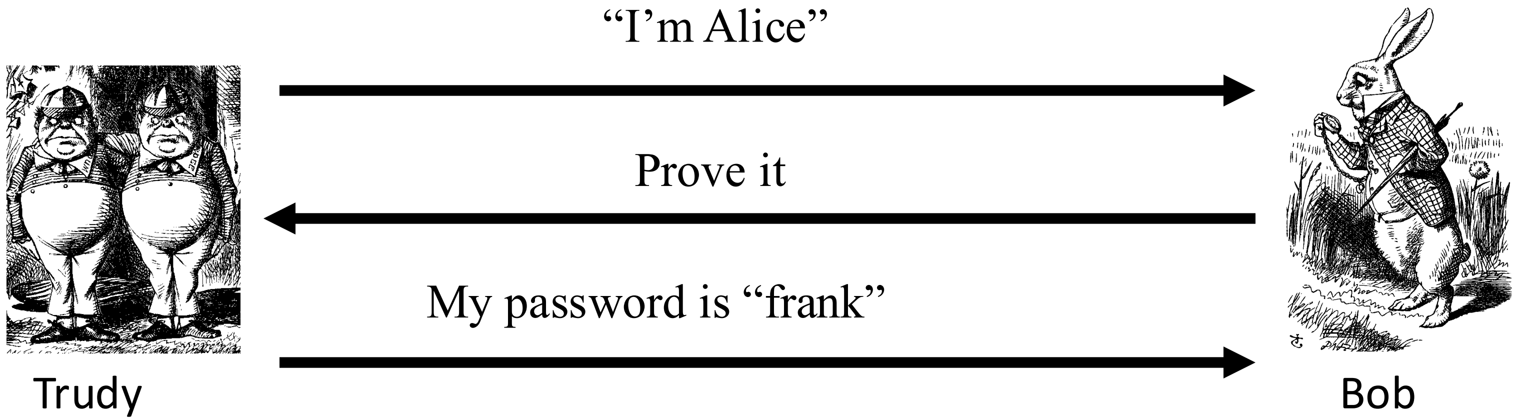



- Simple and may be OK for standalone system
- But highly insecure for networked system
  - Subject to a **replay** attack (next 2 slides)
    - Even if “frank” is encrypted or hashed!
  - Also, Bob must know Alice’s password

# Authentication Attack



# Authentication Attack




- This is an (trivial) example of a **replay** attack
- How can we prevent a replay?

# Simple Authentication

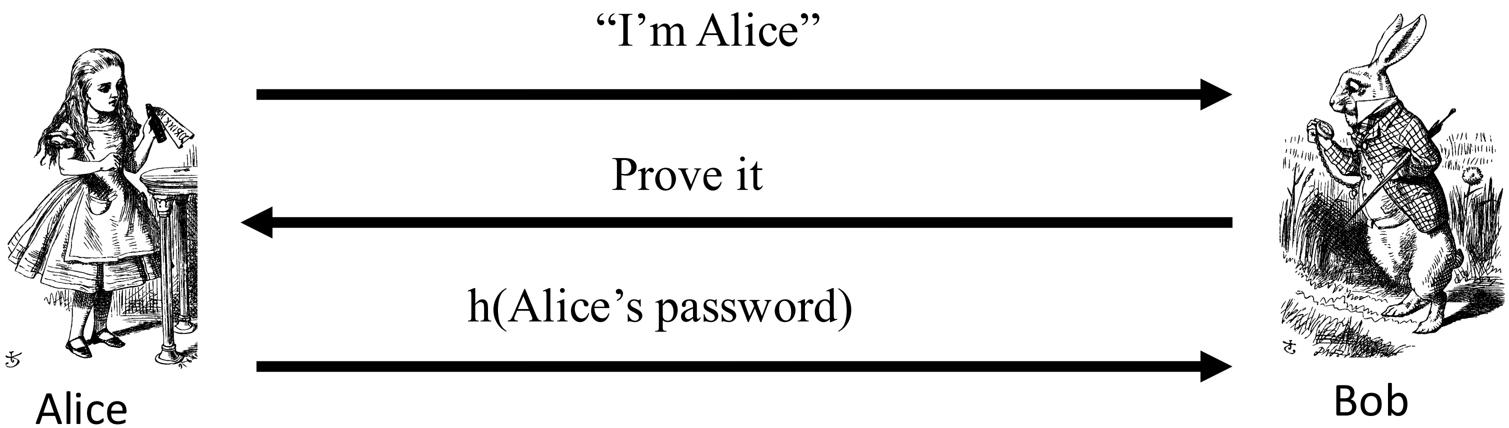
I'm Alice, my password is "frank"

Alice

Bob

- More efficient, but…
- … same problem as previous version

# Better Authentication




- This approach hides Alice’s password
  - From both Bob and Trudy
- But still subject to replay attack

# Challenge-Response

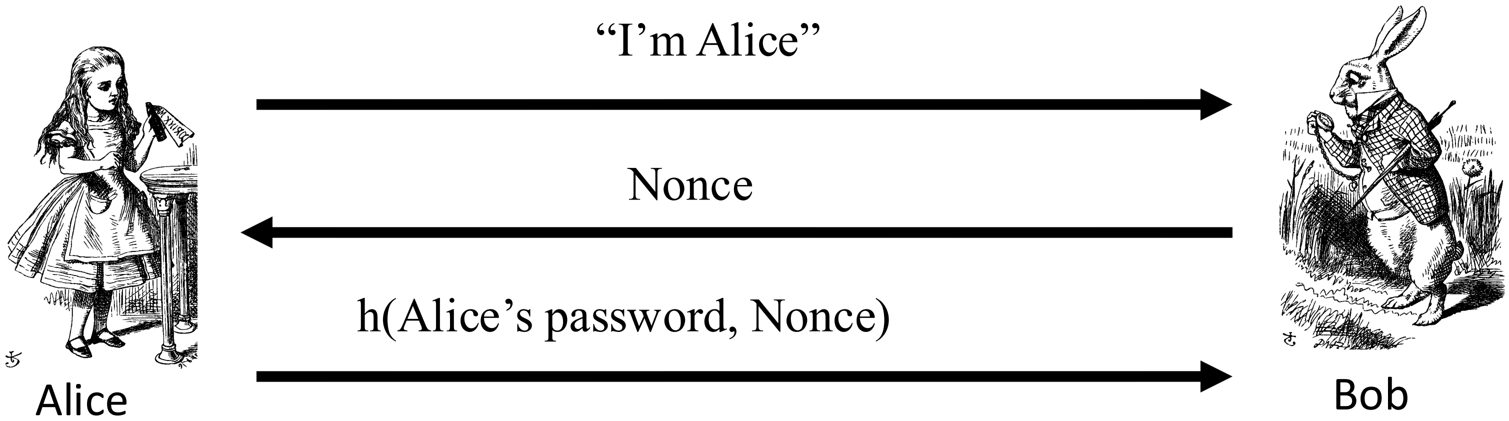
- To prevent replay, use ***challenge-response***
  - Goal is to ensure “freshness”
- Suppose Bob wants to authenticate Alice
  - ***Challenge*** sent from Bob to Alice
- Challenge is chosen so that…
  - Replay is not possible
  - Only Alice can provide the correct ***response***
  - Bob can verify the response

Nouce vs. Salt:
- conceptually irrelevant
– implementation-wise, similar given both of them are random numbers

# Nonce

- To ensure freshness, can employ a **nonce**
  - Nonce == **n**umber used **once**
- What to use for nonces?
  - That is, what is the challenge?
- What should Alice do with the nonce?
  - That is, how to compute the response?
- How can Bob verify the response?

# Challenge-Response




- Nonce is the challenge
- The hash is the response
- Nonce prevents replay (ensures freshness)
- Password is something Alice knows
- Note: Bob must know Alice’s pwd to verify
  - Can we do better?

# Symmetric Key Notation

- Encrypt plaintext $P$ with key $K$

$$C = E(P,K)$$

- Decrypt ciphertext $C$ with key $K$

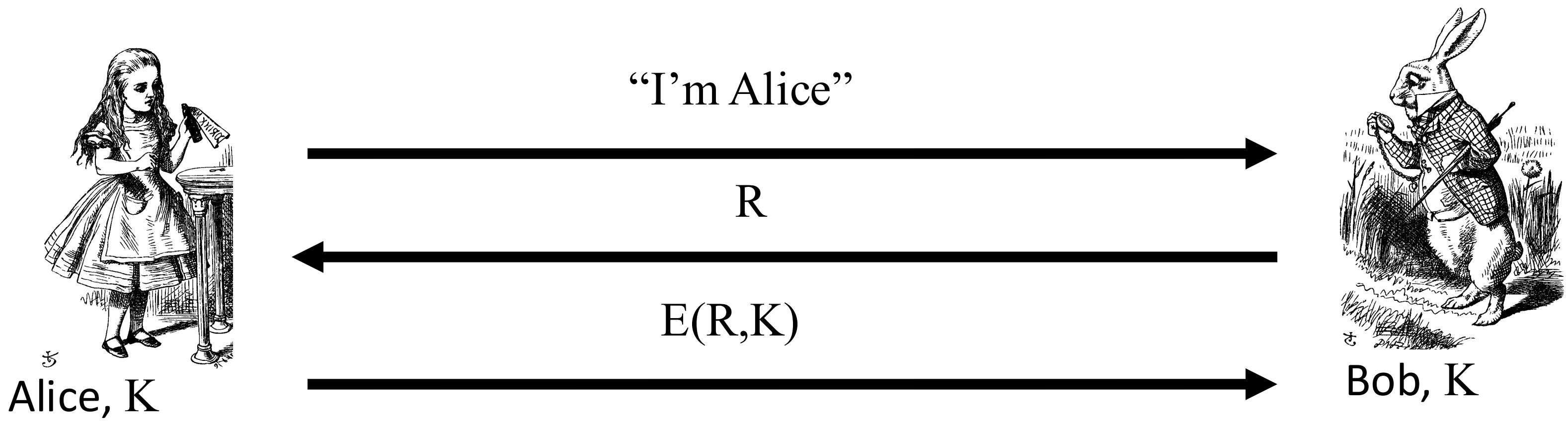
$$P = D(C,K)$$

- Here, we are concerned with attacks on protocols, ***not*** attacks on cryptography
  - So, we assume crypto algorithms are secure

# Authentication: Symmetric Key

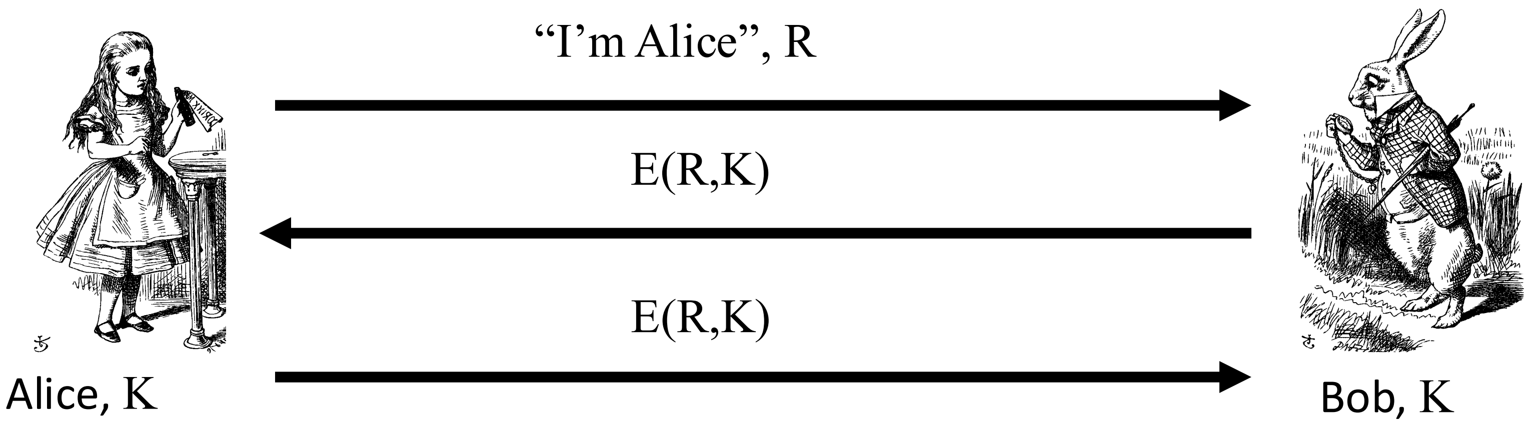
- Alice and Bob share symmetric key $K$
- Key $K$ known only to Alice and Bob
- Authenticate by proving knowledge of shared symmetric key
- How to accomplish this?
  - Cannot reveal key, must not allow replay (or other) attack, must be verifiable, …

# Authenticate Alice Using Symmetric Key




- Secure method for Bob to authenticate Alice
- But, Alice does not authenticate Bob
- So, can we achieve mutual authentication?

# Mutual Authentication?

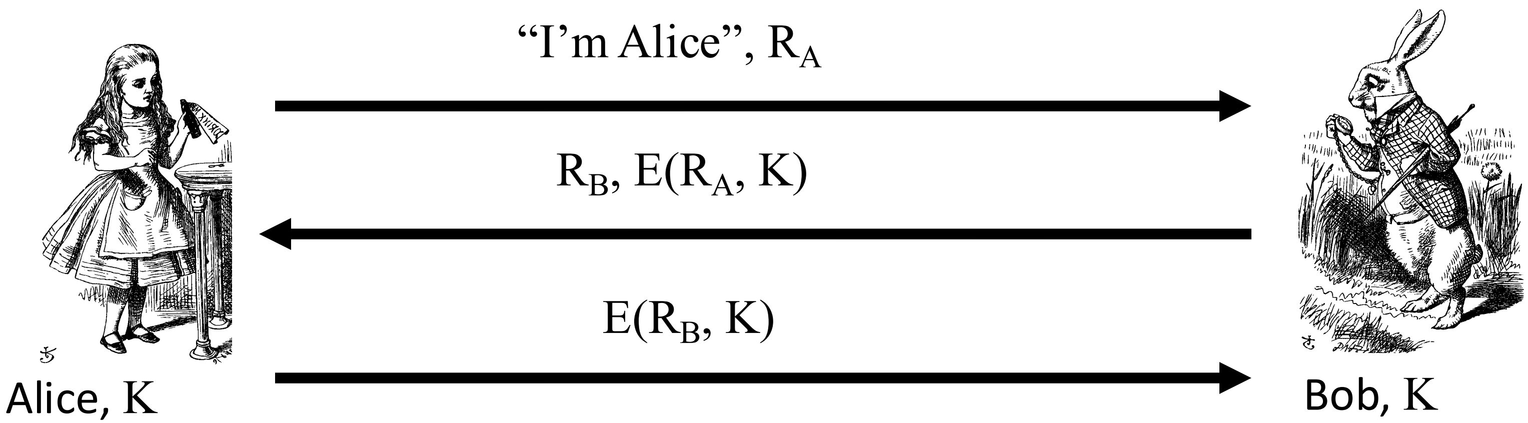



- What's wrong with this picture?

# Mutual Authentication

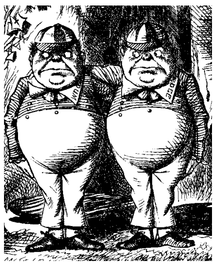
- Since we have a secure one-way authentication protocol…
- The obvious thing to do is to use the protocol twice
    - Once for Bob to authenticate Alice
    - Once for Alice to authenticate Bob
- This has got to work…

# Mutual Authentication




- This provides mutual authentication…
- …or does it? Subject to **reflection** attack
  - Next slide

# Mutual Authentication Attack



Trudy

1. "I'm Alice", $R_A$

2. $R_B$, $E(R_A, K)$

5. $E(\text{"Client"}, R_B, K)$

Bob, K

---

Trudy

3. "I'm Alice", $R_B$

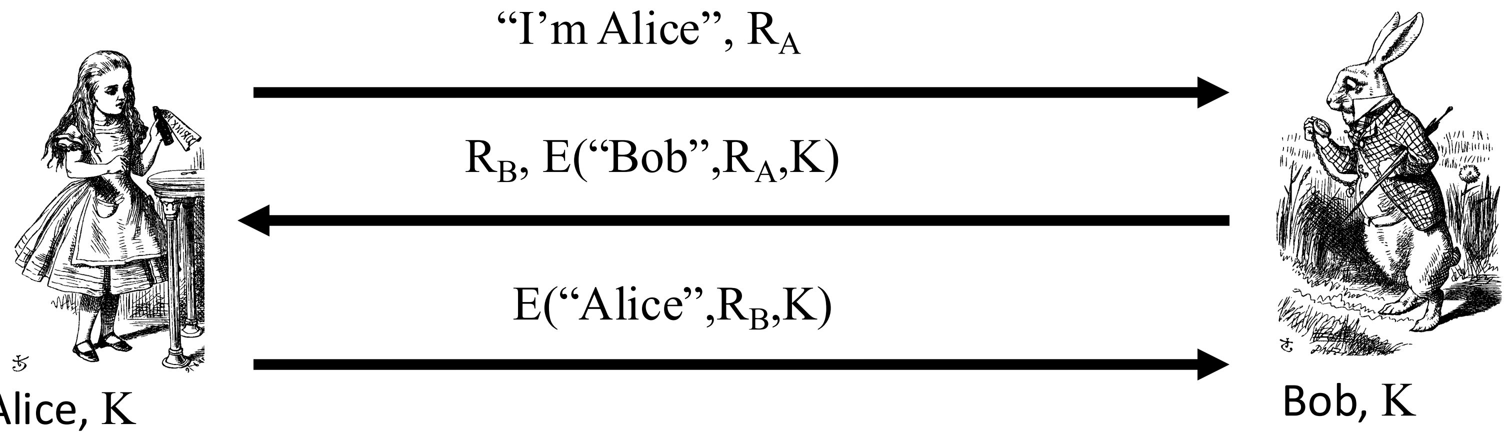
4. $R_C$, $E(\text{"Server"}, R_B, K)$

Bob, K

# Mutual Authentication

- Our one-way authentication protocol is **not** secure for mutual authentication
  - Protocols are subtle!
  - In this case, “obvious” solution is not secure
- Also, if assumptions or environment change, protocol may not be secure
  - This is a common source of security failure
  - For example, Internet protocols

# Symmetric Key Mutual Authentication

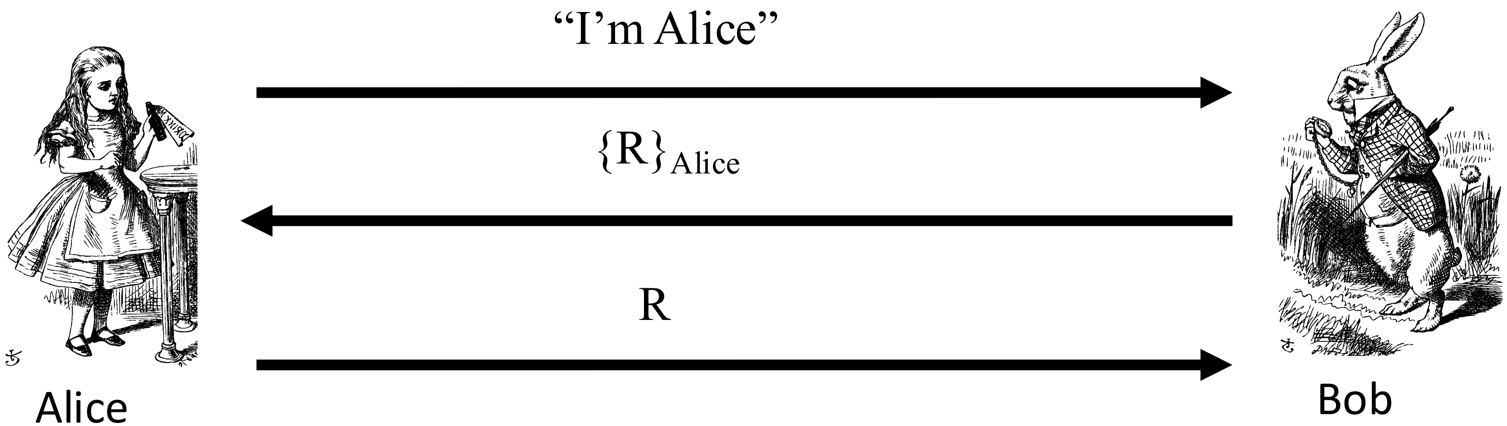



- Do these “insignificant” changes help?
- Yes!

# Public Key Notation

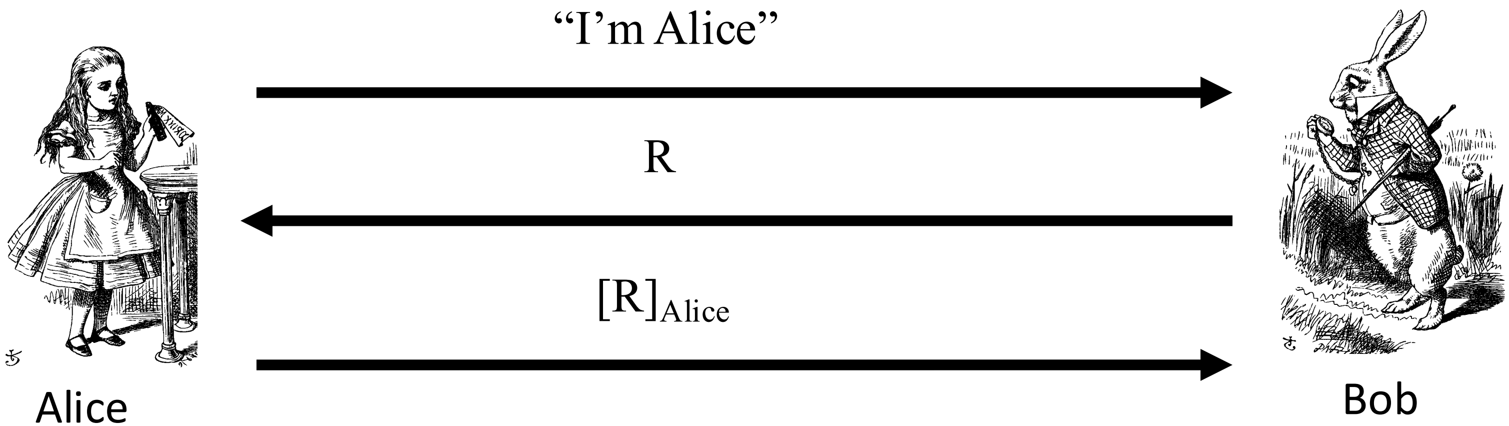
- Encrypt $\mathrm{M}$ with Alice's public key: $\{\mathrm{M}\}_{\mathrm{Alice}}$
- Sign $\mathrm{M}$ with Alice's private key: $[\mathrm{M}]_{\mathrm{Alice}}$
- Then
  - $[\{\mathrm{M}\}_{\mathrm{Alice}}]_{\mathrm{Alice}} = \mathrm{M}$
  - $\{[\mathrm{M}]_{\mathrm{Alice}}\}_{\mathrm{Alice}} = \mathrm{M}$
- **Anybody** can use Alice's **public key**
- Only **Alice** can use her **private key**

# Public Key Authentication




- Is this secure?
- But usually use two key pairs (why?)

# Public Key Authentication




- Is this secure?
- But usually use two key pairs (why?)

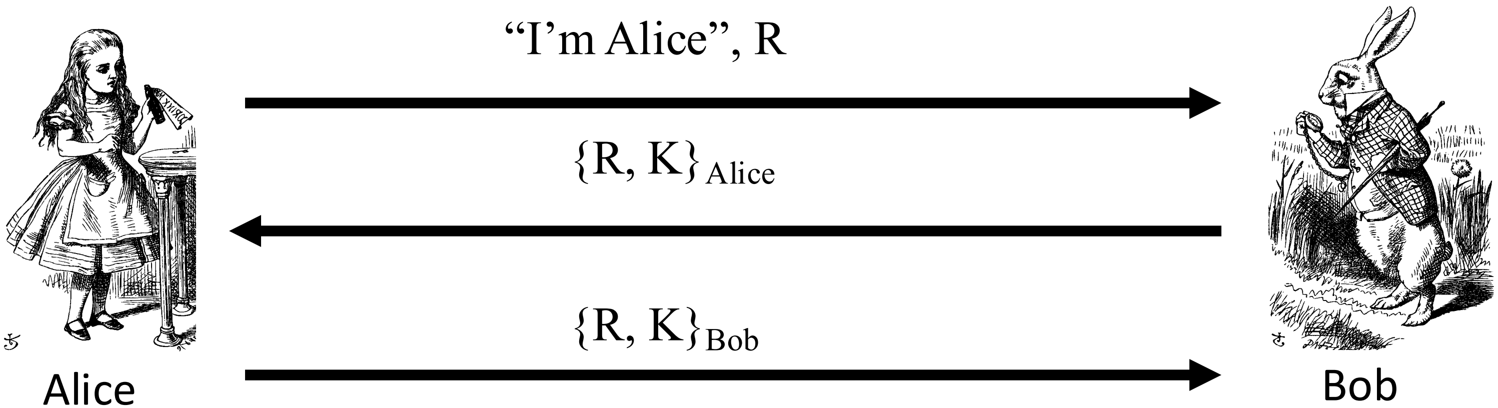
# Public Keys

- Generally, a bad idea to use the same key pair for encryption and signing
- Instead, should have…
  - …one key pair for encryption/decryption and signing/verifying signatures…
  - …and a different key pair for authentication

# Session Key

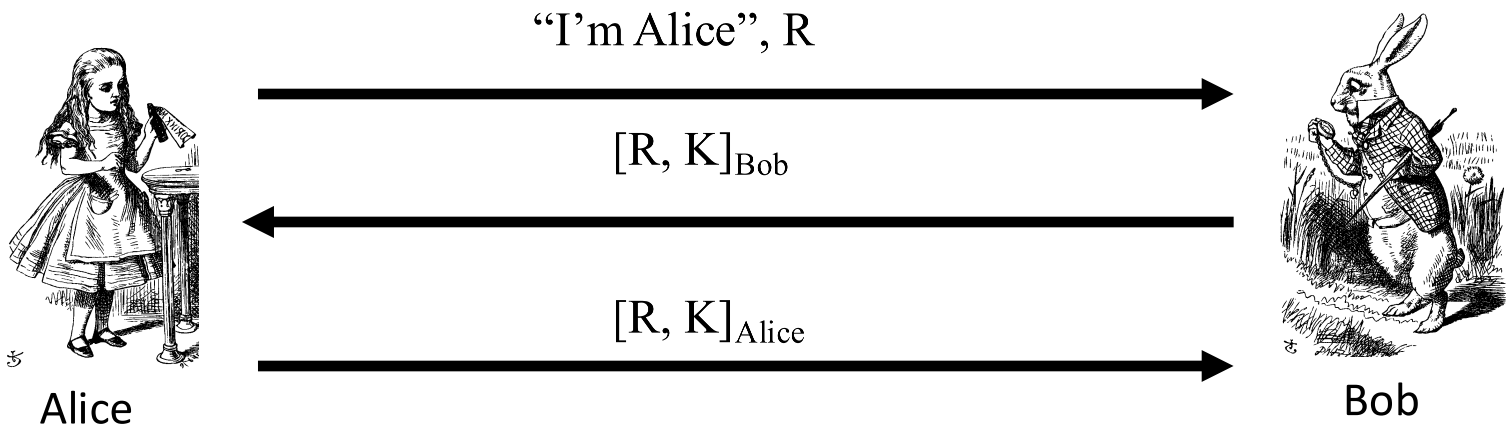
- Usually, a **session key** is required
  - A symmetric key for current session
  - Used for confidentiality and/or integrity
- Ideal case
  - When authentication completed, Alice and Bob share a session key
  - Trudy cannot break the authentication…
  - …***and*** Trudy cannot determine the session key

# Authentication & Session Key




- Is this secure?
  - Alice is authenticated and session key is secure
  - Alice’s “nonce”, R, useless to authenticate Bob
- No mutual authentication

# Public Key Authentication and Session Key




- Is this secure?
  - Mutual authentication (good), but…
  - … session key is not protected (very bad)

# Public Key Authentication and Session Key

Alice

“I’m Alice”, R

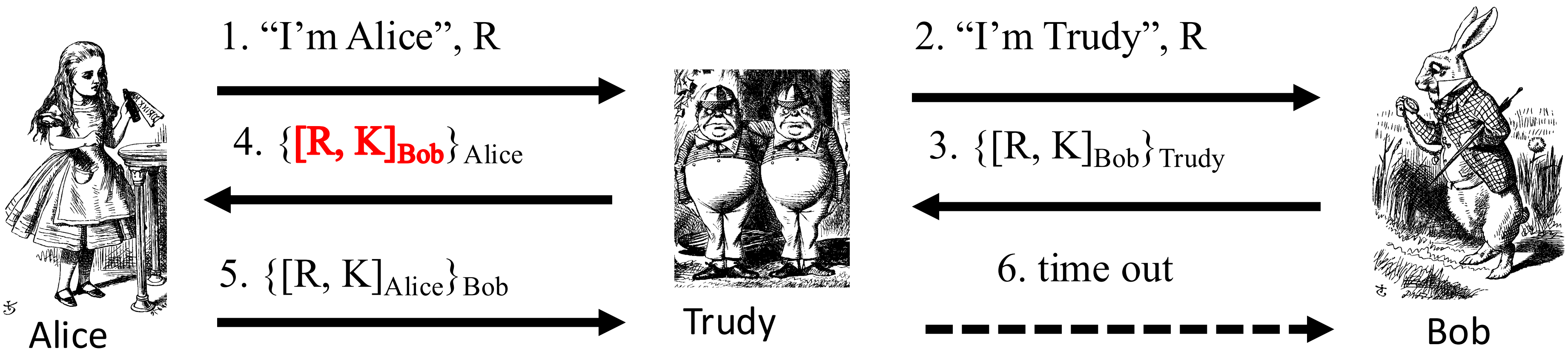
$\{[R, K]_{Bob}\}_{Alice}$

$\{[R, K]_{Alice}\}_{Bob}$

Bob

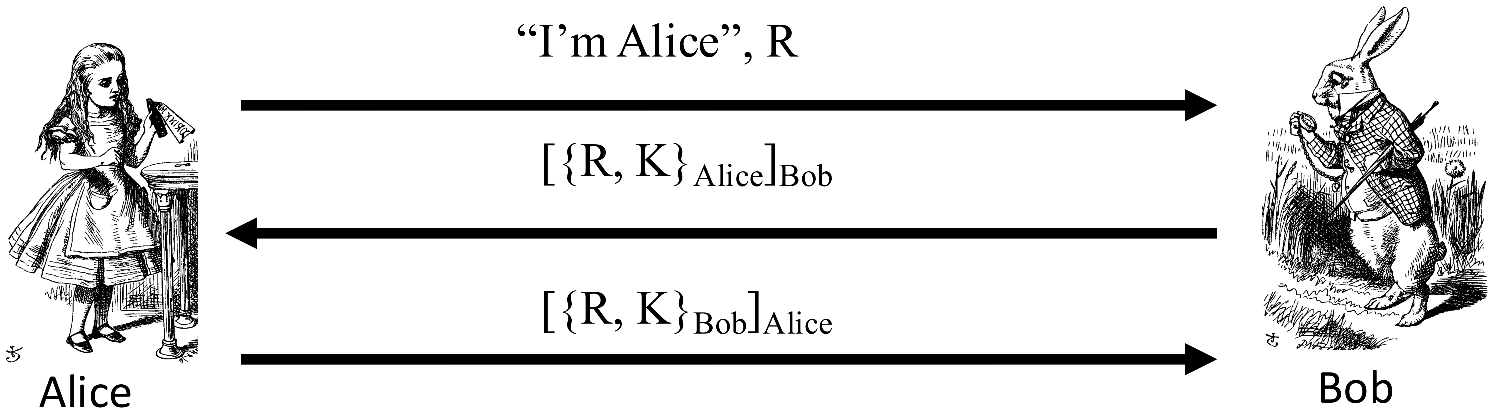
- Is this secure?
- No! It’s subject to subtle MiM attack
  - See the next slide…

# Public Key Authentication and Session Key




- Trudy can get $[R, K]_{Bob}$ and $K$ from 3.
- Alice uses this same key $K$
- And Alice thinks she’s talking to Bob

# Public Key Authentication and Session Key




- Is this secure?
- should be OK
  - Anyone can see $\{R, K\}_{Alice}$ and $\{R, K\}_{Bob}$

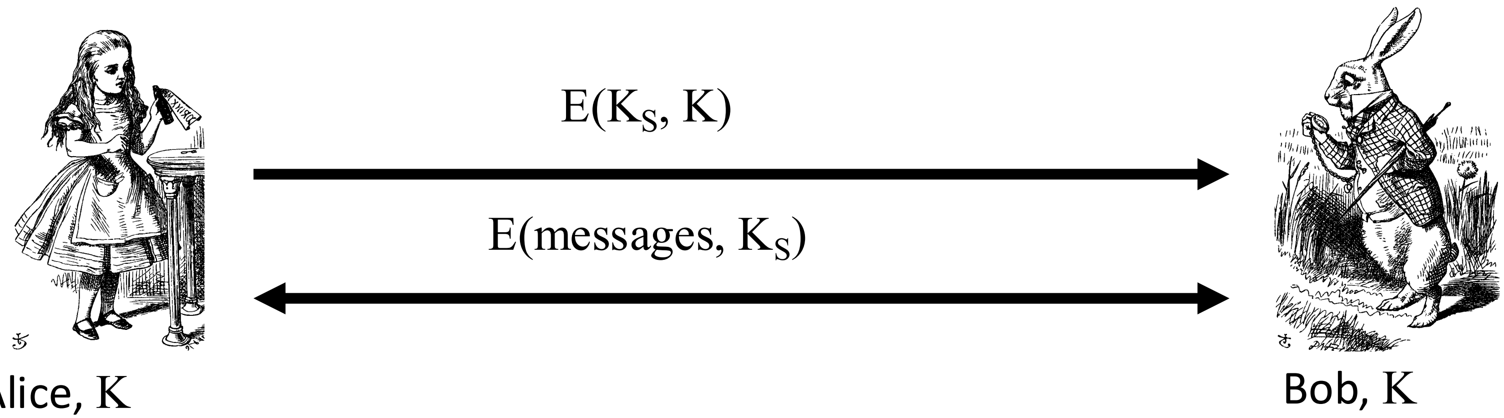
# Perfect Forward Secrecy

- Consider this "issue"…
  - Alice encrypts message with shared key $\mathrm{K}$ and sends ciphertext to Bob
  - Trudy **records** ciphertext and later attacks Alice's (or Bob's) computer to recover $\mathrm{K}$
  - Then Trudy decrypts **recorded** messages
- **Perfect forward secrecy (PFS):** Trudy cannot later decrypt recorded ciphertext
  - Even if Trudy gets key $\mathrm{K}$ or other secret(s)
- Is PFS possible?

# Perfect Forward Secrecy

- Suppose Alice and Bob share key $K$
- For perfect forward secrecy, Alice and Bob cannot use $K$ to encrypt
- Instead they must use a session key $K_S$ and forget it after it's used
- Can Alice and Bob agree on session key $K_S$ in a way that provides PFS?

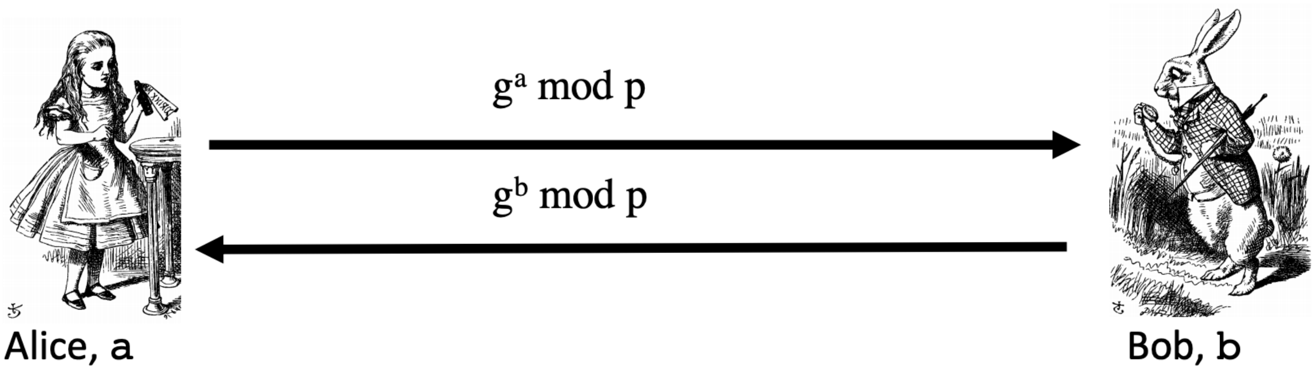
# Naïve Session Key Protocol




- Trudy could record $E(K_S, K)$
- If Trudy later gets $K$ then she can get $K_S$
  - Then Trudy can decrypt recorded messages
- ***No*** perfect forward secrecy in this case

# Diffie-Hellman

- **Public:** g and p
- **Private:** Alice's exponent a, Bob's exponent b




- ❑ Alice computes $(g^b)^a = g^{ba} = g^{ab} \bmod p$
- ❑ Bob computes $(g^a)^b = g^{ab} \bmod p$
- ❑ They can use $K = g^{ab} \bmod p$ as symmetric key

In case you already forgot…

# Diffie–Hellman -- An Example

1. Alice and Bob publicly agree to use a modulus $p = 23$ and base $g = 5$ (which is a primitive root modulo 23).
2. Alice chooses a secret integer $a = 4$, then sends Bob $A = g^a \bmod p$
   - $A = 5^4 \bmod 23 = 4$
3. Bob chooses a secret integer $b = 3$, then sends Alice $B = g^b \bmod p$
   - $B = 5^3 \bmod 23 = 10$
4. Alice computes $s = B^a \bmod p$
   - $s = 10^4 \bmod 23 = 18$
5. Bob computes $s = A^b \bmod p$
   - $s = 4^3 \bmod 23 = 18$
6. Alice and Bob now share a secret (the number 18).

Both Alice and Bob have arrived at the same values because under mod p,

$$A^b \bmod p = g^{ab} \bmod p = g^{ba} \bmod p = B^a \bmod p$$

More specifically,

$$(g^a \bmod p)^b \bmod p = (g^b \bmod p)^a \bmod p$$

# Perfect Forward Secrecy

$E(g^a \bmod p, K) \longrightarrow$

$\longleftarrow E(g^b \bmod p, K)$

Alice: K, a

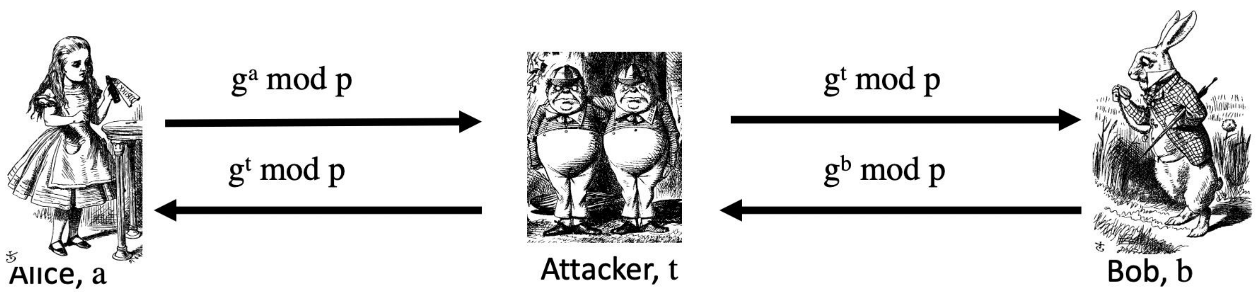
Bob: K, b

- Session key $K_S = g^{ab} \bmod p$
- Alice **forgets** a, Bob **forgets** b
- Neither Alice nor Bob can later recover $K_S$

# But … Diffie-Hellman is subject to man-in-the-middle attack, isn’t it?

## Diffie-Hellman

- Subject to man-in-the-middle (MiM) attack




- Trudy shares secret $g^{at} \bmod p$ with Alice
- Trudy shares secret $g^{bt} \bmod p$ with Bob
- Alice and Bob don’t know attacker is MiM

We don’t worry about that, why? Authentication!

# Mutual Authentication, Session Key and PFS

Alice

“I’m Alice”, $R_A$

$R_B$, $[\{R_A, g^b \bmod p\}_{Alice}]_{Bob}$

$[\{R_B, g^a \bmod p\}_{Bob}]_{Alice}$

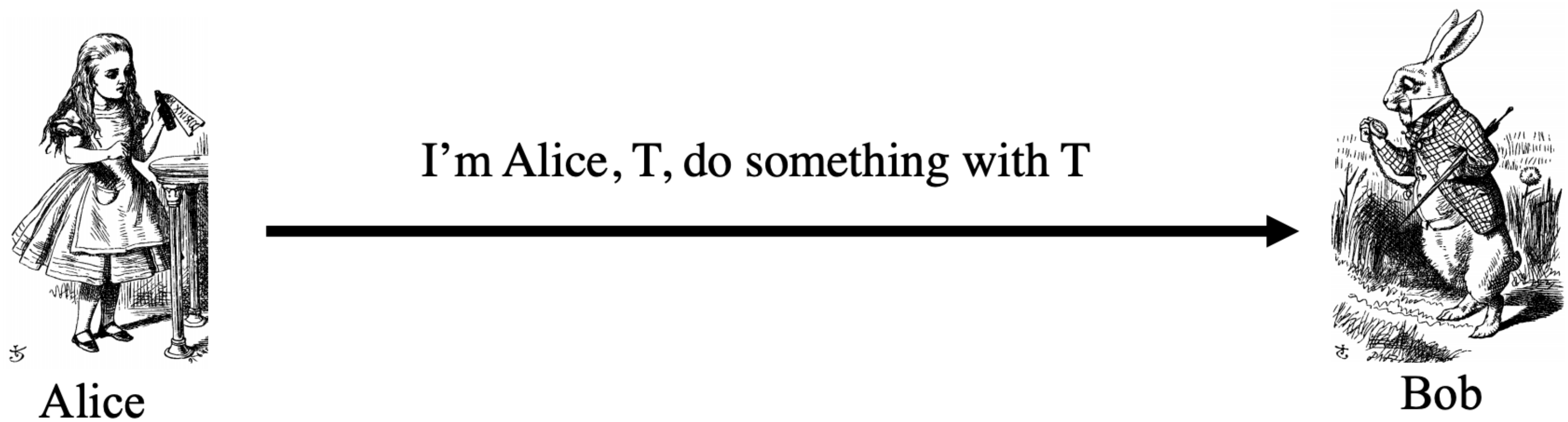
Bob

- Session key is $K = g^{ab} \bmod p$
- Alice forgets $a$ and Bob forgets $b$
- If Trudy later gets Bob’s and Alice’s secrets, she cannot recover session key $K$

# Timestamps

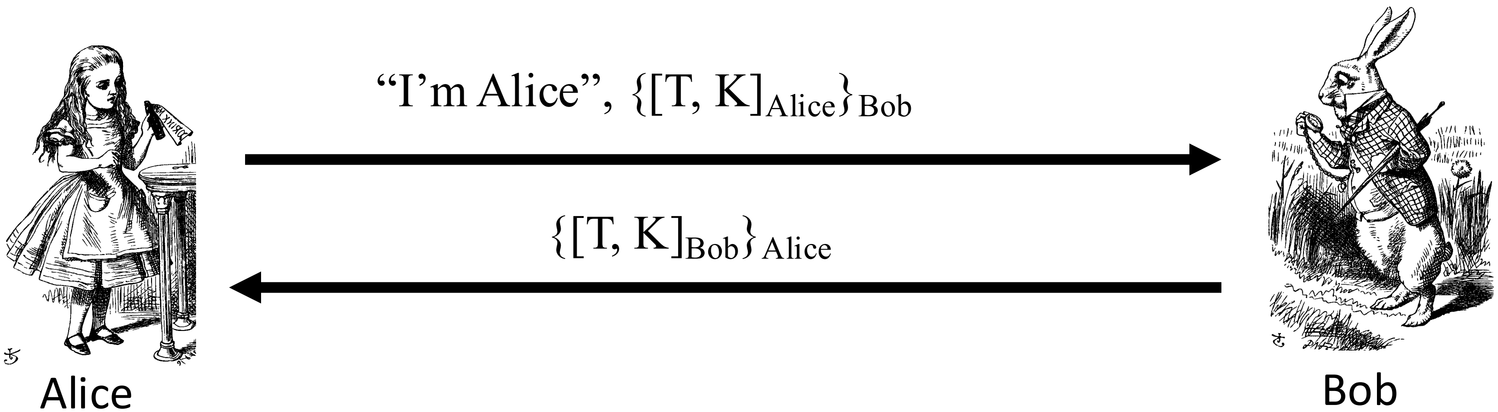
- A timestamp $\mathrm{T}$ is derived from current time
- Timestamps can be used to prevent replay attack
  - A challenge that both sides know in advance
- "Time" is a security-critical parameter (bad)
  - Sometimes timestamp is used to as the seed for random numbers
- More importantly, clocks not same and/or network delays, so must allow for **clock skew** — creates risk of replay
  - During the time-skew window, there is still a room for reply attack for Attacker to impersonate Alice.
  - How much clock skew is enough?
    - Too much? Too little?
    - Real world: 5 mins….

# Timestamp Example, High Level




1. Alice gets the time T and performs her calculations
2. Alice sends her message along with the timestamp T
3. Bob checks the time and verifies it is within the skew
4. If so, Bob verifies Alice's calculations

# Public Key Authentication with Timestamp T




- Secure mutual authentication?
- Session key secure?
- Seems to be OK

# Public Key Authentication with Timestamp T

Alice

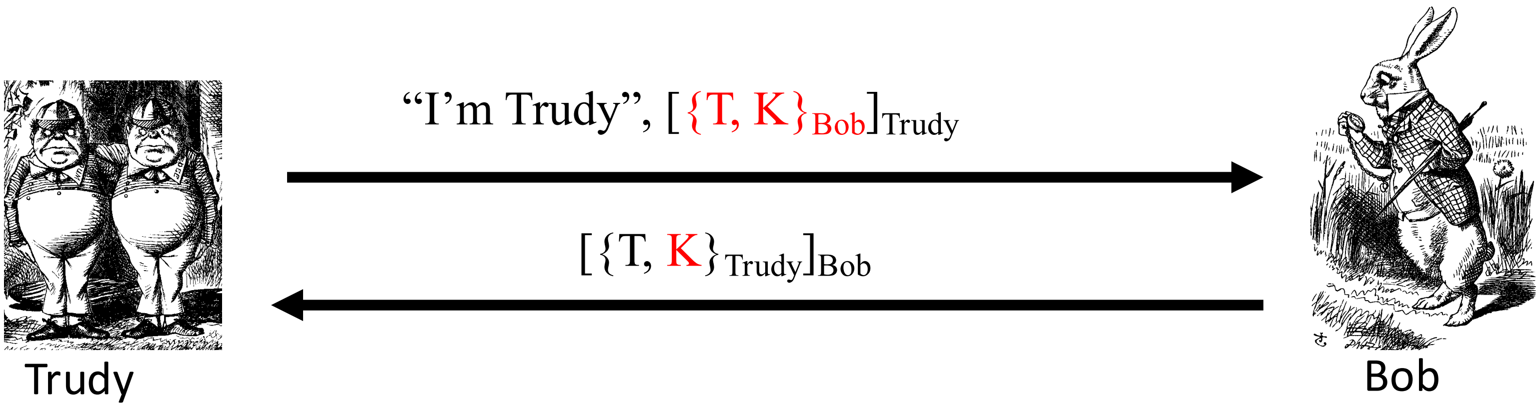
“I’m Alice”, $[\{T, K\}_{Bob}]_{Alice}$

$[\{T, K\}_{Alice}]_{Bob}$

Bob

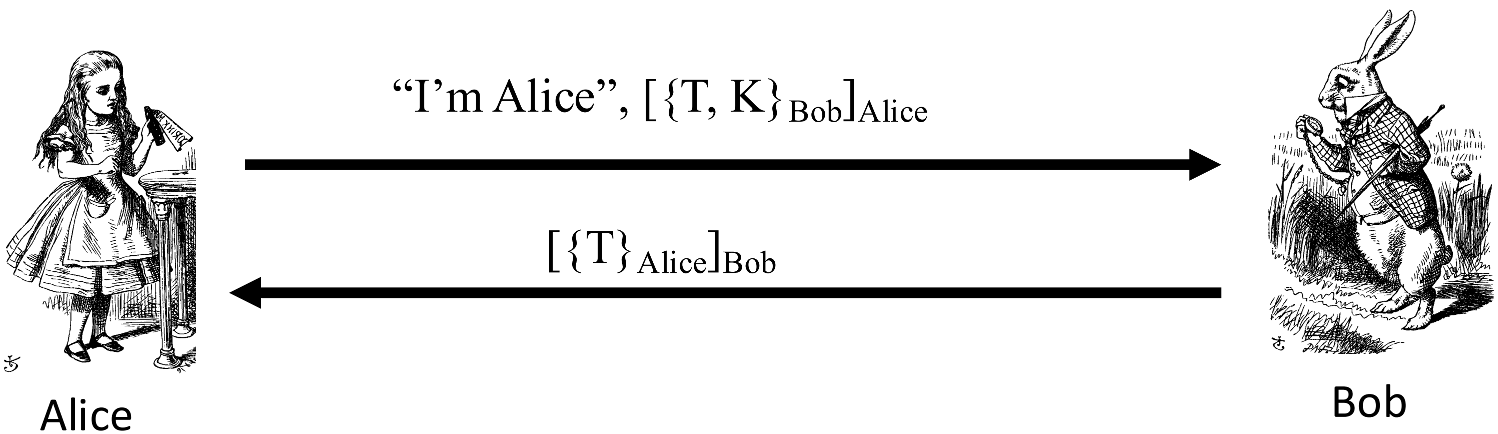
- Secure authentication and session key?
- Trudy can use Alice’s public key to find $\{T, K\}_{Bob}$ and then…

# Public Key Authentication with Timestamp $T$




- Trudy obtains Alice-Bob session key $K$
- Note: Trudy must act within clock skew

# Public Key Authentication with Timestamp T




- Can we fix the previous flaw?
  - No session key is needed in the second round!

# Real-World Protocols

- Some real secure protocols
  - SSH — relatively simple & useful protocol
    - Mutual authentication, session key and PFS
  - SSL — practical security on the Web
  - IPSec — security at the IP layer
  - GSM — mobile phone (in)security

# Secure Shell (SSH)

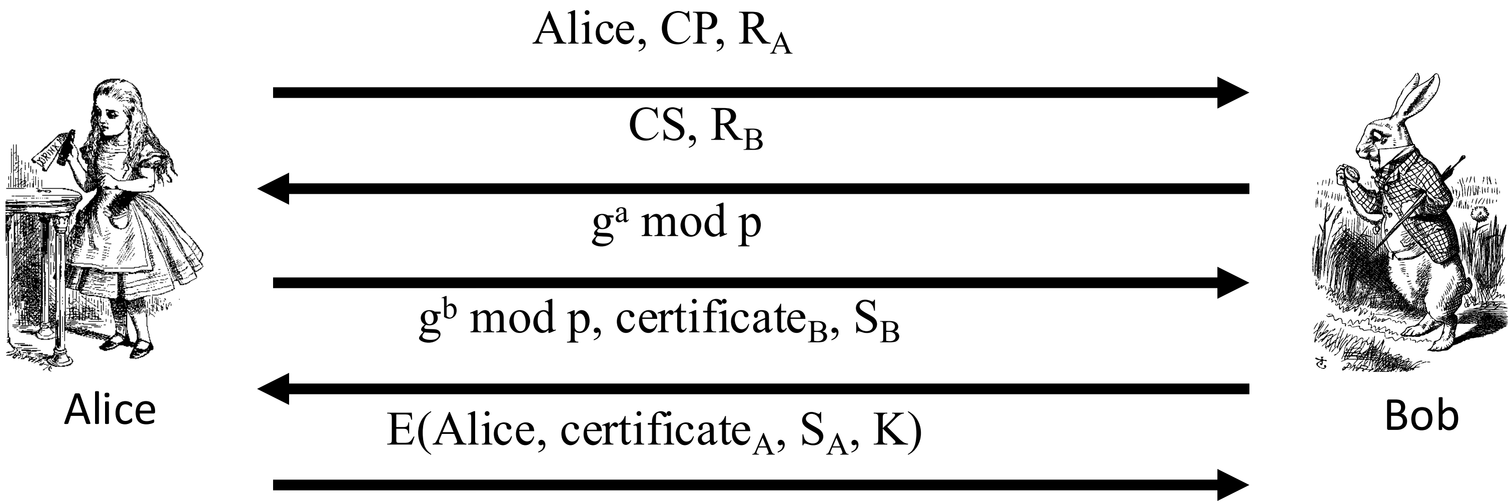
# SSH

- Creates a “secure tunnel”
- Insecure command sent thru SSH “tunnel” are then secure
- SSH is a relatively simple protocol

# SSH

- SSH authentication can be based on:
  - Public keys
    - The default setup if you use Amazon AWS and want to get authenticated.
    - In this case, “private key” is often referred to as “identity key”
  - Digital certificates
    - Create a key pair (with ssh-keygen) // often we don’t use our own key but create a new one on the machine.
    - Ask the CA whose info is in your machine to sign the pub key
    - Start to use the newly created certificate to use the **certificate mode**.
  - Passwords
    - Technically similar with “public key”; easier, as people do not need to maintain their private key.
- Here, we consider ***certificate*** mode

# Simplified SSH




- CP = “crypto proposed”, and CS = “crypto selected”
- H = h(Alice,Bob,CP,CS,$R_A$,$R_B$,$g^a$ mod p,$g^b$ mod p,$g^{ab}$ mod p)
- $S_B = [H]_{Bob}$
- $S_A = [H, \text{Alice}, \text{certificate}_A]_{Alice}$
- K = $g^{ab}$ mod p

# MiM Attack on SSH?

Alice

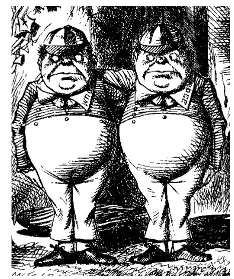
Alice, $R_A$ →

← $R_B$

$g^a \bmod p$ →

← $g^t \bmod p$, $cert_B$, $S_B$

$E(Alice, cert_A, S_A, K)$ →



Trudy

Alice, $R_A$ →

← $R_B$

$g^t \bmod p$ →

← $g^b \bmod p$, $cert_B$, $S_B$

$E(Alice, cert_A, S_A, K)$ →

Bob

- Where does this attack fail?
- Alice computes

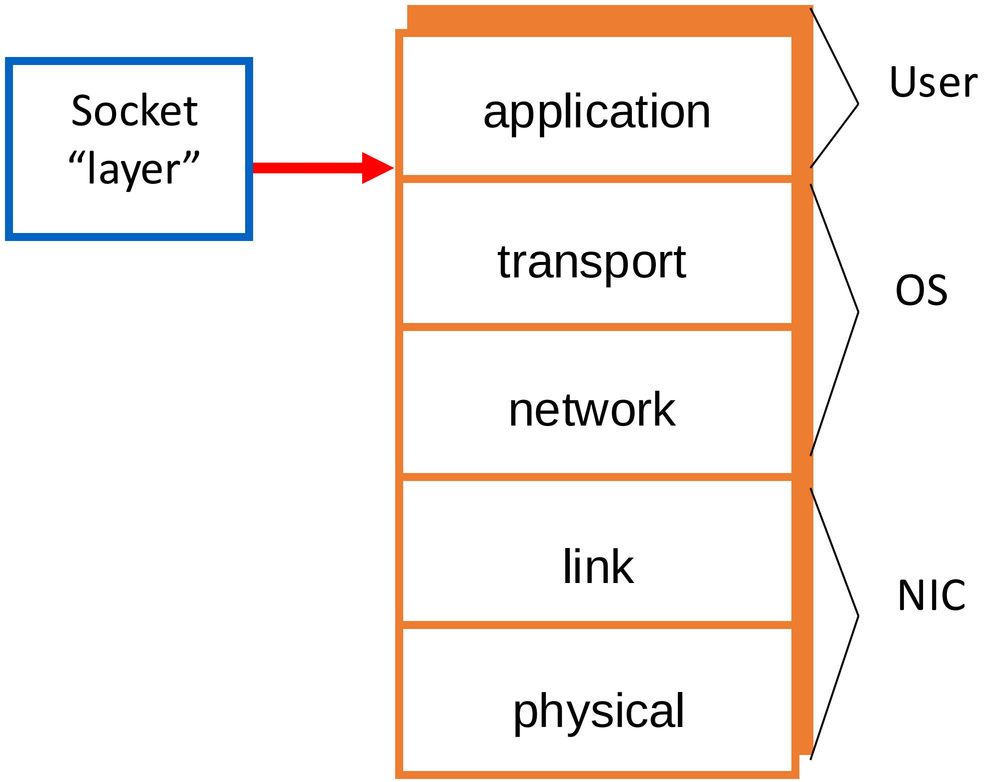
  $H_a = h(Alice, Bob, CP, CS, R_A, R_B, g^a \bmod p, g^t \bmod p, g^{at} \bmod p)$
- But Bob signs

  $H_b = h(Alice, Bob, CP, CS, R_A, R_B, g^t \bmod p, g^b \bmod p, g^{bt} \bmod p)$

# Secure Socket Layer

# Socket layer

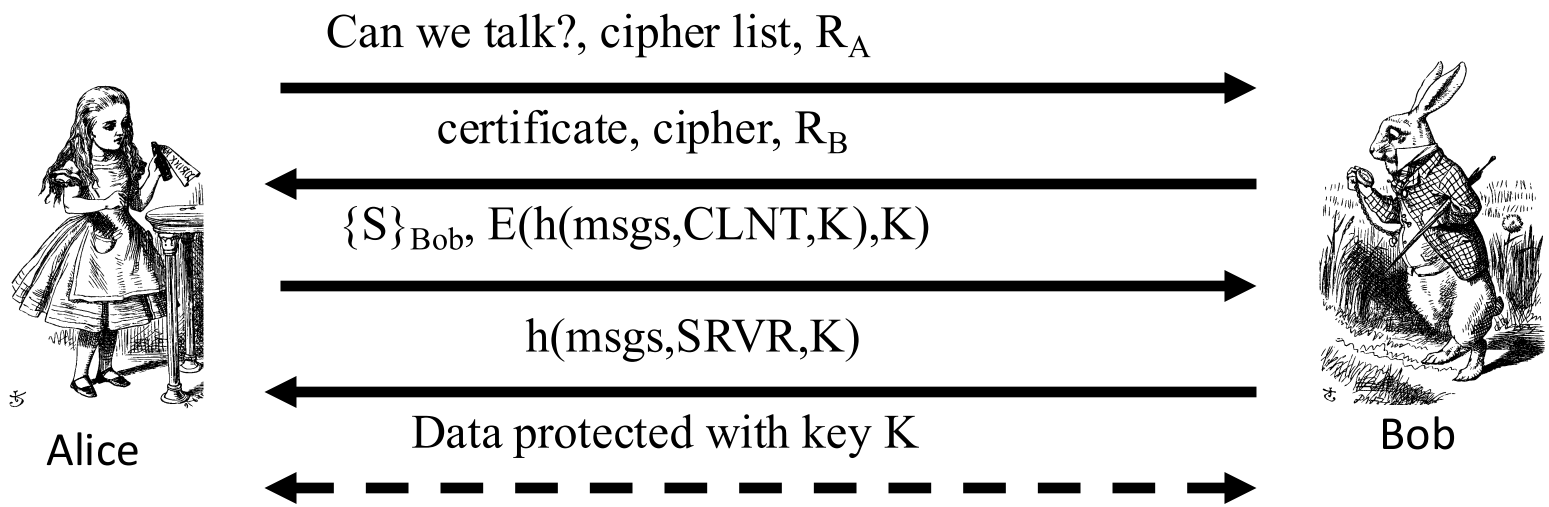
- "Socket layer" lives between application and transport layers
- SSL usually between HTTP and TCP



# What is SSL?

- SSL is the protocol used for majority of secure Internet transactions today
- For example, if you want to buy a book at amazon.com…
  - You want to be sure you are dealing with Amazon (**authentication**)
  - Your credit card information must be protected in transit (**confidentiality** and/or **integrity**)
  - No mutual authentication.
    - Use password, instead.

# Simplified SSL Protocol




- S is the so-called **pre-master secret**
- $K = h(S,R_A,R_B)$
- "msgs" means all previous messages
- CLNT and SRVR are constants

# Implementation considerations

- 6 “keys” derived from $K = h(S, R_A, R_B)$
  - 2 encryption keys: client and server
  - 2 integrity keys: client and server
  - 2 IVs: client and server
  - Why different keys in each direction?
    - Implementation purpose
- Alice authenticates Bob, not vice-versa
  - How does client authenticate server?
  - Why would server not authenticate client?