홍익대학교
HONGIK UNIVERSITY

Network Security

<CH 6>

Youn Kyu Lee
Hongik University

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: TLS, IPSec, Kerberos, etc.

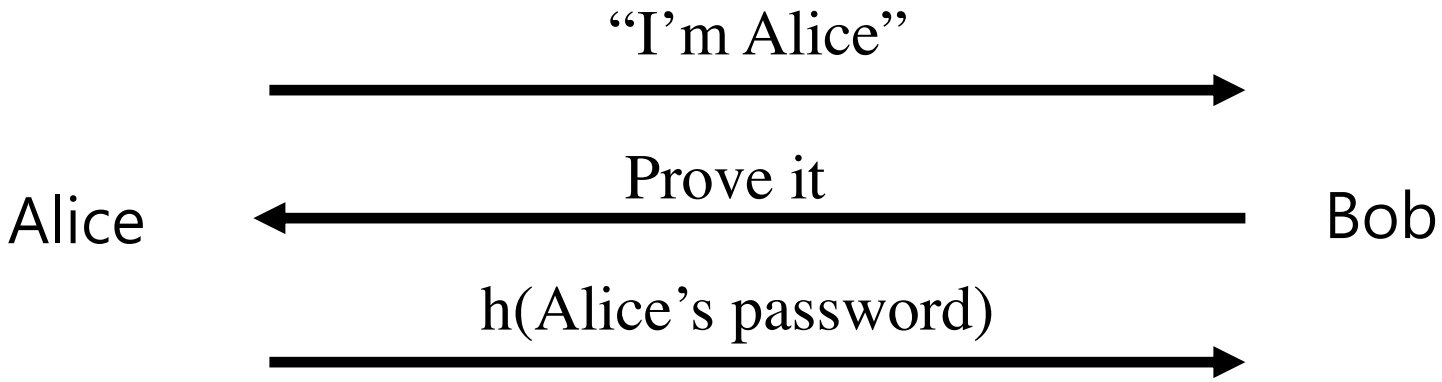
Protocol

- Protocol flaws can be very **subtle**
- Several (old) well-known security protocols have significant flaws
  - Including WEP, GSM, and early IPSec
- Implementation errors can occur
  - 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 and implement, flexible…
- Difficult to satisfy all of these!

Very Simple Authentication



- it hides Alice's password from Trudy(eavesdropper)
- But subject to "replay"
- generally, authentication over Internet is challenging
  - Attacker can passively observe messages
  - Attacker can replay, reflect messages
  - Active attacks possible (insert, delete, change)

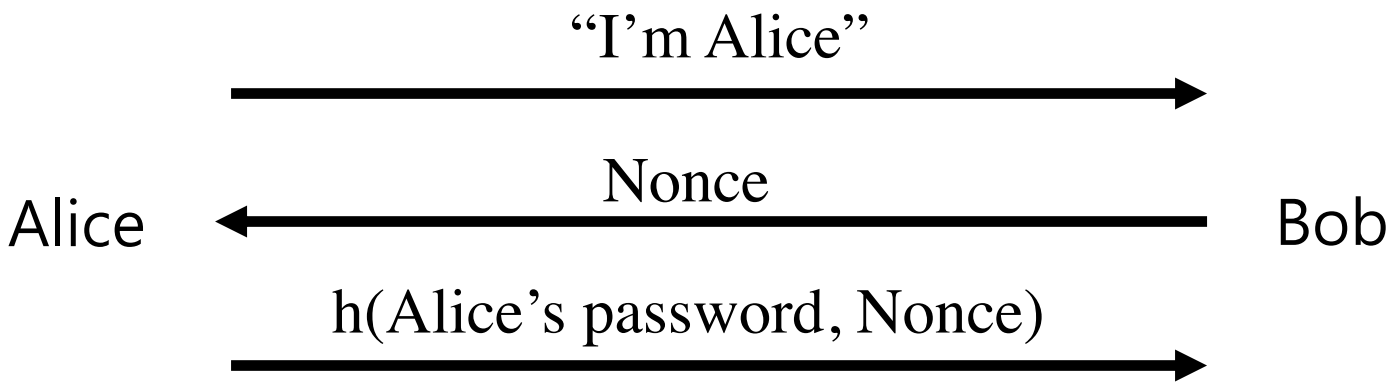
Challenge-Response

- 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

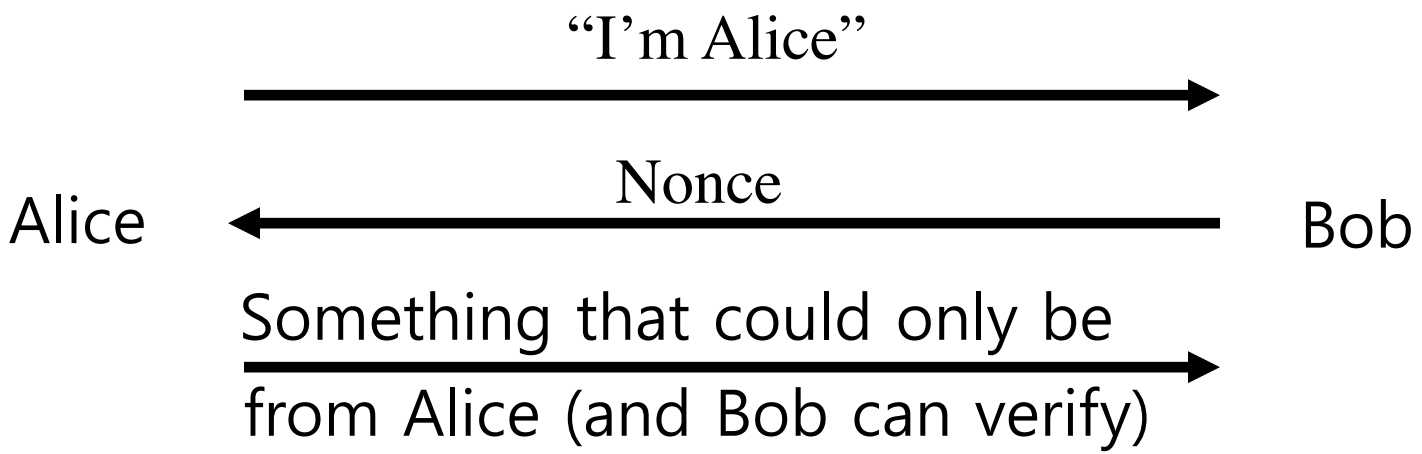
- To ensure freshness, can employ a **nonce**
  - Nonce == **n**umber used **once**
- What to use for nonces?
  - That is, what is the challenge?
- What should Alice do with the nonce?
  - That is, how to compute the response?
- How can Bob verify the response?
- Should we rely on passwords or keys?

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 pwd to verify

Generic Challenge-Response



- In practice, how to achieve this?
- Hashed password works, but…
- Encryption also can be applied here

Case Study: PKES(Passive Keyless Entry Systems)

Car controller → Key fob: N
Car controller ← Key fob: ID_{KeyFob} , $E_K(ID_{KeyFob,}$ N)
(K: crypto key shared between key fob(w/ ID_{KeyFob}) and car controller)

- Allow keyless entry and engine start with a push button(or touch screen)
- If you walk up to your car, it will unlock and start automatically when you push the button; if you walk away, it will lock

Case Study: PKES(Passive Keyless Entry Systems)

Car controller → Key fob: N
Car controller ← Key fob: ID_{KeyFob} , $E_K(ID_{KeyFob,}$ N)
(K: crypto key shared between key fob(w/ ID_{KeyFob}) and car controller)

- Thieves can use devices amplifying or relaying the signals -> then?

(countermeasure) new radio protocol based on UWB, which measures the distance from the key fob to the car with a precision of 10cm up to a range of 150m : but it's very complex to do properly

Two-factor Authentication

- An example: Password generators(P) + PIN to logon to corporate computer systems

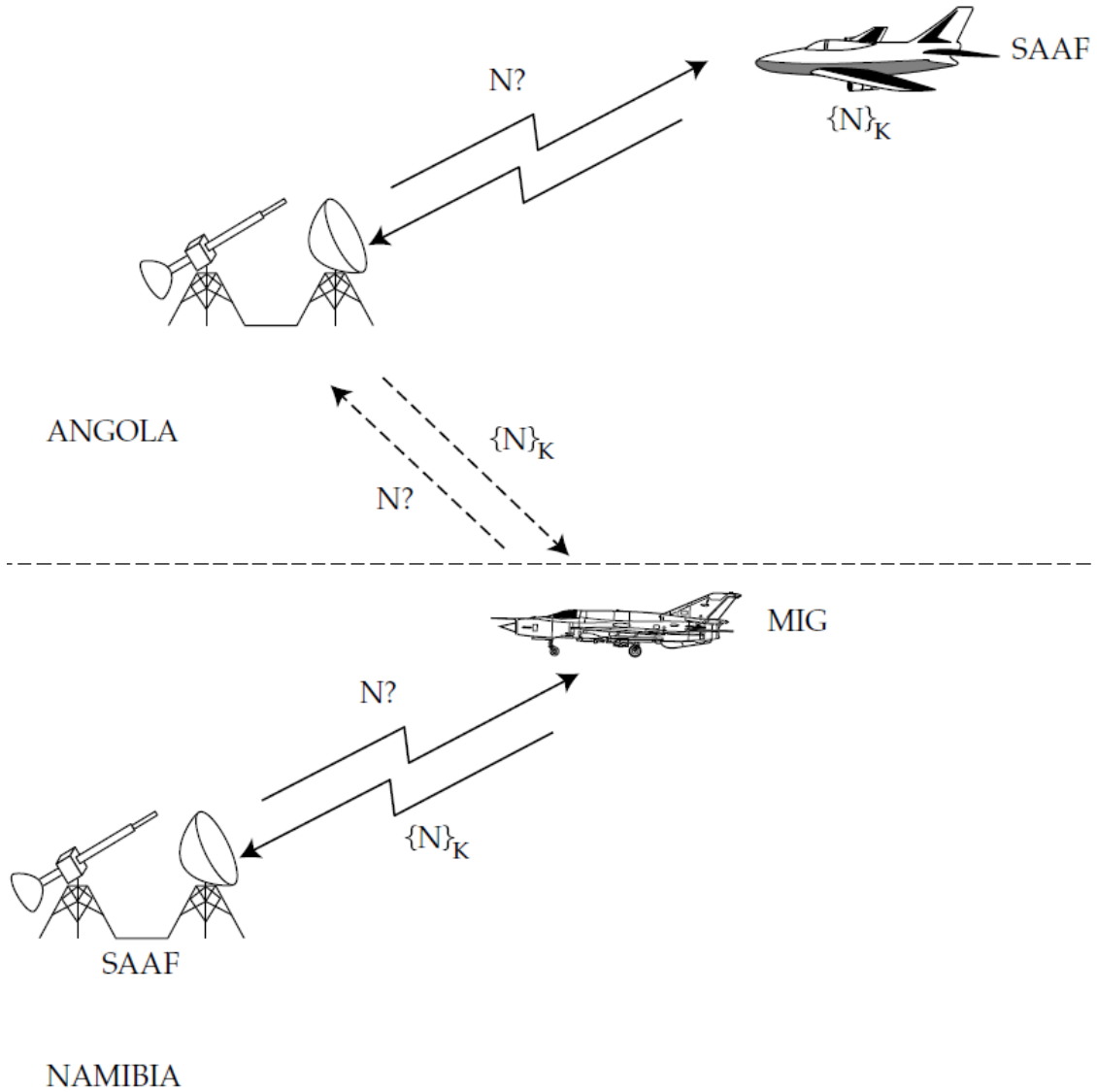
$S \rightarrow U$: N

$U \rightarrow P$: N, PIN

$U \leftarrow P$: E_K(N, PIN)

$S \leftarrow U$: E_K(N, PIN)

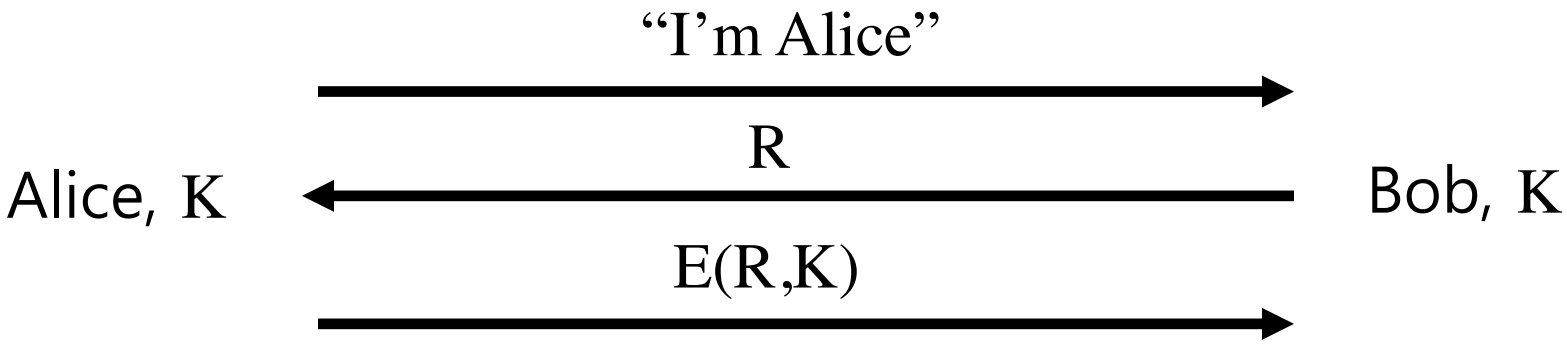
MIG-in-the-Middle Attack (based on “relaying”)



Authentication w/ Symmetric Key

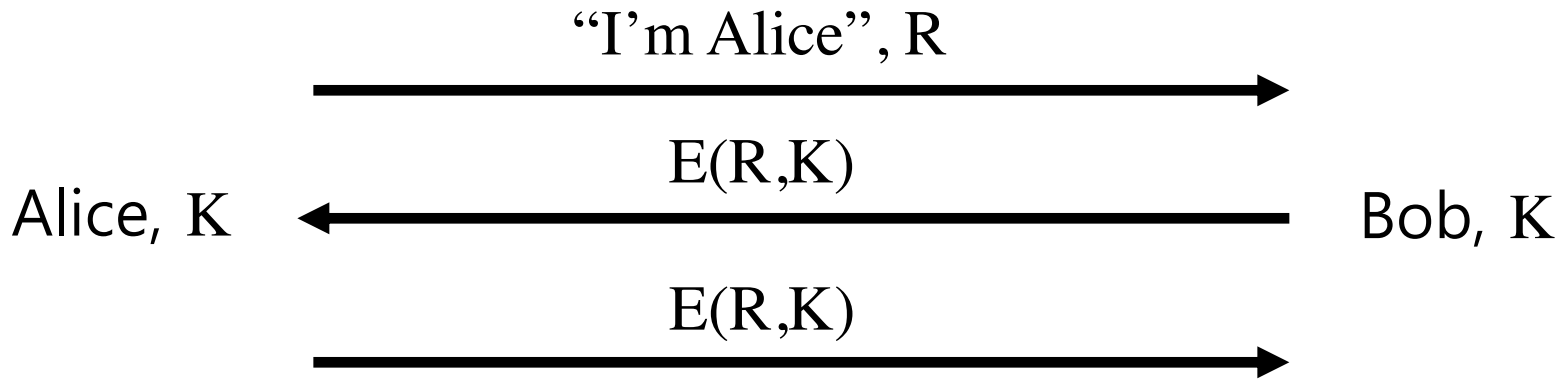
- Alice and Bob share symmetric key $\mathbf{K}$
- Key $\mathbf{K}$ known only to Alice and Bob
- Authenticate by proving knowledge of shared symmetric key
- How to accomplish this?
  - Cannot reveal key, must not allow replay (or other) attack, must be verifiable, …

Authentication with Symmetric Key



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

Mutual Authentication?



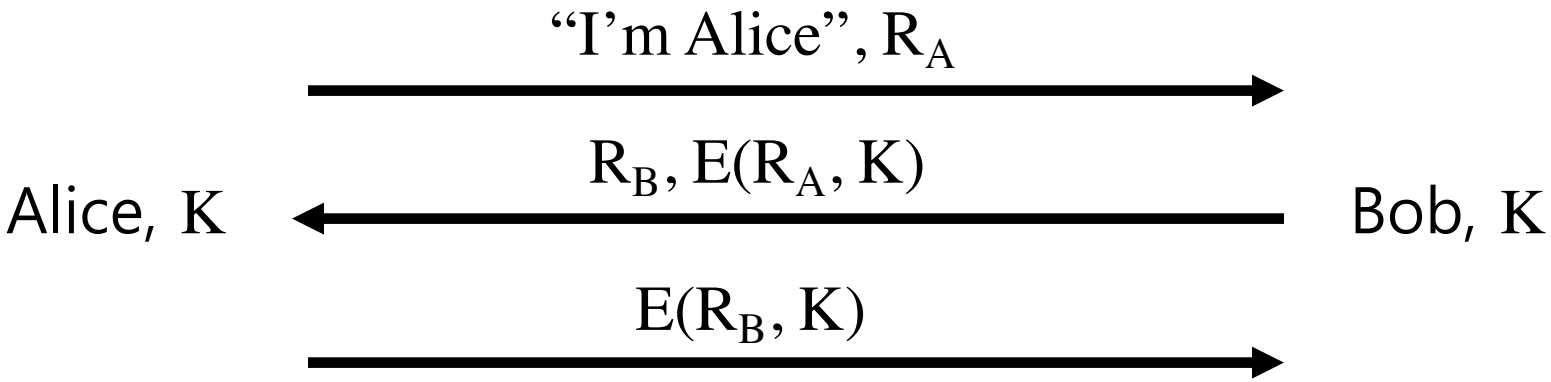
- What's wrong with this picture?

-"Alice" could be Trudy (or anybody else)!

Mutual Authentication

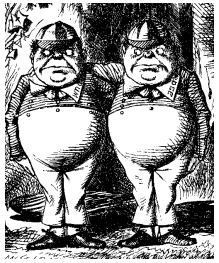
- Since we have a secure one-way authentication protocol…
- The obvious thing to do is to use the protocol twice
  - Once for Bob to authenticate Alice
  - Once for Alice to authenticate Bob
- This has got to work…

Mutual Authentication?



- This provides mutual authentication…
- …or does it?

Reflection Attacks in *Mutual* Authentication



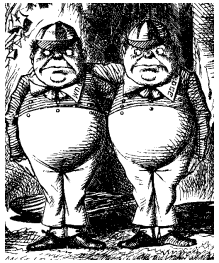
Trudy

1. "I'm Alice", R_A

2. R_B, $E_K(R_A)$

5. $E_K(R_B)$

Bob, K



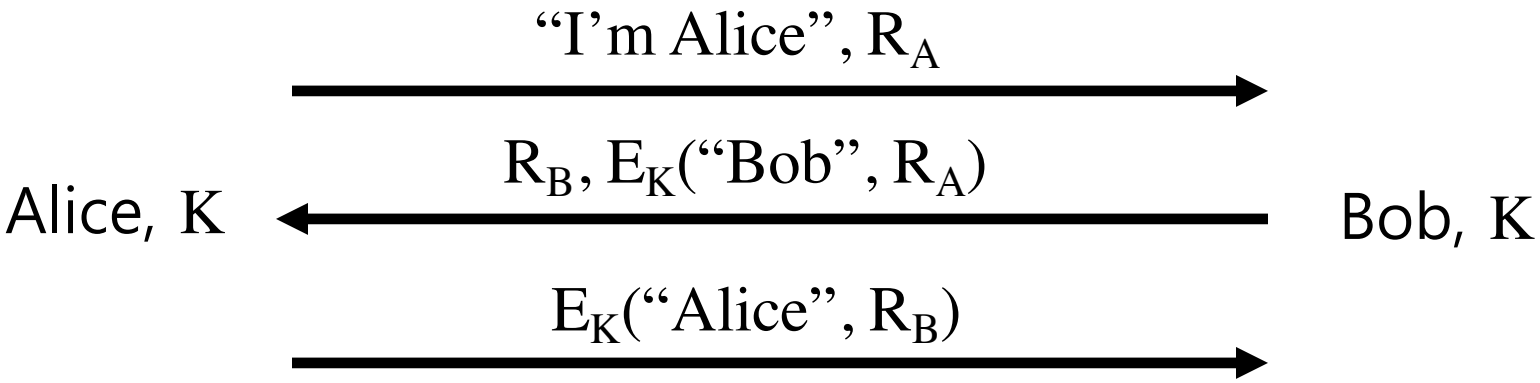
Trudy

3. "I'm Alice", R_B

4. R_C, $E_K(R_B)$

Bob, K

Reflection Attacks in *Mutual* Authentication



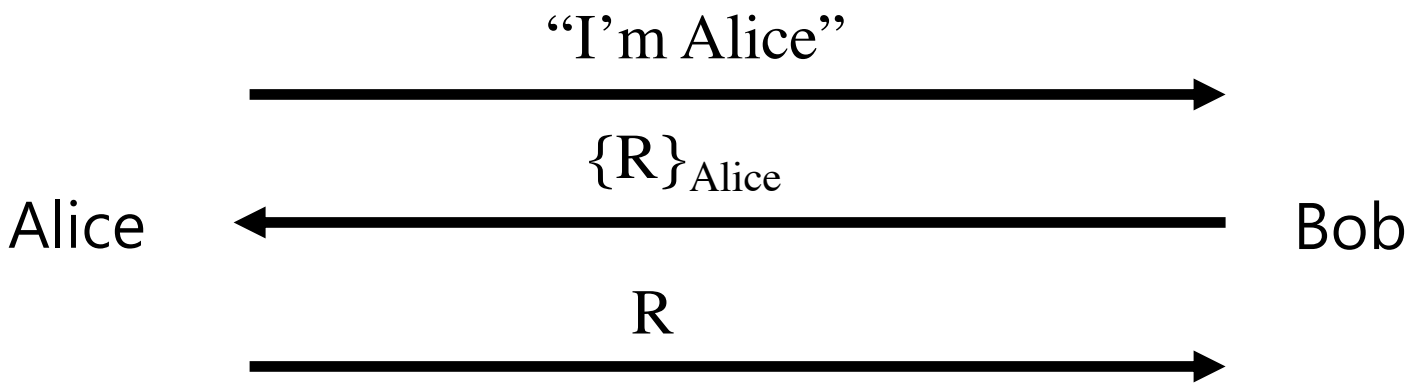
- Do these "insignificant" changes help?
- Yes!

More on Authentication Protocols

- Do we always need ‘mutual authentication’?
- Can we do it with public key cryto?
- What properties do we want in the authentication process?
  - Session key establishment
  - Perfect forward secrecy
  - Minimized # of messages exchanged
- What if environment changes?
  - This is a common source of security failure
  - Eg. ATM machine fraud:

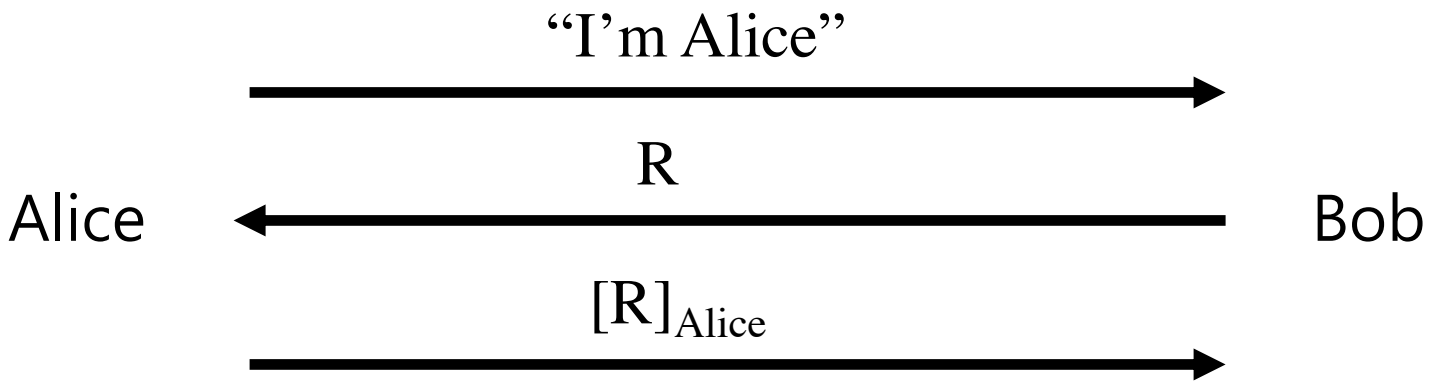
    magnetric-strip+PIN vs. chip+PIN

Public Key based Authentication



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

Public Key based Authentication



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

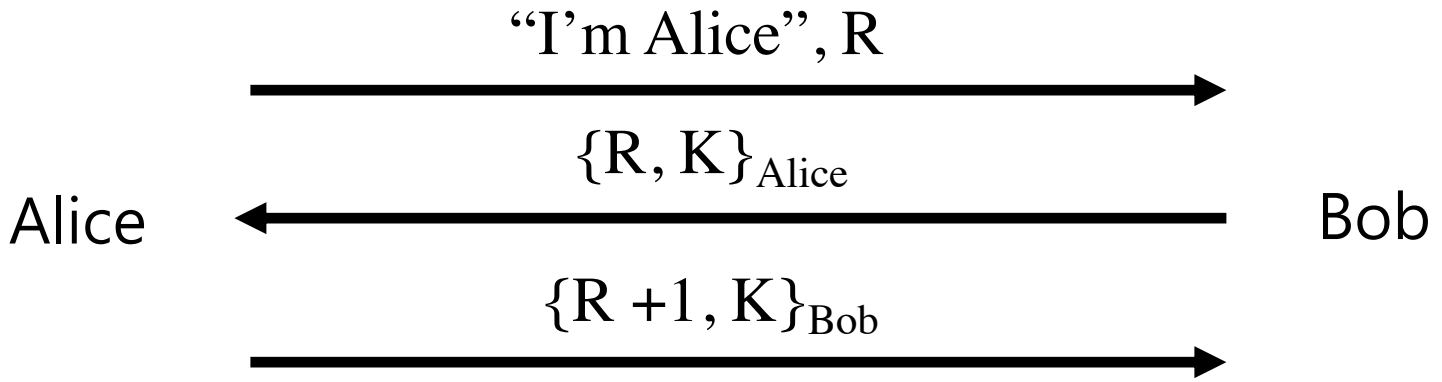
Public Keys

- Generally, it's a very bad idea to use the same key pair for encryption and signing
- Instead, you must have…
  - …one key pair for encryption/decryption…
  - …and a different key pair for signing/verifying signatures

Session Key

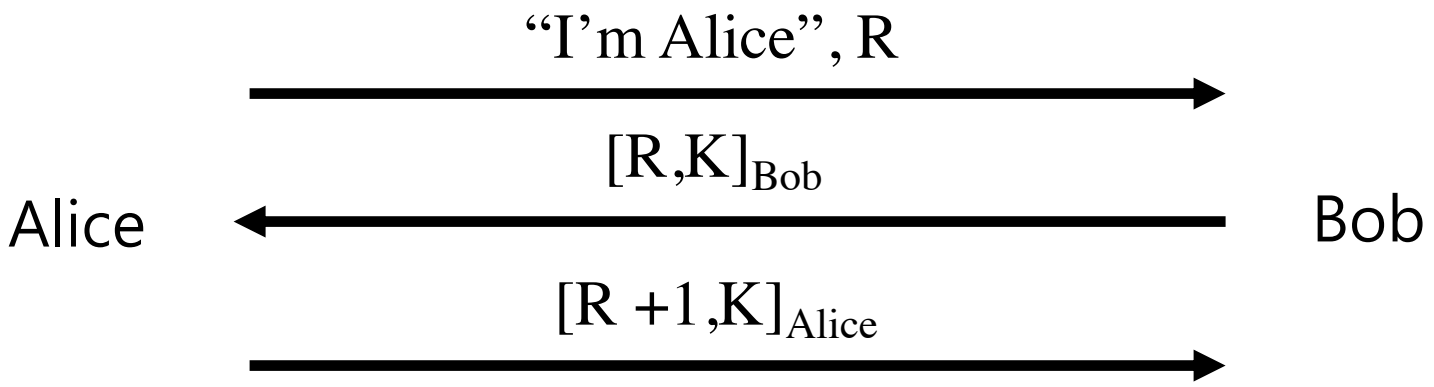
- Usually, a **session key** is required
  - i.e., a symmetric key for a particular session
  - Used for confidentiality and/or integrity
- How to authenticate AND establish a session key (i.e., shared symmetric key)?
  - When authentication completed, want Alice and Bob to share a session key
  - Trudy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

Authentication & Session Key



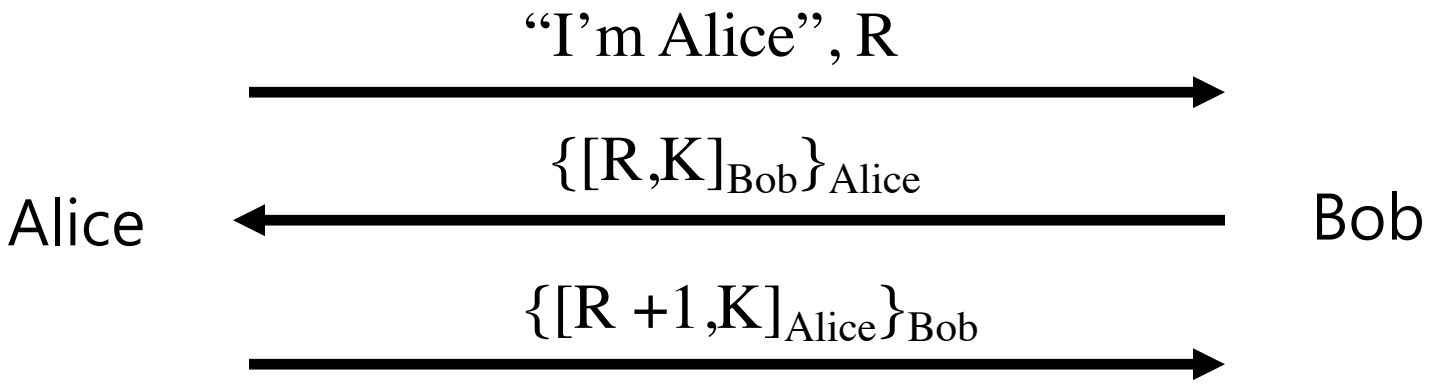
- Is this secure?
  - Alice is authenticated and session key is secure
  - Alice's "nonce", R, useless to authenticate Bob
  - The key K is acting as Bob's nonce to Alice
- No mutual authentication

Public Key based Authentication and Session Key



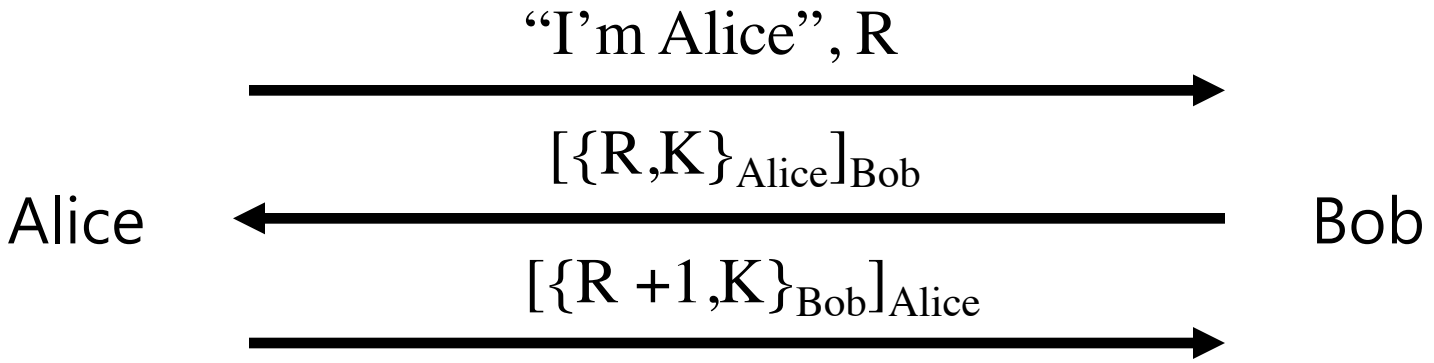
- Is this secure?
  - Mutual authentication (good), but…
  - … session key is not secret (very bad)

Public Key based Authentication and Session Key



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

Public Key based Authentication and Session Key

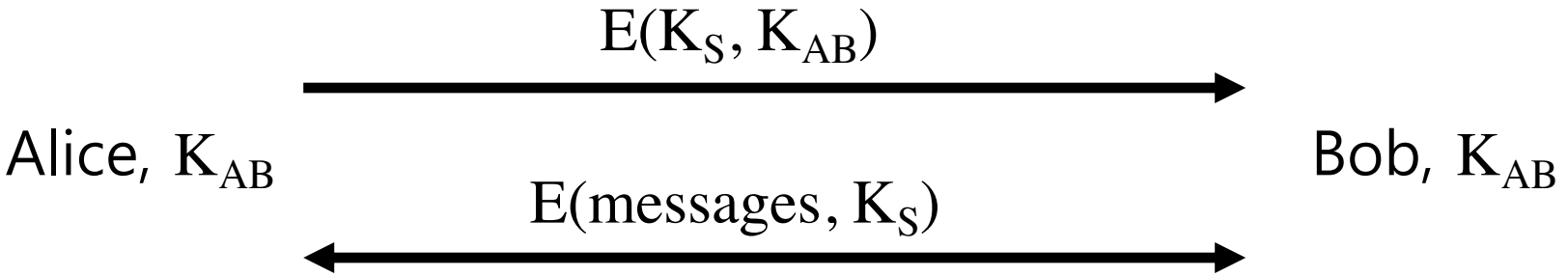


- Is this secure?
- Seems to be OK
  - Anyone can see $\{R,K\}_{Alice}$ and $\{R+1,K\}_{Bob}$

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 (guaranteed!)
  - Even if Trudy gets key K or other secret(s)
- Is PFS possible?

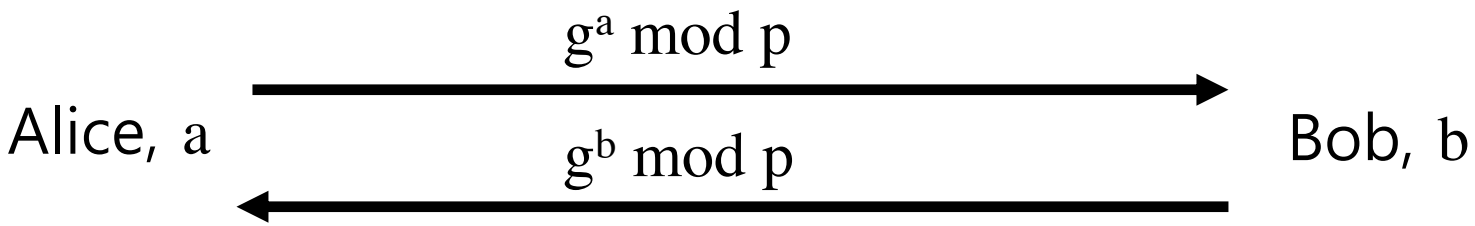
Naïve Session Key Protocol



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

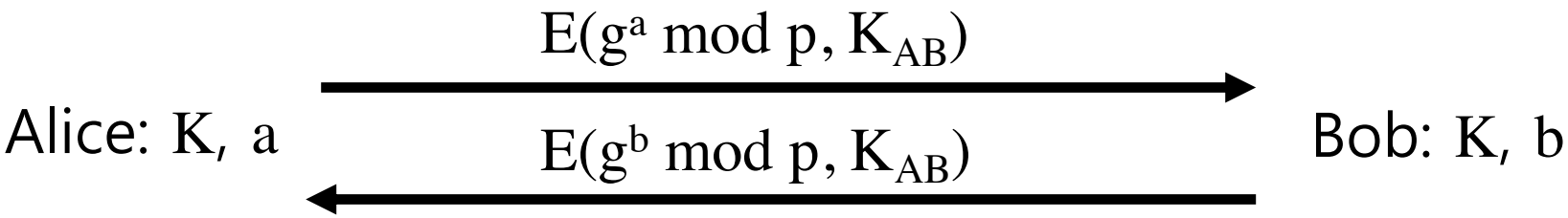
Perfect Forward Secrecy

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



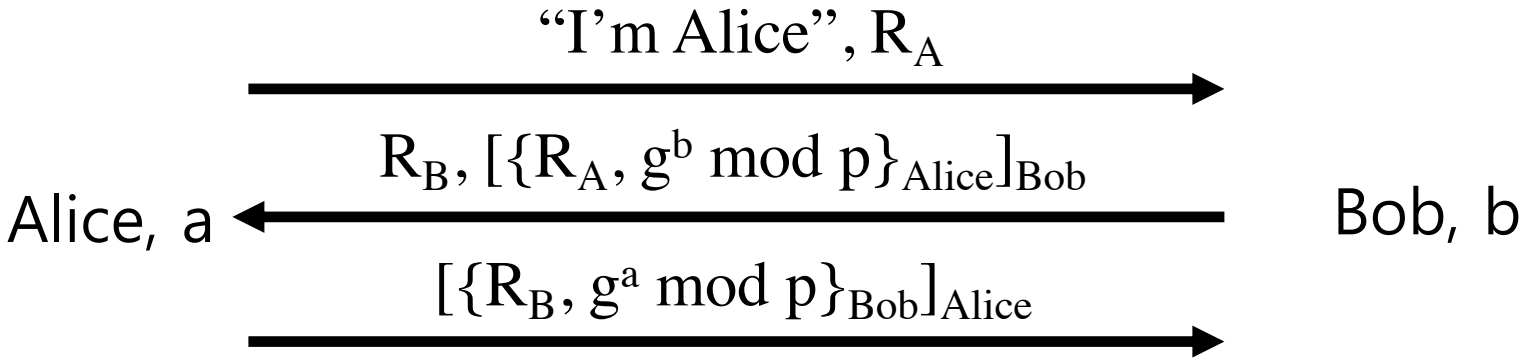
- But Diffie-Hellman is subject to MiM
- How to get PFS and prevent MiM?

Perfect Forward Secrecy



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

Mutual Authentication, Session Key and PFS

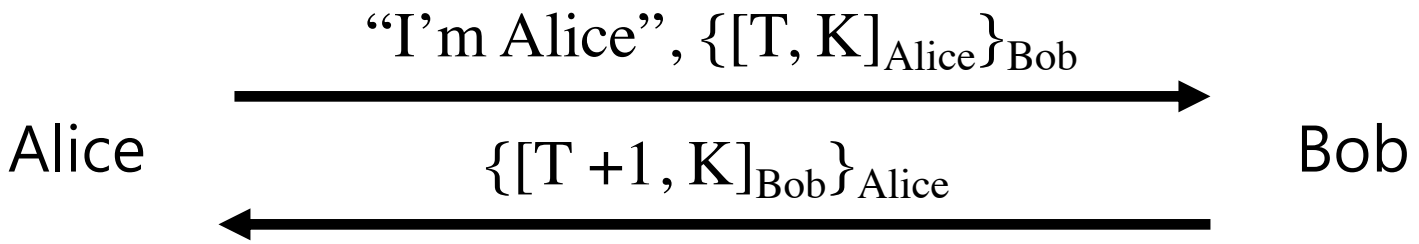


- Session key is $K = g^{ab} \bmod p$
- Alice forgets a and Bob forgets b
- If Trudy later gets Bob's and Alice's secrets, she cannot recover session key K

Timestamps

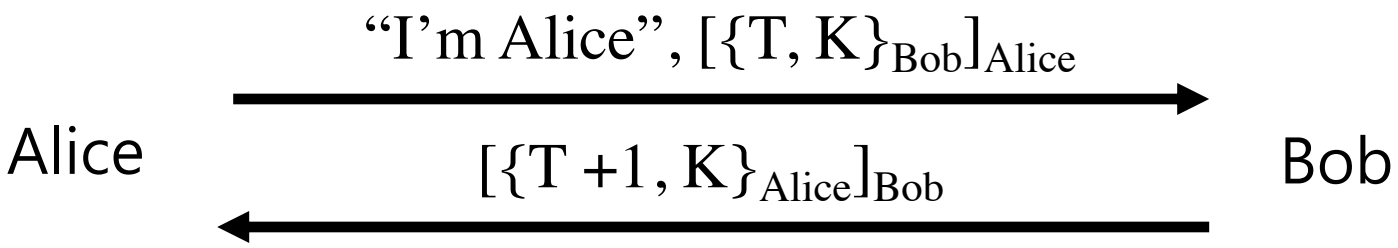
- A timestamp T is derived from current time
- Timestamps used in some security protocols
  - Kerberos, for example
- Timestamps reduce number of messages (good)
  - Like a nonce that both sides know in advance
- "Time" is a security-critical parameter
- Clocks never exactly the same, so must allow for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

Public Key based Authentication with Timestamp T



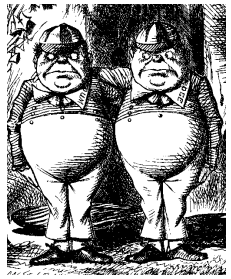
- Secure mutual authentication?
- Session key?
- Seems to be OK

Public Key based Authentication with Timestamp T

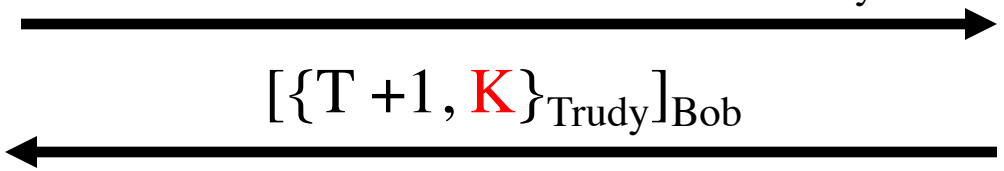


- Secure authentication and session key?
- Trudy can use Alice's public key to find $\{T, K\}_{Bob}$ and then…

Public Key based Authentication with Timestamp T



Trudy



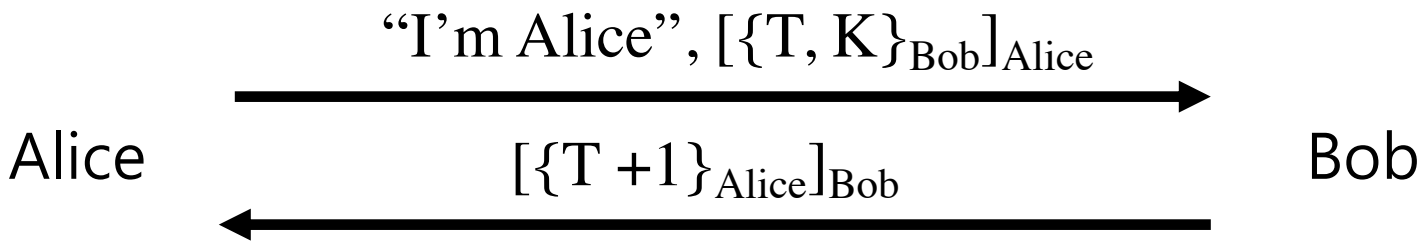
Bob

- 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 based Authentication with Timestamp T



- - Is this "encrypt and sign" secure?
  - Yes, seems to be OK

Manipulating the Message

An example:

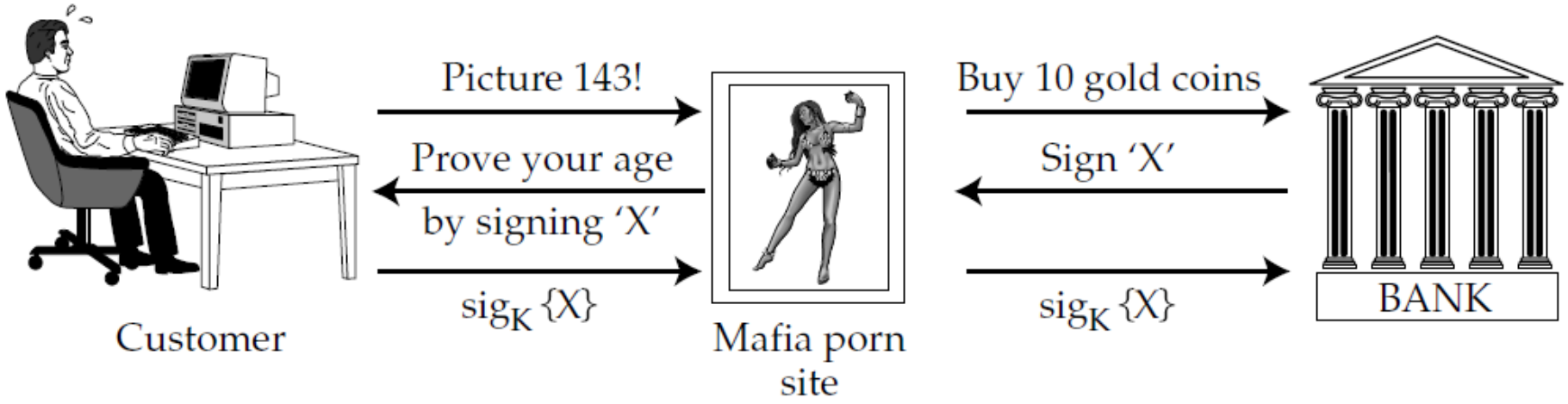
- Lots of households in UK + other countries have an electricity meter that accepts encrypted tokens: the householder buys a magic number and types it into the meter, which then dispenses the purchased quantity of energy.

- One "early" meter widely used in South Africa checked only that the nonce was different from last time.

- So the customer could charge their meter indefinitely by buying two low-value power tickets and then …

Chosen Protocol Attacks

- "Multifunction authentication" devices for being used in a wide range of transactions, such as, ID, banking, transport ticketing, …
- Now, what if users can be fooled into reusing the same token or crypto key … in *other applications*?



The Mafia-in-the-Middle Attack

Remote Key Management

- Key distribution protocols eg. Kerberos
- Assume existence of a trusted 3rd party, say S(am)

- A simple authentication protocol using S(am):

T: timestamp

$A \rightarrow S$: A, B

$A \leftarrow S$: $E_{K_{AS}}(A,B, K_{AB},T)$, $E_{K_{BS}}(A,B, K_{AB},T)$

$A \rightarrow B$: $E_{K_{BS}}(A,B, K_{AB},T)$, $E_{K_{AB}}(MSG)$

Remote Key Management

- *Needham-Schroeder* Protocol

$A \rightarrow S: A, B, N_A$

$A \leftarrow S: E_{K_{AS}}(N_A, B, K_{AB}, E_{K_{BS}}(K_{AB}, A))$

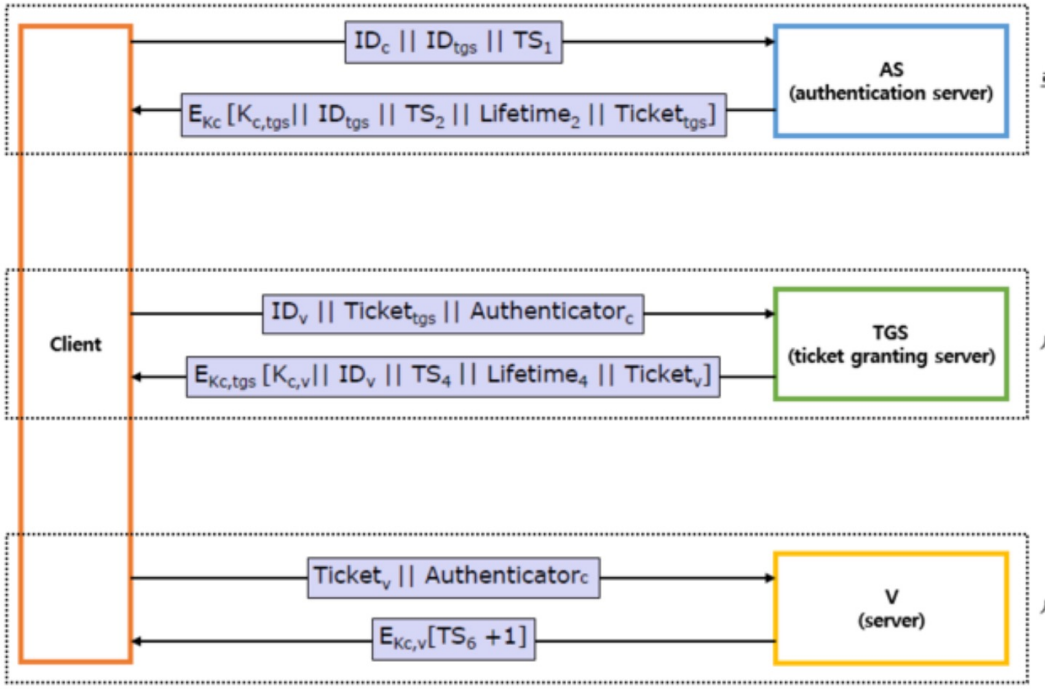
$A \rightarrow B: E_{K_{BS}}(K_{AB}, A)$

$A \leftarrow B: E_{K_{AB}}(N_B)$

$A \rightarrow B: E_{K_{AB}}(N_B - 1)$

Remote Key Management

- *Kerberos* : a distributed access control system that originated at MIT (become part of the basic mechanisms of authentication over a LAN for both Windows and Linux)
- authentication servers and ticket granting servers



Remote Key Management

Kerberos

A → S: A, B // S: ticket granting server

A ← S: $E_{K_{AS}}(T_S, L, K_{AB}, B, E_{K_{BS}}(T_S, L, K_{AB}, A))$

A → B: $E_{K_{BS}}(T_S, L, K_{AB}, A), E_{K_{AB}}(A, T_A)$

A ← B: $E_{K_{AB}}(T_A+1)$

T_S : timestamp included by S

L : lifetime

T_A : timestamp included by A

-What if clocks on clients and servers get out of sync?

Q & A

aiclasshongik@gmail.com