Part III: Protocols

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: SSL, IPSec, Kerberos, etc.

Protocols

- Protocol flaws can be very **subtle**
- Several well-known security protocols have significant flaws
  - Including WEP,WPA2/3, GSM, and IPSec
- Implementation errors can also occur
  - Recently, IE implementation of SSL
- 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!

Chapter 9: Simple Security Protocols

"I quite agree with you," said the Duchess; "and the moral of that is
'Be what you would seem to be' or
if you'd like it put more simply 'Never imagine yourself not to be
otherwise than what it might appear to others that what you were
or might have been was not otherwise than what you
had been would have appeared to them to be otherwise.' "
Lewis Carroll, *Alice in Wonderland*

Seek simplicity, and distrust it.
Alfred North Whitehead

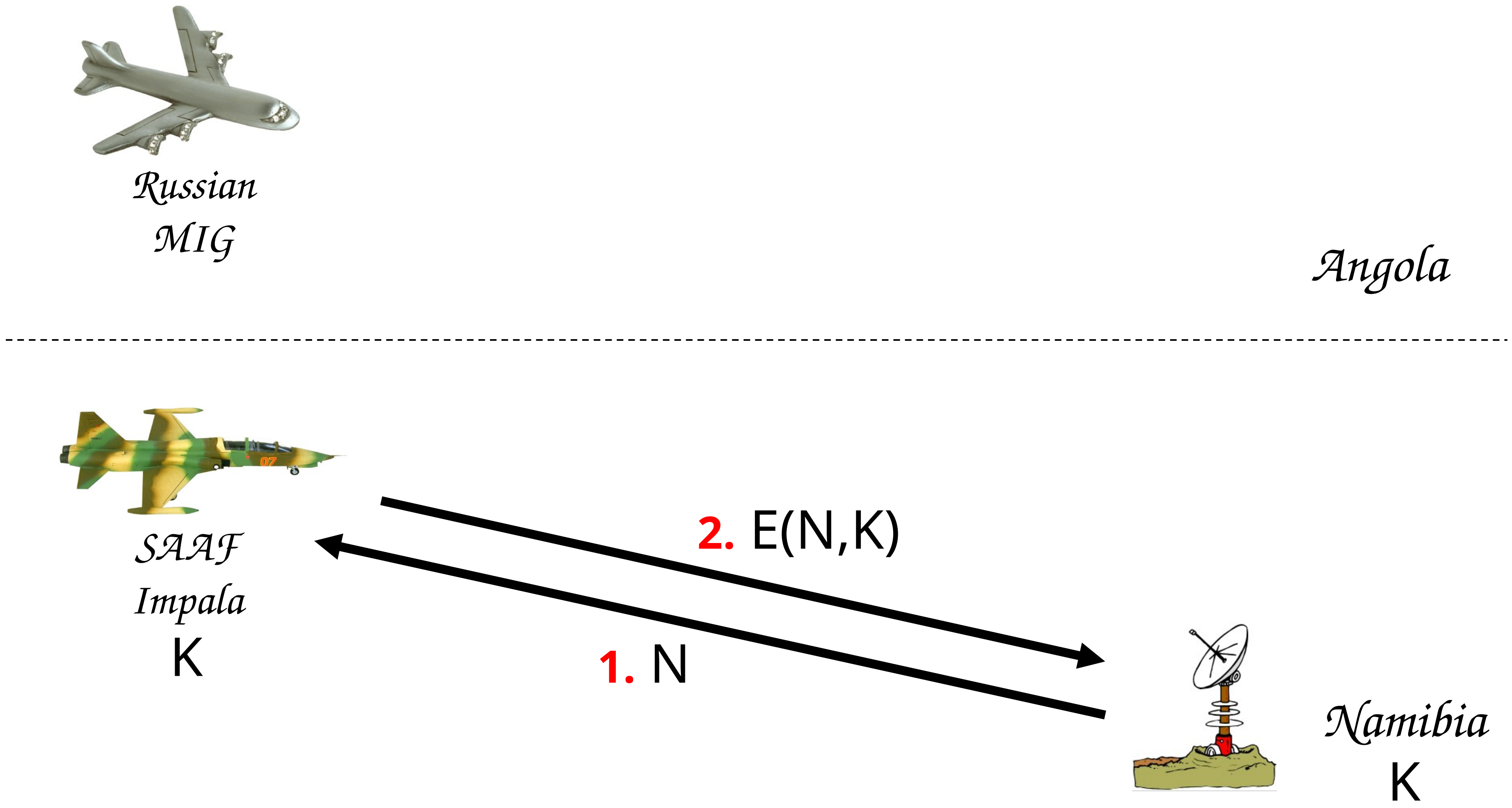
Secure Entry to NSA

1. Insert badge into reader
2. Enter PIN
3. Correct PIN?
    - **Yes?** Enter
    - **No?** Get shot by security guard

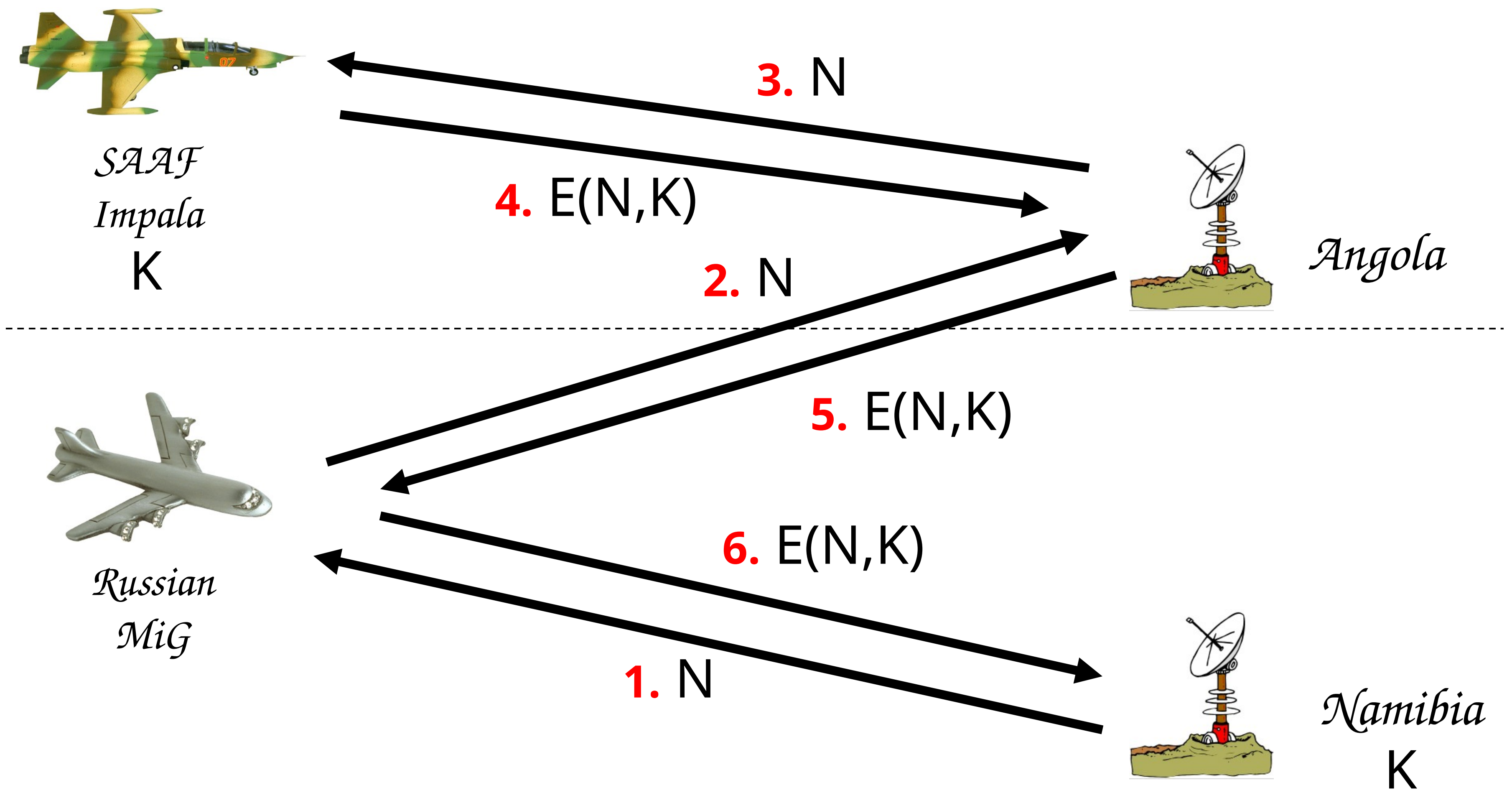
ATM Machine Protocol

1. Insert ATM card
2. Enter PIN
3. Correct PIN?
   - **Yes?** Conduct your transaction(s)
   - **No?** Machine (eventually) eats card

Identify Friend or Foe (IFF)



MIG in the Middle



Authentication Protocols

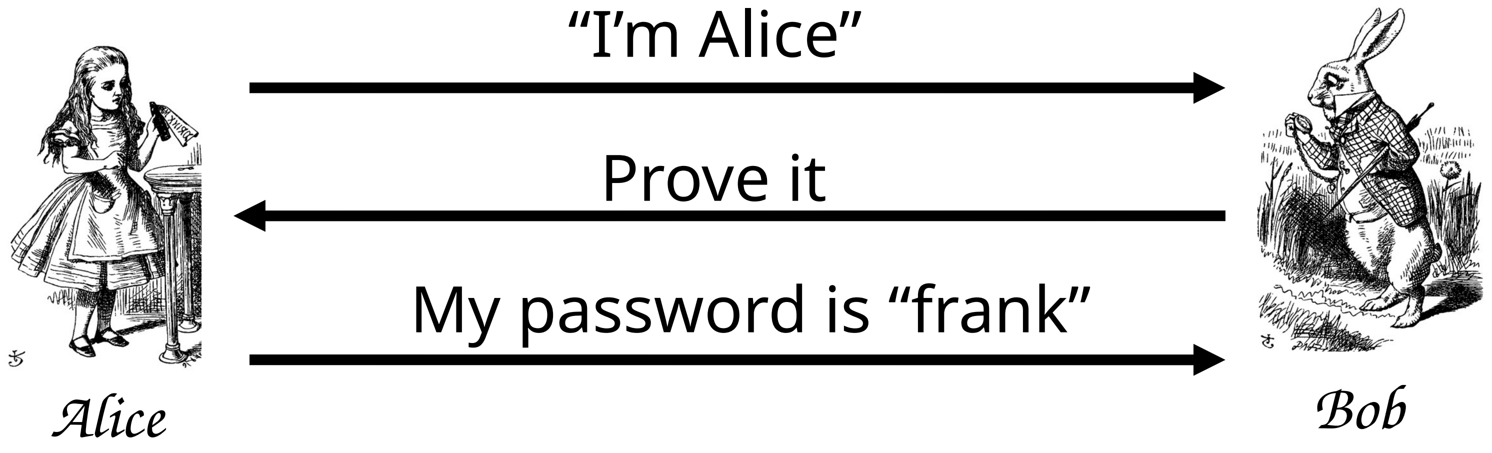
Authentication

- 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

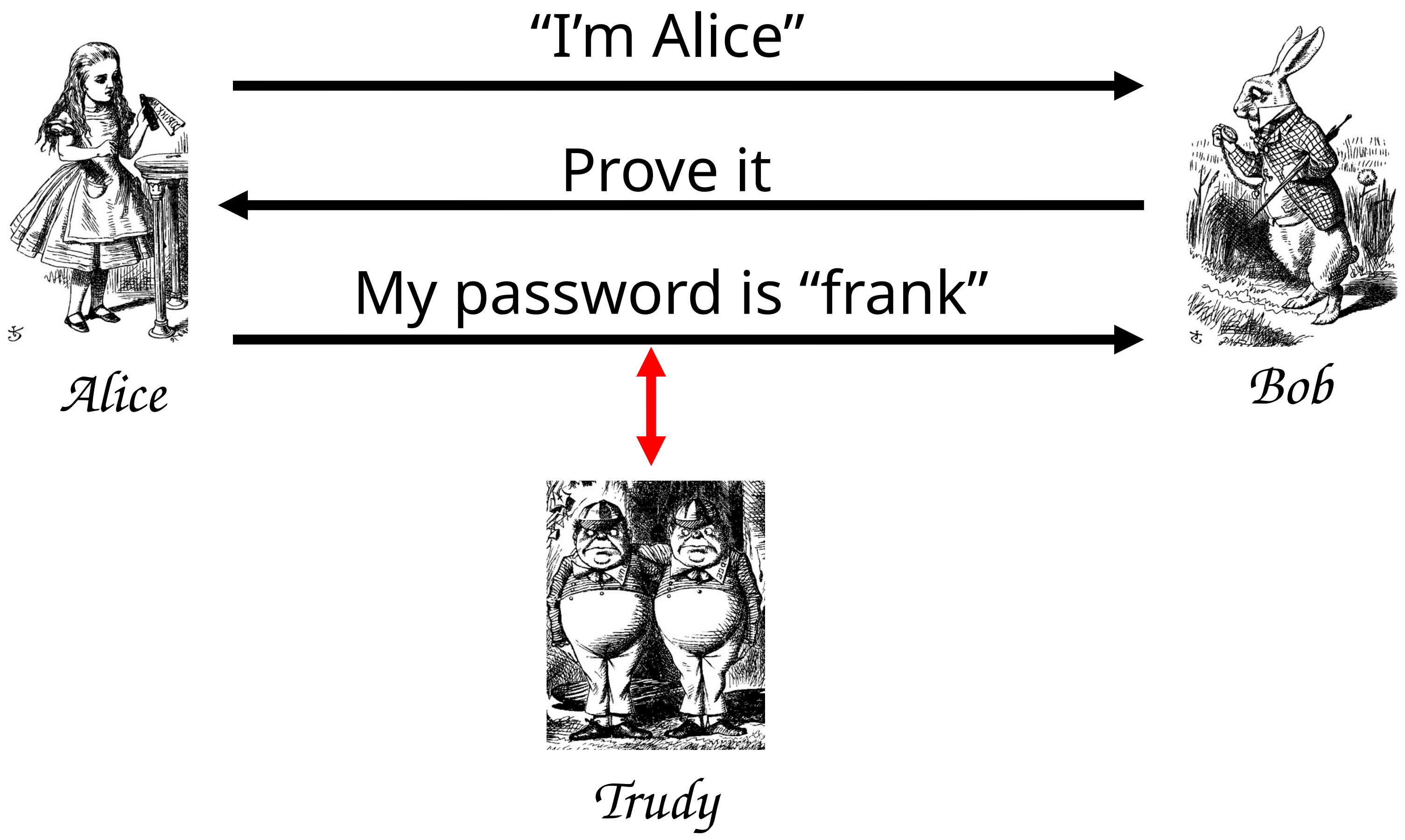
- Authentication on a stand-alone computer is relatively simple
  - For example, hash a password with a salt
  - "Secure path," attacks on authentication software, keystroke logging, etc., can be issues
- Authentication over a network is challenging
  - Attacker can passively observe messages
  - Attacker can replay messages
  - Active attacks possible (insert, delete, change)

Simple Authentication

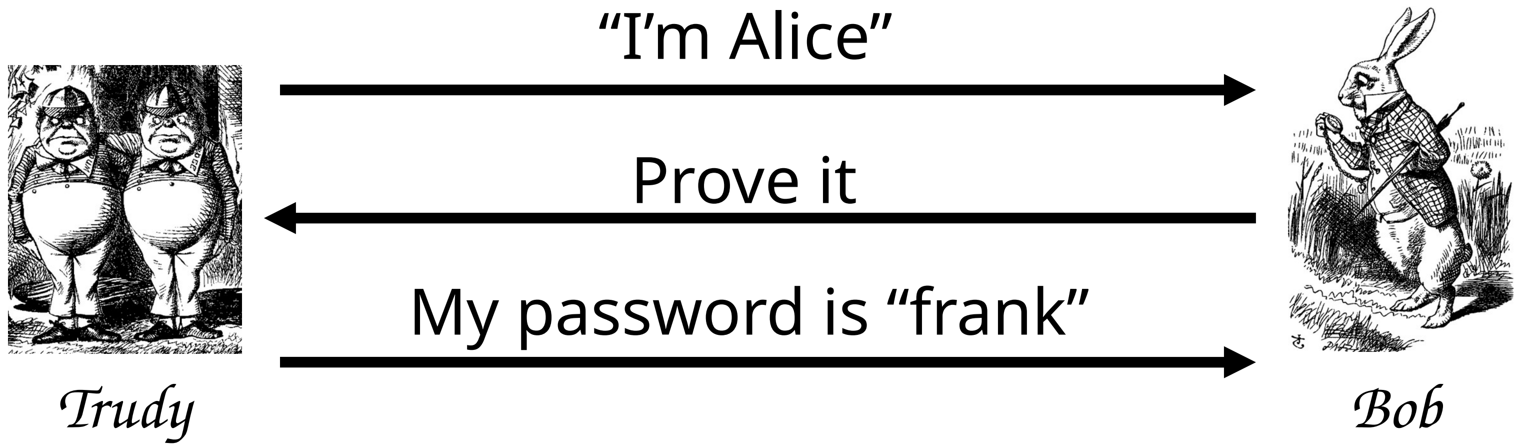


- Simple and may be OK for standalone system
- But highly insecure for networked system
  - Subject to a **replay** attack (next 2 slides)
  - Also, Bob must know Alice's password

Authentication Attack

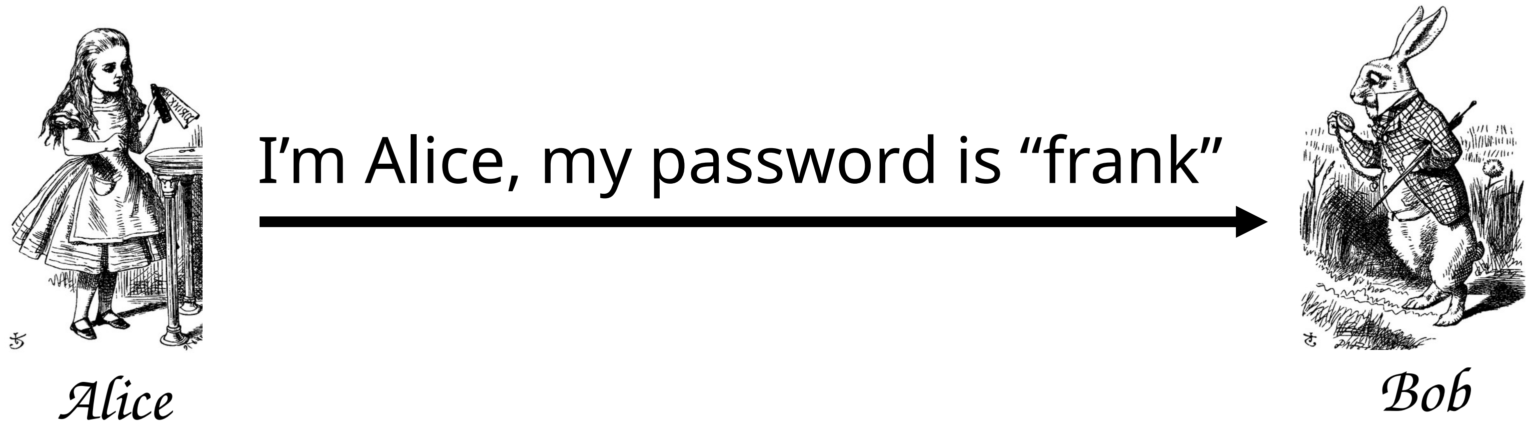


Authentication Attack



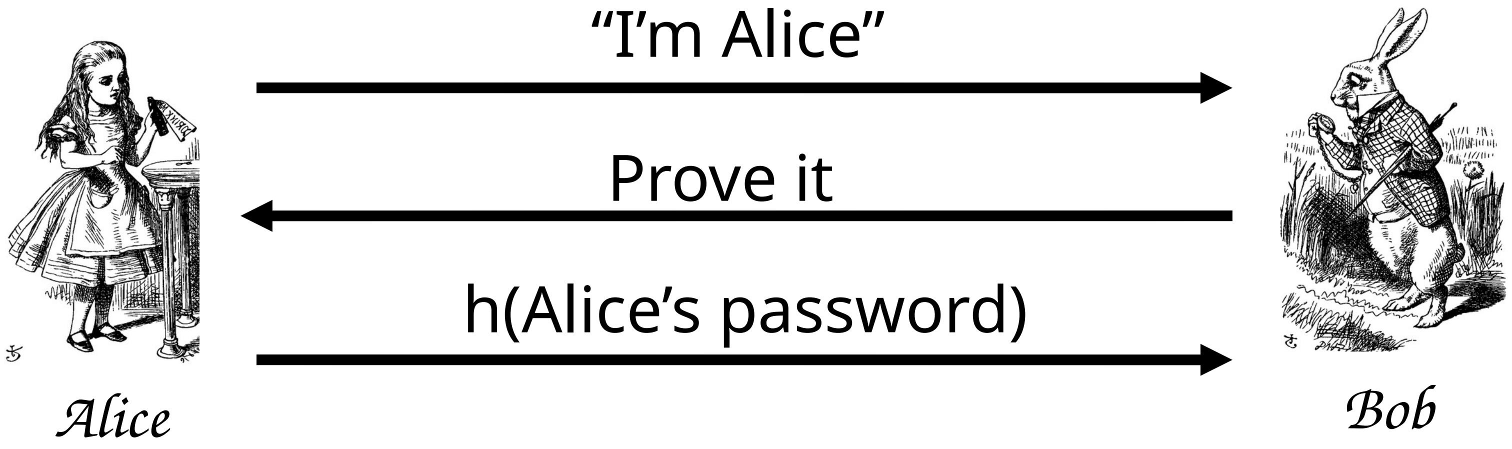
- *This is an example of a* ***replay*** *attack*
- *How can we prevent a replay?*

Simple Authentication



- 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*

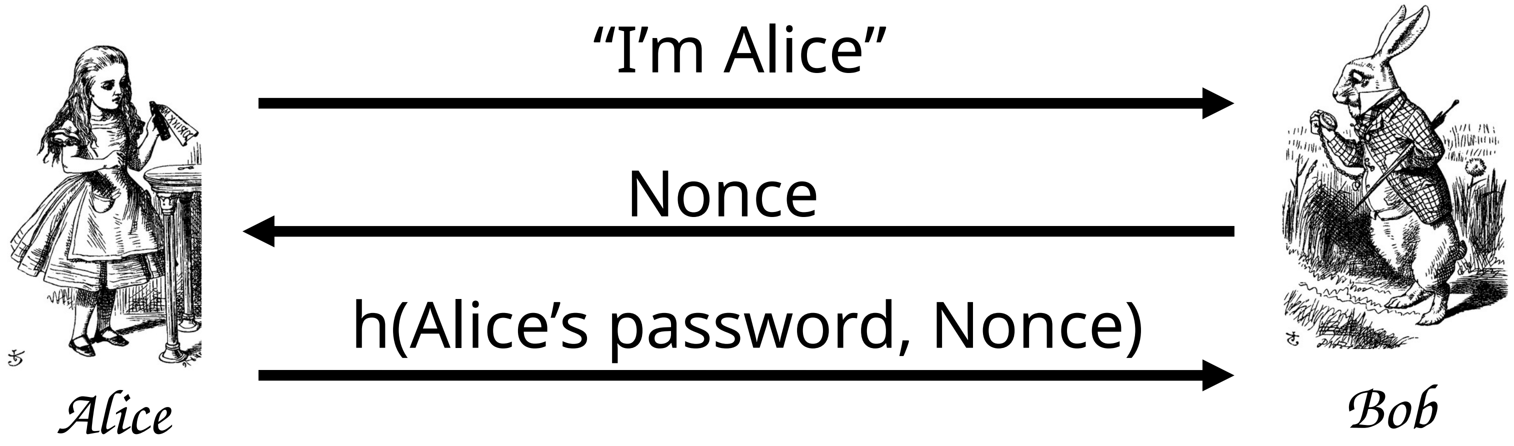
Challenge-Response

- 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

Nonce

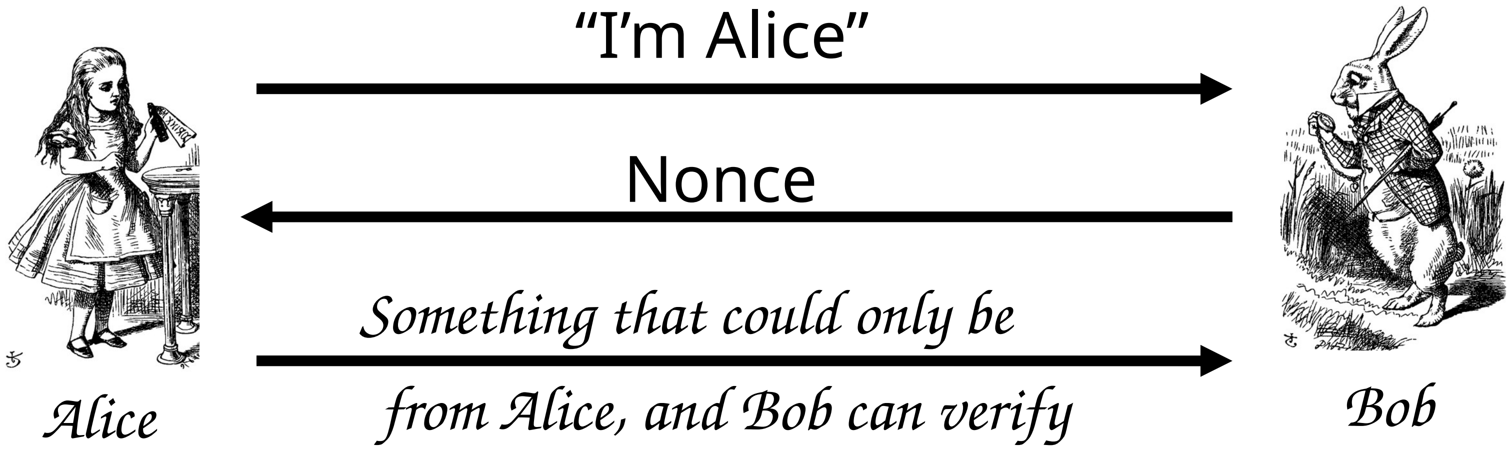
- To ensure freshness, can employ a ***nonce***
  - Nonce == ***n***umber used ***once***
- What to use for nonces?
  - That is, what is the challenge?
- What should Alice do with the nonce?
  - That is, how to compute the response?
- How can Bob verify the response?
- Should we use passwords or keys?

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*

Generic Challenge-Response



- *In practice, how to achieve this?*
- *Hashed password works, but…*
- *…encryption is much better here (why?)*

Symmetric Key Notation

- *Encrypt plaintext* P *with key* K

  C = E(P,K)

- *Decrypt ciphertext* C *with key* K

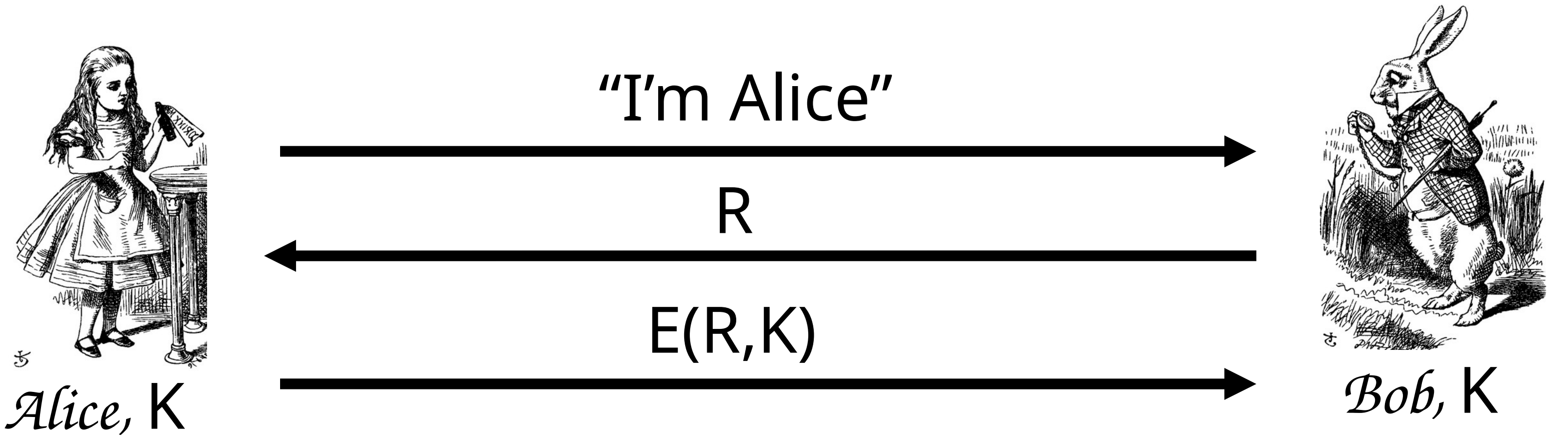
  P = D(C,K)

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

Authentication: Symmetric Key

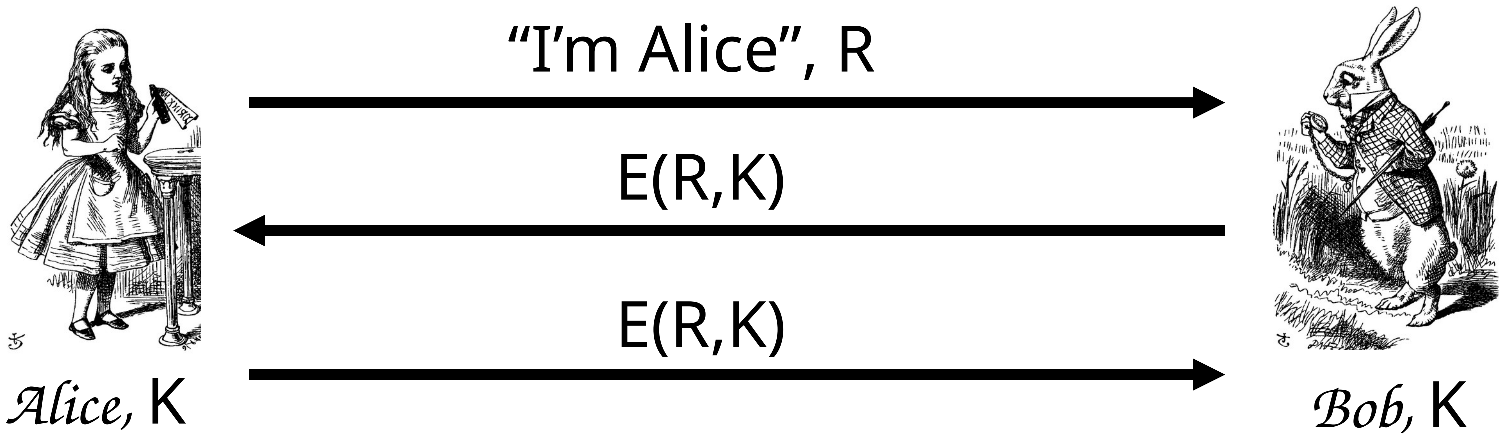
- Alice and Bob share symmetric key K
- Key K known only to Alice and Bob
- Authenticate by proving knowledge of shared symmetric key
- How to accomplish this?
  - Cannot reveal key, must not allow replay (or other) attack, must be verifiable, …

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?

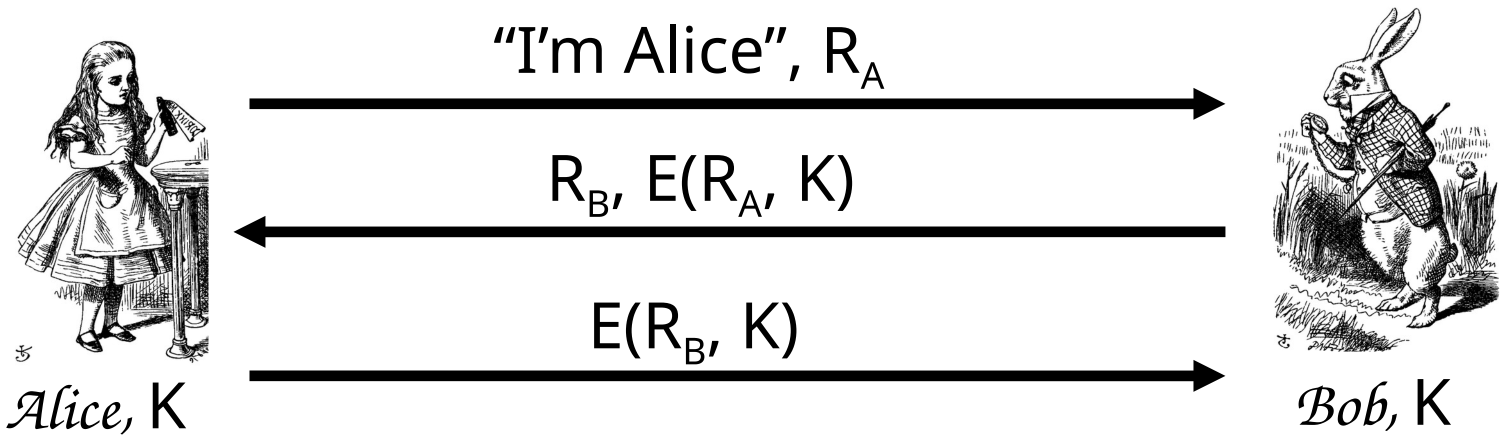


- *What’s wrong with this picture?*
- *“Alice” could be Trudy (or anybody else)!*

Mutual Authentication

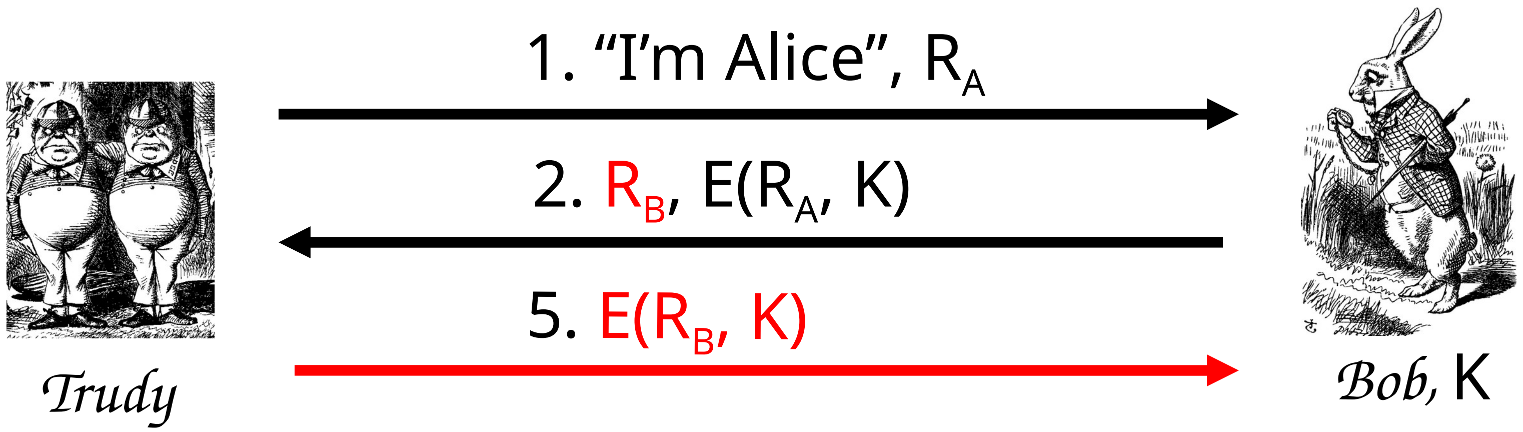
- Since we have a secure one-way authentication protocol…
- The obvious thing to do is to use the protocol twice
  - Once for Bob to authenticate Alice
  - Once for Alice to authenticate Bob
- This has got to work…

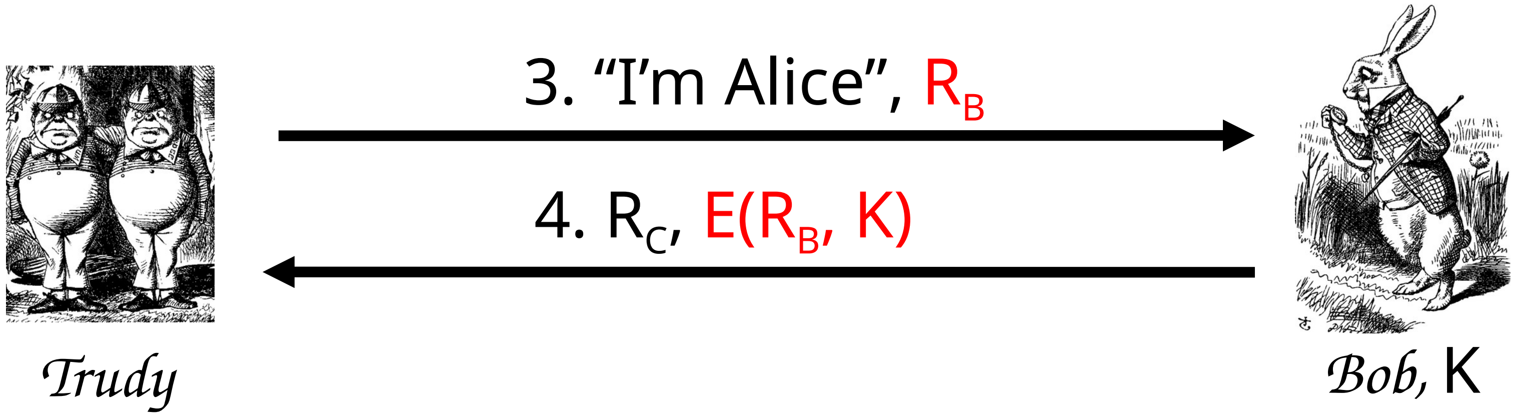
Mutual Authentication



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

Mutual Authentication Attack

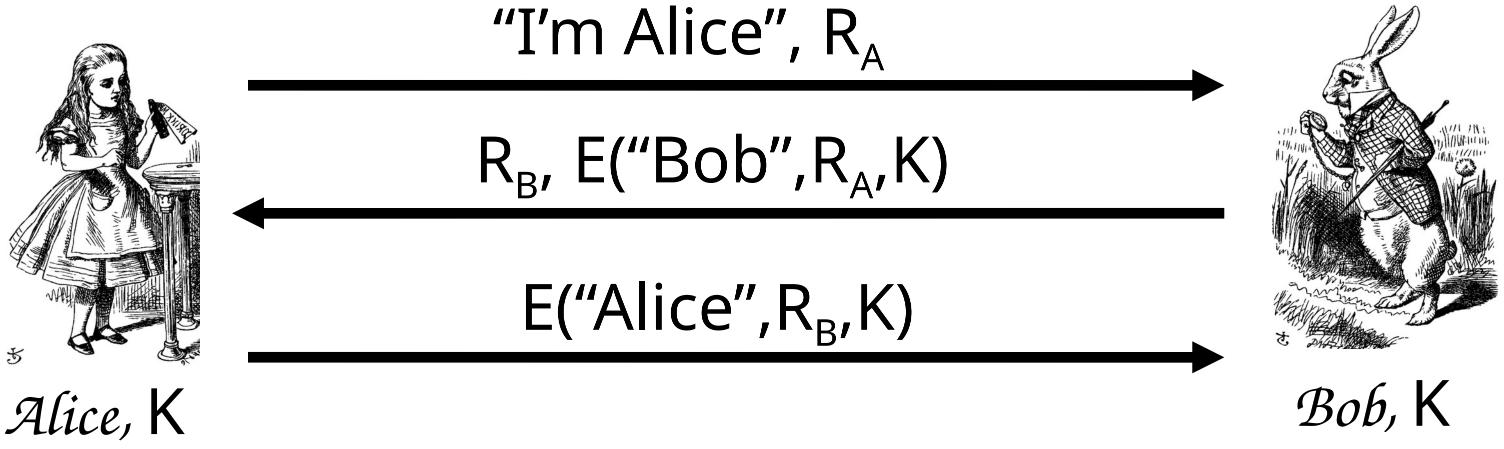




Mutual Authentication

- 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

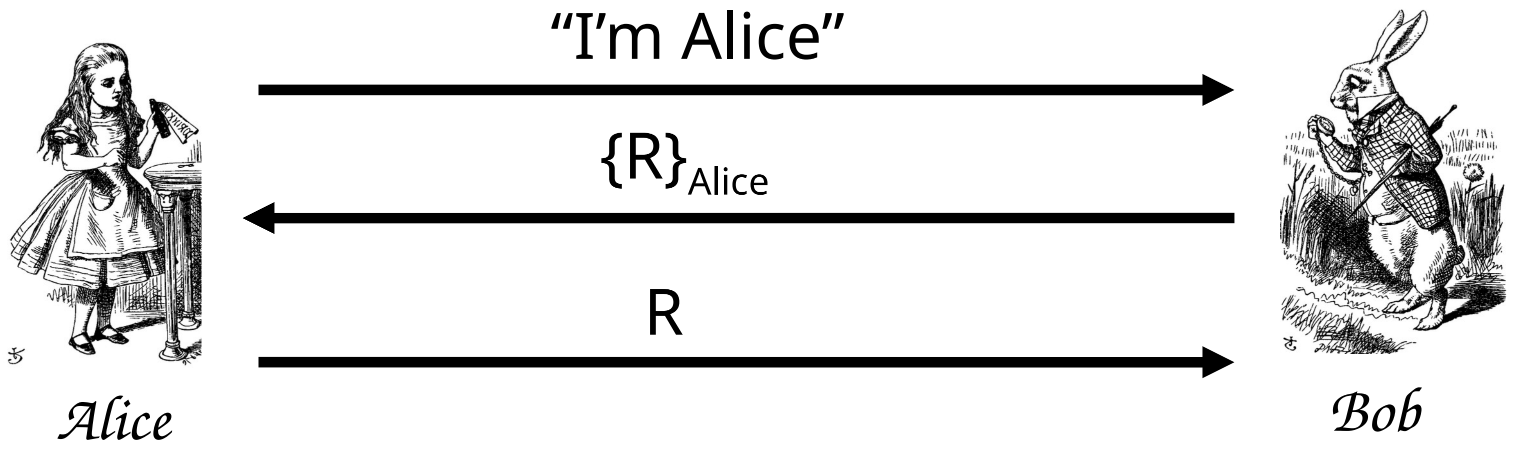


- Do these "insignificant" changes help?
- Yes!

Public Key Notation

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

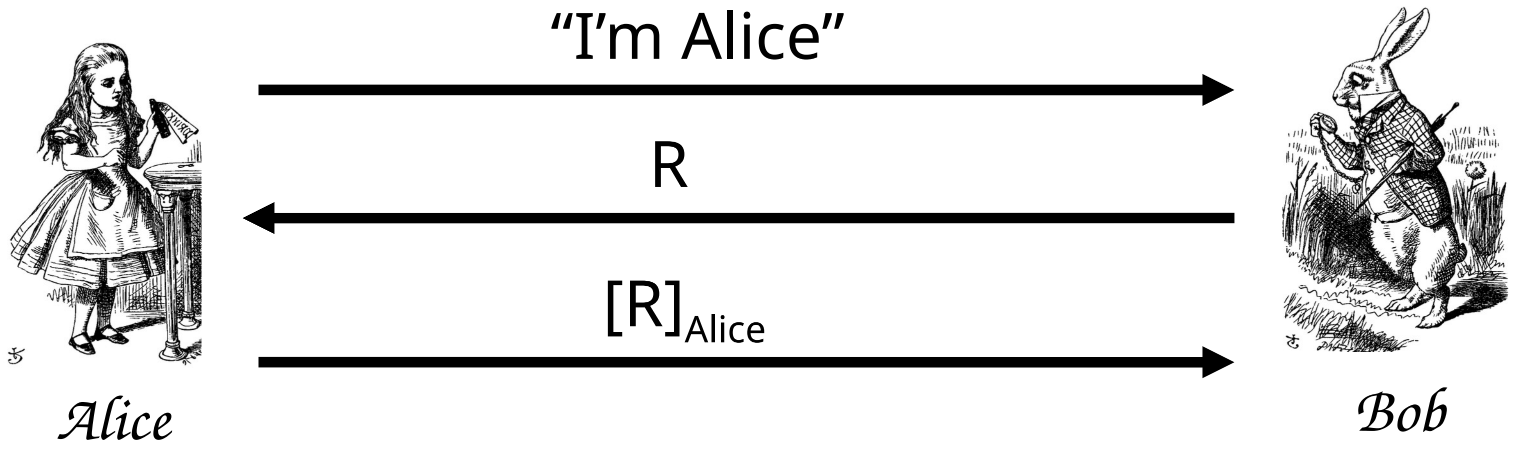
Public Key Authentication



- Is this secure?
- Trudy can get Alice to decrypt anything!

  Prevent this by having two key pairs

Public Key Authentication



- Is this secure?
- Trudy can get Alice to sign anything!
  - Same a previous   should have two key pairs

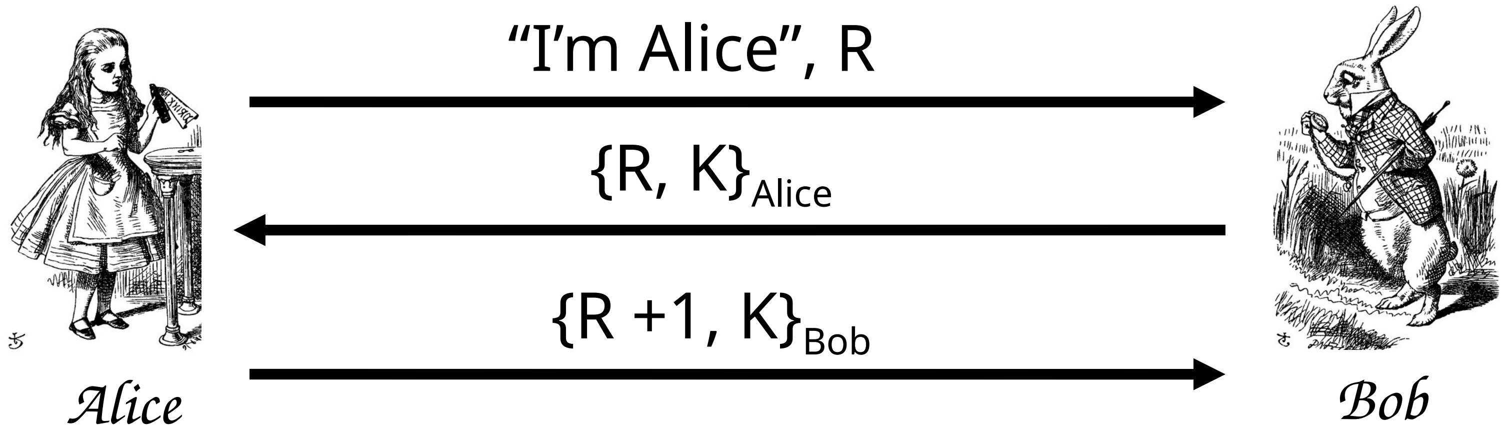
Public Keys

- 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

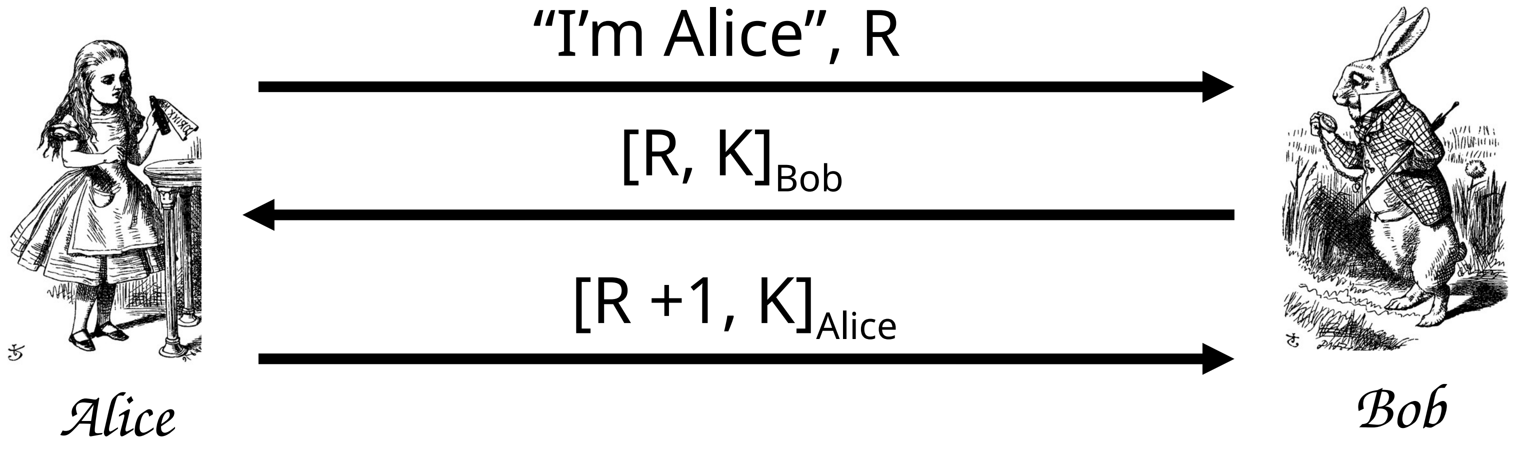
- Usually, a **session key** is required
  - A symmetric key for current session
  - Used for confidentiality and/or integrity
- How to authenticate **and** establish a session key (i.e., shared symmetric key)?
  - When authentication completed, Alice and Bob share a session key
  - Trudy cannot break the authentication…
  - …**and** Trudy cannot determine the session key

Authentication & Session Key



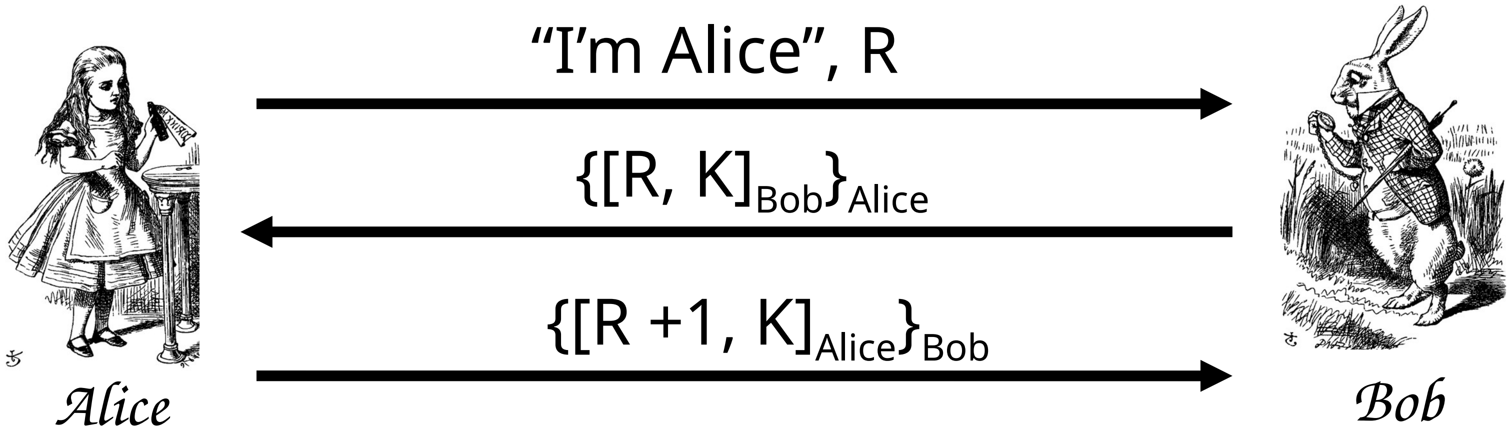
- Is this secure?
  - Alice is authenticated and session key is secure
  - Alice's "nonce", R, useless to authenticate Bob
  - The key K is acting as Bob's nonce to Alice
- No mutual authentication

Public Key Authentication and Session Key



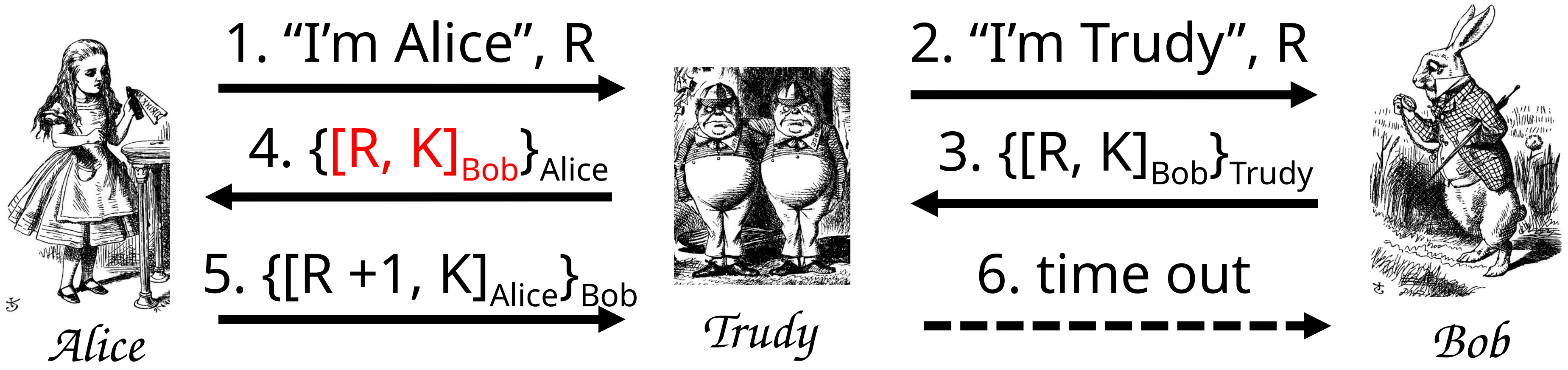
- Is this secure?
  - Mutual authentication (good), but…
  - … session key is not protected (very bad)

Public Key Authentication and Session Key


"I'm Alice", R
$\{[R, K]_{\text{Bob}}\}_{\text{Alice}}$
$\{[R+1, K]_{\text{Alice}}\}_{\text{Bob}}$
Alice
Bob

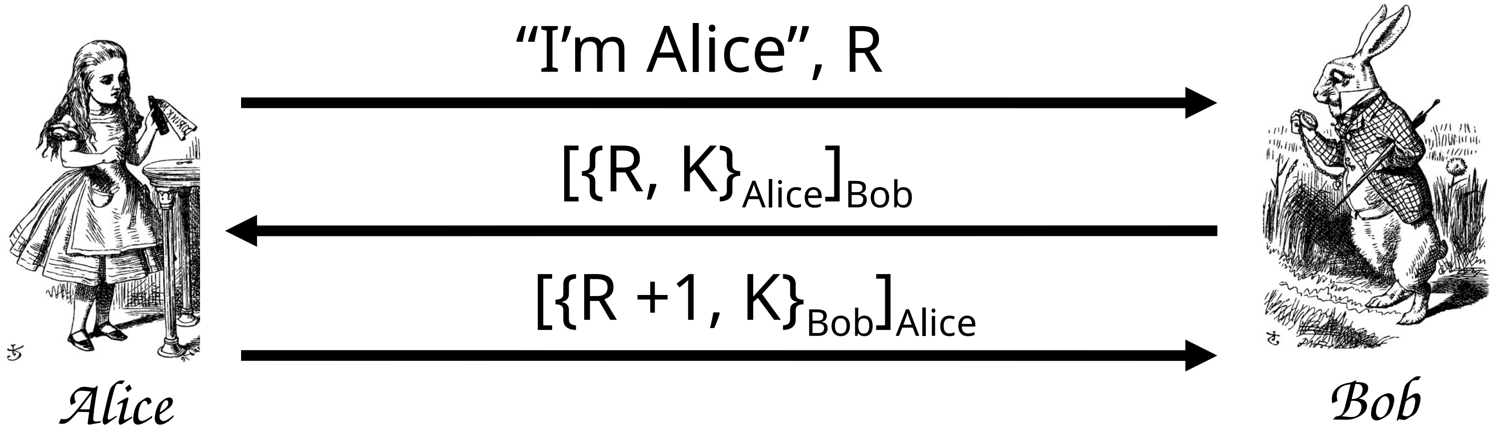
- *Is this secure?*
- *No! It's subject to subtle MiM attack*
    - *See the next slide…*

Public Key Authentication and Session Key


1. "I'm Alice", R
4. $\{[R, K]_{Bob}\}_{Alice}$
5. $\{[R+1, K]_{Alice}\}_{Bob}$
Alice
Trudy
2. "I'm Trudy", R
3. $\{[R, K]_{Bob}\}_{Trudy}$
6. time out
Bob

- *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

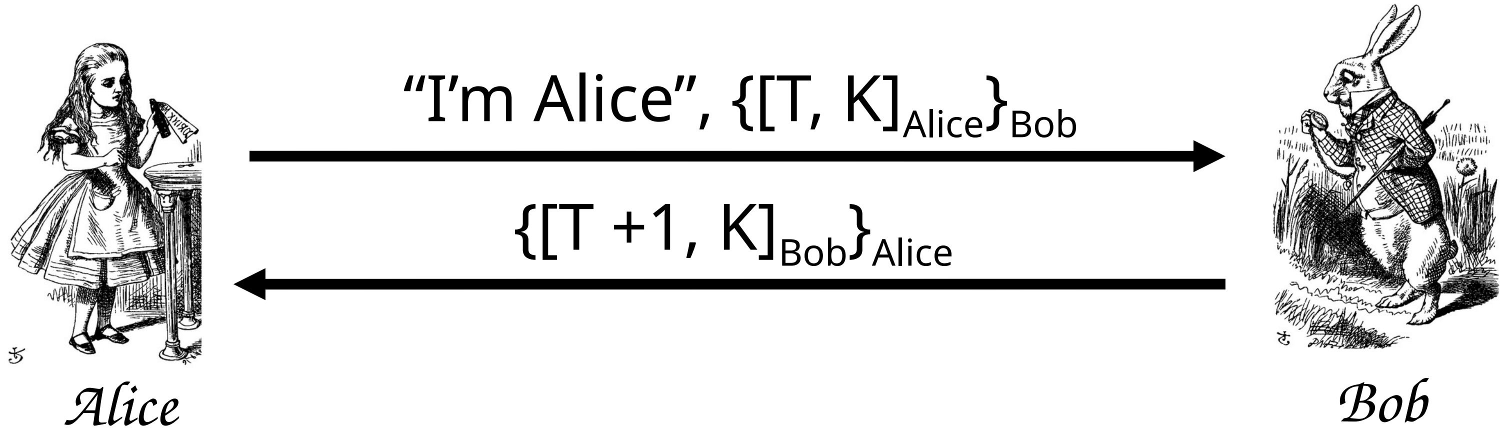

"I'm Alice", R
$[\{R, K\}_{\text{Alice}}]_{\text{Bob}}$
$[\{R+1, K\}_{\text{Bob}}]_{\text{Alice}}$
Alice
Bob

- Is this secure?
- Seems to be OK
  - Anyone can see $\{R, K\}_{\text{Alice}}$ and $\{R+1, K\}_{\text{Bob}}$

Timestamps

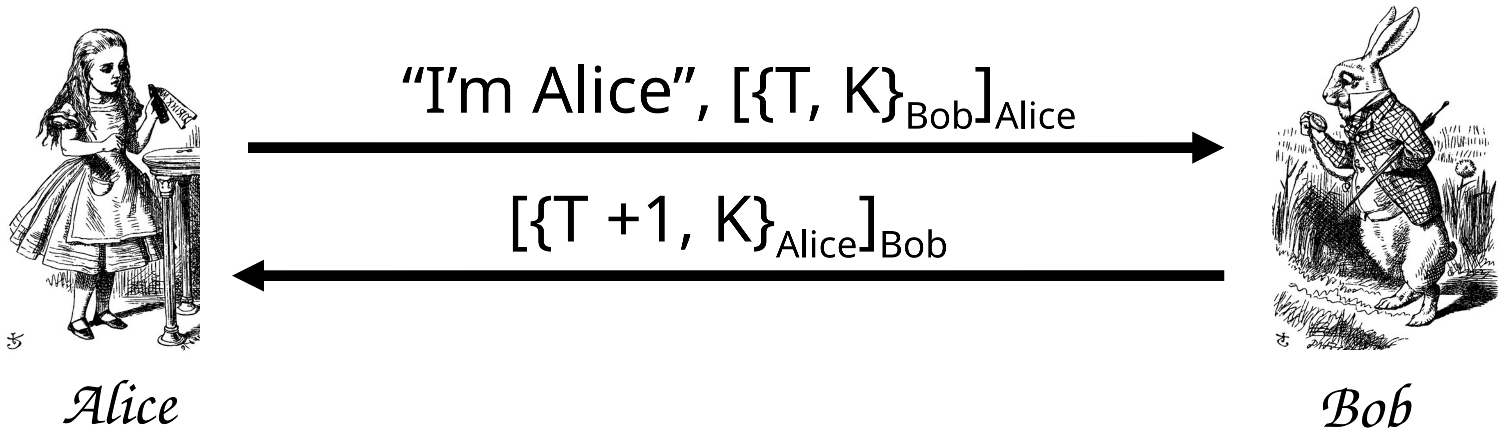
- A timestamp T is derived from current time
- Timestamps can be used to prevent replay
  - Used in Kerberos, for example
- Timestamps reduce number of msgs (good)
  - A challenge that both sides know in advance
- "Time" is a security-critical parameter (bad)
  - Clocks not same and/or network delays, so must allow for **clock skew** creates risk of replay
  - How much clock skew is enough?

Public Key Authentication with Timestamp T


"I'm Alice", $\{[T, K]_{Alice}\}_{Bob}$
$\{[T+1, K]_{Bob}\}_{Alice}$
Alice
Bob

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

Public Key Authentication with Timestamp T


"I'm Alice", $[\{T, K\}_{Bob}]_{Alice}$
$[\{T+1, K\}_{Alice}]_{Bob}$
Alice
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

"I'm Trudy", $[\{T, K\}_{Bob}]_{Trudy}$ $\longrightarrow$

$\longleftarrow$ $[\{T+1, K\}_{Trudy}]_{Bob}$

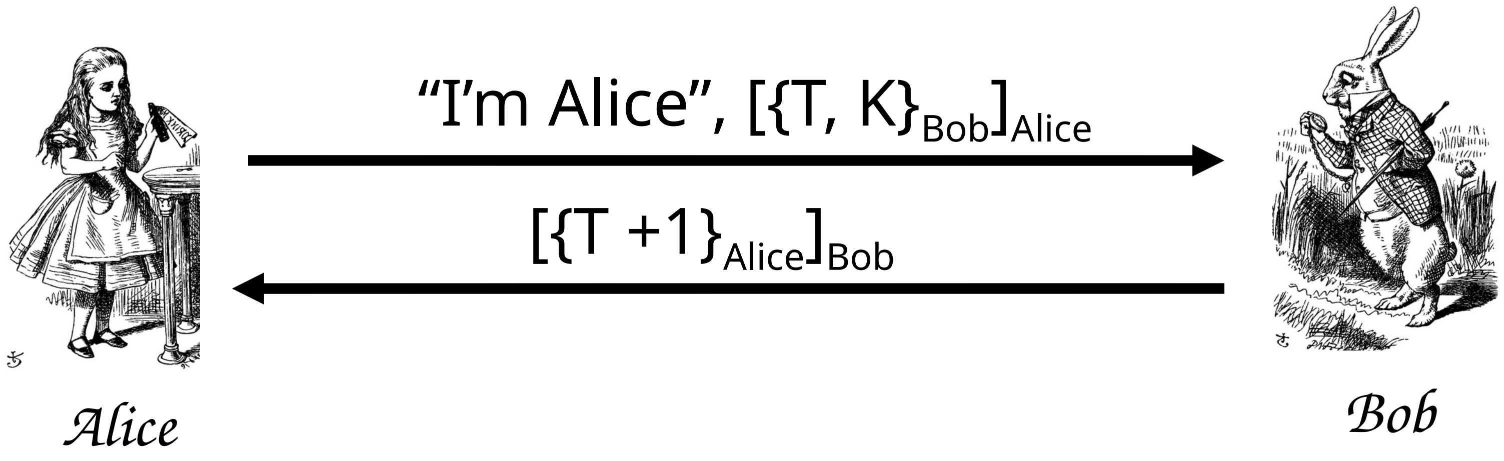
Bob

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

Public Key Authentication

- Sign and encrypt with nonce…
  - Insecure
- Encrypt and sign with nonce…
  - Secure
- Sign and encrypt with timestamp…
  - Secure
- Encrypt and sign with timestamp…
  - Insecure
- Protocols can be subtle!

Public Key Authentication with Timestamp T


"I'm Alice", $[\{T, K\}_{Bob}]_{Alice}$
$[\{T+1\}_{Alice}]_{Bob}$
Alice
Bob

- Is this "encrypt and sign" secure?
  - Yes, seems to be OK
- Does "sign and encrypt" also work here?

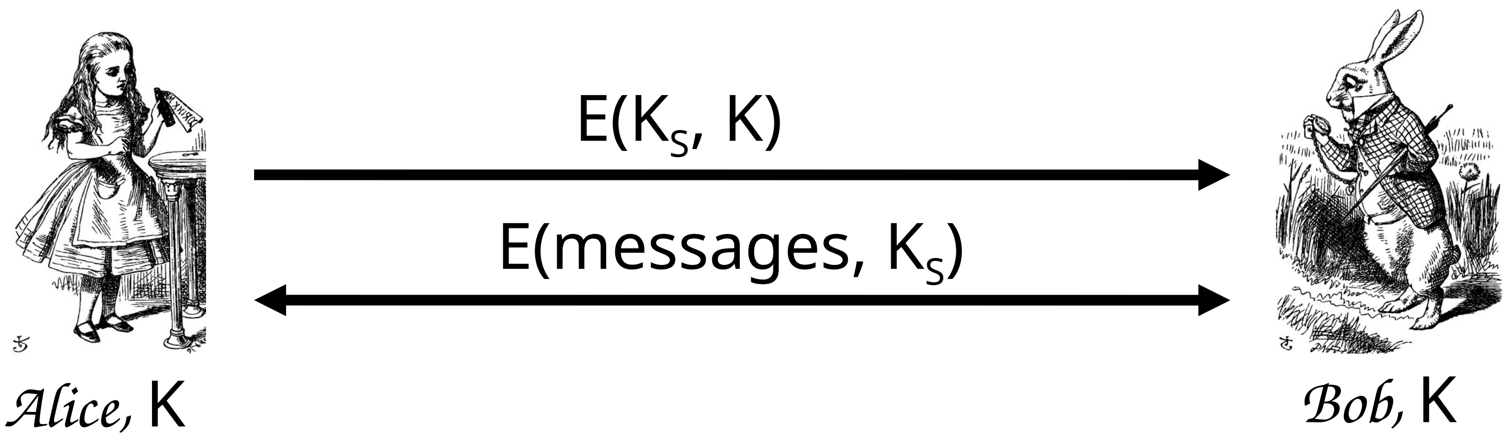
Perfect Forward Secrecy

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

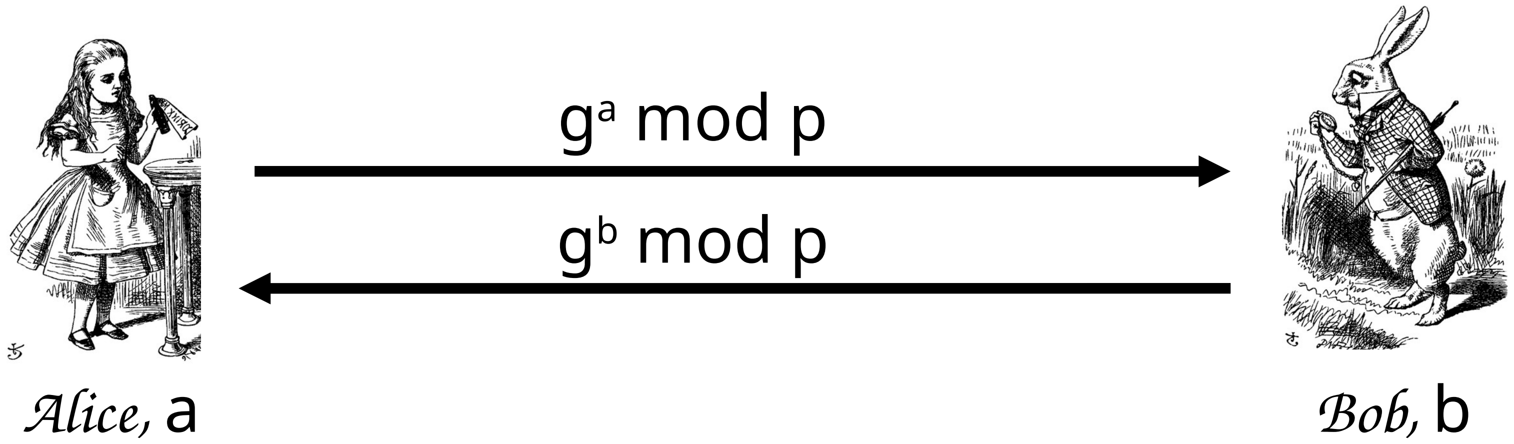
Naïve Session Key Protocol



- *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*

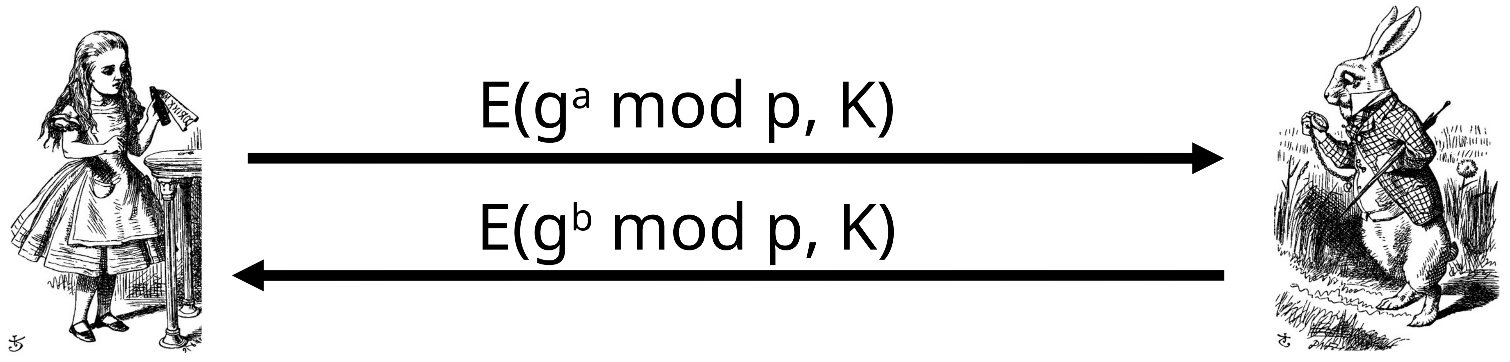
Perfect Forward Secrecy

- We can use **Diffie-Hellman** for PFS
- Recall: public g and p



- But Diffie-Hellman is subject to MiM
- How to get PFS and prevent MiM?

Perfect Forward Secrecy

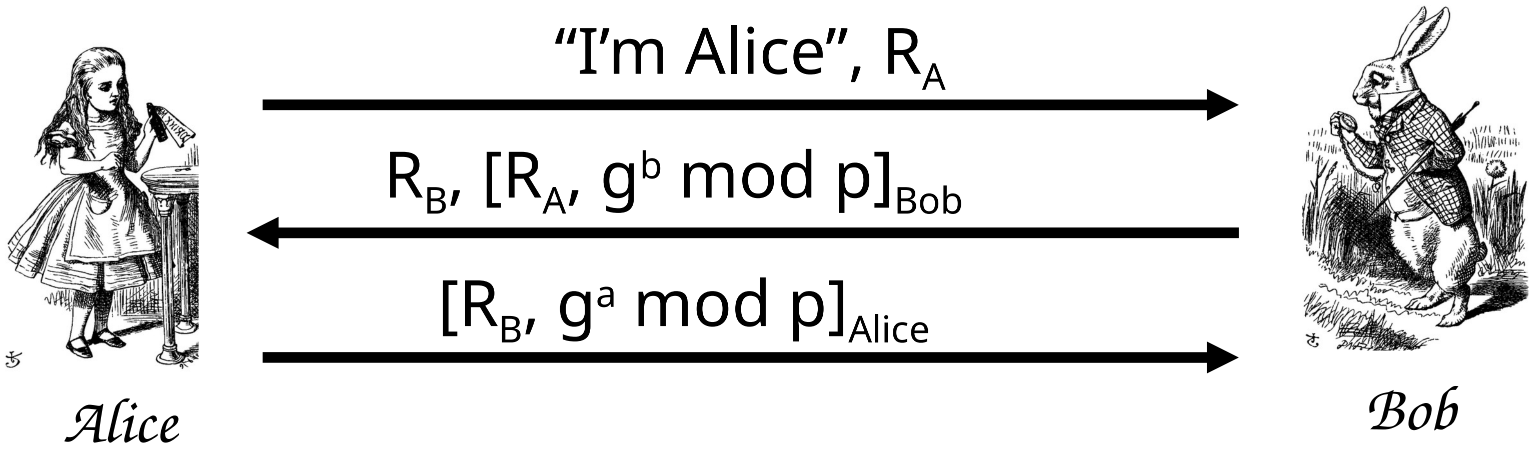


Alice: K, a

Bob: K, b

- *Session key* $K_S = g^{ab} \bmod p$
- *Alice* ***forgets*** a, *Bob* ***forgets*** b
- *This is known as* ***Ephemeral Diffie-Hellman***
- *Neither Alice nor Bob can later recover* K_S
- *Are there other ways to achieve PFS?*

Mutual Authentication, Session Key and PFS


"I'm Alice", R_A
R_B, $[R_A, g^b \bmod p]_{Bob}$
$[R_B, g^a \bmod p]_{Alice}$
Alice
Bob

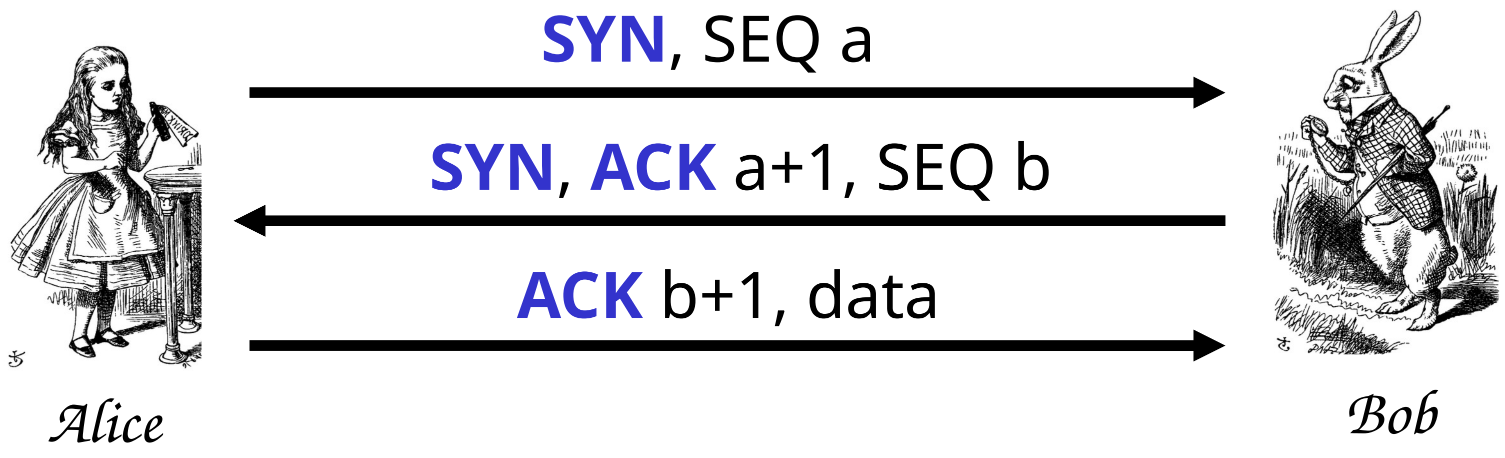
- Session key is $K = g^{ab} \bmod p$
- Alice forgets a and Bob forgets b
- If Trudy later gets Bob's and Alice's secrets, she cannot recover session key K

Authentication and TCP

TCP-based Authentication

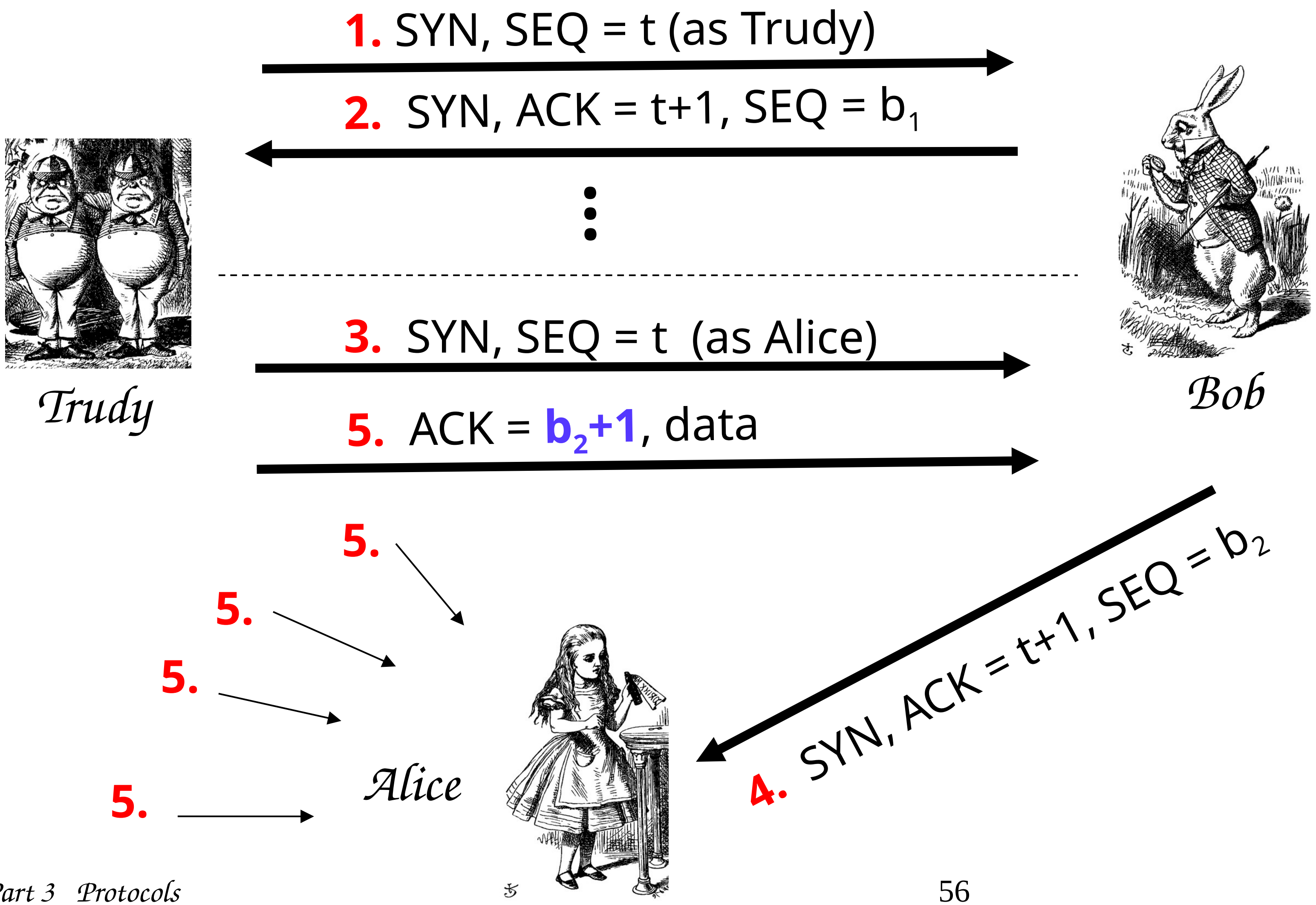
- TCP not intended for use as an authentication protocol
- But IP address in TCP connection may be (mis)used for authentication
- Also, one mode of IPSec relies on IP address for authentication

TCP 3-way Handshake

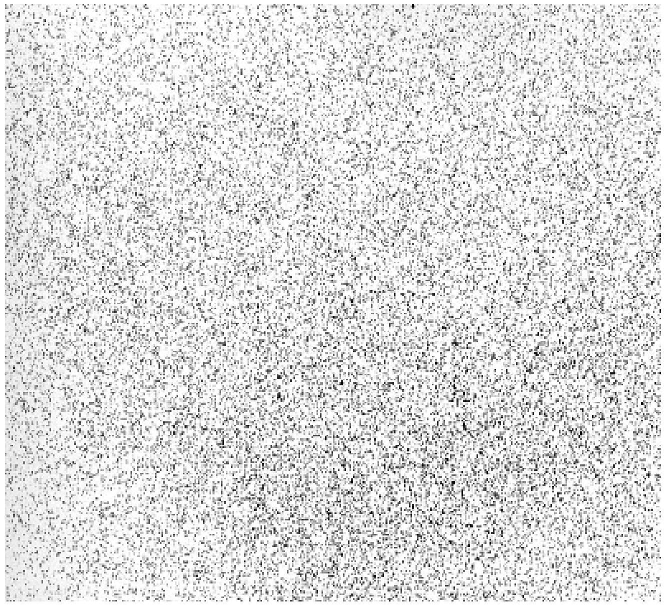


- *Initial sequence numbers:* SEQ a *and* SEQ b
  - *Supposed to be selected at random*
- *If not, might have problems…*

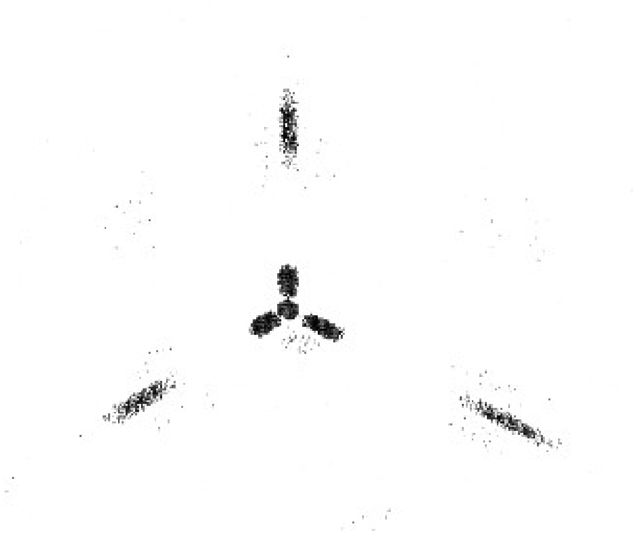
TCP Authentication Attack


1. SYN, SEQ = t (as Trudy)
2. SYN, ACK = t+1, SEQ = b₁
⋮
3. SYN, SEQ = t (as Alice)
5. ACK = b₂+1, data
4. SYN, ACK = t+1, SEQ = b₂
5.
5.
5.
5.
Trudy
Bob
Alice

TCP Authentication Attack



Random SEQ *numbers*



Initial SEQ numbers
Mac OS X

- *If initial* SEQ *numbers not very random…*
- *…possible to guess initial* SEQ *number…*
- *…and previous attack will succeed*

TCP Authentication Attack

- Trudy cannot see what Bob sends, but she can send packets to Bob, while posing as **Alice**
- Trudy must prevent Alice from receiving Bob's response (or else connection will terminate)
- If **password** (or other authentication) required, this attack fails
- If TCP connection is relied on for authentication, then attack might succeed
- **Bad idea** to rely on TCP for authentication

Best Authentication Protocol?

- It depends on…
  - The sensitivity of the application/data
  - The delay that is tolerable
  - The cost (computation) that is tolerable
  - What crypto is supported (public key, symmetric key, …)
  - Whether mutual authentication is required
  - Whether PFS, anonymity, etc., are concern
- …and possibly other factors

CAPTCHA

Turing Test

- Proposed by Alan Turing in 1950
- Human asks questions to a human and a computer, without seeing either
- If questioner cannot distinguish human from computer, computer passes
- This is the gold standard in AI
- No computer can pass this today
  - But some claim they are close to passing

CAPTCHA

- CAPTCHA
  - Completely Automated Public Turing test to tell Computers and Humans Apart
- Completely Automated test is generated and scored by a computer
- Public program and data are public
- Turing test to tell… humans can pass the test, but machines cannot
  - Also known as HIP == Human Interactive Proof
- Like an inverse Turing test (sort of…)

CAPTCHA Paradox?

- "…CAPTCHA is a program that can generate and grade tests that it itself cannot pass…"
- "…much like some professors…"
- Paradox computer creates and scores test that it itself cannot pass!
- CAPTCHA purpose?
  - Only humans get access (not bots/computers)
- So, CAPTCHA is for **access control**

CAPTCHA Uses?

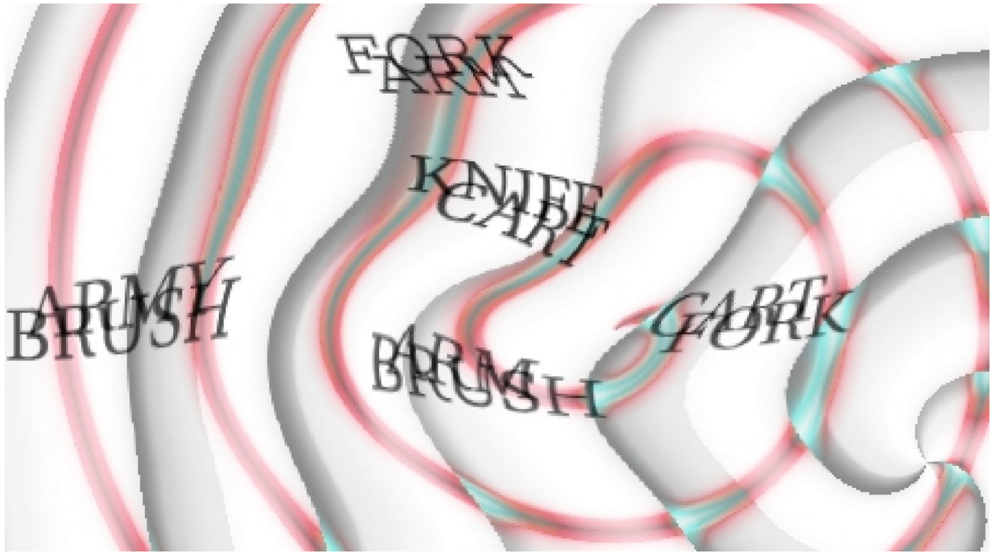
- Original motivation?
  - Automated bots stuffed ballot box in vote for best CS grad school
  - SJSU vs Stanford? No, it was MIT vs CMU
- Free email services   spammers like to use bots to sign up for 1000s of email accounts
  - CAPTCHA employed so only humans get accounts
- Sites that do not want to be automatically indexed by search engines
  - CAPTCHA would force human intervention

CAPTCHA: Rules of the Game

- *Easy for most humans to pass*
- *Difficult or impossible for machines to pass*
  - ***Even with access to CAPTCHA software***
- *From Trudy's perspective, the only unknown is a random number*
  - *Similar to Kerckhoffs' Principle*
- *Good to have different CAPTCHAs in case someone cannot pass one type*
  - *E.g., blind person could not pass visual CAPTCHA*

Do CAPTCHAs Exist?

- Test: Find 2 words in the following



- Easy for most humans
- A (difficult?) OCR problem for computer
  - OCR Optical Character Recognition

CAPTCHAs

- Current types of CAPTCHAs
  - Visual like previous example
  - Audio distorted words or music
- No text-based CAPTCHAs
  - Maybe this is impossible…

CAPTCHA's and AI

- OCR is a challenging AI problem
  - Hardest part is the **segmentation problem**
  - Humans good at solving this problem
- Distorted sound makes good CAPTCHA
  - Humans also good at solving this
- Hackers who break CAPTCHA have solved a hard AI problem (such as OCR)
  - So, putting hacker's effort to good use!
- Other ways to defeat CAPTCHAs???

Coming Protocols Attraction

- Generic authentication protocols
  - Protocols are subtle!
- SSH
- SSL
- IPSec
- Kerberos
- Wireless: GSM, WEP, WPA, WPA2, WPA3