PROTOCOL

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, GSM, and IPSec
- Implementation errors can also occur
  - Old 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!

Secure Entry to NSA

1. Insert badge into reader
2. Enter PIN
3. Correct PIN?

Yes? Enter

No? Get shot by security guard

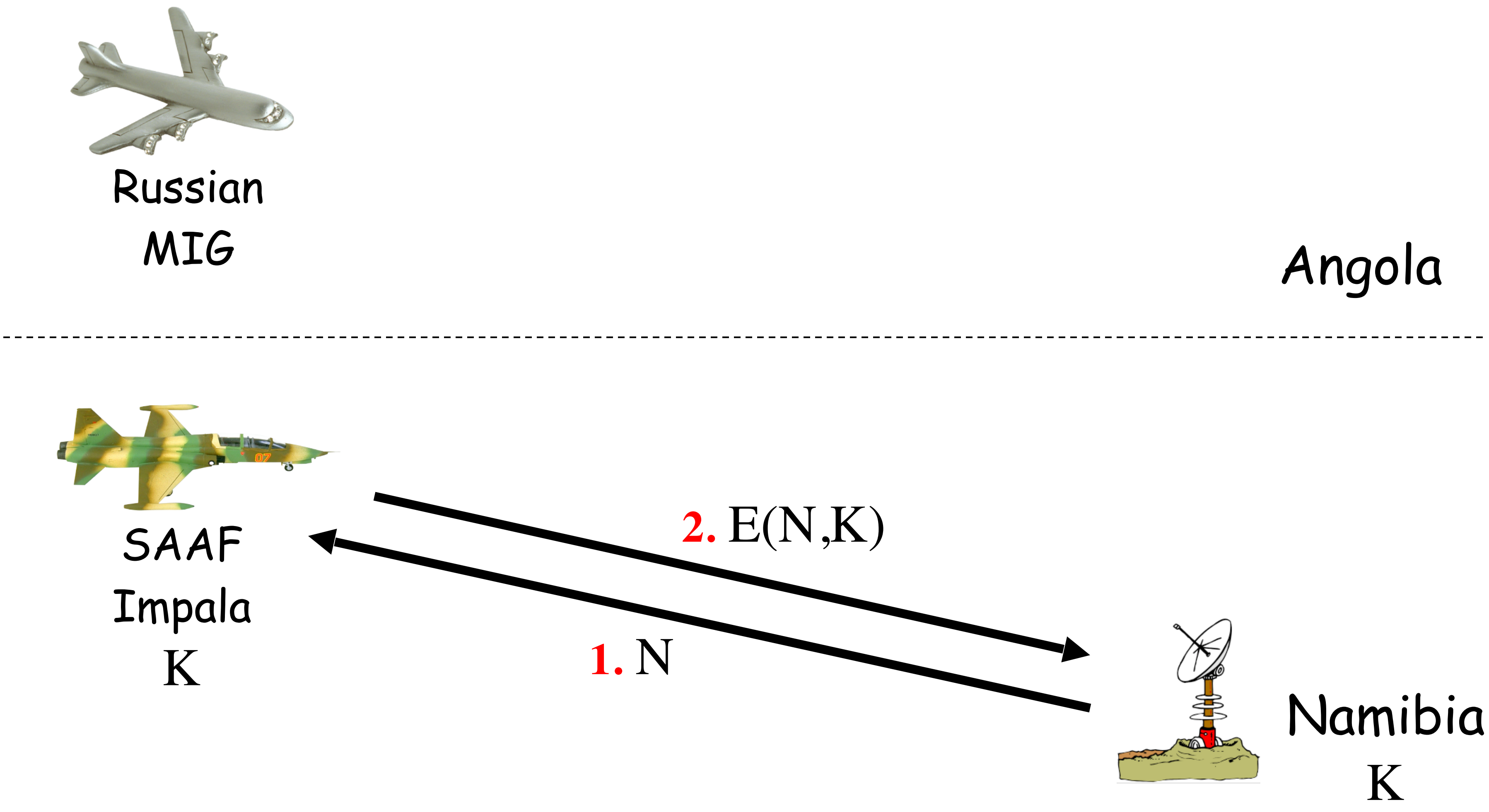
ATM Machine Protocol

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)



Russian MIG

Angola

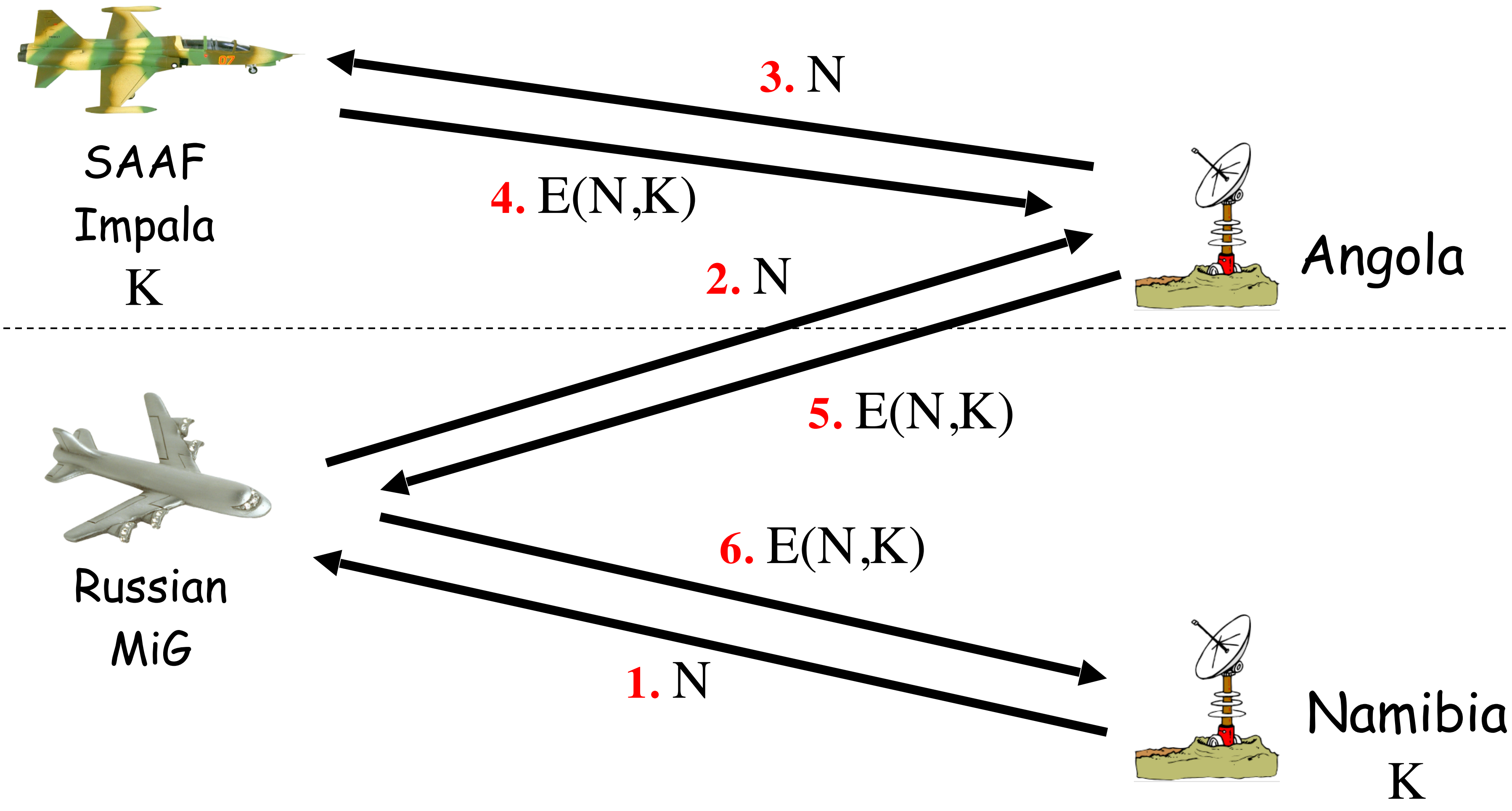
SAAF Impala K

2. E(N,K)

1. N

Namibia K

MIG in the Middle



SAAF
Impala
K

3. N

4. E(N,K)

2. N

Angola

5. E(N,K)

Russian
MiG

6. E(N,K)

1. N

Namibia
K

Authentication Protocols

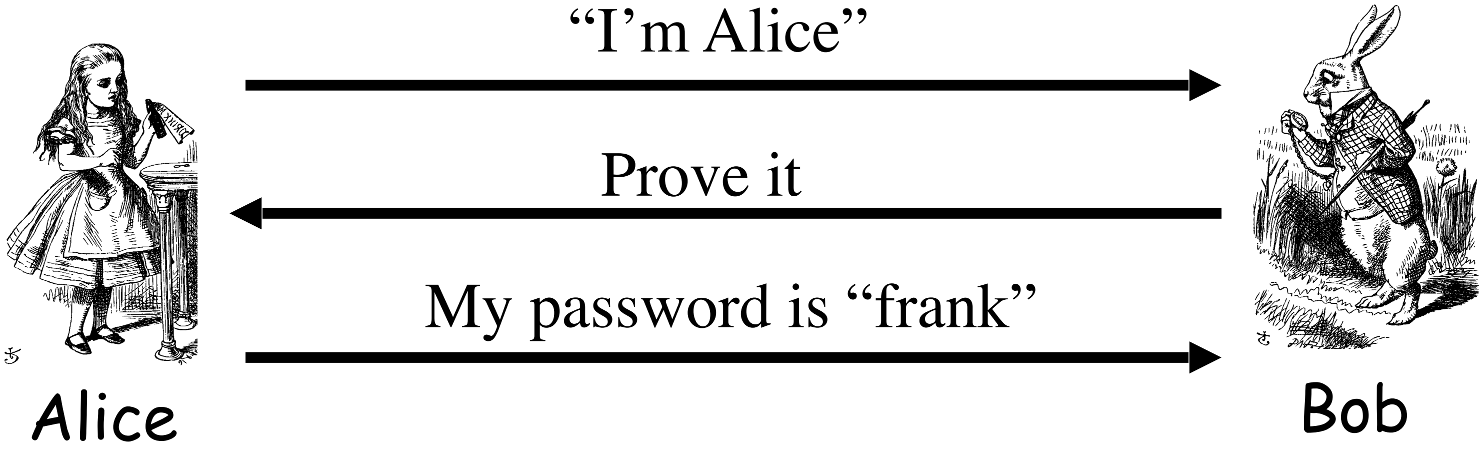
Authentication

- 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
    - Hash password with 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



"I'm Alice"

Prove it

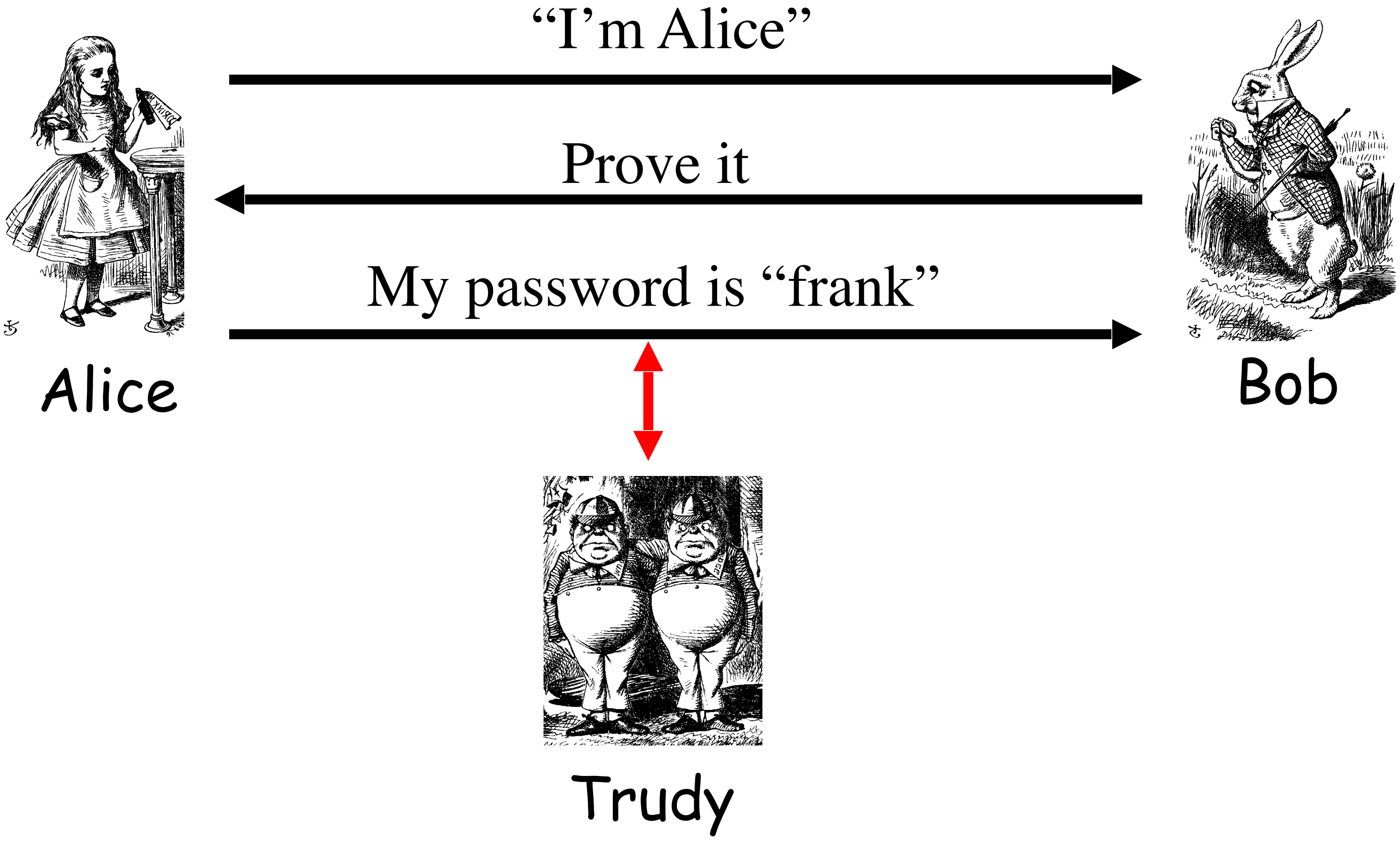
My password is "frank"

Alice

Bob

- 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



“I’m Alice”

Prove it

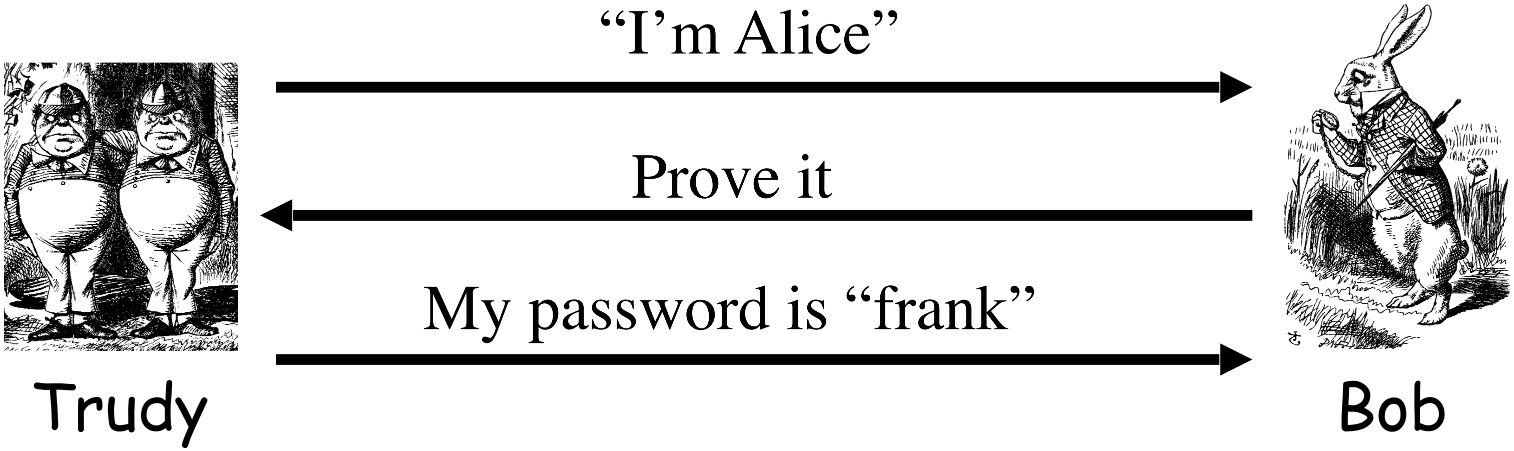
My password is “frank”

Alice

Bob

Trudy

Authentication Attack



“I’m Alice”

Prove it

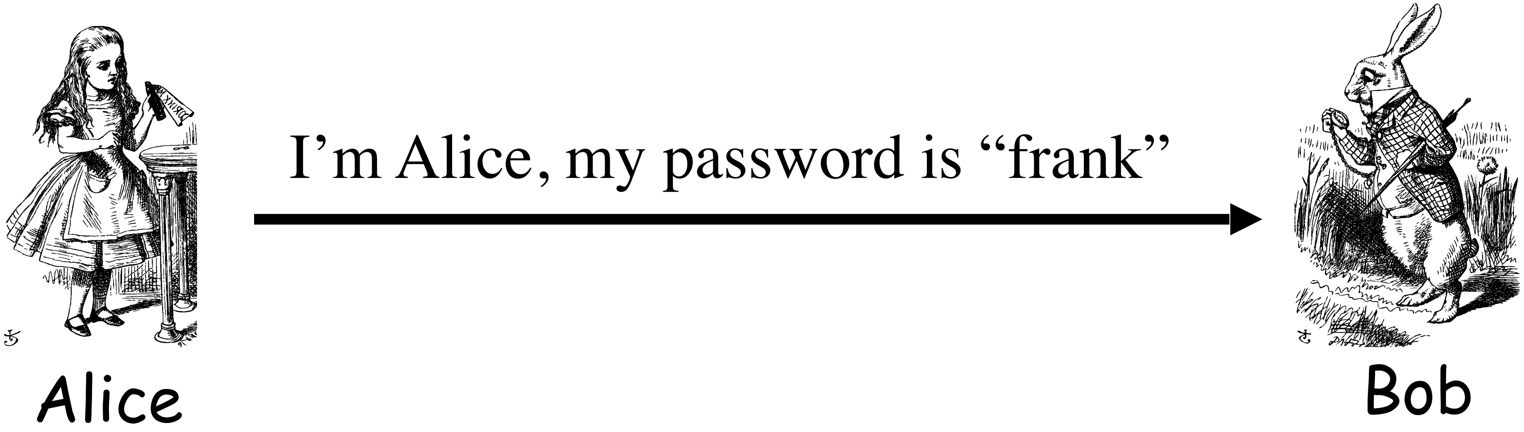
My password is “frank”

Trudy

Bob

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

Simple Authentication



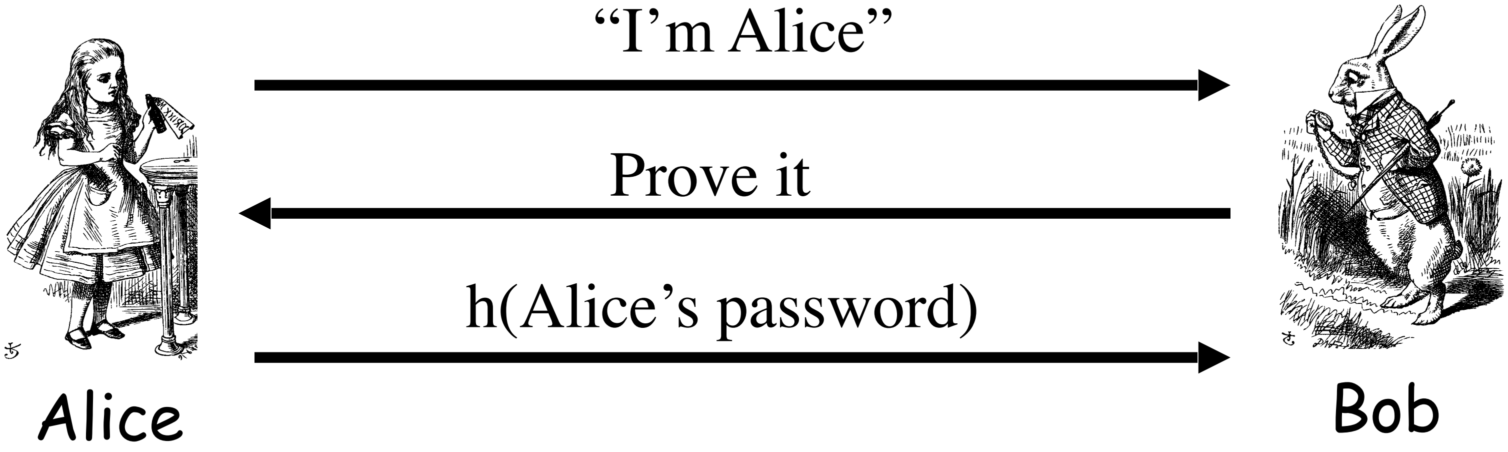
I’m Alice, my password is “frank”

Alice

Bob

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

Better Authentication



“I’m Alice”

Prove it

h(Alice’s password)

Alice

Bob

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

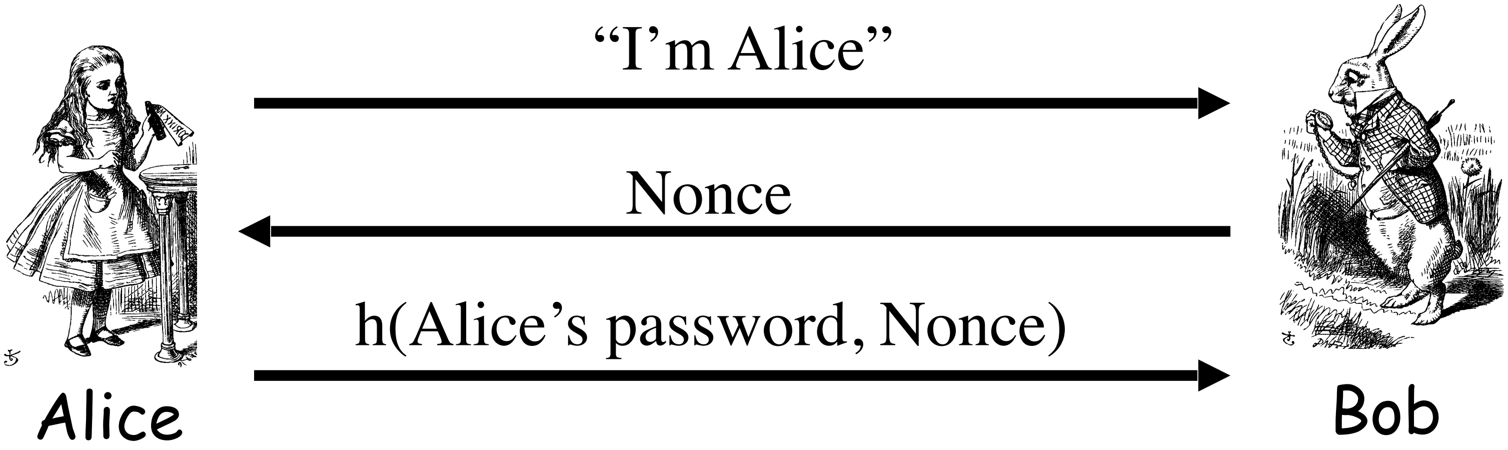
Challenge-Response

- 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



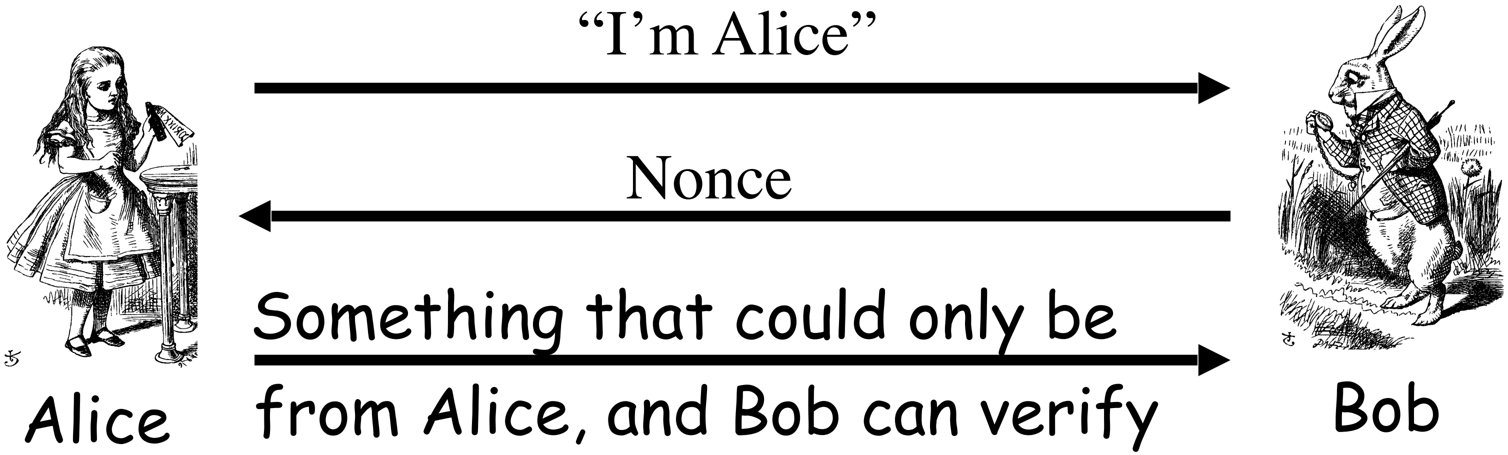
“I’m Alice”

Nonce

h(Alice’s password, Nonce)

- 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



"I'm Alice"

Nonce

Something that could only be from Alice, and Bob can verify

Alice

Bob

- 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

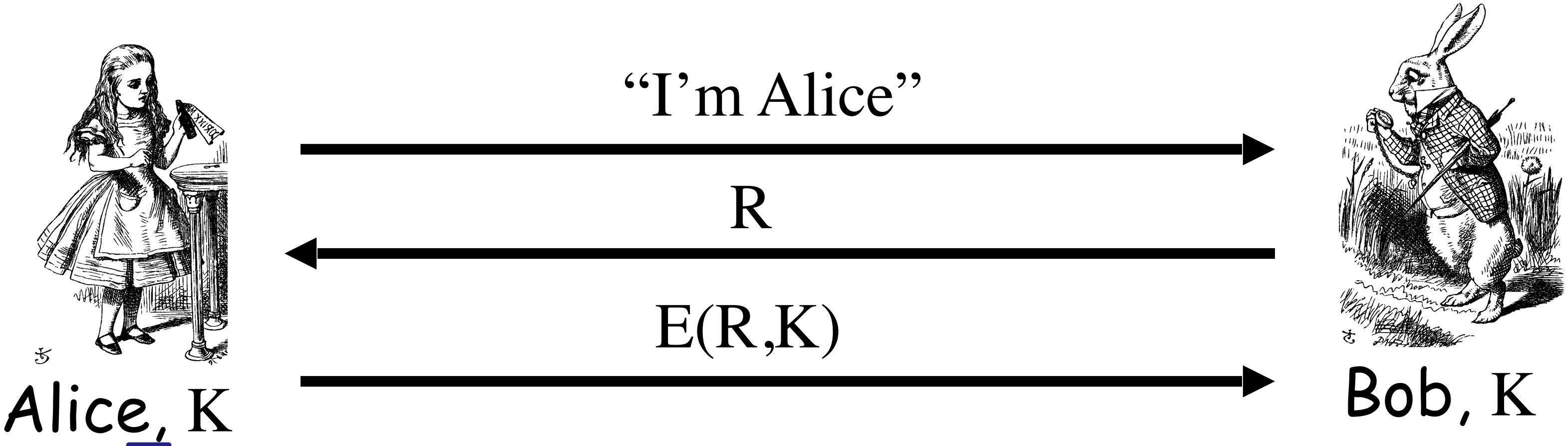
  $P = D(C,K)$

- 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



“I’m Alice”

R

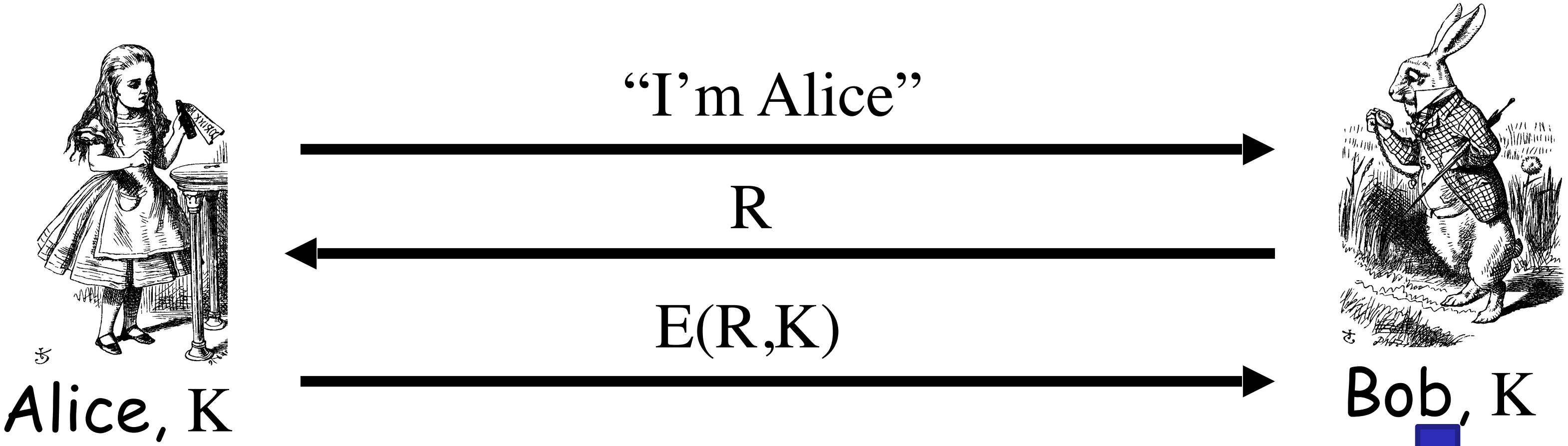
E(R,K)

Alice, K

Bob, K

Alice has to use her key K to encrypt the value R.
She is “proving herself”, because she and Bob are the only ones that can use that key!

Authenticate Alice Using Symmetric Key



"I'm Alice"

R

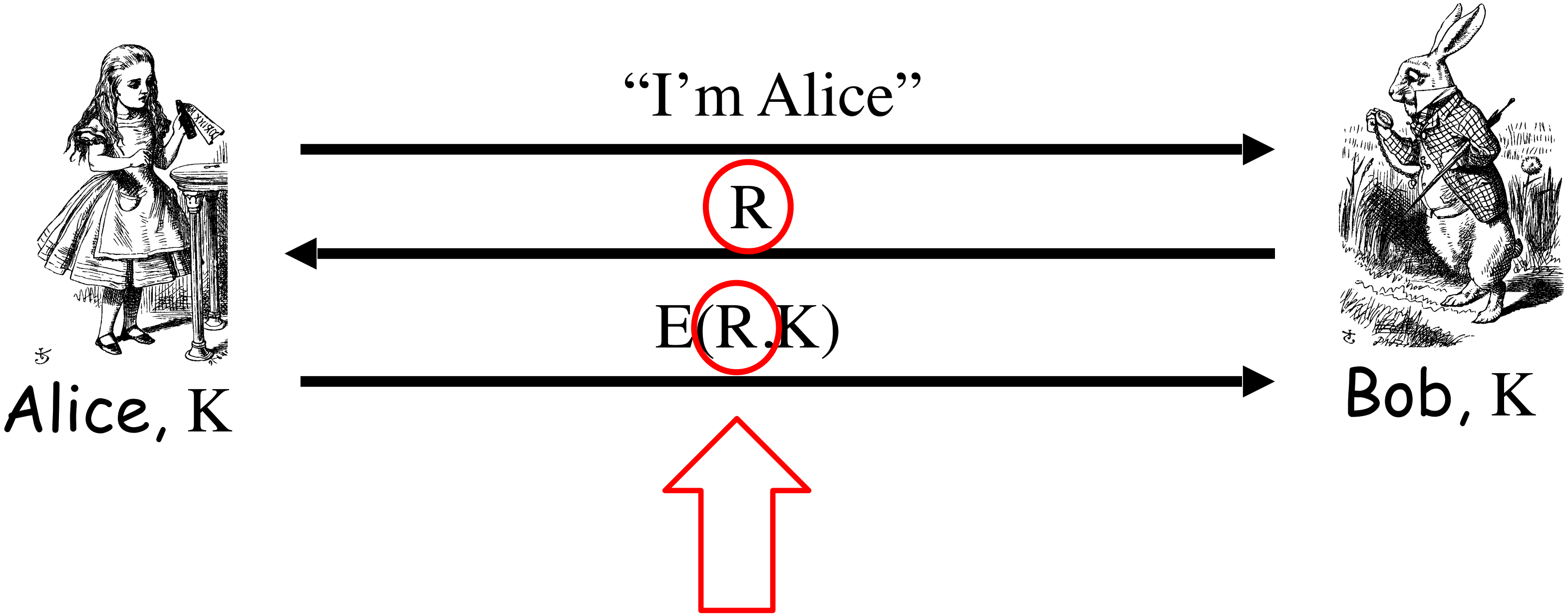
E(R,K)

Alice, K

Bob, K

Bob can verify that the message has been encrypted using the key K, and because the only person, except him, to be able to use it is Alice, he knows that is talking to

Authenticate Alice Using Symmetric Key



“I’m Alice”

R

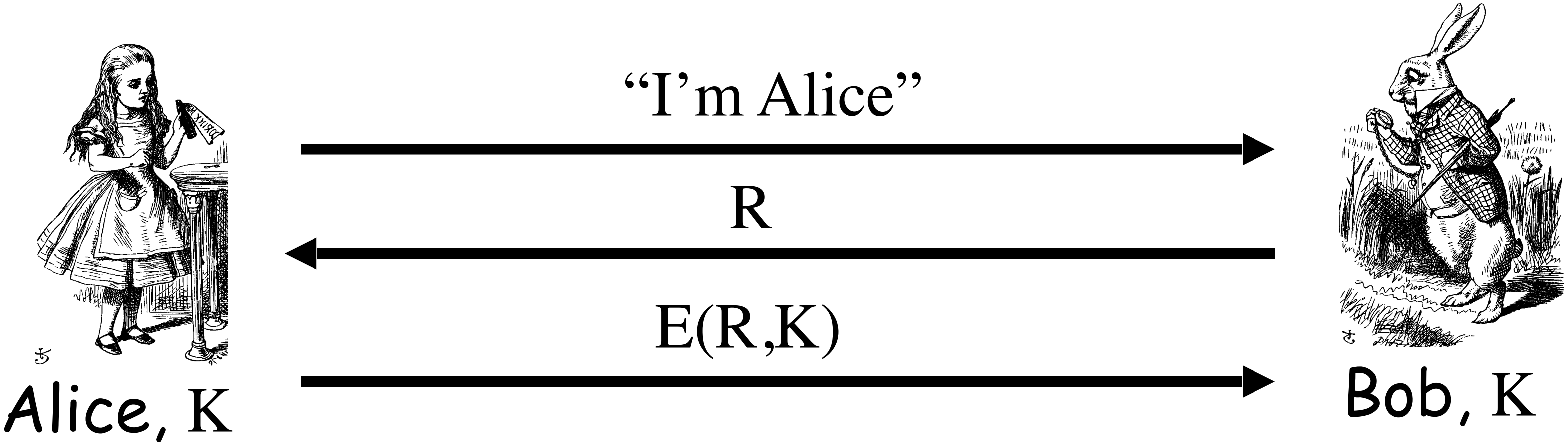
E(R.K)

Alice, K

Bob, K

The third message is Fresh. Because can exists only after THIS exact second message.

Authenticate Alice Using Symmetric Key



“I’m Alice”

R

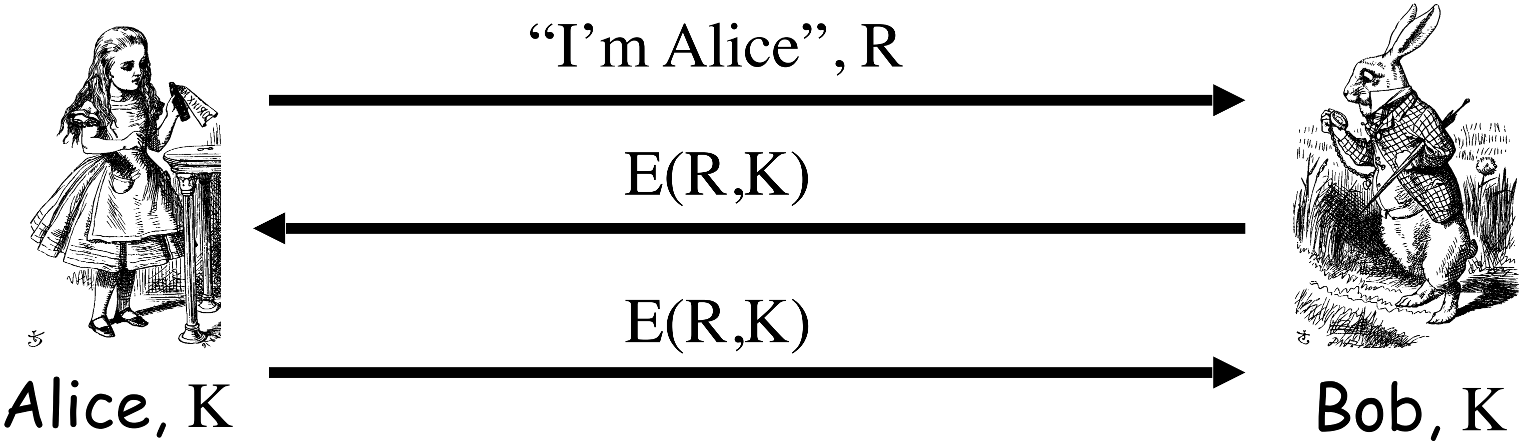
E(R,K)

Alice, K

Bob, K

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

Mutual Authentication?



“I’m Alice”, R

E(R,K)

E(R,K)

Alice, K

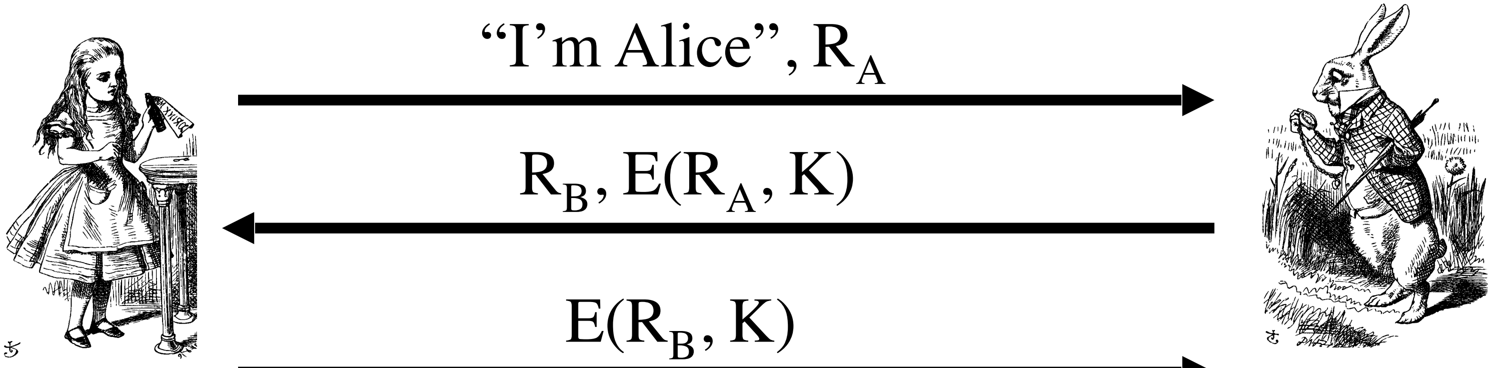
Bob, K

- 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



"I'm Alice", R_A

R_B, $E(R_A, K)$

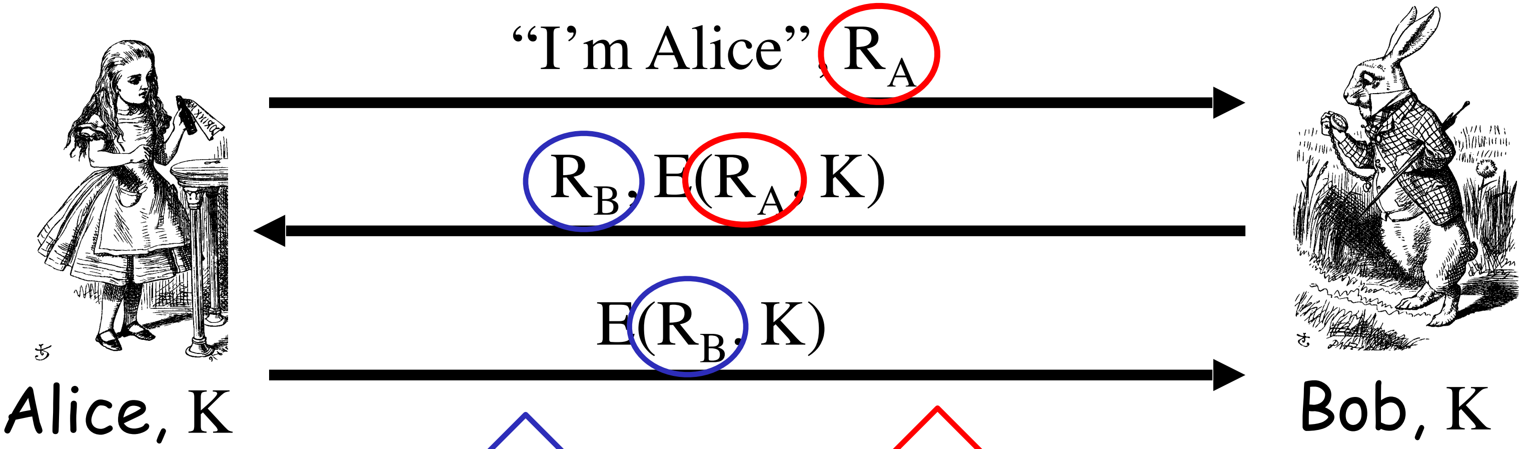
$E(R_B, K)$

Alice, K

Bob, K

- This provides mutual authentication…
- …or does it? See the next slide

Mutual Authentication



“I’m Alice”, R_A

R_B, $E(R_A, K)$

$E(R_B, K)$

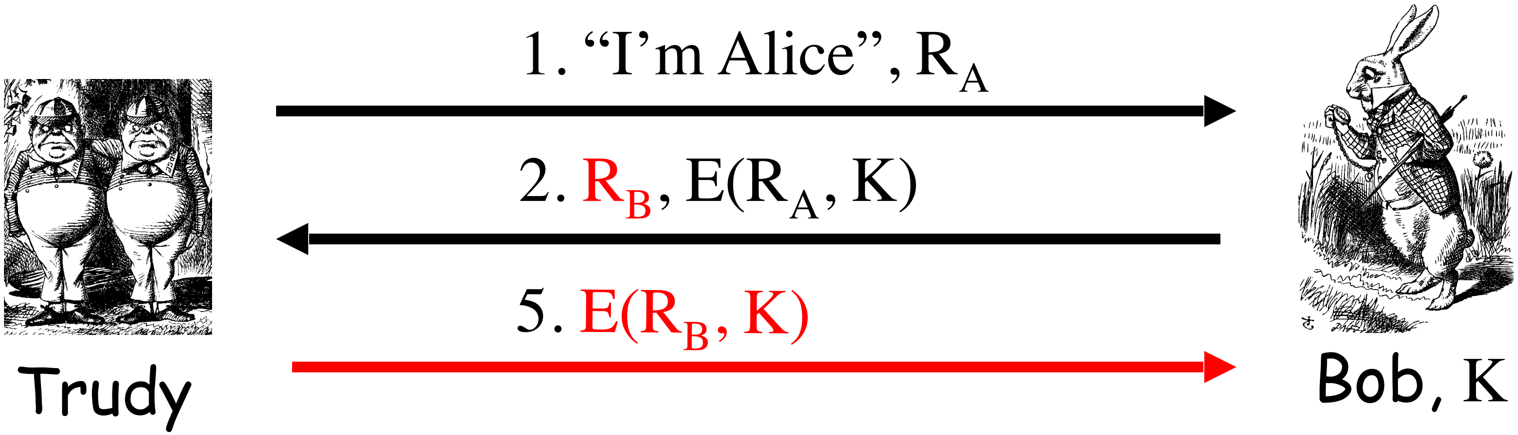
Alice, K

Bob, K

The 3nd message is Fresh. Because can exists only after THIS exact 2nd one.

The 2nd message is Fresh. Because can exists only after THIS exact 1st one.

Mutual Authentication Attack



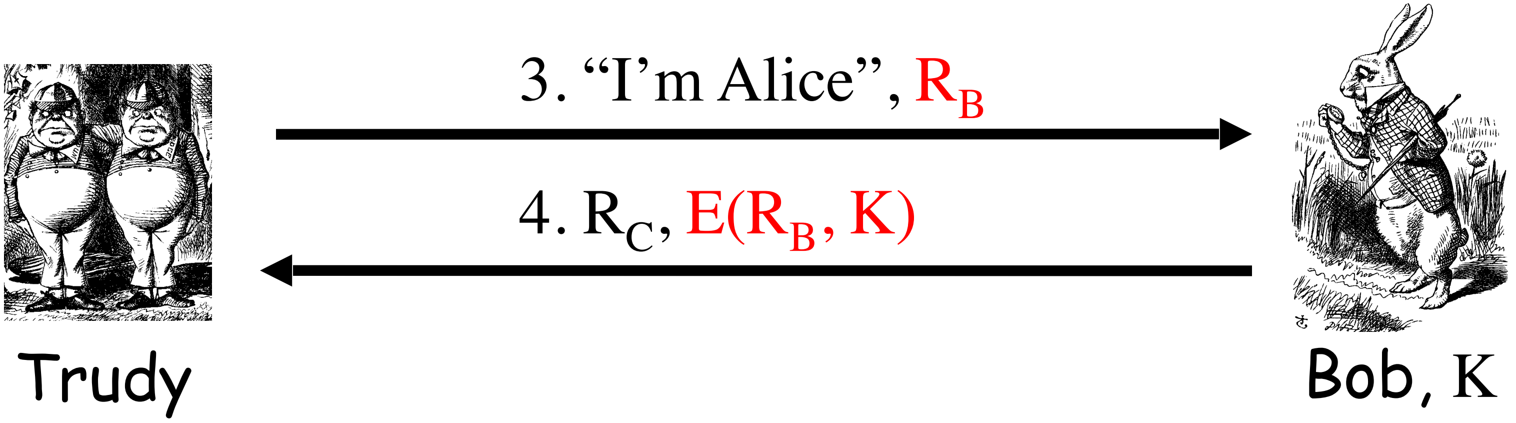
Trudy

1. "I'm Alice", R_A

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

5. $E(R_B, K)$

Bob, K



Trudy

3. "I'm Alice", R_B

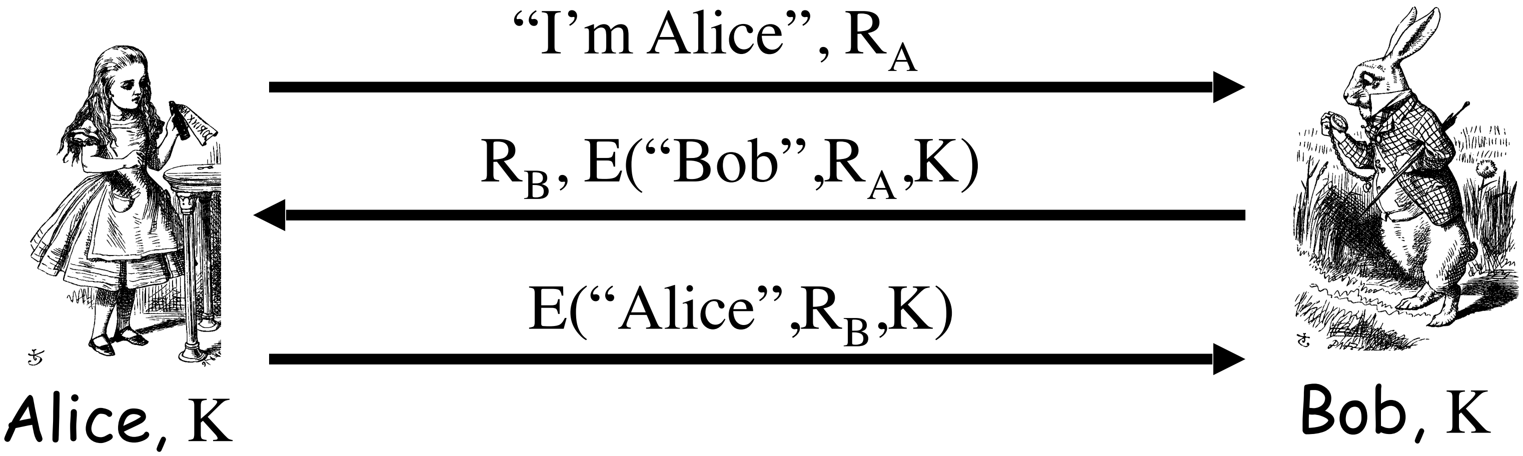
4. R_C, $E(R_B, K)$

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



“I’m Alice”, R_A

R_B, E(“Bob”,R_A,K)

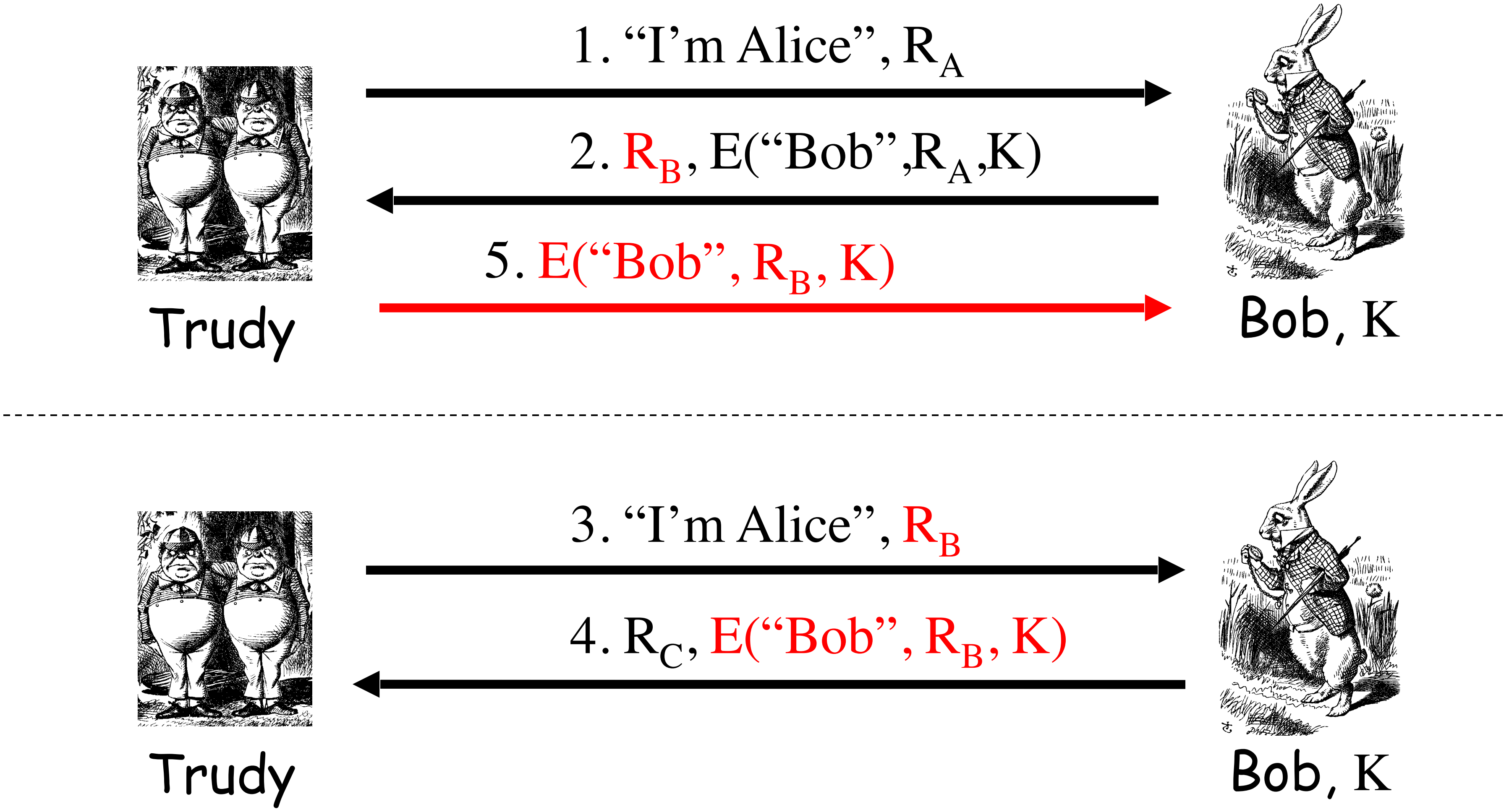
E(“Alice”,R_B,K)

Alice, K

Bob, K

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

Mutual Authentication Attack



1. “I’m Alice”, R_A

2. R_B, E(“Bob”,R_A,K)

5. E(“Bob”, R_B, K)

Trudy

Bob, K

3. “I’m Alice”, R_B

4. R_C, E(“Bob”, R_B, K)

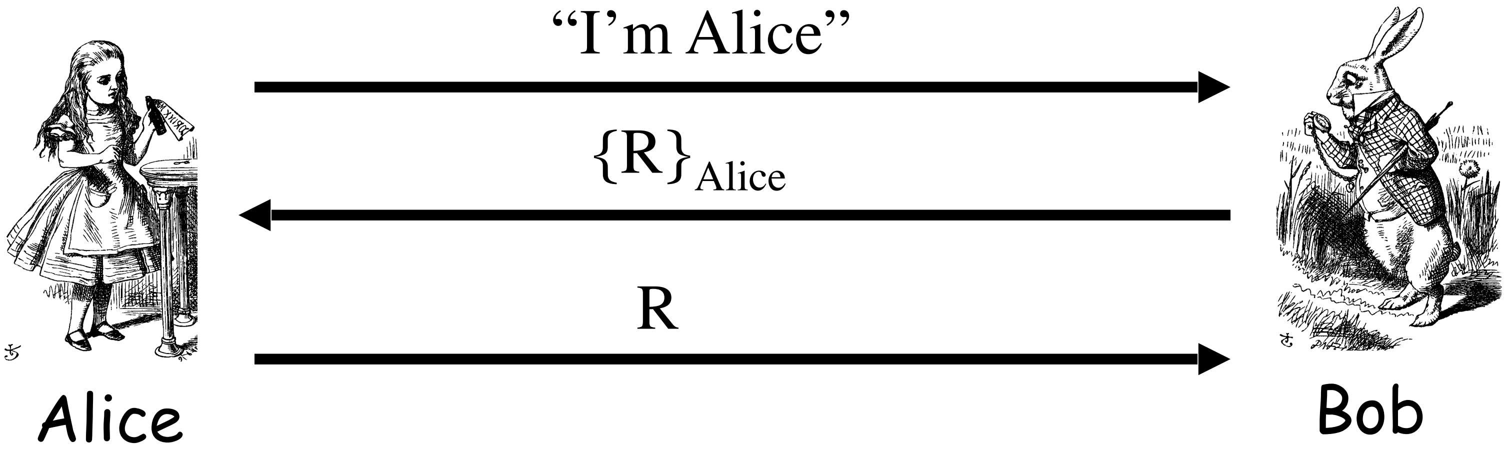
Trudy

Bob, K

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



"I'm Alice"

$\{R\}_{\text{Alice}}$

R

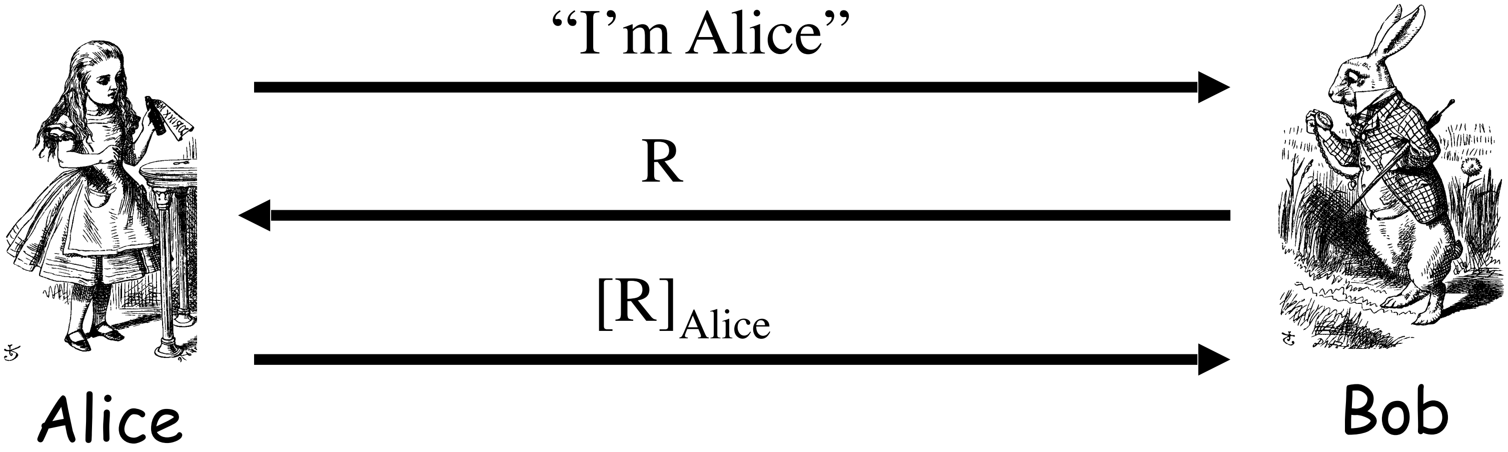
Alice

Bob

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

  Prevent this by having two key pairs

Public Key Authentication



"I'm Alice"

R

$[R]_{\text{Alice}}$

Alice

Bob

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

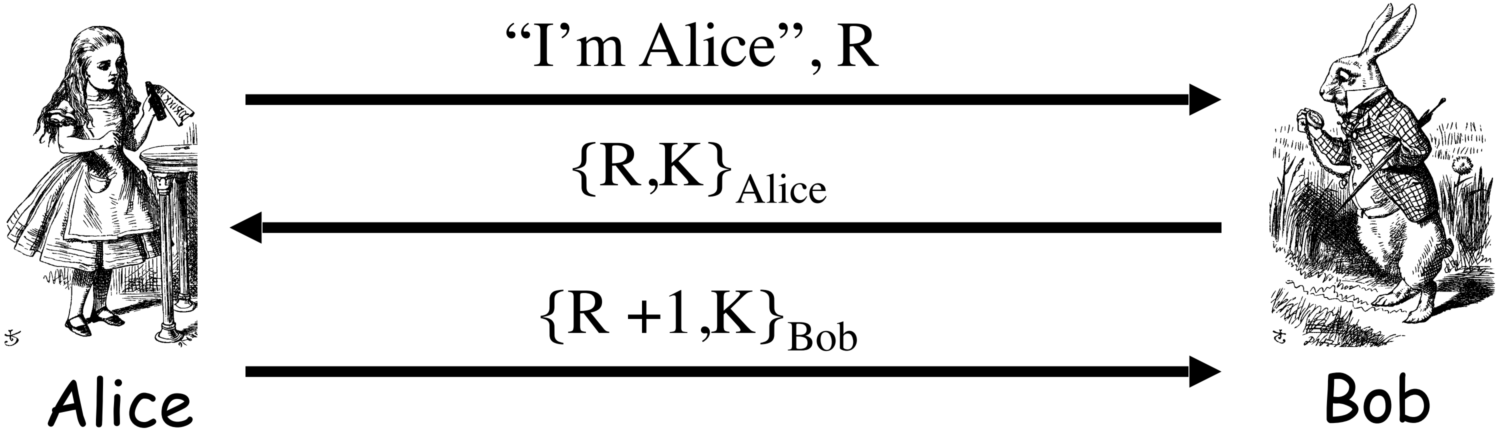
Public Keys

- 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
  - I.e., 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



"I'm Alice", R

$\{R,K\}_{\text{Alice}}$

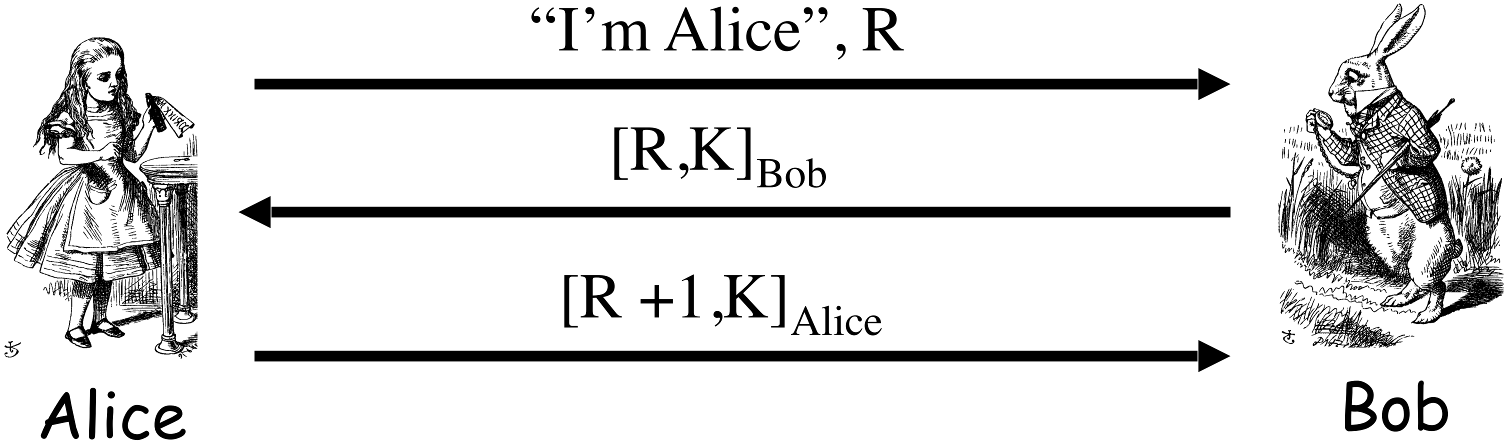
$\{R+1,K\}_{\text{Bob}}$

Alice

Bob

- 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



"I'm Alice", R

$[R,K]_{Bob}$

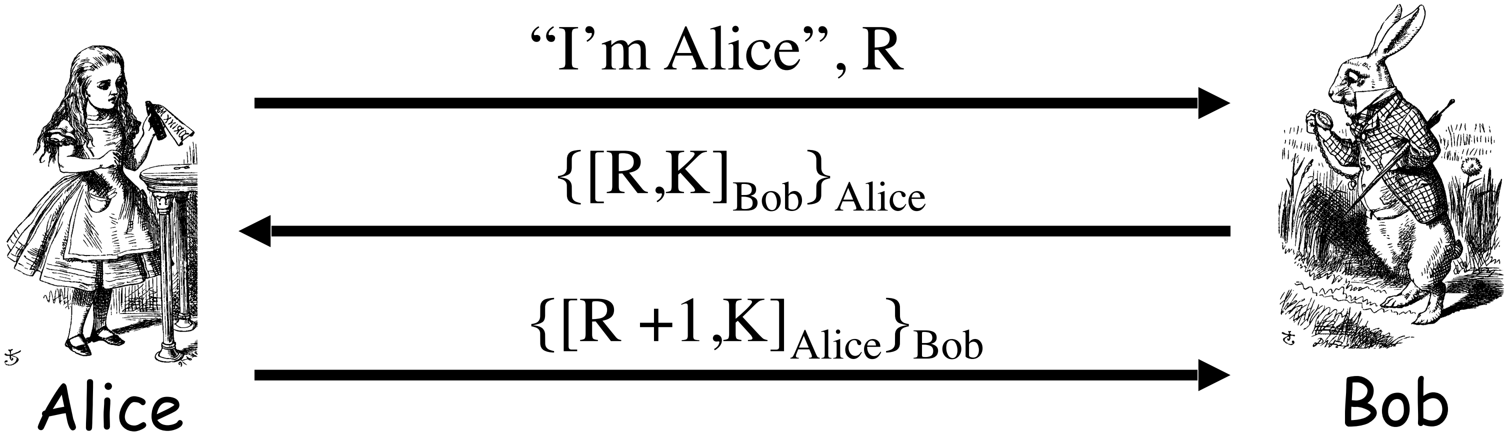
$[R+1,K]_{Alice}$

Alice

Bob

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

Public Key Authentication and Session Key



“I’m Alice”, R

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

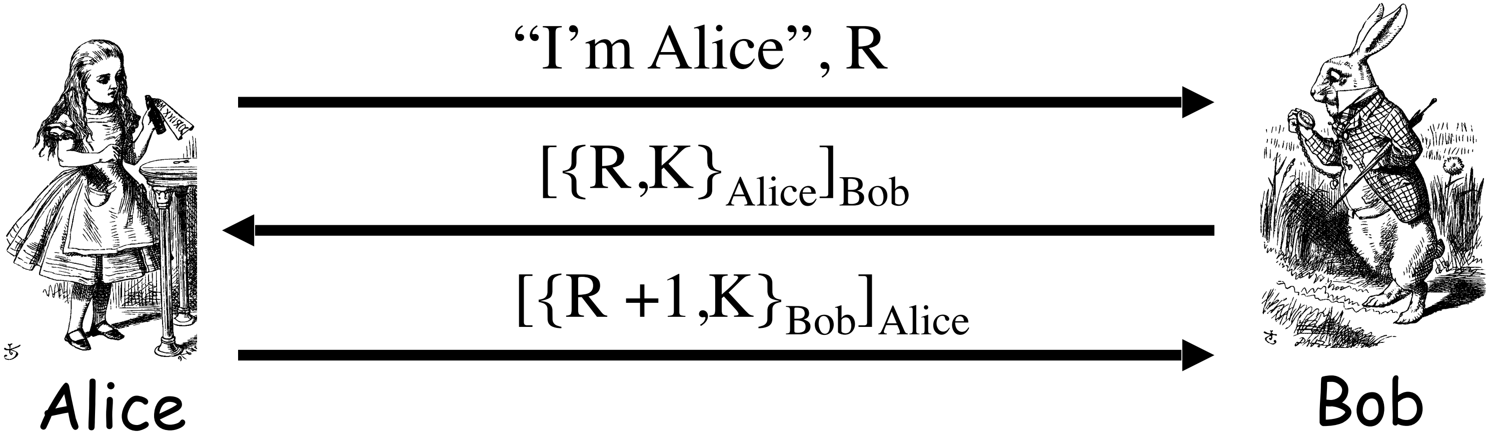
$\{[R+1,K]_{Alice}\}_{Bob}$

Alice

Bob

- Is this secure?
- Seems to be OK
- Mutual authentication and session key!

Public Key Authentication and Session Key



“I’m Alice”, R

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

$[\{R+1,K\}_{\text{Bob}}]_{\text{Alice}}$

Alice

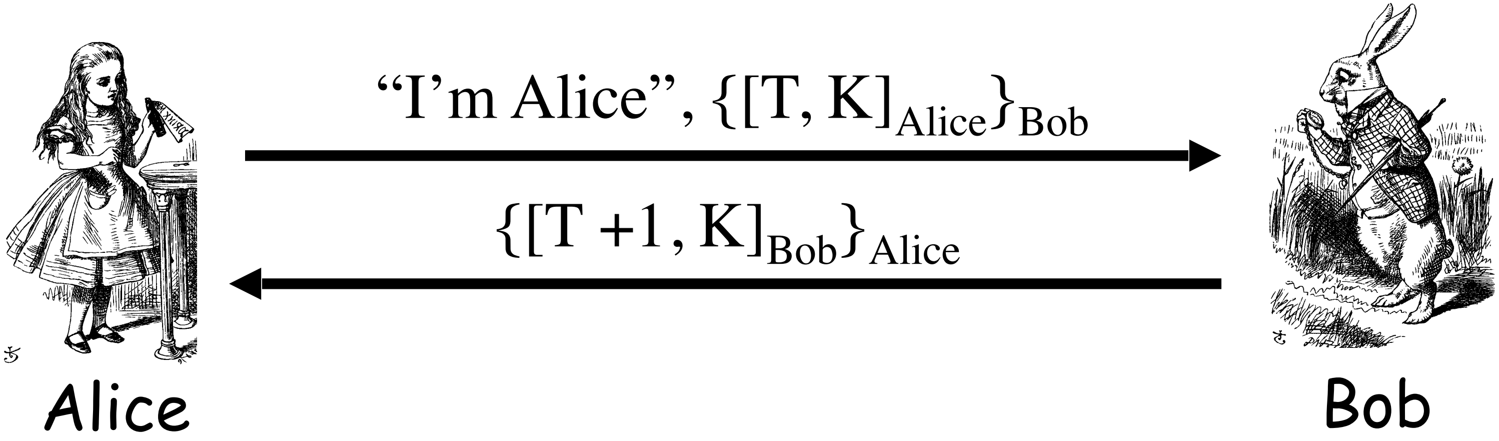
Bob

- 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



Alice

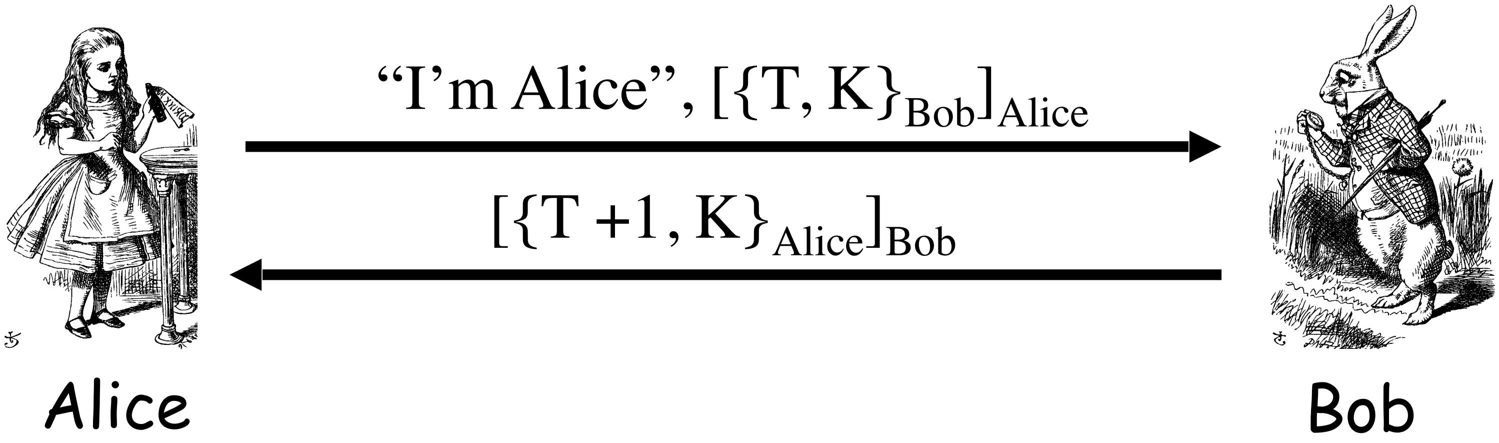
“I’m Alice”, $\{[T, K]_{\text{Alice}}\}_{\text{Bob}}$

$\{[T+1, K]_{\text{Bob}}\}_{\text{Alice}}$

Bob

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

Public Key Authentication with Timestamp T

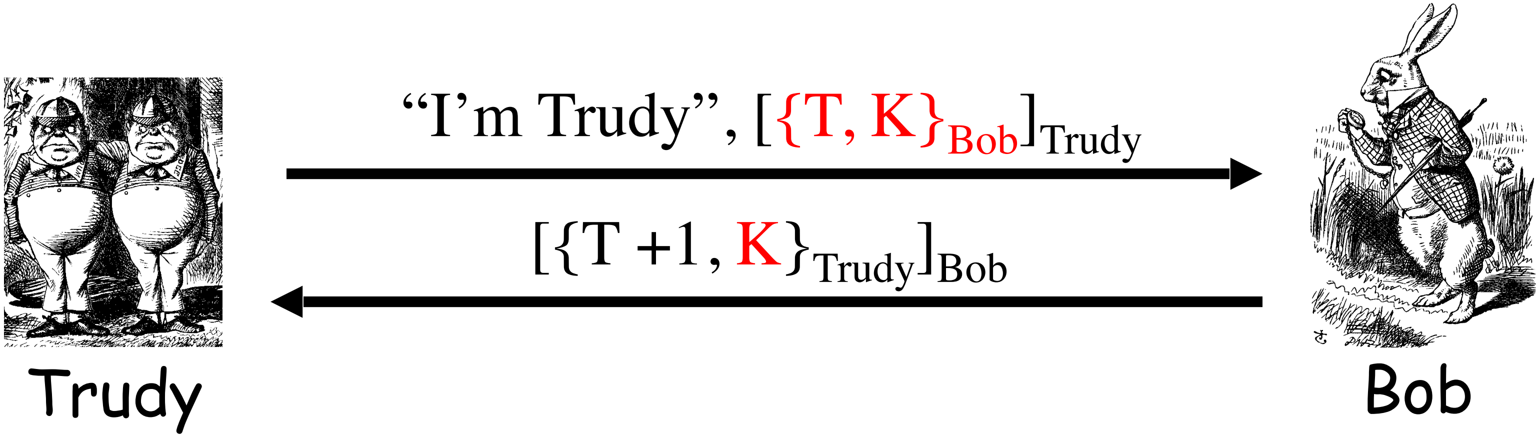


“I’m Alice”, $[\{T, K\}_{\text{Bob}}]_{\text{Alice}}$

$[\{T+1, K\}_{\text{Alice}}]_{\text{Bob}}$

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

Public Key Authentication with Timestamp T



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

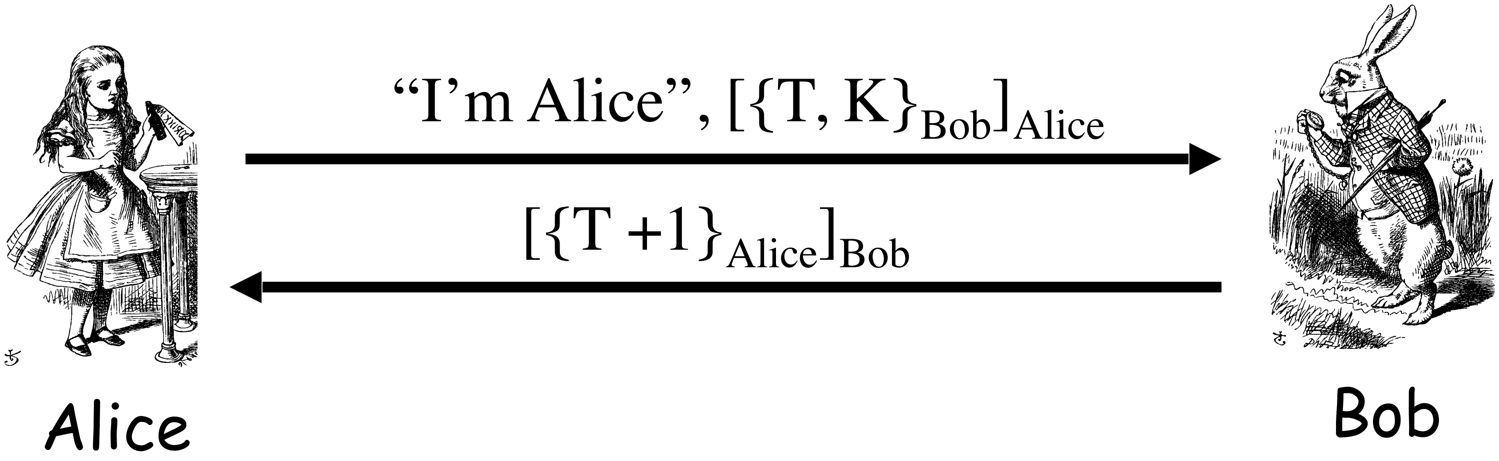
$[\{T+1, K\}_{Trudy}]_{Bob}$

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

Public Key Authentication

- Sign and encrypt with nonce…
  - Secure
- 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", $[\{T, K\}_{Bob}]_{Alice}$

$[\{T+1\}_{Alice}]_{Bob}$

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

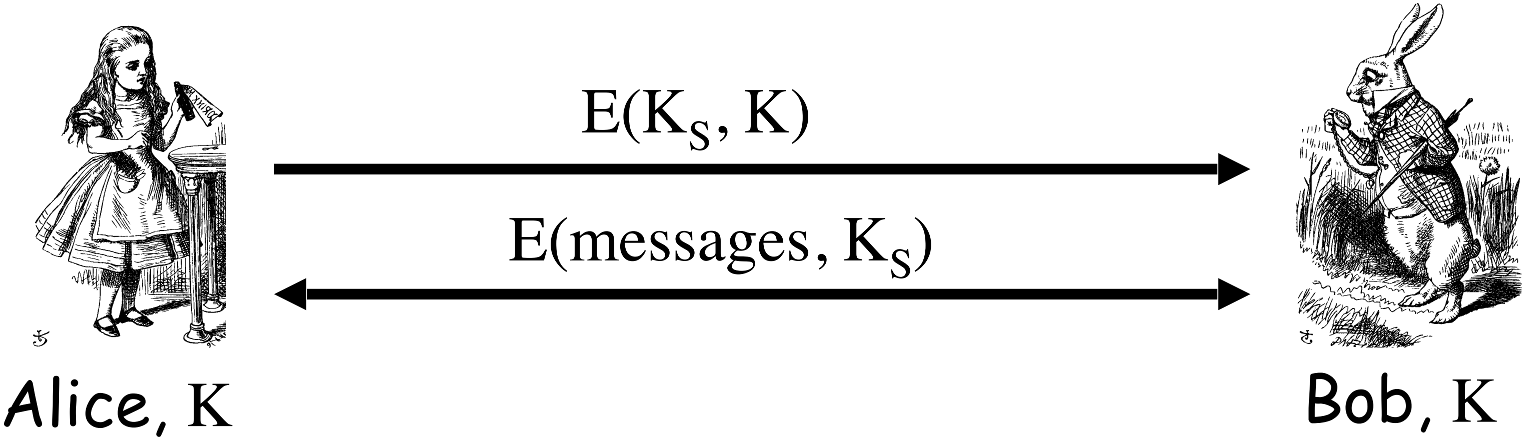
Perfect Forward Secrecy

- 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



E(K_S, K)

E(messages, K_S)

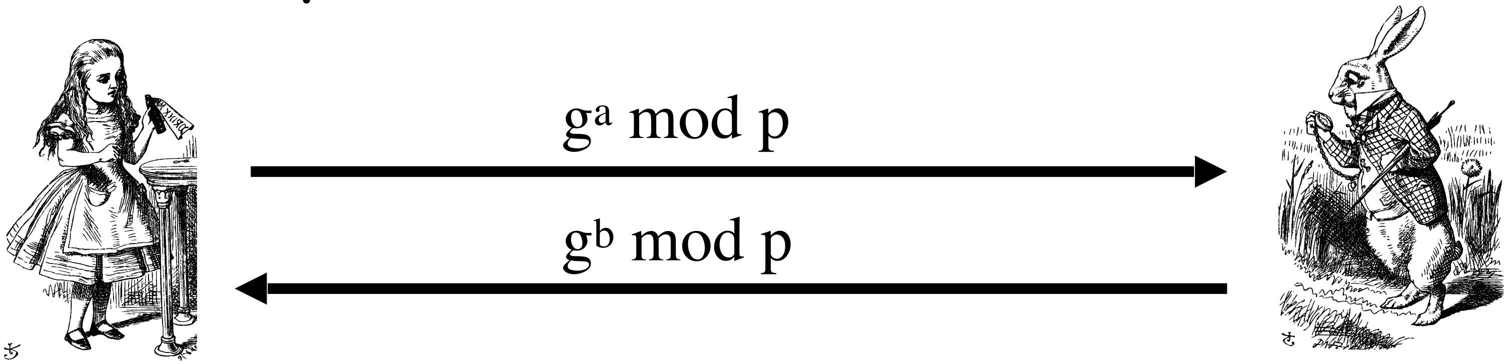
Alice, K

Bob, K

- 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



$g^a \bmod p$

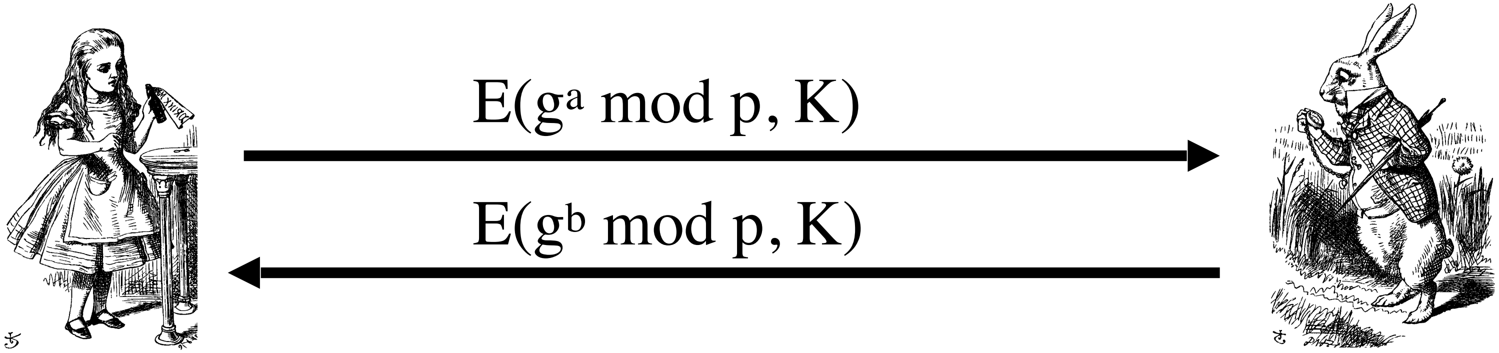
$g^b \bmod p$

Alice, a

Bob, b

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

Perfect Forward Secrecy



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

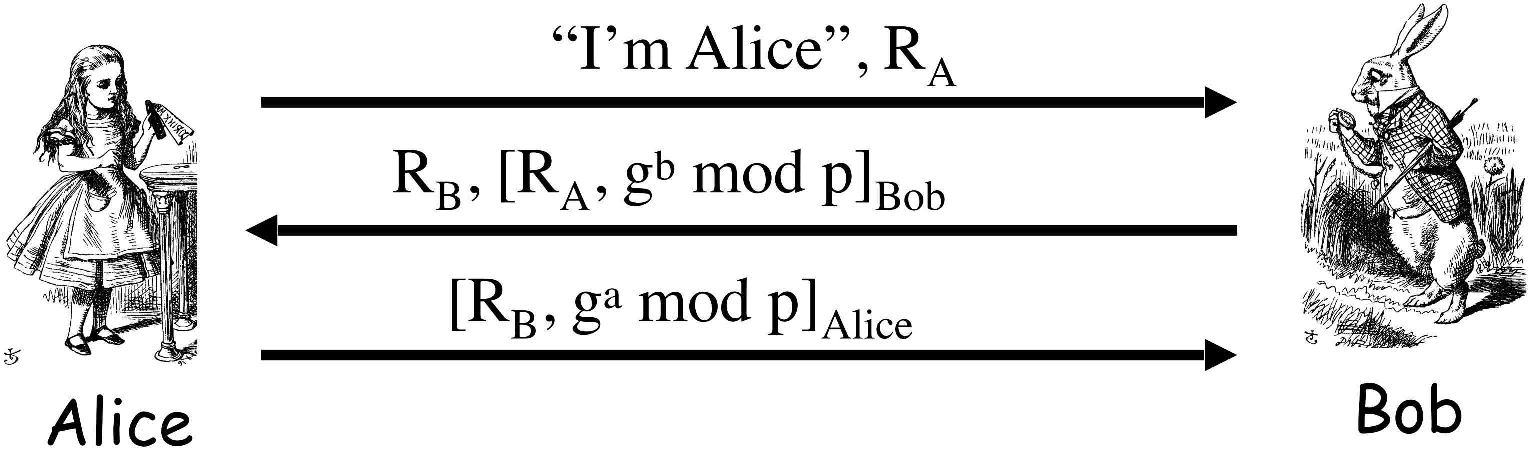
$E(g^b \bmod p, K)$

Alice: K, a

Bob: K, b

- Session key $K_S = g^{ab} \bmod p$
- Alice **forgets** a, Bob **forgets** b
- 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, $[R_A, g^b \bmod p]_{Bob}$

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

- 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

Judging a Protocol

- Remember these questions:
  - Is the protocol safe against the Replay Attack?
  - Is the protocol safe against MiM?
  - Is Mutual Authentication guaranteed?
  - Is the Session Key safe?
  - If requested, is PFS guaranteed?