

Authentication Protocols

If security were all that mattered, computers would never be turned on, let alone hooked into a network with literally millions of potential intruders.

—Dan Farmer, American computer security researcher and programmer

Tamer ABUHMED

School of Computer and Information Engineering

INHA University

Outline

- Introduction
- Ideal Security Protocol Properties
- Authentication Protocols
 - Simple Design
 - Common Attacks on the Authentication Protocols
- Better Authentication Protocol Designs
 - Challenge-Response design
 - Authentication Protocols by: Symmetric Key Settings
 - Authentication Protocols by: Public Key Settings

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 occur
 - Recent IE implementation of SSL
- Not easy to get protocols right...

Ideal Security Protocol

- Must satisfy security requirements
 - Requirements need to be precise
- Efficient
 - Small computational requirement
 - Small bandwidth usage, minimal delays...
- Robust
 - Works when attacker tries to break it
 - Works even if environment changes
- Easy to use & implement, flexible...
- Difficult to satisfy all of these!

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



- 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



Alice

“I’m Alice”



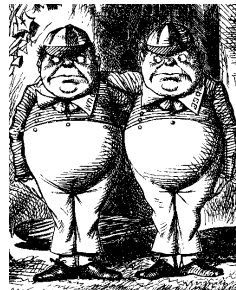
Prove it



My password is “frank”



Bob



Trudy

Authentication Attack



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

Simple Authentication



Alice

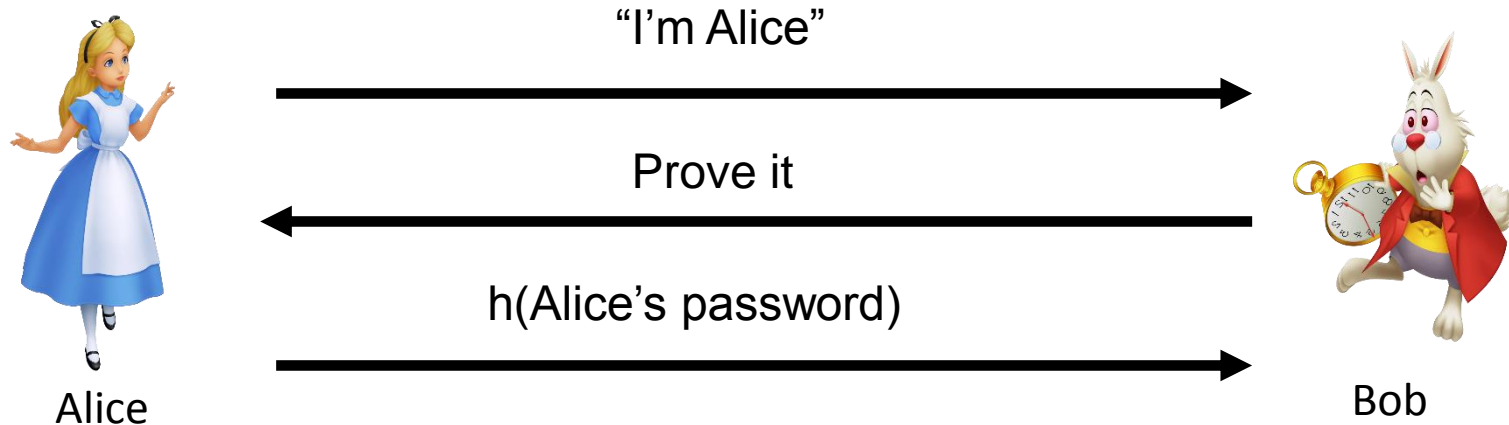
I'm Alice, my password is "frank"



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

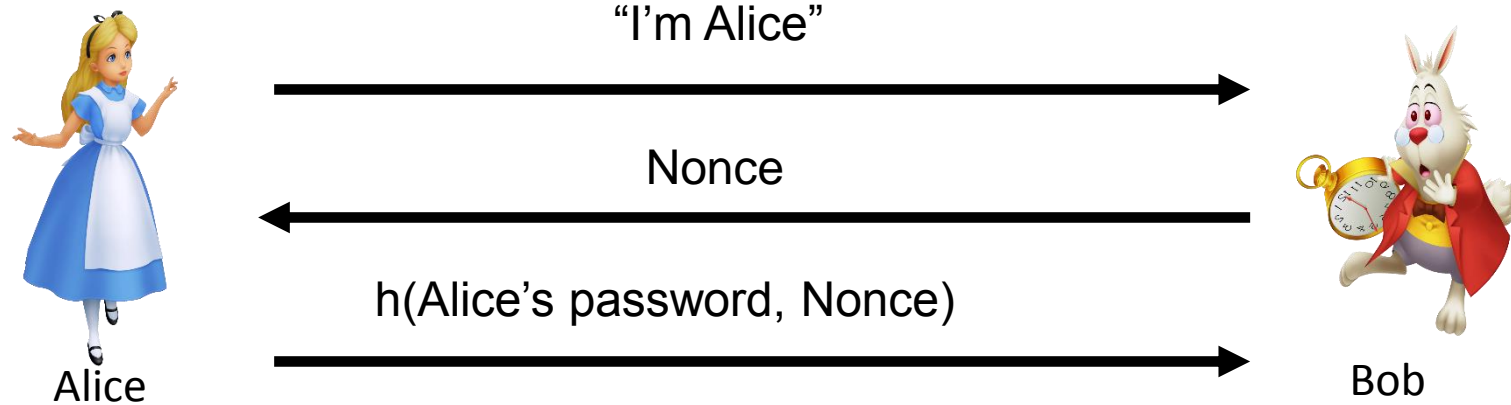
Challenge-Response

- 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

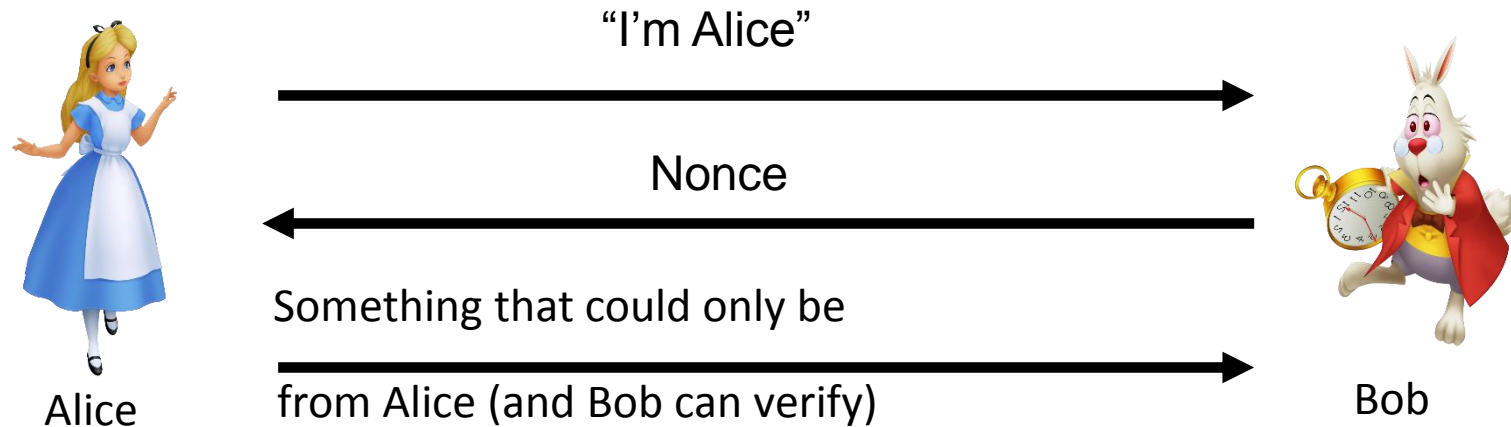
- To ensure freshness, can employ a **nonce**
 - Nonce == **number used once**
- What to use for nonces?
 - That is, what is the challenge?
- What should Alice do with the nonce?
 - That is, how to compute the response?
- How can Bob verify the response?
- Should we rely on passwords or keys?

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 password to verify

Generic Challenge-Response



- In practice, how to achieve this?
- Hashed password works, but...
- Encryption is better here (Why?)

Symmetric Key Notation

- Encrypt plaintext P with key K

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

- Decrypt ciphertext C with key K

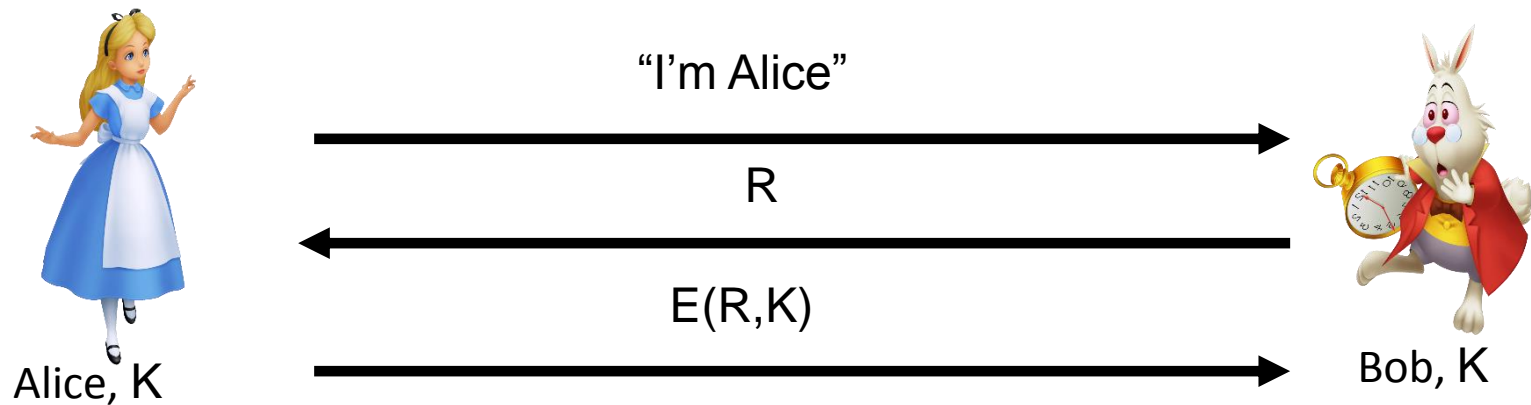
$$P = D(C,K)$$

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

Authentication: Symmetric Key

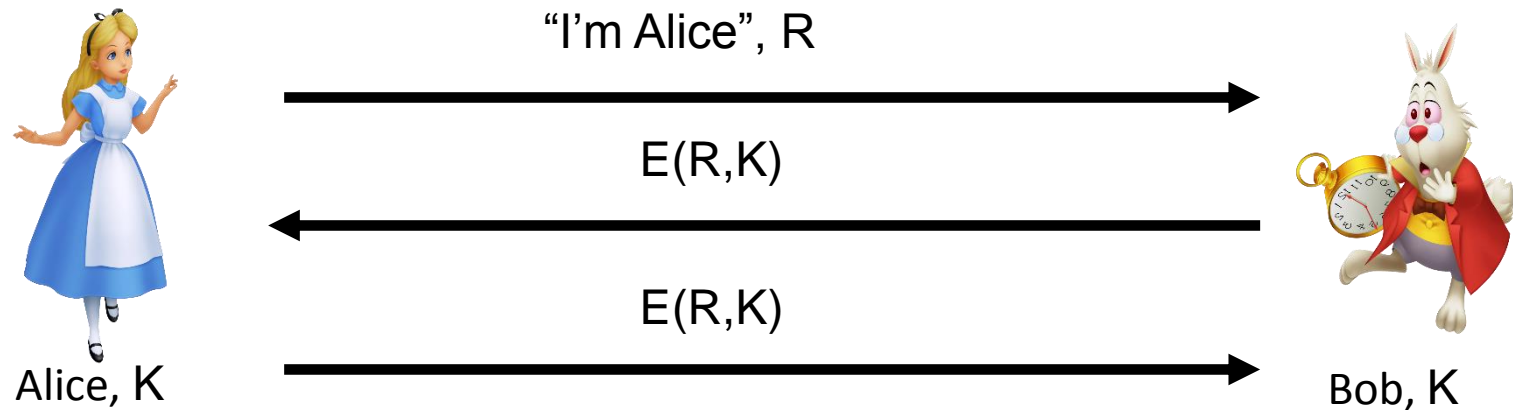
- Alice and Bob share symmetric key K
- Key K known only to Alice and Bob
- Authenticate by proving knowledge of shared symmetric key
- How to accomplish this?
 - Cannot reveal key, must not allow replay (or other) attack, must be verifiable, ...

Authentication with Symmetric Key



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

Mutual Authentication?

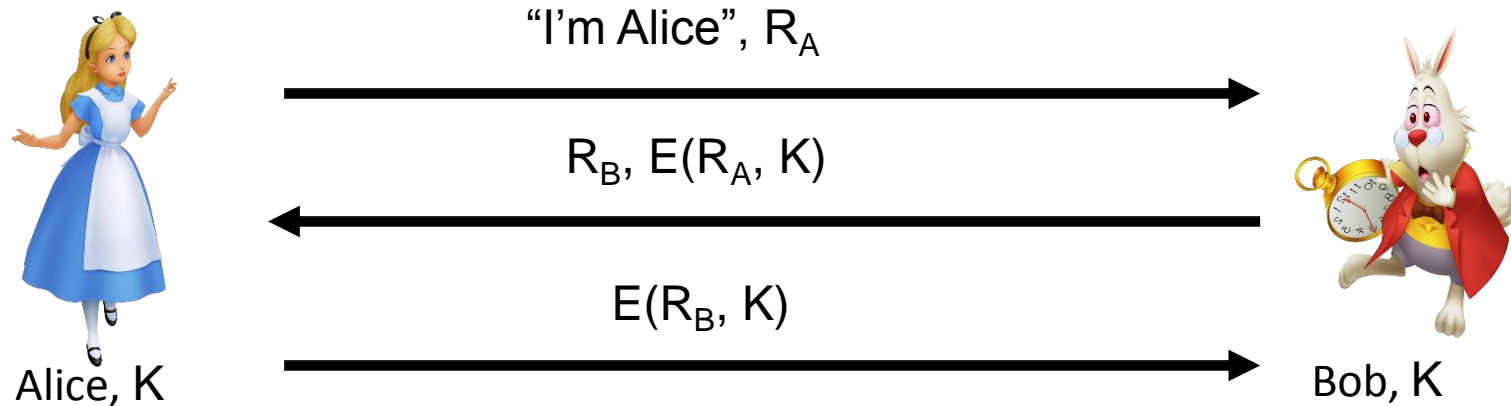


- What's wrong with this picture?
- “Alice” could be Trudy (or anybody else)!

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? See the next slide

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 , $E(R_B, K)$

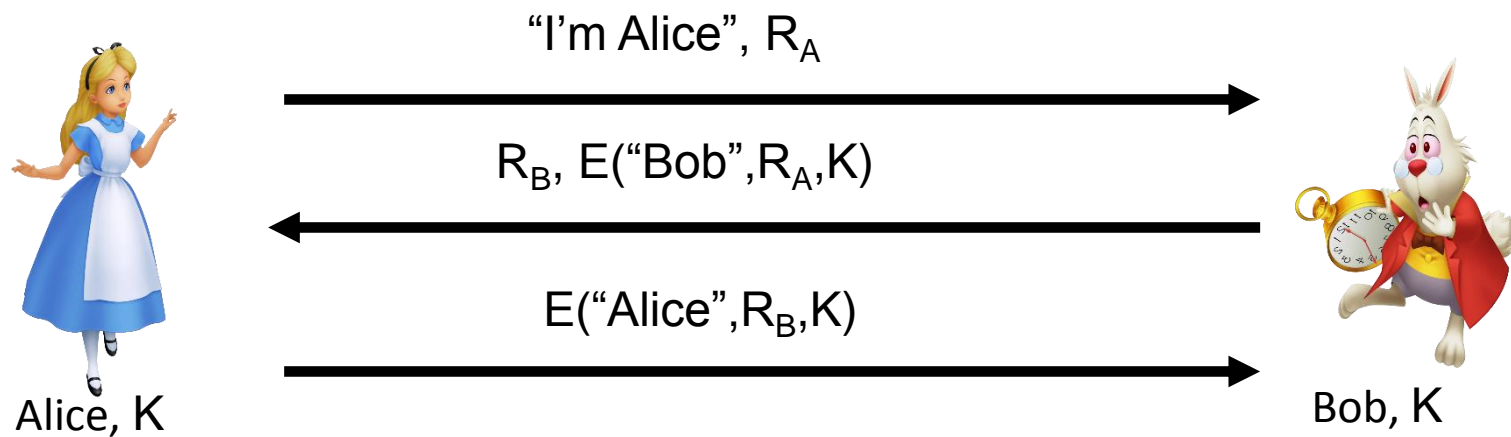


Bob, K

Mutual Authentication

- Our one-way authentication protocol is **not** secure for mutual authentication
 - Protocols are subtle!
 - 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

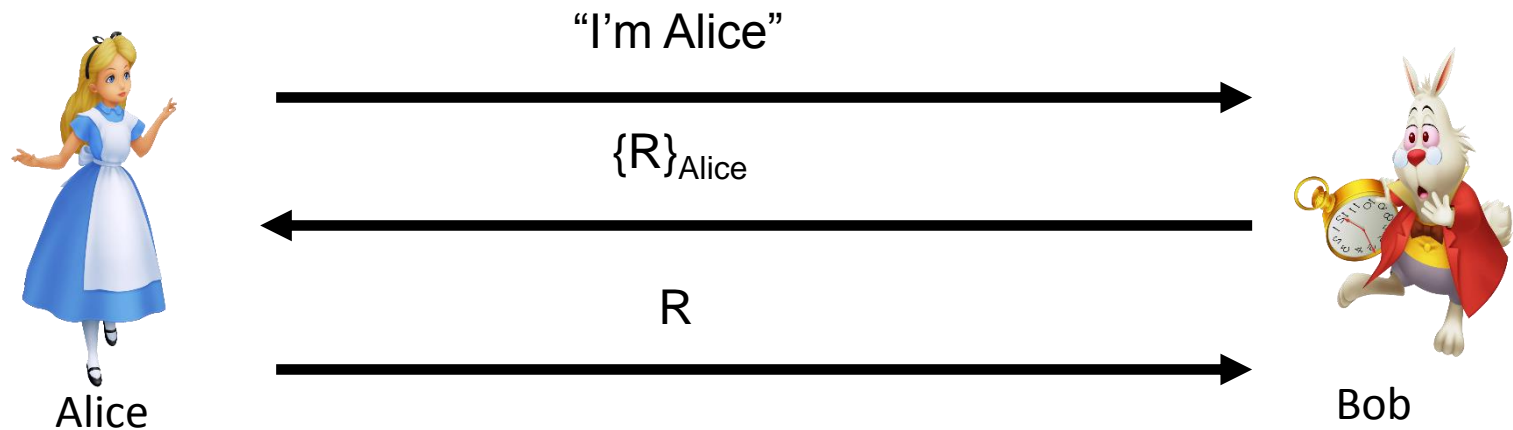


- Do these “insignificant” changes help?
- Yes!

Public Key Notation

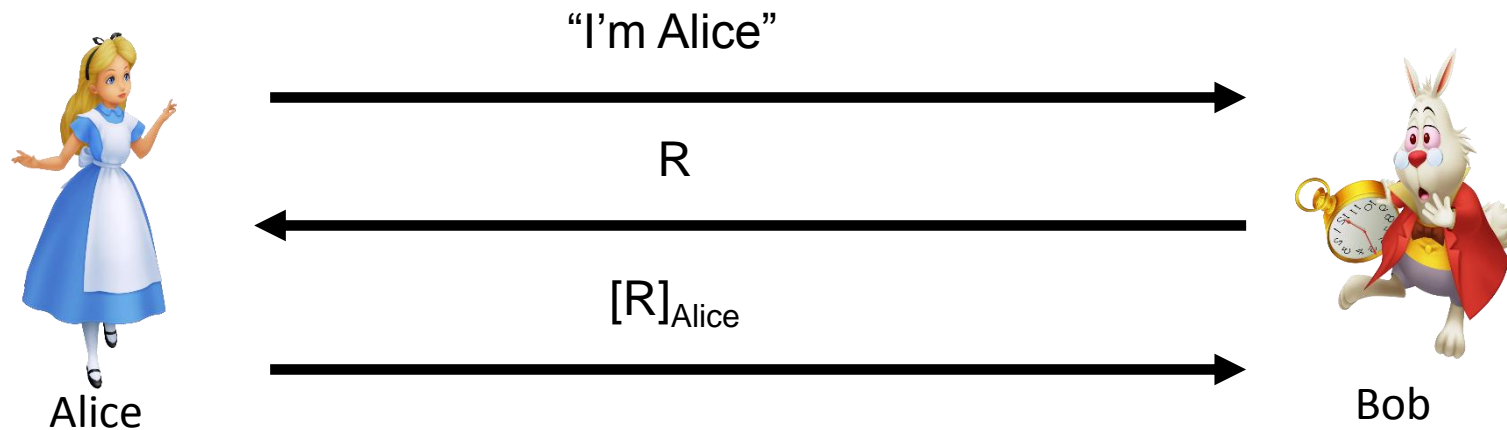
- Encrypt M with Alice's public key: $\{M\}_{\text{Alice}}$
- Sign M with Alice's private key: $[M]_{\text{Alice}}$
- Then
 - $[\{M\}_{\text{Alice}}]_{\text{Alice}} = M$
 - $\{[M]_{\text{Alice}}\}_{\text{Alice}} = M$
- **Anybody** can use Alice's **public key**
- Only **Alice** can use her **private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

Public Key Authentication



- Is this secure?
- Trudy can get Alice to decrypt anything!
 - So, should have two key pairs

Public Key Authentication



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

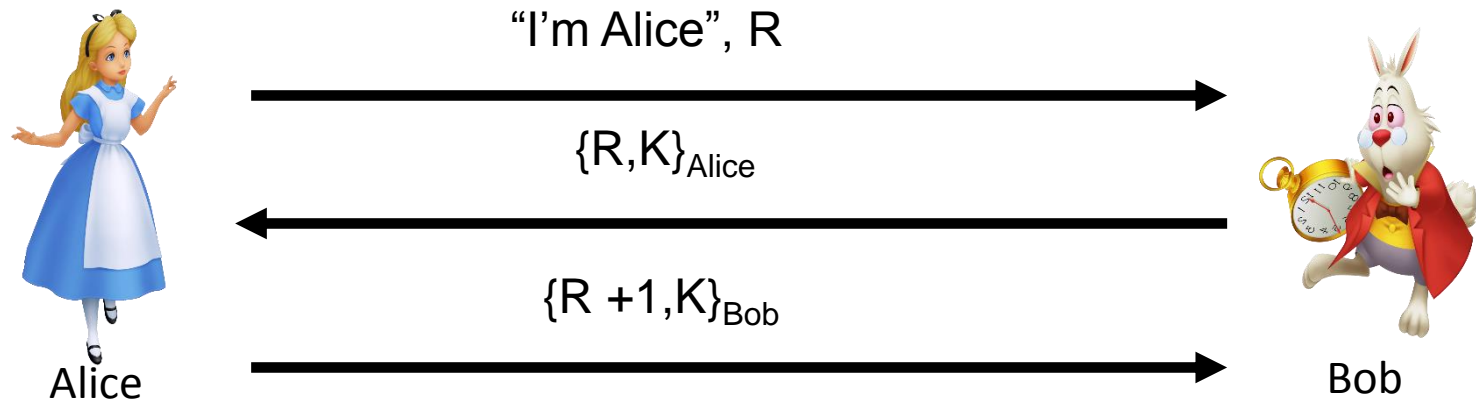
Public Keys

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

Session Key

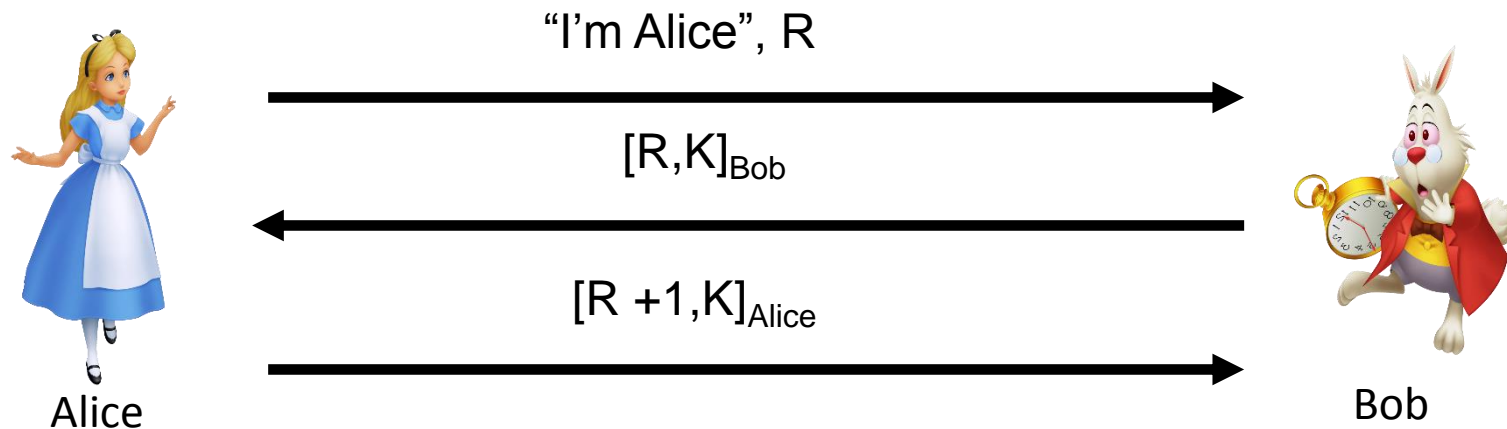
- Usually, a **session key** is required
 - I.e., a symmetric key for a particular session
 - Used for confidentiality and/or integrity
- How to authenticate and establish a session key (i.e., shared symmetric key)?
 - When authentication completed, want Alice and Bob to share a session key
 - Trudy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

Authentication & Session Key



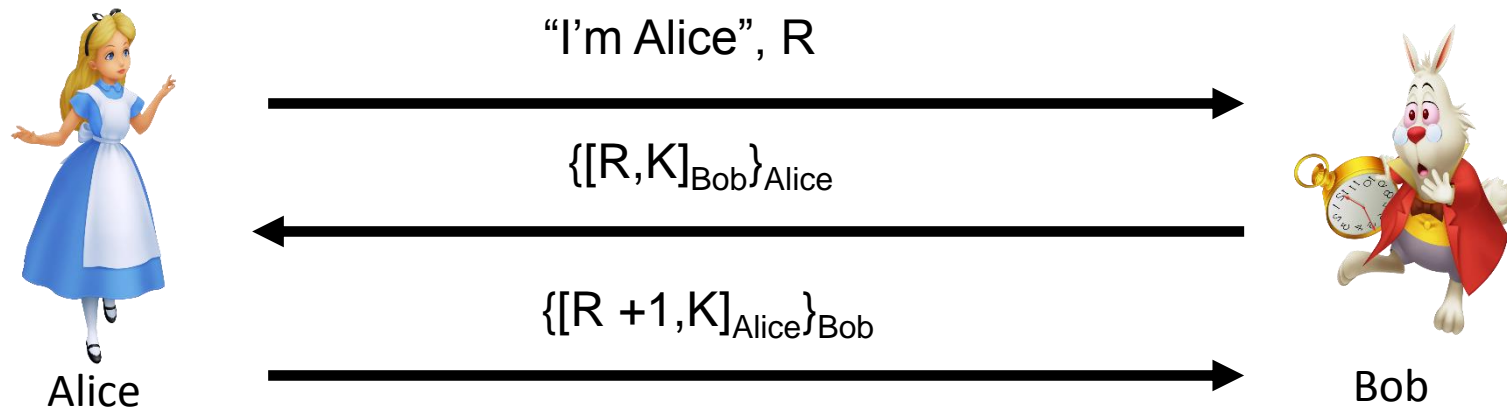
- Is this secure?
 - Alice is authenticated and session key is secure
 - Alice's "nonce", R , useless to authenticate Bob
 - The key K is acting as Bob's nonce to Alice
- No mutual authentication

Public Key Authentication and Session Key



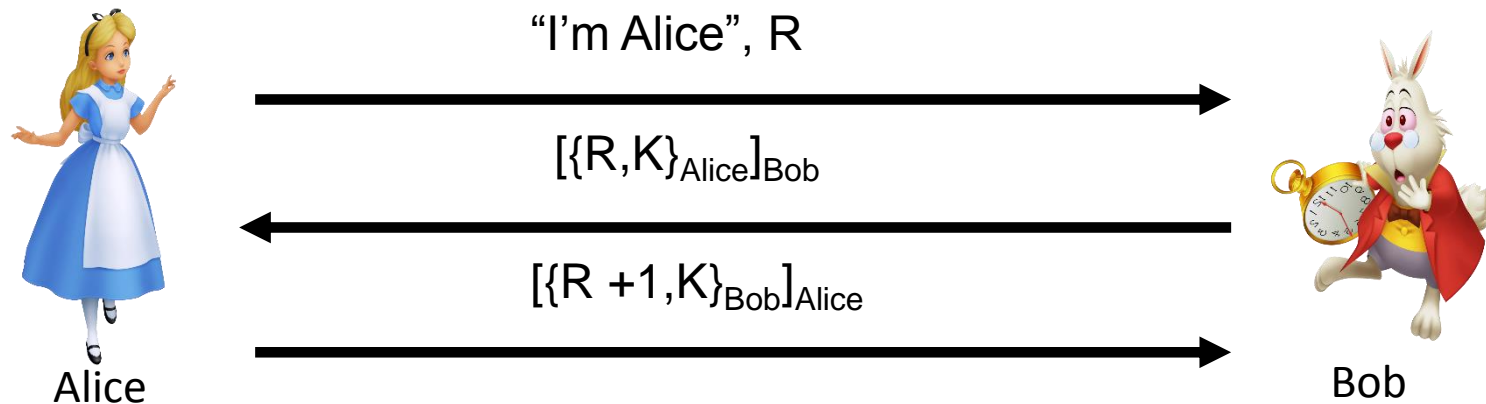
- Is this secure?
 - Mutual authentication (good), but...
 - ... session key is not secret (very bad)

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

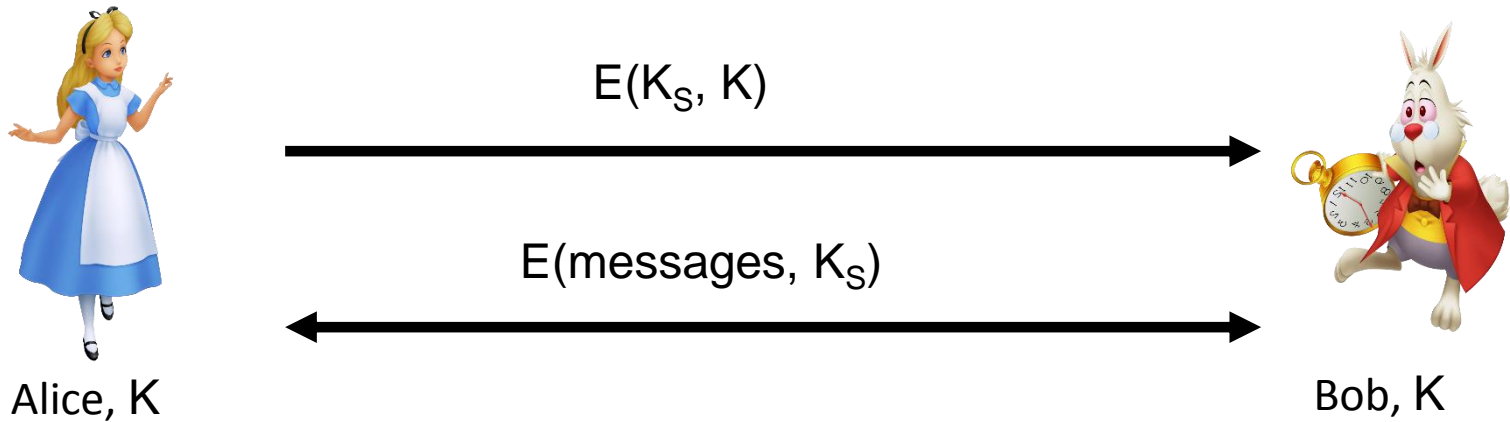
Perfect Forward Secrecy

- 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 ensures PFS?

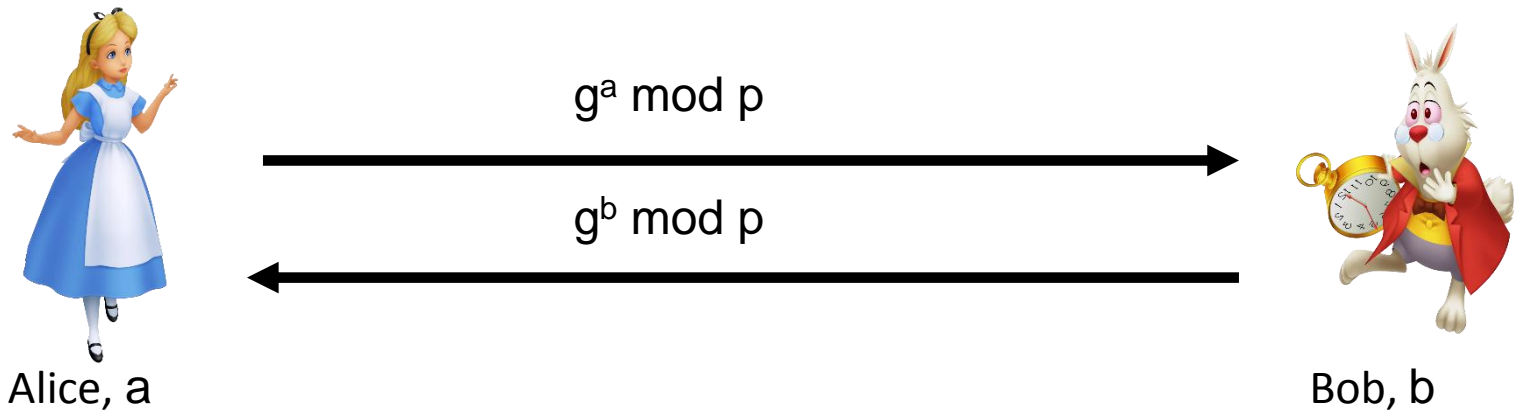
Naïve Session Key Protocol



- Trudy could record $E(K_S, K)$
- If Trudy later gets K then she can get K_S
 - Then Trudy can decrypt recorded messages

Perfect Forward Secrecy

- We 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

$E(g^a \text{ mod } p, K)$



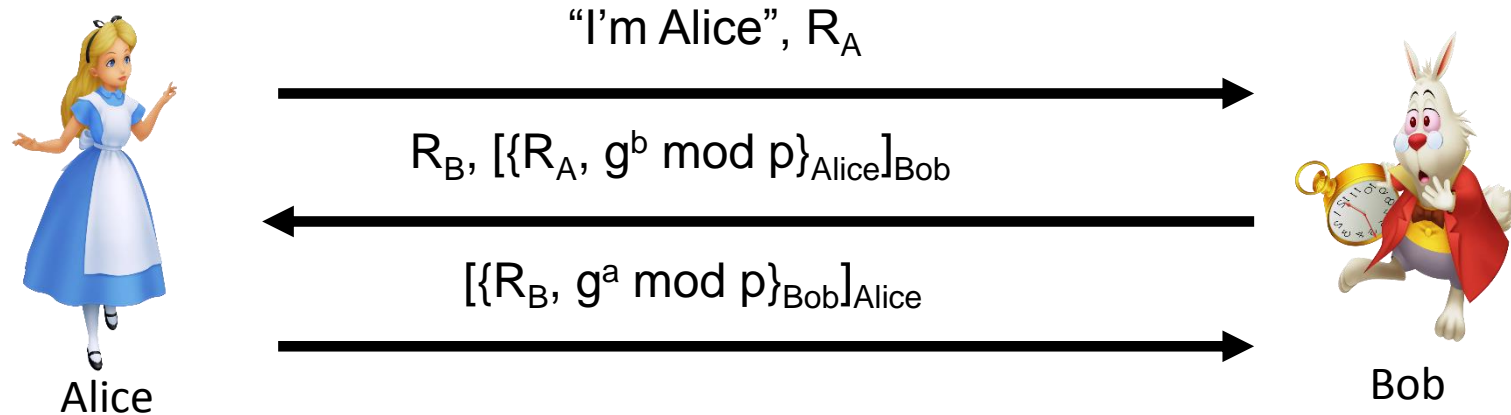
$E(g^b \text{ mod } p, K)$



Bob: K, b

- Session key $K_S = g^{ab} \text{ mod } p$
- Alice **forgets** a , Bob **forgets** b
- So-called **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



- Session key is $K = g^{ab} \text{ mod } p$
- 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 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)
- Clocks never exactly the same, 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?
- Seems to be OK

Public Key Authentication with Timestamp T



Alice

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



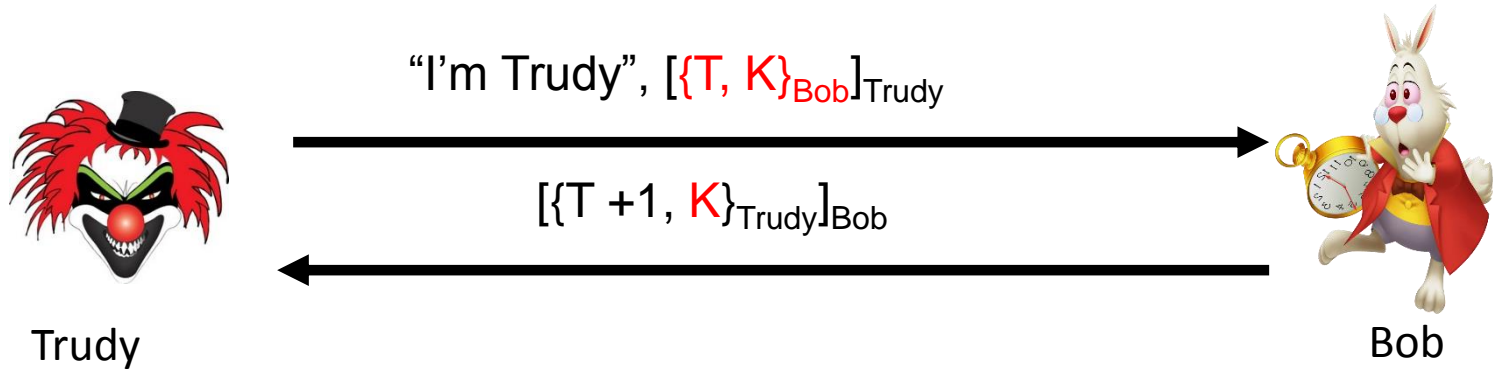
$[\{T + 1, K\}_{Alice}]_{Bob}$



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

- 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



Alice

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



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



Bob

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