Quiz

Quiz

- **What is the primary purpose of a Trusted Platform Module (TPM)?**
  - A) To serve as the primary storage unit in secure computing environments.
  - B) To manage user passwords and login credentials.
  - C) To provide hardware-based security functions like secure generation of cryptographic keys.
  - D) To monitor network traffic for malicious activity.

Quiz

- **What role does attestation play in TPM?**
  - A) It allows a device to prove to remote parties that it is running specified software.
  - B) It increases the data storage capacity of secure computing environments.
  - C) It provides users with regular updates on system performance.
  - D) It helps users reset their passwords securely.

Quiz

- **Which type of security policy is primarily used by SELinux?**
  - A) Role-based Access Control (RBAC)
  - B) Mandatory Access Control (MAC)
  - C) Discretionary Access Control (DAC)
  - D) Attribute-based Access Control (ABAC)

Quiz

- **How does SELinux enhance security compared to traditional Unix permissions?**
  - A) By allowing only root users to set permissions for files and processes.
  - B) By enforcing security policies that are defined outside of the traditional user and group permission system.
  - C) By disabling network access to improve security.
  - D) By using more complex password policies.

Quiz

- **What role does the Java Security Manager play in JVM security?**
  - A) It manages user passwords and access levels.
  - B) It controls access to system resources and capabilities, like file I/O and network access, based on a security policy.
  - C) It encrypts and decrypts data stored by Java applications.
  - D) It ensures that Java applications are distributed securely.

Quiz

- **What is the purpose of an access control list (ACL) in authorization?**
  - A) It lists all users in a system and their password hashes.
  - B) It defines what actions specific users or groups can take with respect to objects in a system.
  - C) It monitors and logs all user activities within the system.
  - D) It encrypts data as per user specifications.

Quiz

- **What is the primary difference between authentication and authorization?**
  - A) Authentication is about determining if a user is allowed to access a system, while authorization is about monitoring user actions.
  - B) Authentication verifies a user’s identity, whereas authorization determines the resources a user can access and the actions they can perform.
  - C) Authentication encrypts user data, and authorization decrypts it.
  - D) There is no difference; both terms are interchangeable.

Quiz

- **Which of the following is a common method of authentication?**
  - A) Encryption
  - B) Firewalls
  - C) Antivirus software
  - D) Password

Basic Authentication Protocol

Protocol: a key concept in network security

- **Human protocols** — the rules followed in human interactions
  - Example: handshaking
- **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.

Simple Security Protocols in Real Life

Secure Entry to NSA

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

    Yes? Enter

    No? Get shot by security guard

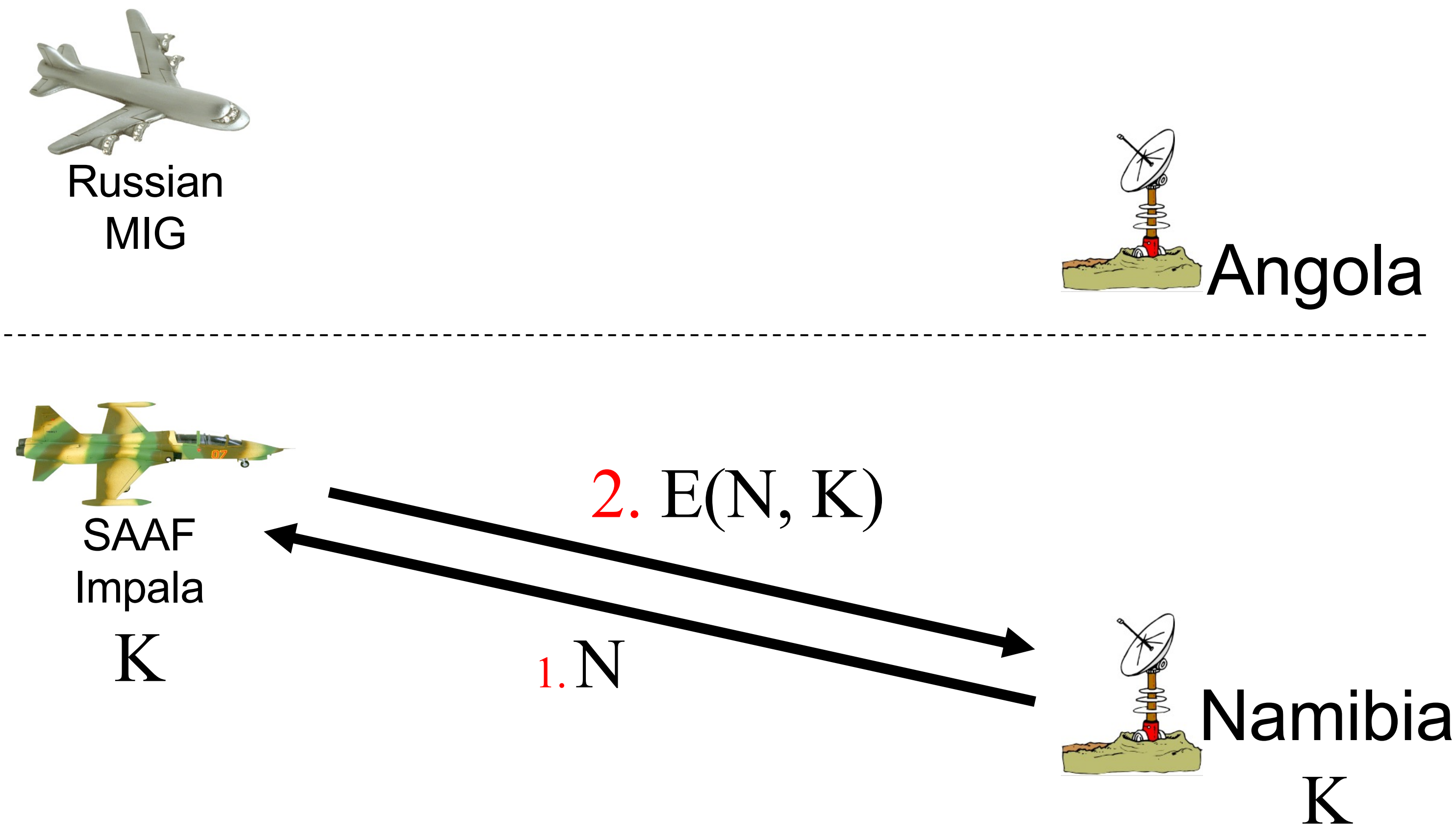
ATM Machine Protocol

1. Insert ATM card
2. Enter PIN
3. Correct PIN?

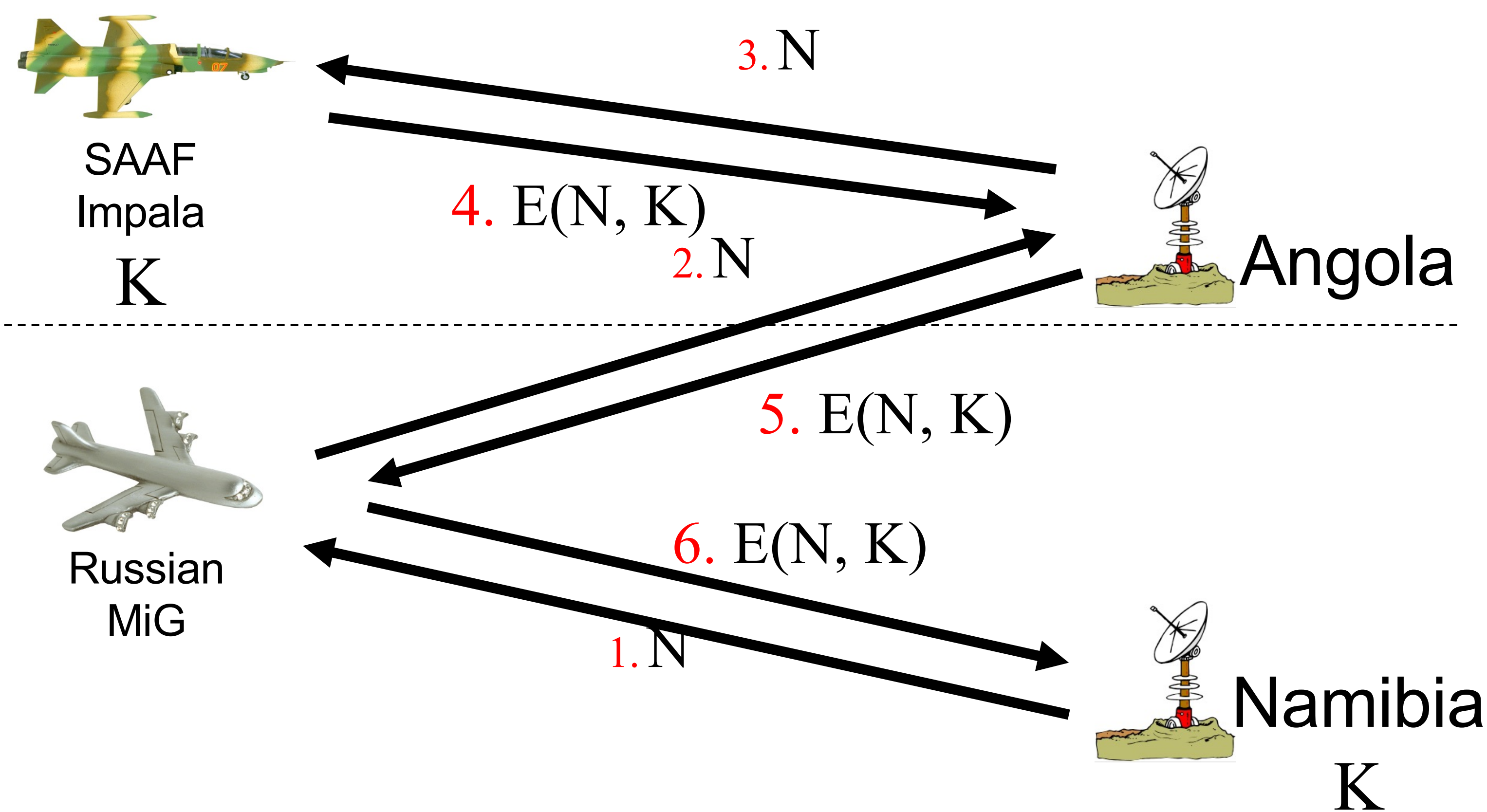
    Yes? Conduct your transaction(s)

    No? Machine (eventually) eats card

Military Security Protocol: Identify Friend or Foe (IFF)



MIG in the Middle



Simple Authentication Protocol

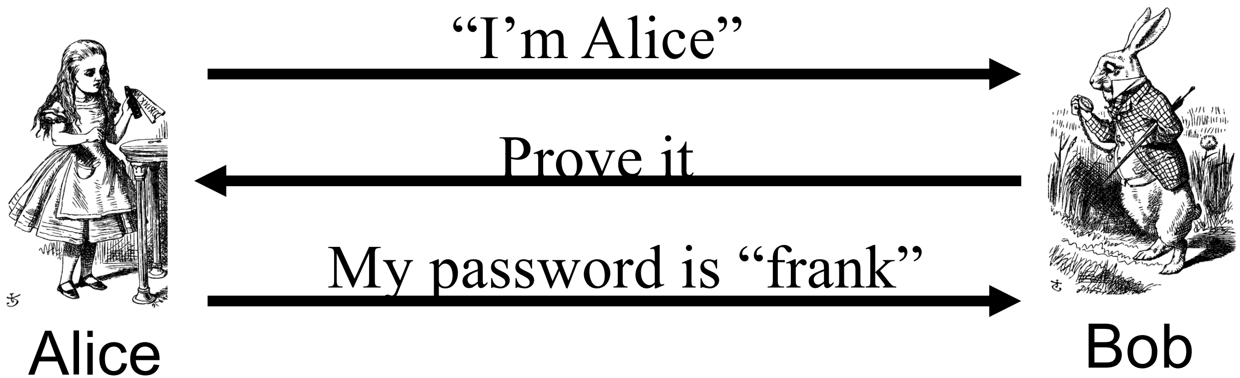
Authentication Related Issues

- Alice must prove her identity to Bob
  - Alice and Bob can be humans or **computers**
  - Communication **over network**
- May also require Bob to prove he's Bob (**mutual authentication**)
  - Not always secure
- Probably need to establish a **session key**
  - Symmetric key to provide confidentiality/integrity
- May have other requirements, such as
  - Use public keys
  - Use symmetric keys
  - Use hash functions
  - Anonymity, plausible deniability, etc., etc.

Authentication

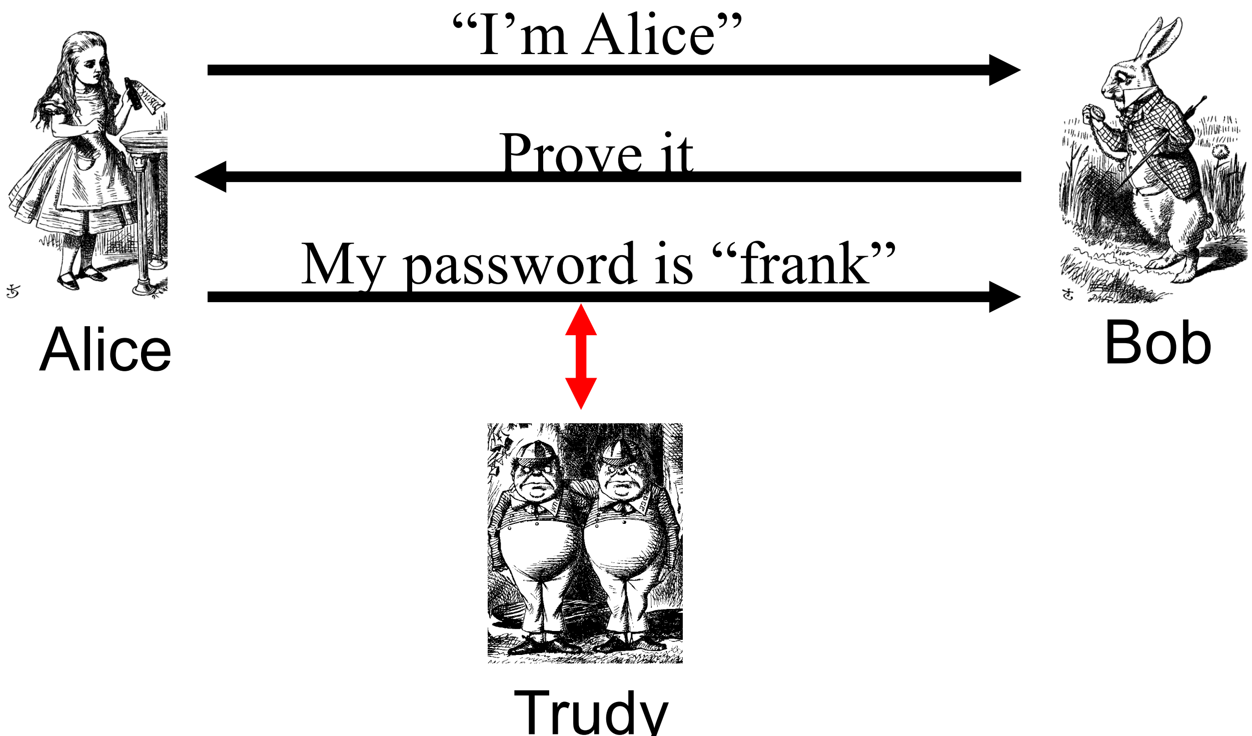
- Authentication on a stand-alone computer is relatively simple
  - Hash password with salt
  - “trusted path,” attacks on authentication software, keystroke logging, etc., can be issues
- Authentication over a network is challenging
  - Attacker can passively **observe** messages
  - Attacker can **replay**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

Simple Authentication

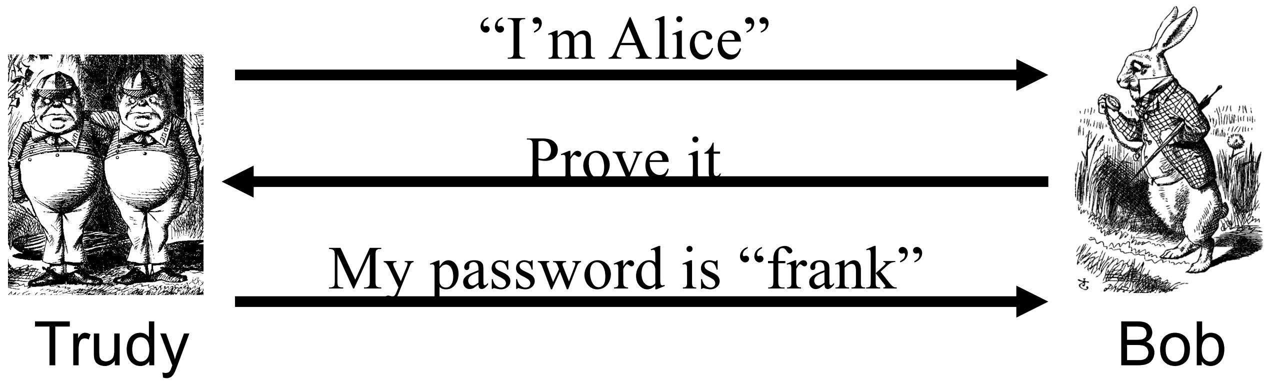


- Simple and may be OK for standalone system
- But insecure for networked system
  - Subject to a **replay** attack (next 2 slides)
  - Also, Bob must know Alice’s password

Authentication Attack



Authentication Attack



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

Simple Authentication

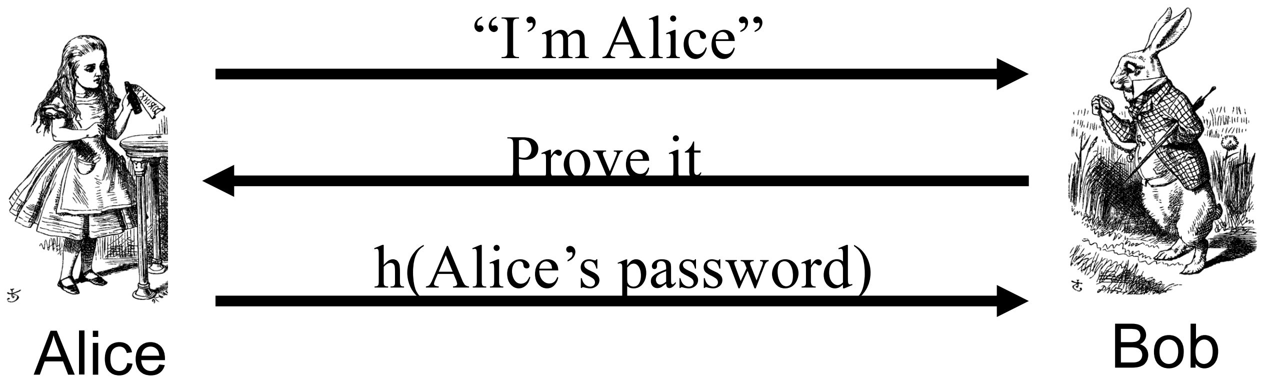
I'm Alice, my password is "frank"

Alice

Bob

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

Better Authentication



- Better since it hides Alice’s password
  - From both Bob and Trudy
- But still subject to **replay**

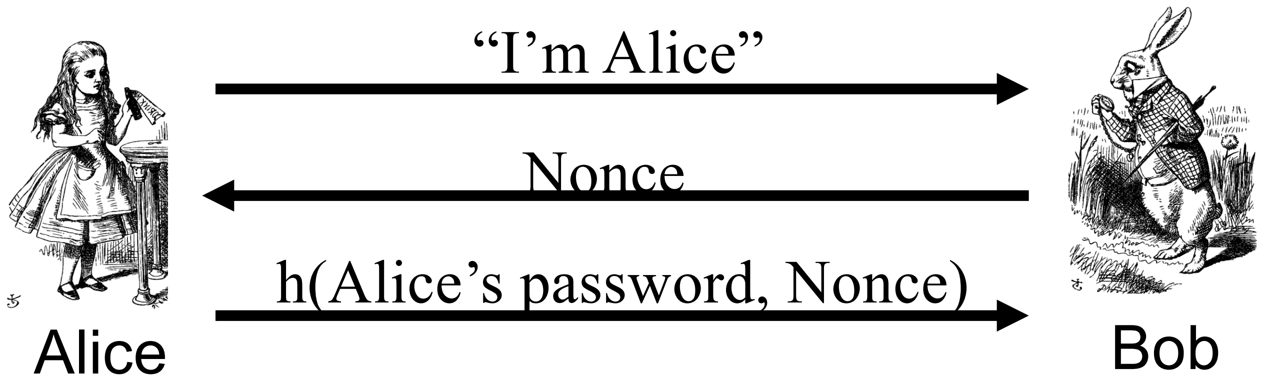
Challenge-Response

- 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

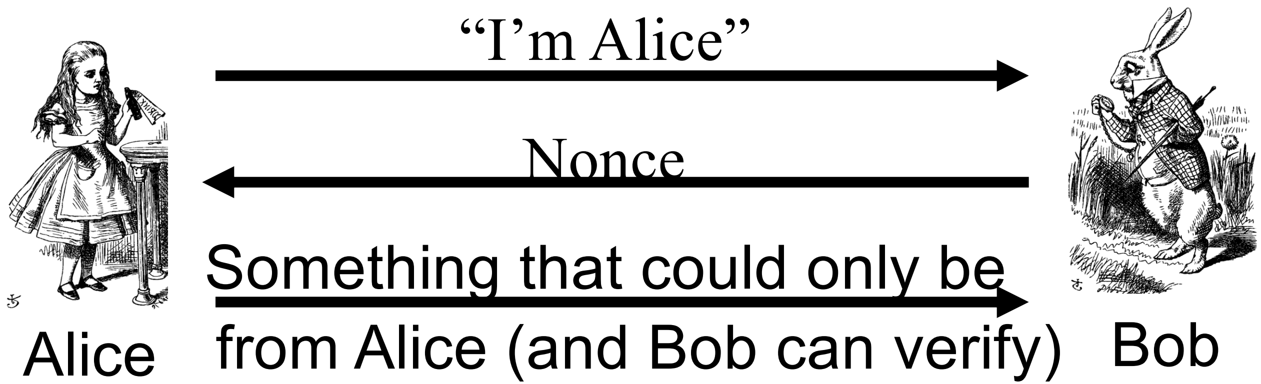
- To ensure freshness, can employ a **nonce**
  - Nonce == **n**umber used **once**

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 better here

Mutual Authentication based on Symmetric Key Crypto

Symmetric Key Notation

- Encrypt plaintext P with key K

  $C = E(P, K)$

- Decrypt ciphertext C with key K

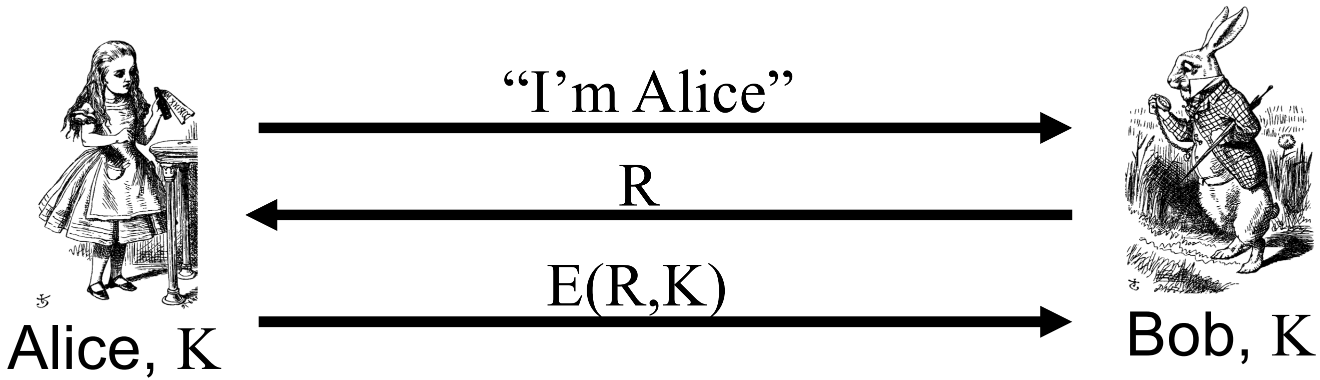
  $P = D(C, K)$

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

Authentication: Symmetric Key

- Alice and Bob share symmetric key K
- Key $\mathbf{K}$ 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, …

Authentication with Symmetric Key

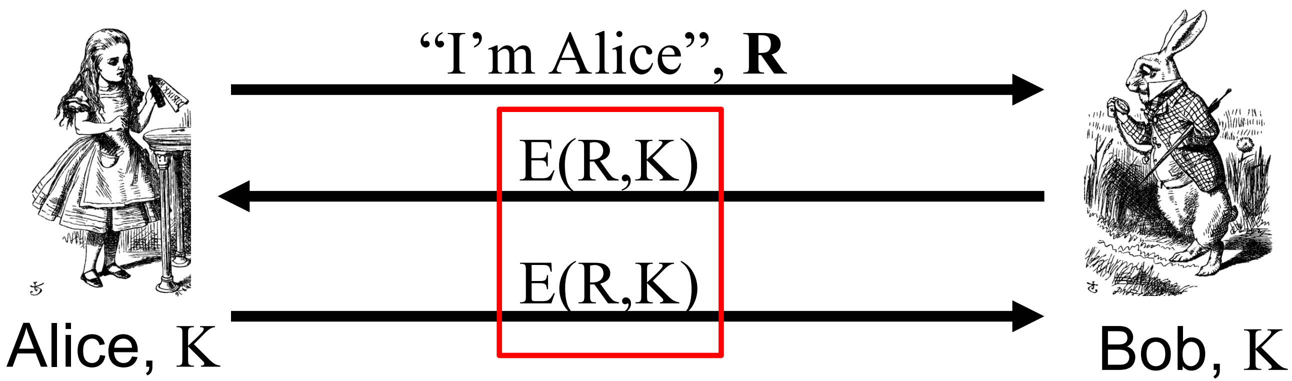


- Secure method for Bob to authenticate Alice
  - Alice does not authenticate Bob
- So, can we achieve mutual authentication?

Mutual Authentication

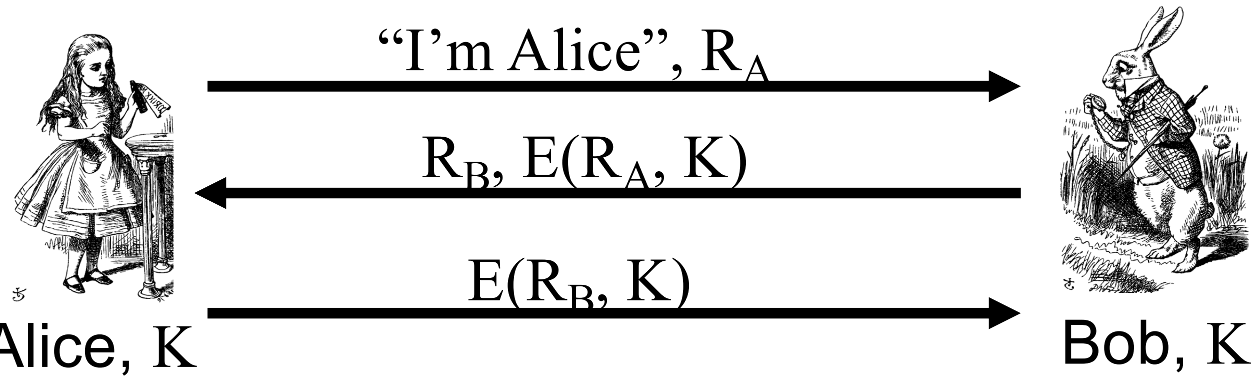
- Since we have a secure one-way authentication protocol…
- The obvious thing to do is to use the protocol **twice**
  - Once for Bob to authenticate Alice
  - Once for Alice to authenticate Bob
- This has got to work…

Mutual Authentication?



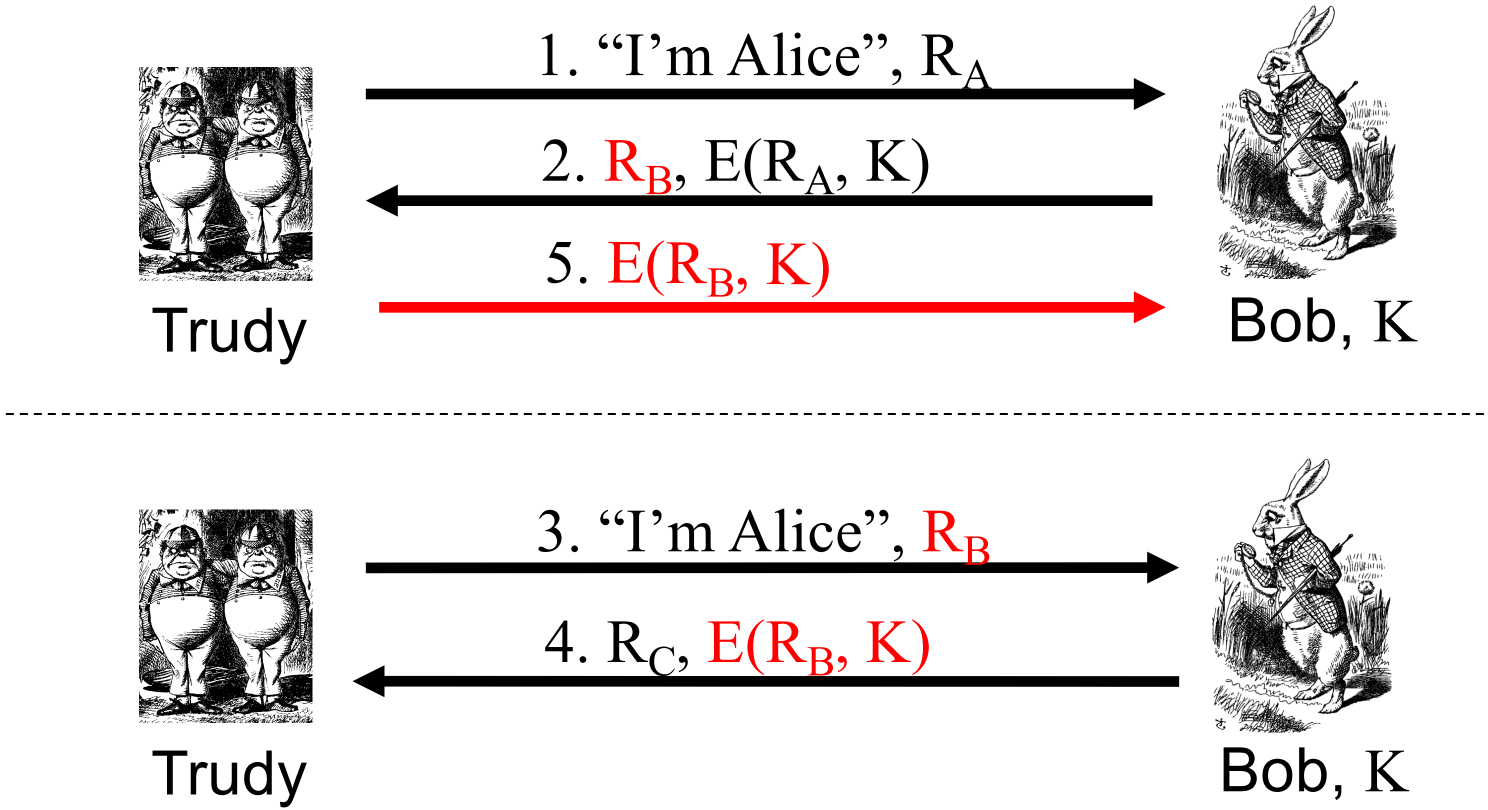
- What’s wrong with this?
- “Alice” could be Trudy (or anybody else)!

Mutual Authentication



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

Mutual Authentication Attack



BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Mutual Authentication

- Our one-way authentication protocol is **not** secure for mutual authentication
  - Protocols are subtle!
  - The “obvious” thing may not be 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)

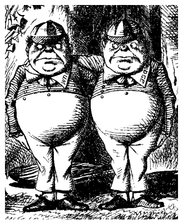
E("**Alice**",R_B,K) →

Alice, K

Bob, K

- Do these "insignificant" changes help?

Mutual Authentication Attack



Trudy

1. “I’m Alice”, R_A

2. R_B, E(“Bob”,R_A,K)

5. E(“Bob”, R_B, K)

Bob, K

Expecting: E(“Alice”, R_B, K)

Trudy

3. “I’m Alice”, R_B

4. R_C, E(“Bob”, R_B, K)

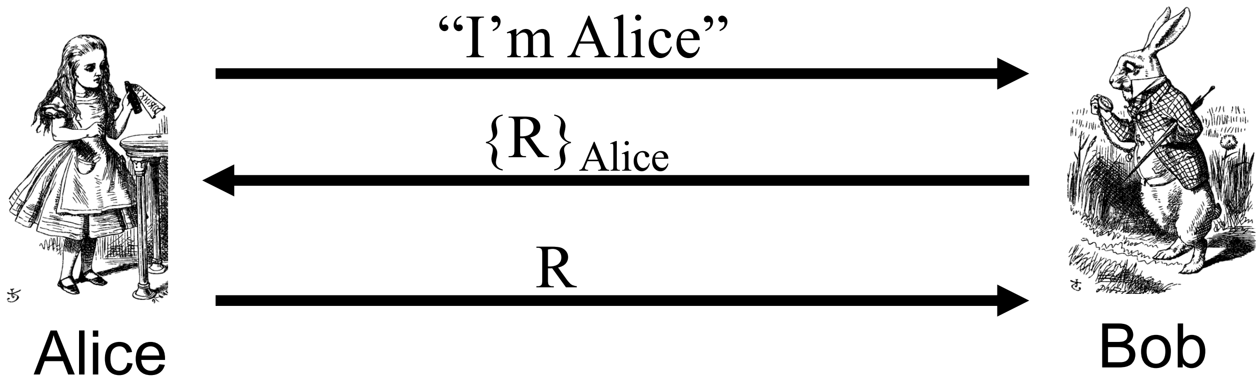
Bob, K

Mutual Authentication based on Public Key Crypto

Public Key Notation

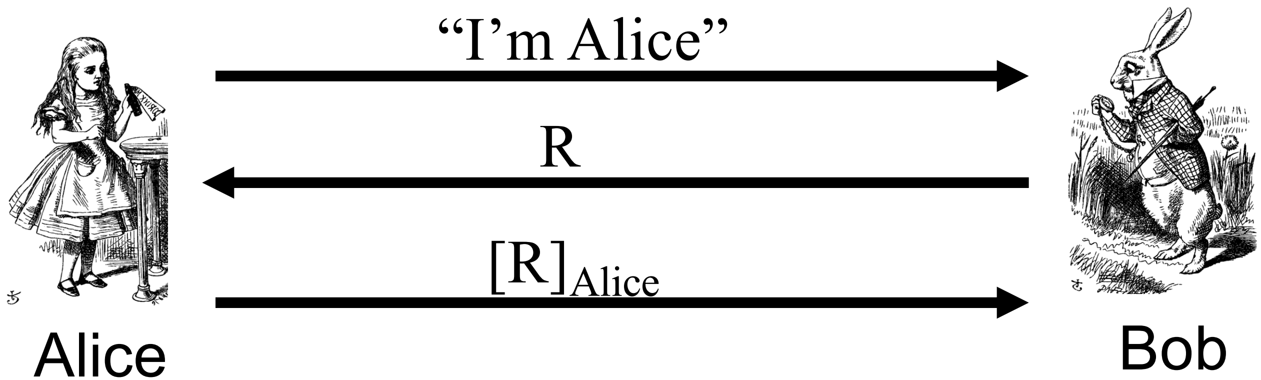
- Encrypt M with Alice's public key: $\{M\}_{\text{Alice}}$
- Sign M with Alice's private key: $[M]_{\text{Alice}}$
- Then
  - $[\{M\}_{\text{Alice}}]_{\text{Alice}} = M$
  - $\{[M]_{\text{Alice}}\}_{\text{Alice}} = M$
- **Anybody** can use Alice's **public key**
- Only **Alice** can use her **private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

Public Key Authentication



- Is this secure?
- Trudy can get Alice to decrypt anything!
  - So, should have two key pairs (one for authentication, one for encryption)

Public Key Authentication



- Is this secure?
- Trudy can get Alice to sign anything!
  - Same as previous — should have two key pairs (one for authentication, and one for encryption)

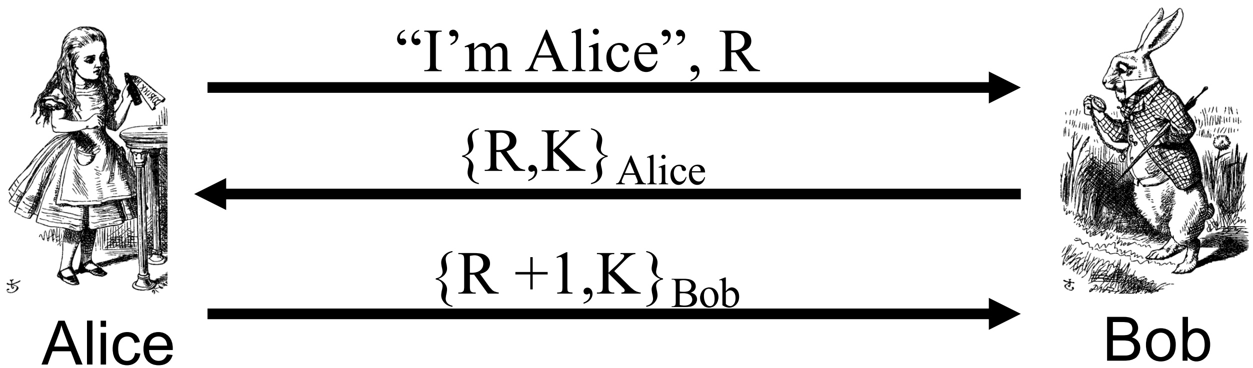
Public Keys

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

Session Key

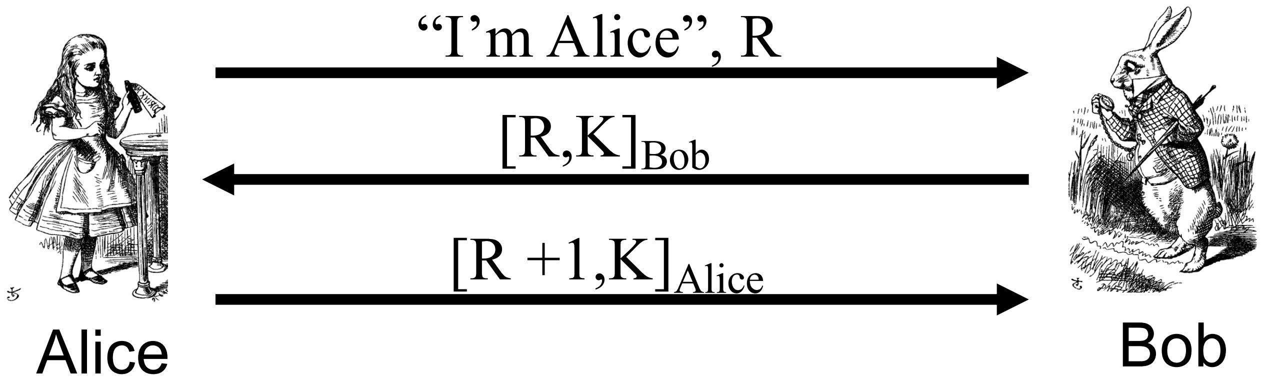
- Usually, a **session key** is required
  - I.e., a **symmetric key** for a particular session
  - Used for **confidentiality** and/or **integrity**
- How to authenticate and **establish** a session key (i.e., shared symmetric key)?
  - When authentication completed, want Alice and Bob to share a session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

Authentication & Session Key



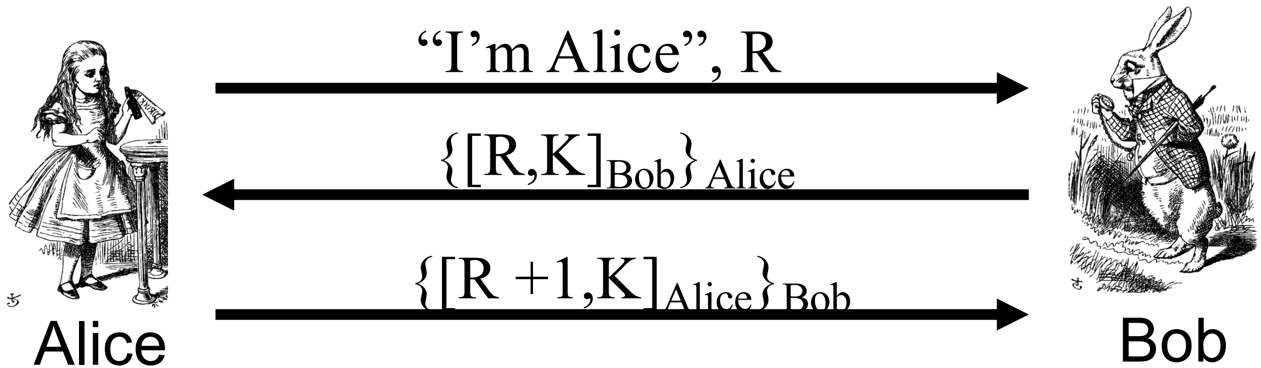
- Is this secure?
  - Alice is authenticated and session key is secure
  - Alice's "nonce", R, useless to authenticate Bob
  - The key K is acting as Bob's nonce to Alice
- No mutual authentication

Public Key Authentication and Session Key



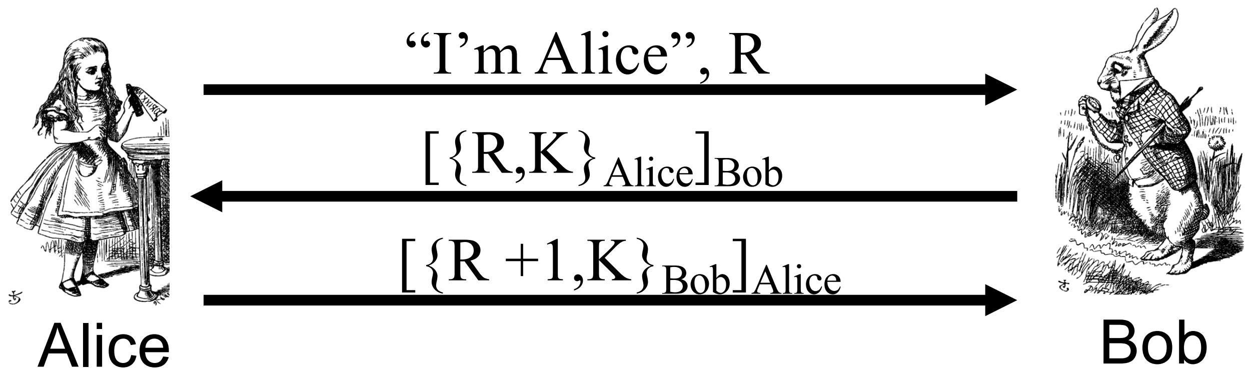
- Is this secure?
  - Mutual authentication (good), but…
  - … session key is not secret (very bad)

Public Key Authentication and Session Key



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

Public Key Authentication and Session Key



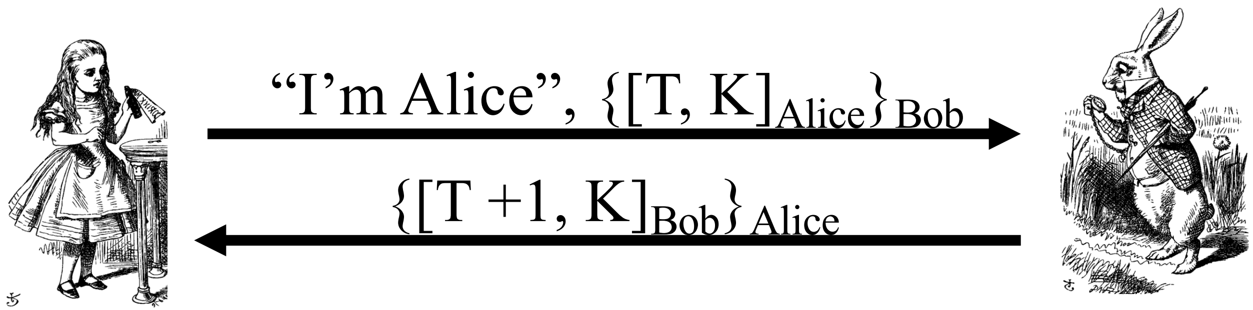
- Is this secure?
- Seems to be OK
  - Anyone can see $\{R,K\}_{Alice}$ and $\{R+1,K\}_{Bob}$

BINGHAMTON
UNIVERSITY
STATE UNIVERSITY OF NEW YORK

Timestamps

- A timestamp T is derived from current time
- Timestamps used in some security protocols
  - Kerberos, for example
- Timestamps reduce number of msgs (good)
  - Like a **nonce** that both sides know in advance
- "Time" is a security-critical parameter (bad)
  - Attacker can attack the system clock
- Clocks never exactly the same, so must allow for **clock skew** — creates risk of replay
  - Accept time "**close enough**" to the current time
  - How much clock skew is enough?
  - Open small window for replay attack

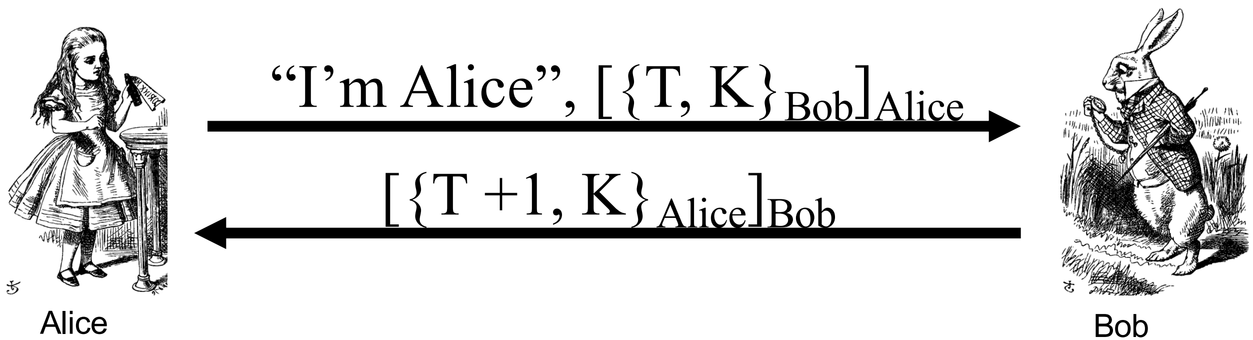
Public Key Authentication with Timestamp T



Alice Bob

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

Public Key Authentication with Timestamp T



- 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”, $[\{T, K\}_{Bob}]_{Trudy}$ →

← $[\{T+1, K\}_{Trudy}]_{Bob}$

Bob

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

Public Key Authentication

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