Authentication

Authentication

- How to prove that you are whom you claim to be?
  - Indispensible for access control

- User authentication
  - Allows a user to prove his identity
  - E.g., telephone voice authentication
  - Prover: entity to be authenticated
  - Verifier: entity checking pover identity
  - Proof: information allowing prover to be authenticated (prove his ID)
- Message authentication
  - Verify that a message is authentic
  - E.g., receiving a signed email

Authentication mechanisms

- Identification: who are you?
- Authentication: prove it
- Authorization: you can do it

- Password
  - Fixed password
  - One-time password
- Cryptographic authentication protocols: challenge-response
  - A prover proves demonstrates knowing a seret
    - Symmetric key, private key
- Biometrics

Password-based authentication

- User demonstrates knowledge of password to authenticate
  - most common method of user authentication
- A password should be easy to remember, but hard to guess: difficult!

- Password storage
  - Storing unencrypted passwords in a file is risky
  - Store H(pwd): when user inputs password, compare hash with stored hash

Attacks on password

- Intellegent search

- 10 most popular pwds

  password

  123456

  qwerty

  abc123

  letmein

  monkey

  myspace1

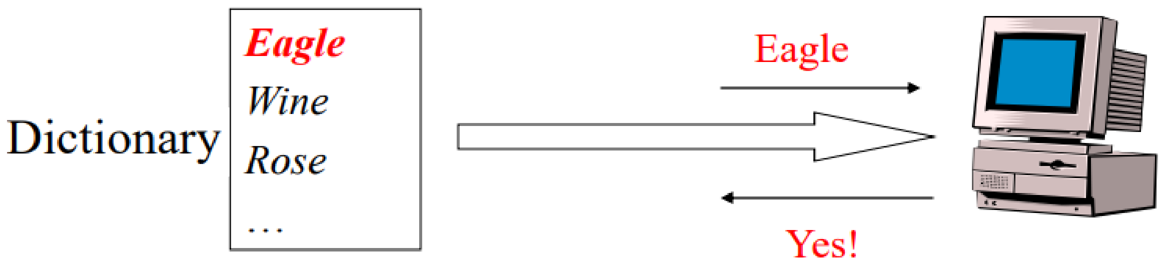
  Password1 (passw0rd)

  blink182

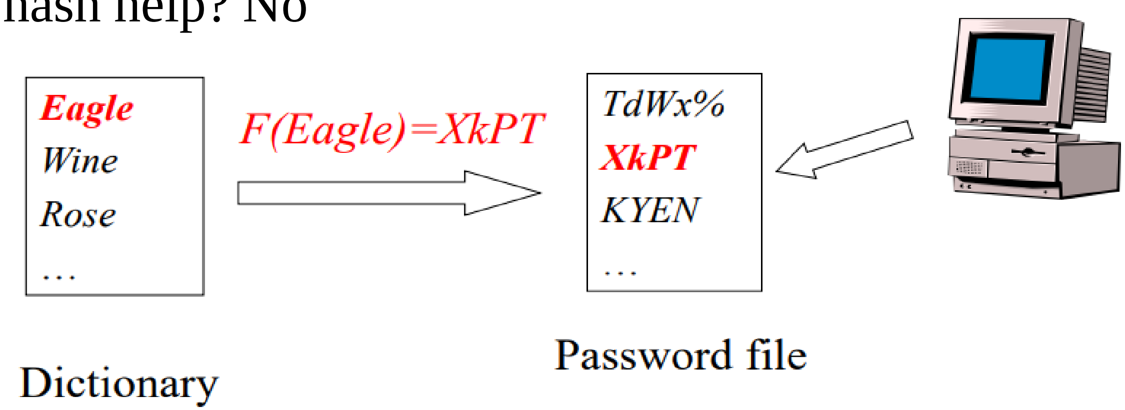
  name

Dictionary Attack

- Online
  - Create a dictionary of commonly used pwds
  - Use these to guess the password

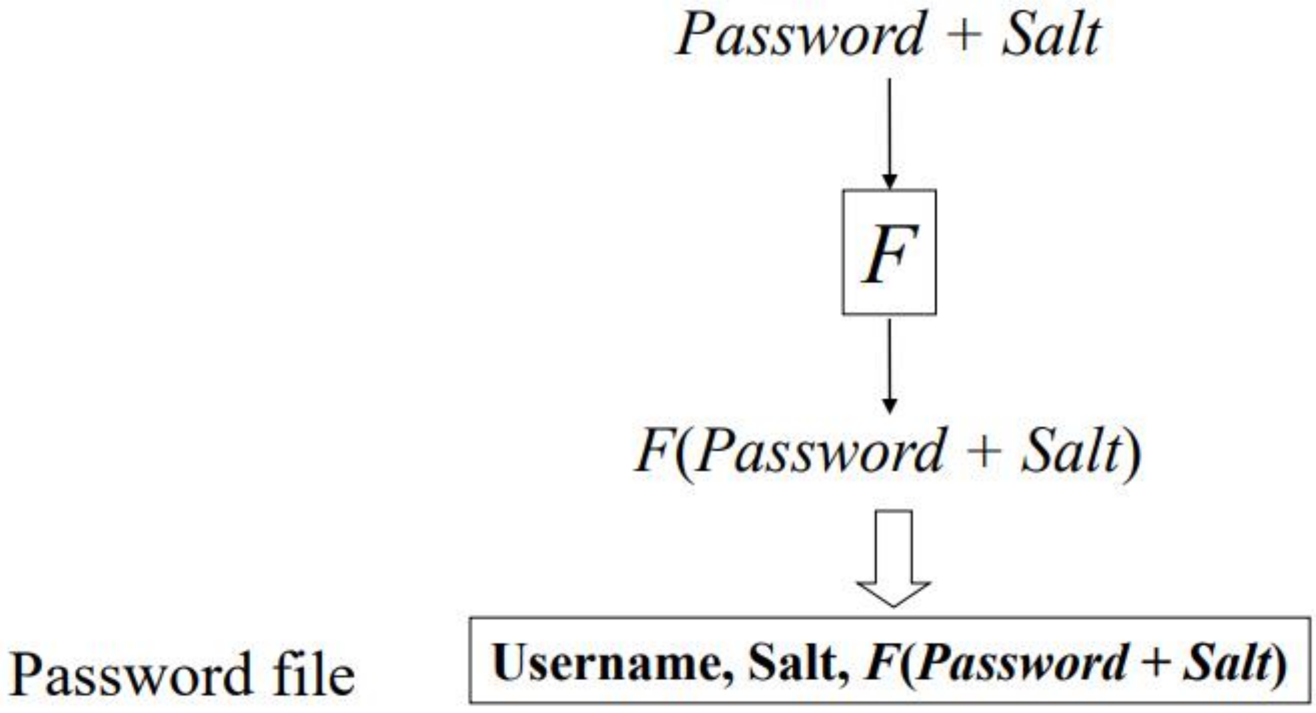


- Offline
  - Obtain the pwd file
  - Compare with the dictionary
  - Does hash help? No



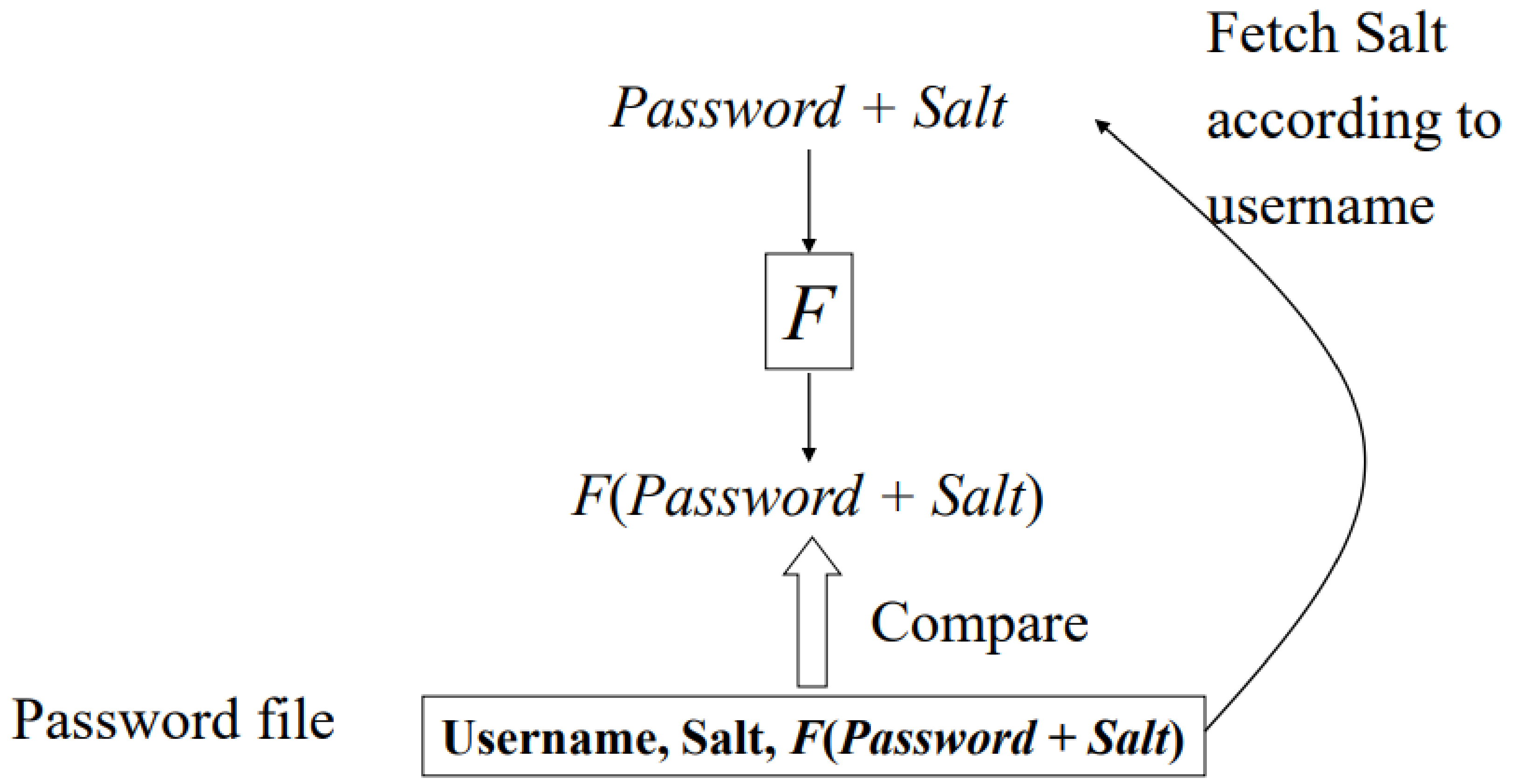
Password Salt

- Storing passwords



Password salt

- Verifying passwords



- Dictionary attack?

Password recommendation

- Initial passwords are system-generated, change on first login
- Change periodically
- Passwords vulnerable to a dictionary attack are rejected
- Do not use same password on multiple sites

- Other password attacks
  - Eavesdropping
  - Backdoor
  - Phishing

One-time password

a
secret: w
hash function: h
number of authentication: 3

b
hash function: h
number of authentication: 3
counter for a : compt_a

Initialisation

- compute $w_0 = h^3(w)$

w_0 — Secure channel → b

- store w_0
- compt_a := 1

Authentication 1

- compute $w_1 = h^2(w)$

$A, 1, w_1$ → b

- check compt_a= 1
- check $w_0 = h(w_1)$
- store w_1, compt_a := 2

Authentication 2

- compute $w_2 = h(w)$

$A, 2, w_2$ → b

- check compt_a= 2
- check $w_1 = h(w_2)$
- store w_2, compt_a := 3

Authentication 3

- use $w_3 = w$

$A, 3, w_3$ → b

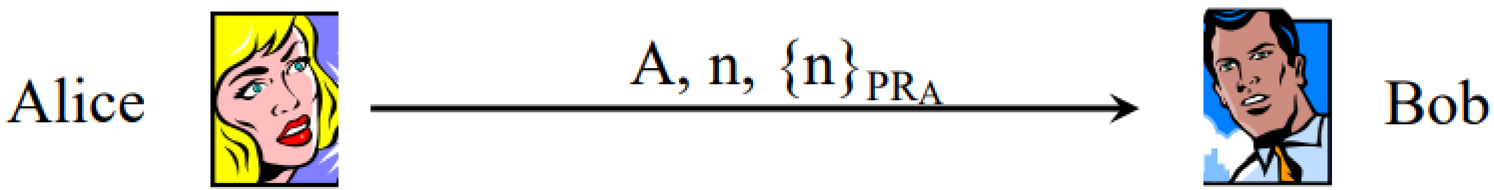
- check compt_a= 3
- check $w_2 = h(w_3)$
- compt_a := 0

A should generate new secret …

Cryptographic authentication protocols

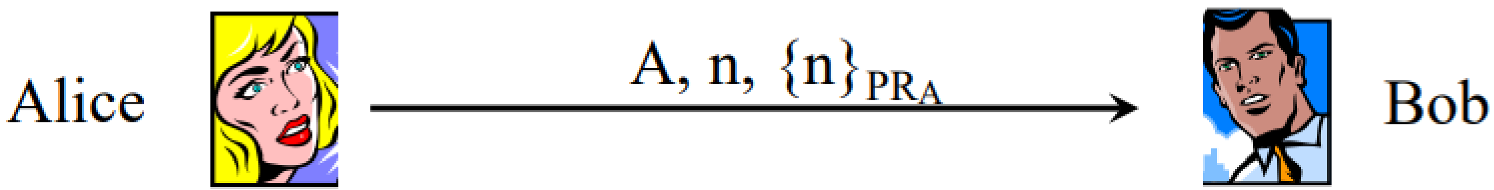
- Goals:
  - Mutual Authentication: each party authenticates itself to the other
  - Key Establishment: establish a session key
    - used to secure communication
- Methods
  - Authentication with asymmetric keys
    - Public key is known to everyone
  - Authentication with symmetric keys
    - A pre-shared secret key
- Attacker can do
  - Message injection, modification, deletion, replay

Authentication with asymmetric keys



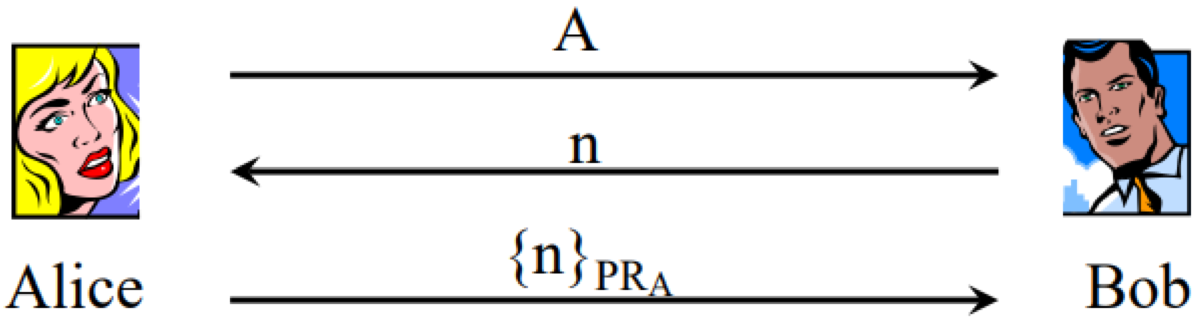
- Private key PR_A, Public key PU_A
- n: nounce
  - An ideal nounce has two properties:
    - Freshness: each nounce is used once during any execution of protocol
    - Unpredictability
  - In practice, it is simulated by a large random number
  - Sometimes we only need freshness
    - Increasing sequence number
    - Time stamp

Version 1



- Can we replace $\{n\}_{PRA}$ by $\{n\}_{PUA}$?
  - Answer: No. Everyone knows PU_A and can compute $\{n\}_{PUA}$.
- What is wrong with this authentication protocol?
  - An attacker can replay this message later to authenticate himself to Bob
  - How to fix this problem?

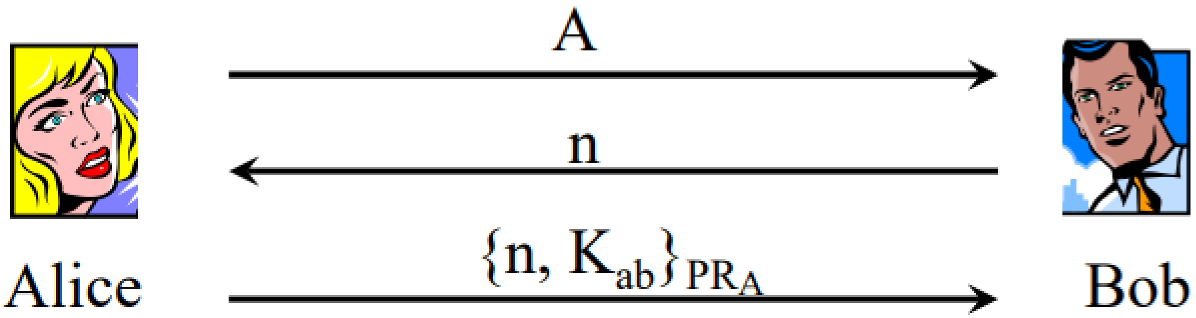
Version 2



- Attacker cannot replay $\{n\}_{PRA}$

- What is wrong with this authentication protocol?
  - No session key is established.
  - Authentication = mutual identity verification + session key establishment
  - How to fix this problem?

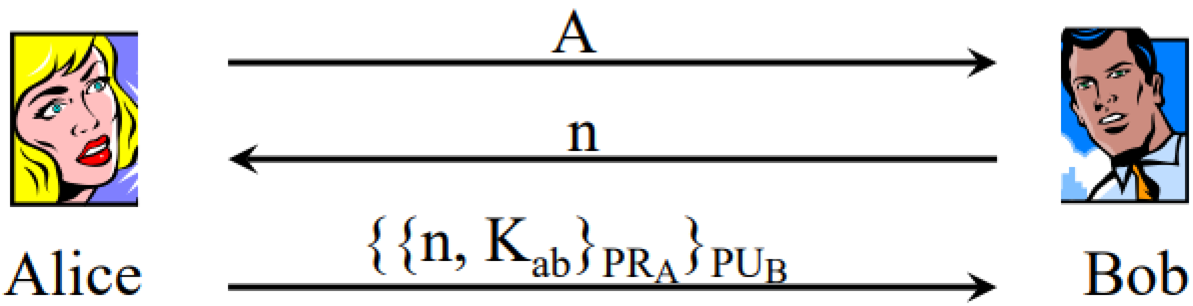
Version 3



K_{ab} denotes a session key

- What is wrong with this authentication protocol?
  - Attacker can see K_{ab} by Alice’s public key.
  - How to fix this problem?

Version 4

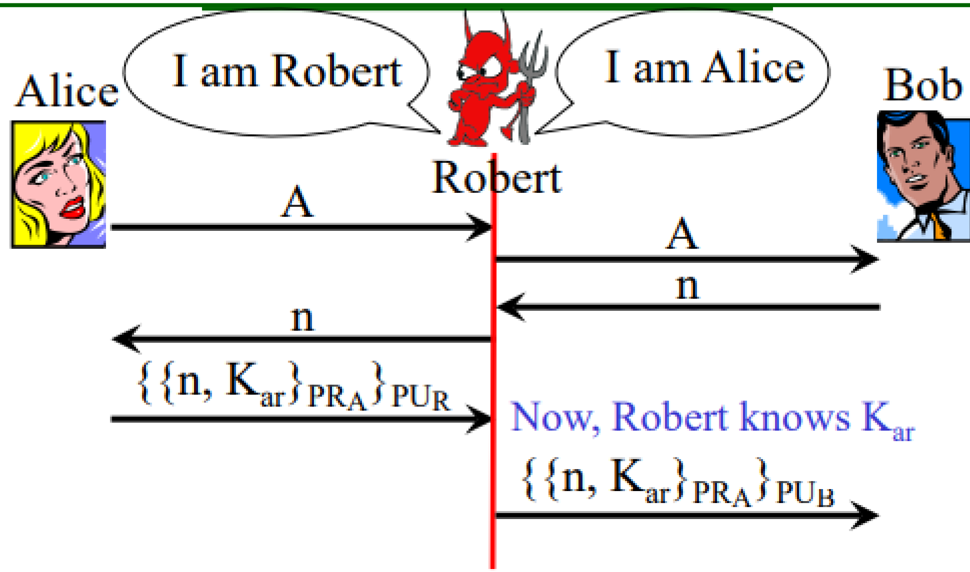


K_{ab} denotes a session key

- Only Bob can decrypt $\{\{n, K_{ab}\}PR_A\}PU_B$
- Protocole developed by Denning & Sacco in 1981

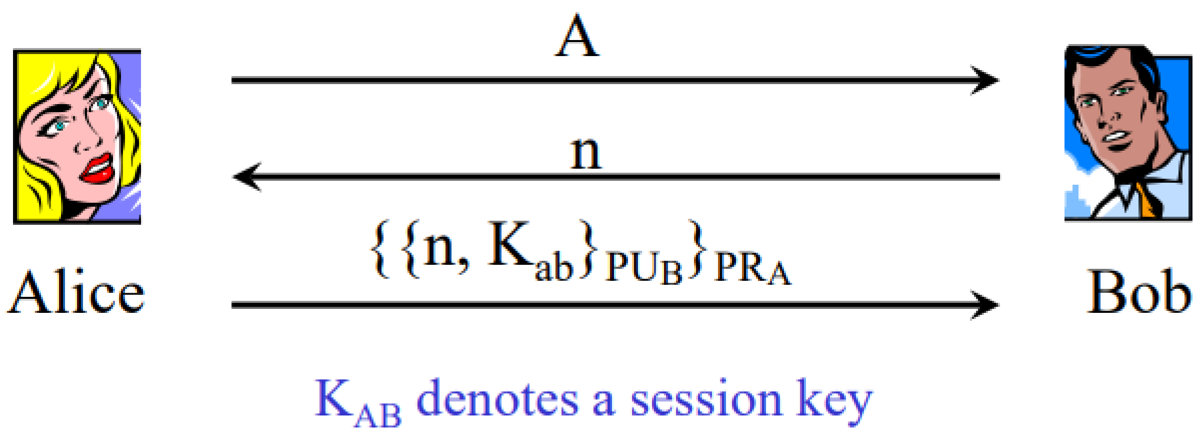
- What is wrong with this authentication protocol?
  - Vulnerable to man-in-the-middle attack
    - attacker makes independent connections with the victims and manipulates messages between them

Man-in-the-middle attack



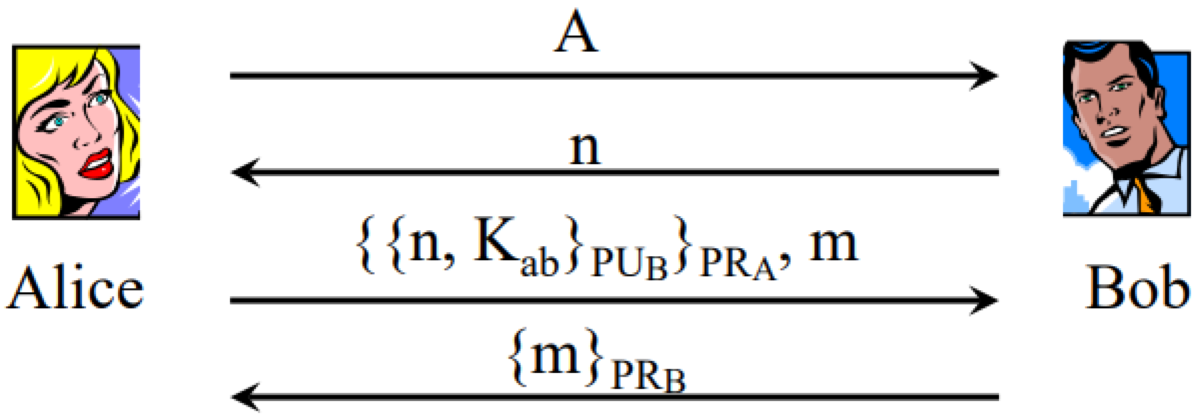
- When Alice begins to talk to Robert, Robert starts to talk to Bob as Alice
- How to fix this problem?
  - Solution 1: use $\{\{n, K_{ar}\}_{\mathbf{PUR}}\}_{\mathbf{PRA}}$ to replace $\{\{n, K_{ar}\}_{PRA}\}_{PUR}$
  - Solution 2: use $\{\{n, \mathbf{R}, K_{ar}\}_{PRA}\}_{PUR}$ to replace $\{\{n, K_{ar}\}_{PRA}\}_{PUR}$
- Principle: Encryption should be inside a signature, otherwise we need to include principal's names.

Version 5



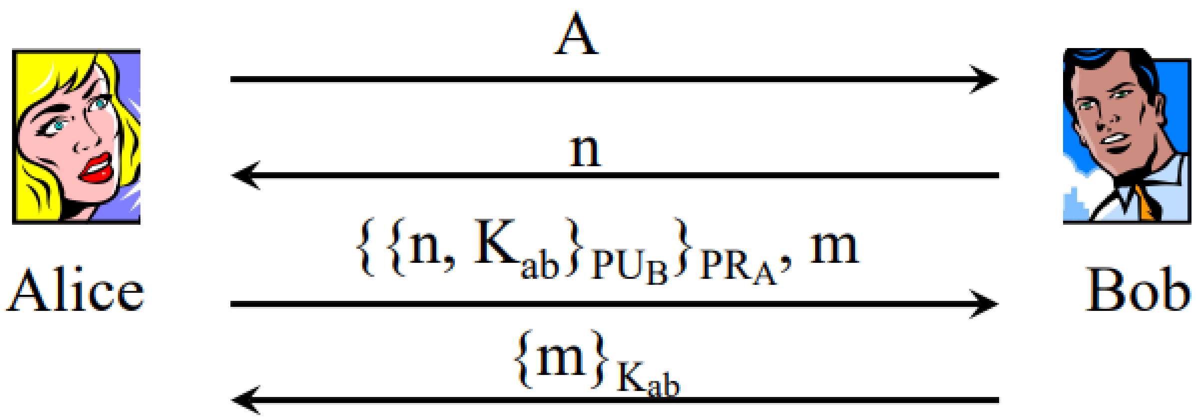
- Now only Alice and Bob know session key K_{ab}
- What is wrong with this authentication protocol?
  - Authentication = mutual identity verification + session key establishment
  - Bob authenticates Alice, but Alice did not authenticate Bob.
  - How to fix this problem?

Version 6

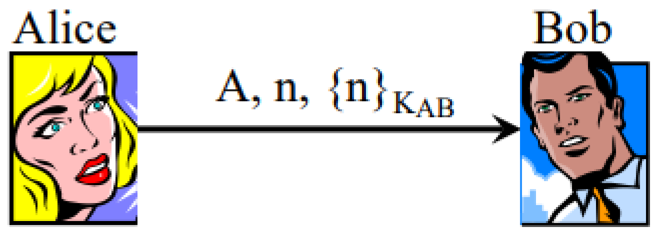


- Now, mutual authentication, and session key established
- Which part of this protocol can be made more efficient?
  - Replace $\{m\}_{PRB}$ by $\{m\}_{Kab}$.
  - Note: attacker can launch man-in-the-middle attack, but cannot learn K_{ab}

Version 7: final

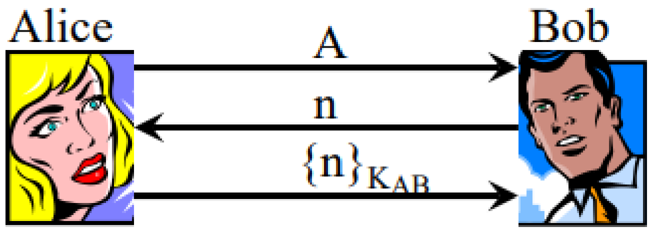


Authentication with symmetric key: V1



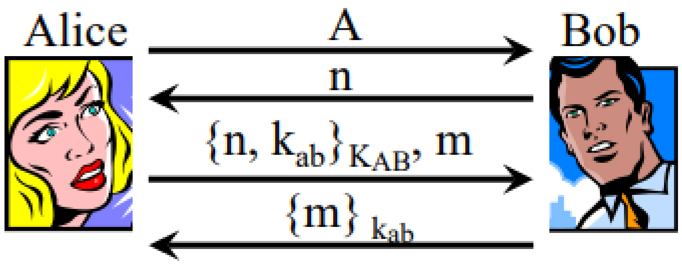
- What is wrong with this authentication protocol?
  - Answer: vulnerable to replay attack
  - How to fix this problem?

Version 2



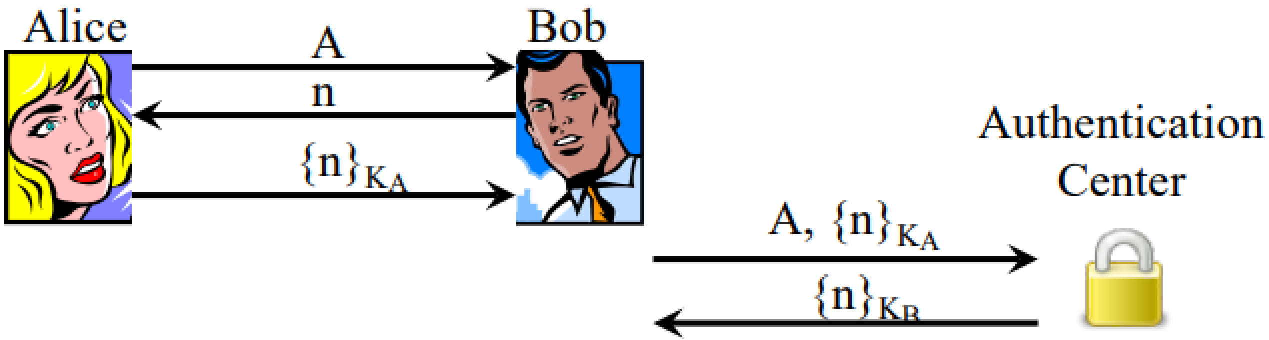
- What is wrong with this authentication protocol?
  - No session key established
  - No mutual authentication.
  - How to fix this problem?
    - Add session key k_{ab}, and a nounce m from Alice

Version 3 : *version finale*



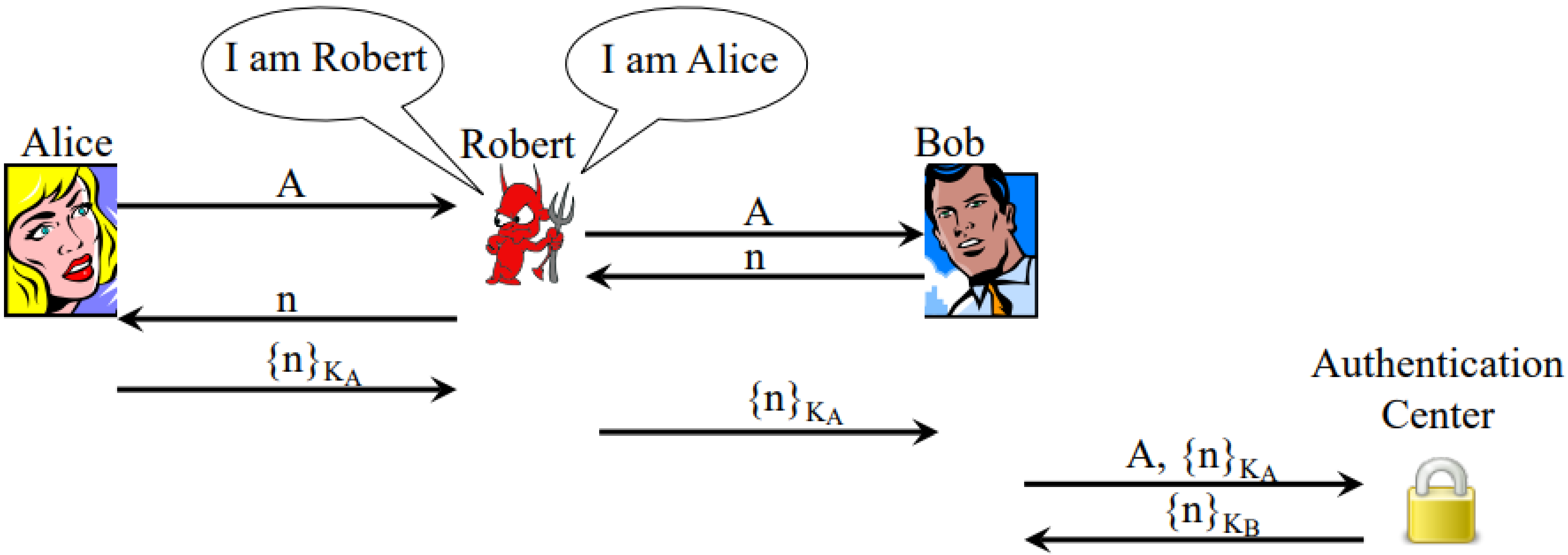
- What is the problem with this protocol
  - In case of authentication among a group of people
    - Every pair of users need to have a shared secret key
    - When a user joins a group, every one in the group needs to configure a new key with him
    - Solution: use a trusted third party
      - reduce $O(n^2)$ keys to $O(n)$ keys.

With trusted third party: Version 1



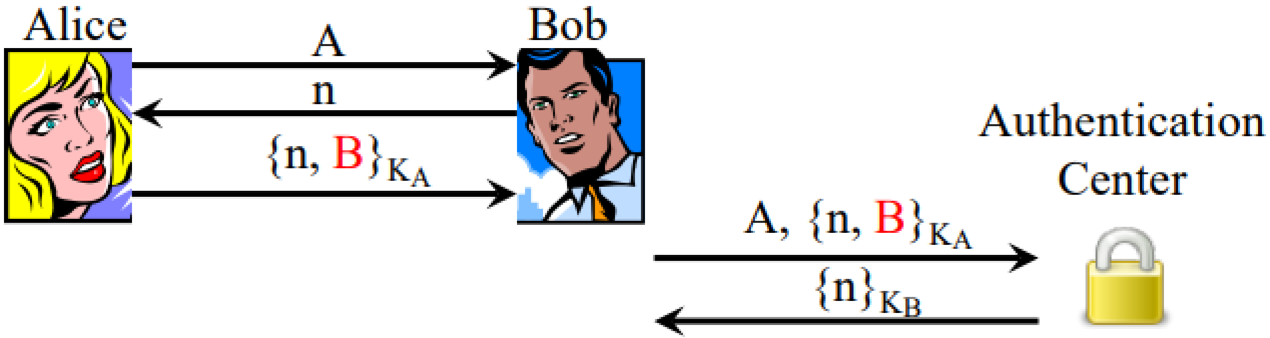
- Is this authentication protocol secure?
  - No, man-in-the-middle attack

Version 2: Man-in-the-middle



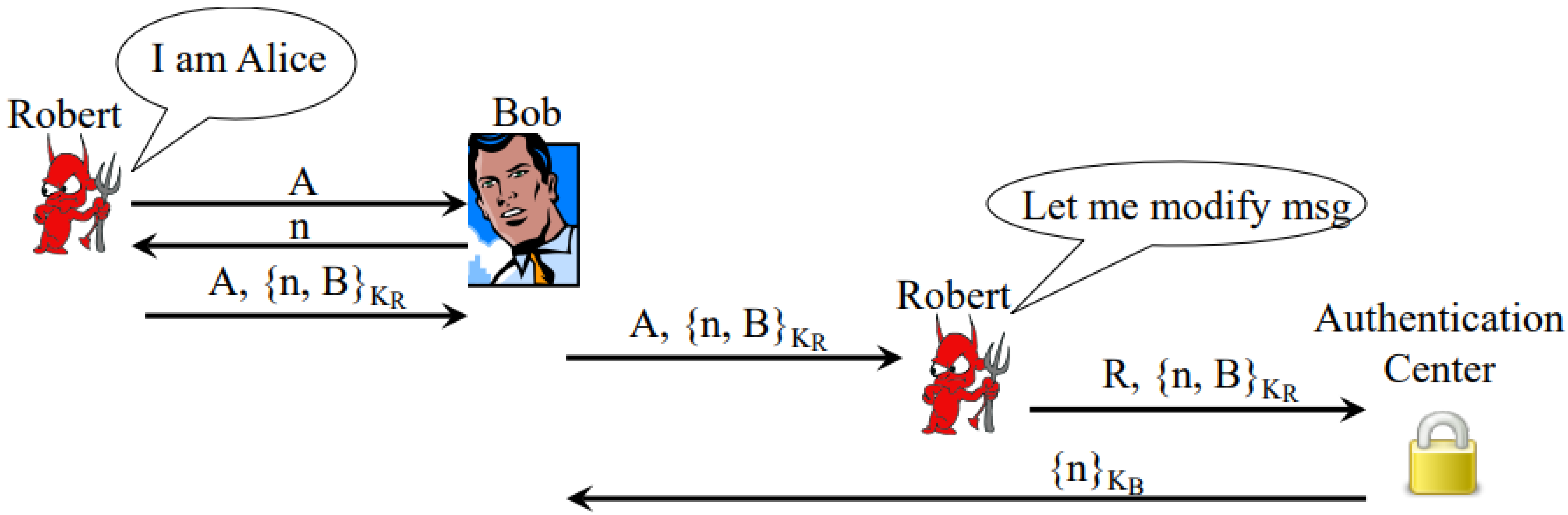
- How to defend against this attack?
  - Add principal name to prevent $\{n\}_{KA}$ from being reused

Version 3



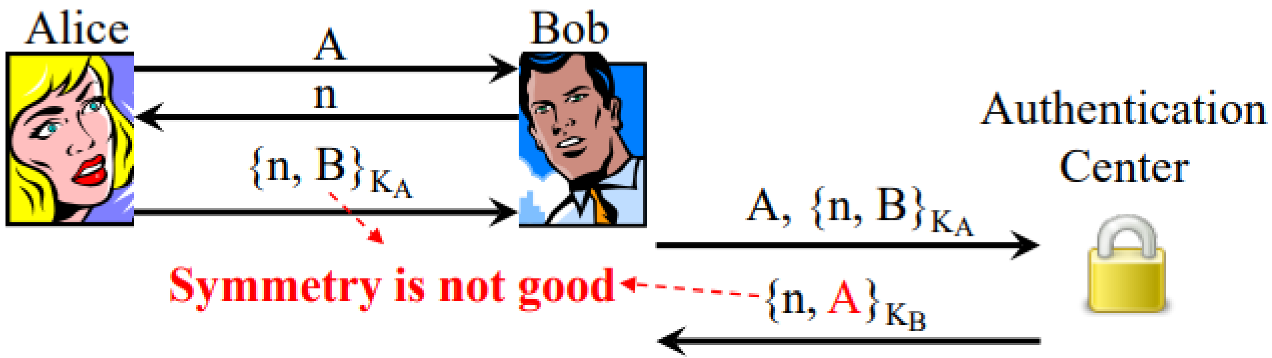
- Is this authentication protocol secure?
  - No

Version 3: attack



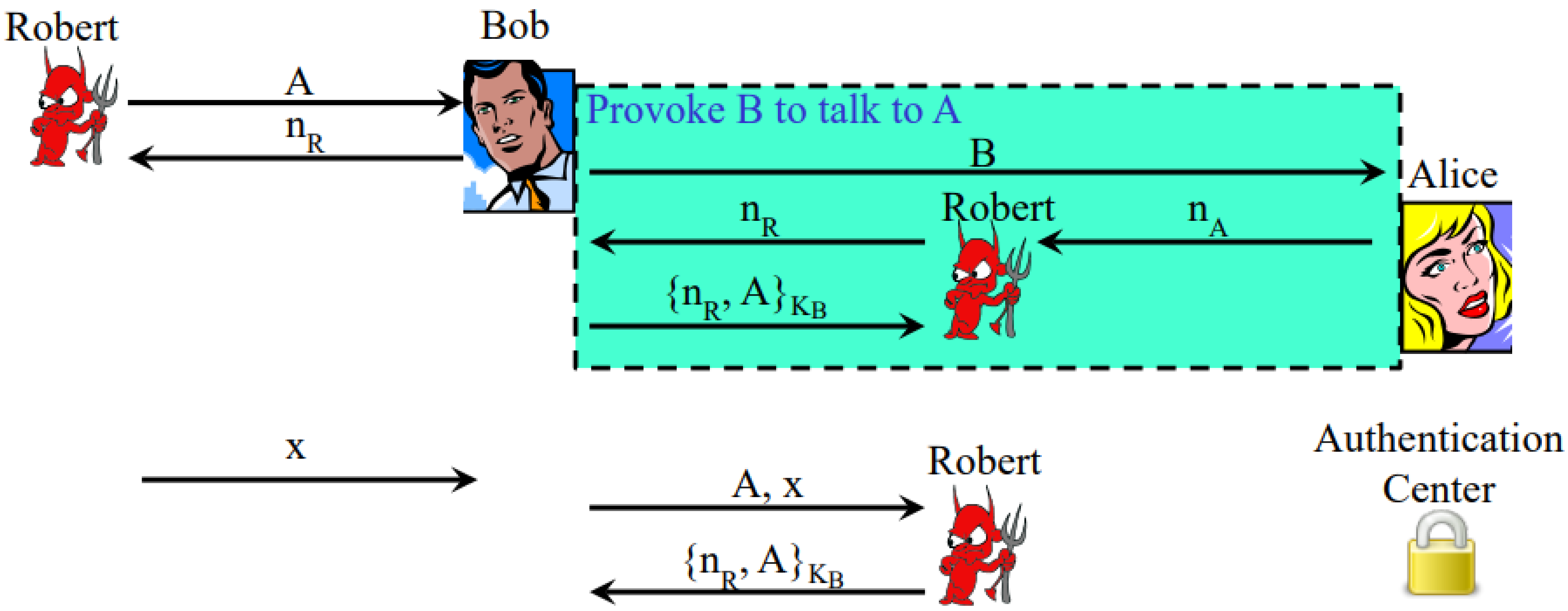
- How to defend against this attack?
  - add principal name into $\{n\}_{KB}$

Version 4



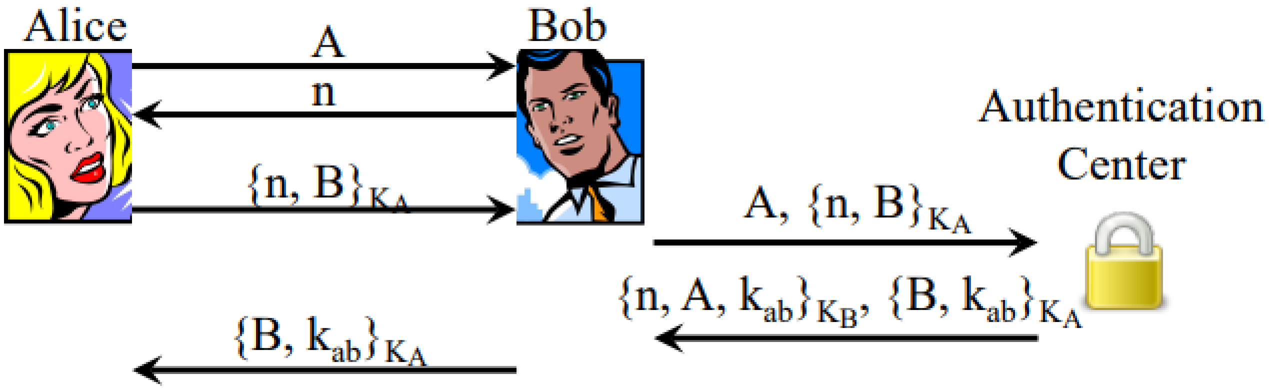
- Is this authentication protocol secure?
  - Answer: No
  - Message symmetry in authentication protocols is not good

Version 4: attack



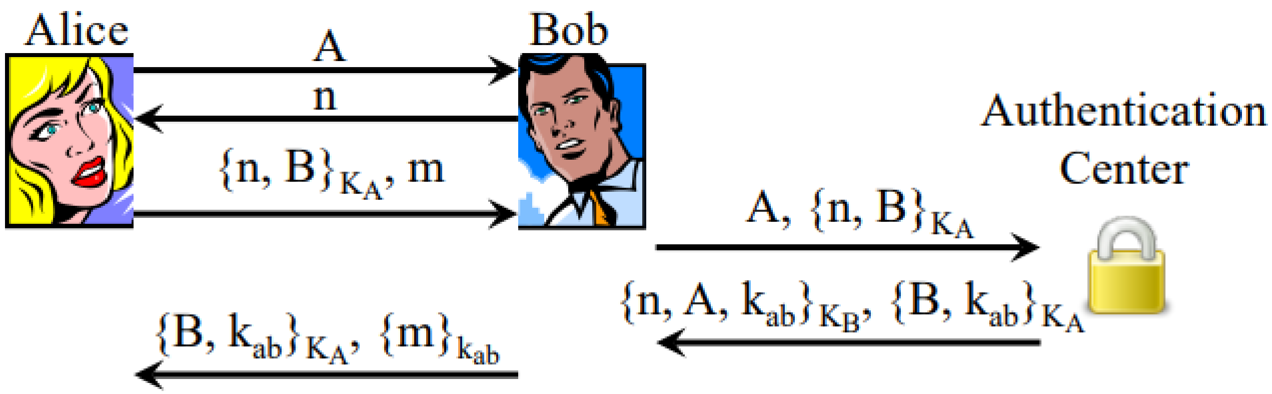
- How to defend against this attack?
  - Break symmetry

Version 5



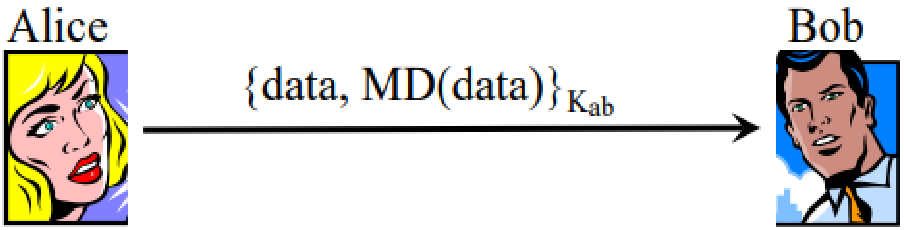
- Almost done there
  - except having mutual authentication

Version 6: fin

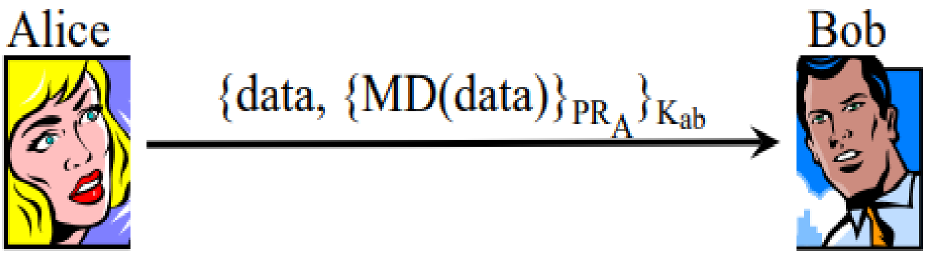


Secrecy, integrity, non-repudiation

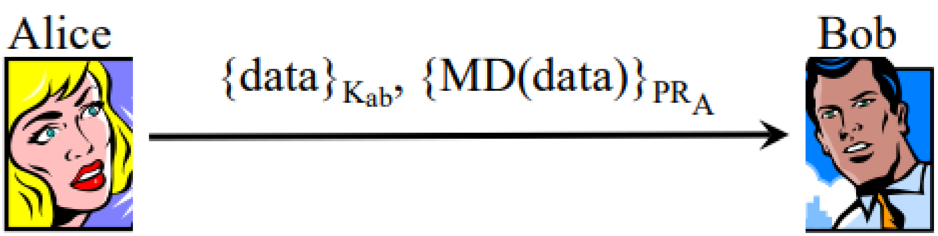
- How to achieve secrecy and integrity?



- How to futher achieve non-repudiation?



- Further improvement



More attacks: freshness

Needham-Schroeder protocol

M_1, $a \rightarrow s : a.\ b.\ r_{\mathbf{a}}$

M_2, $s \rightarrow a : \{r_{\mathbf{a}}\ .\ b\ .\ K_{\mathbf{ab}}\ .\ \{K_{\mathbf{ab}}\ .\ a\}_{\mathbf{Ksb}}\ \}_{\mathbf{Ksa}}$

M_3, $a \rightarrow b : \{K_{\mathbf{ab}}\ .\ a\}_{\mathbf{Ksb}}$

M_4, $b \rightarrow a : \{r_{\mathbf{b}}\}_{\mathbf{Kab}}$

M_5, $a \rightarrow b : \{r_{\mathbf{b}} - 1\}_{\mathbf{Kab}}$

Ksa, Ksb, secret key between s&a, s&b

- Denning-Sacco attack: attacker already obtained $oldK_{ab}$

  M_3', $x/a \rightarrow b : \{oldK_{ab}\ .\ a\}_{Ksb}$

  M_4', $b \rightarrow x/a : \{r_b\}_{oldKab}$

  M_5', $x/a \rightarrow b : \{r_b - 1\}_{oldKab}$

  Solution: add time stamp in M2 and M3

  M_2: $s \rightarrow a : \{\ b\ .\ K_{\mathbf{ab}}\ .\ t_{\mathbf{a}}\ .\ \{a\ .\ K_{\mathbf{ab}}\ .\ t_{\mathbf{a}}\}_{\mathbf{Ksb}}\ \}_{\mathbf{Ksa}}$

  M_3: $a \rightarrow b : \{\ K_{\mathbf{ab}}\ .\ a\ .\ t_{\mathbf{a}}\ \}_{\mathbf{Ksb}}$

Type confusion attack

Otway-Rees protocol (Ksa, Ksb, secret key between s & a, s & b)

M_1: $a \rightarrow b$: $m.a.b.\{n_a.m.a.b\}_{Ksa}$ m is transaction ID
M_2: $b \rightarrow s$: $m.a.b.\{n_a.m.a.b\}_{Ksa}\{n_b.m.a.b\}_{Ksb}$
M_3: $s \rightarrow b$: $m.\{n_a.K_{ab}\}_{Ksa}\{n_b.K_{ab}\}_{Ksb}$
M_4: $b \rightarrow a$: $m.\{n_a.K_{ab}\}_{Ksa}$ m 32 bits, a, b 16 bits, K_{ab} 64 bits

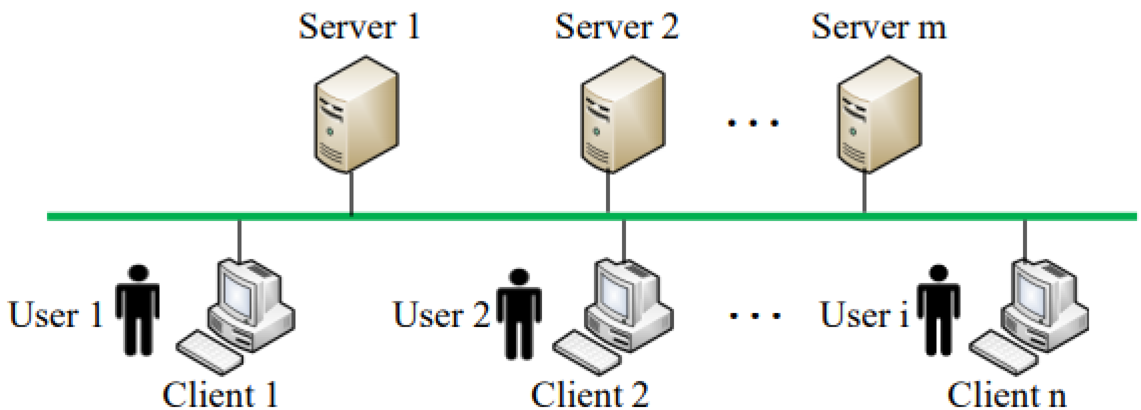
Attack: replay encrypted part in M_1
M_1: $a \rightarrow x/b$: $m.a.b\{n_a.m.a.b\}_{Ksa}$
M_4': $x/b \rightarrow a$: $\{n_a.m.a.b\}_{Ksa}$

Result: a accepts m.a.b as key

Test: find another attack

Kerberos

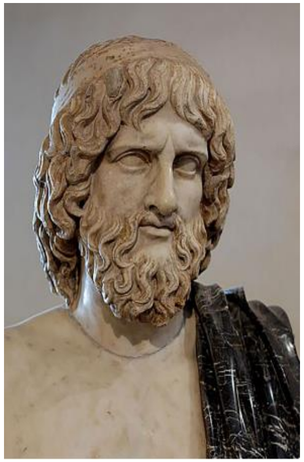
- Authentication service developed by MIT
  - Multi-user, multi-client machine, multi-server machine
  - Authentication
    - Users prove their identities when requesting services at servers from client machines



- Uses a trusted third party & symmetric cryptography
- Based on Needham Schroeder with Denning Sacco
- Passwords not sent in clear text
  - Only the network can be compromised

Kerberos

- Most widely used authentication service
- Kerberos:
  - a many headed dog, commonly three
  - guardian of the entrance of Hadès
    - Zeus, Hadès et Poséidon



- Why Kerberos
  - Want to access resources from anywhere
  - Don’t want to enter password for each access
    - Time comsuming
    - Insecure

Kerberos

- Kerberos realm consists of a
  - Kerberos server
    - Authentication Server (AS)
    - Ticket Granting Server (TGS)
  - Users and servers that are registered with Kerberos server
- Uses ticket
  - Ticket granting Ticket, TGT (issued by AS for user to request for service ticket from TGS)
  - Service Ticket (issued by TGS for user to use service from server)
- Procedure: 4 steps
  - User login: user login at client machine
  - Authentication: user is authenticated
  - Authorization: user is granted access to service
  - Service request: user sends request to server

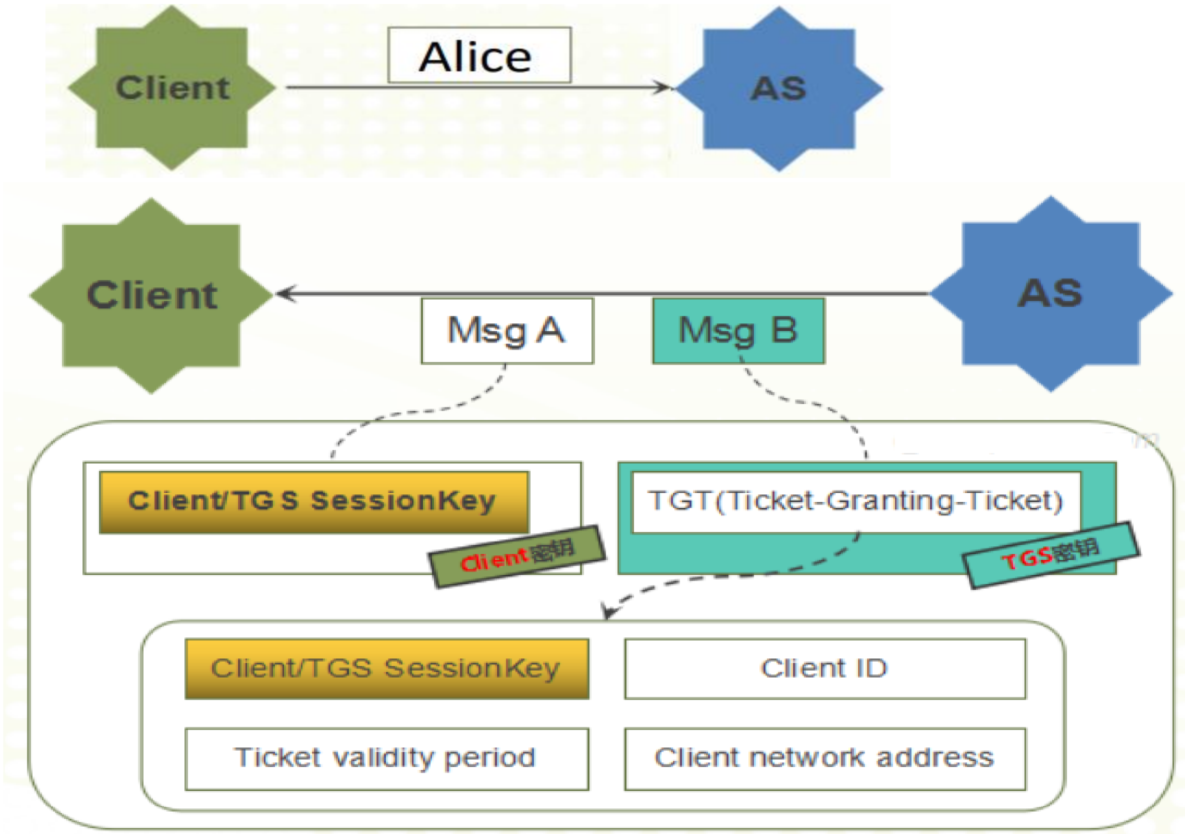
Kerberos

- Application Client (Client): client requesting service at Server
- Authentication Server (AS): authenticates Client
- Ticket-Granting Service (TGS): grants tkt to access service
- Service Server (SS): has the service requested by Client

Step 1: user login

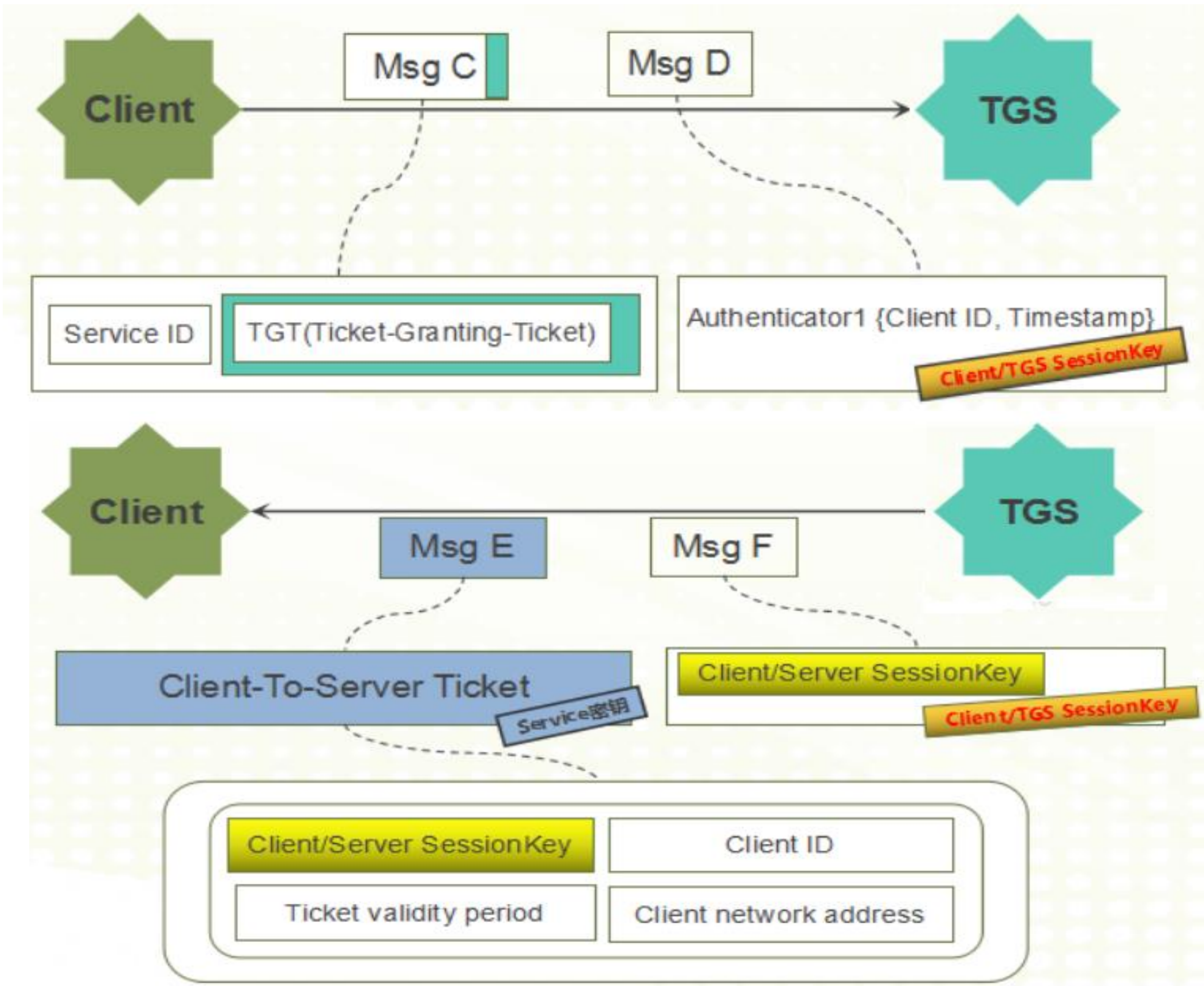
- Alice wants to access a service at Server from Client
- Alice types user ID and password into Client
- Client derives Client key: H(pwd)

Step 2: User authentication



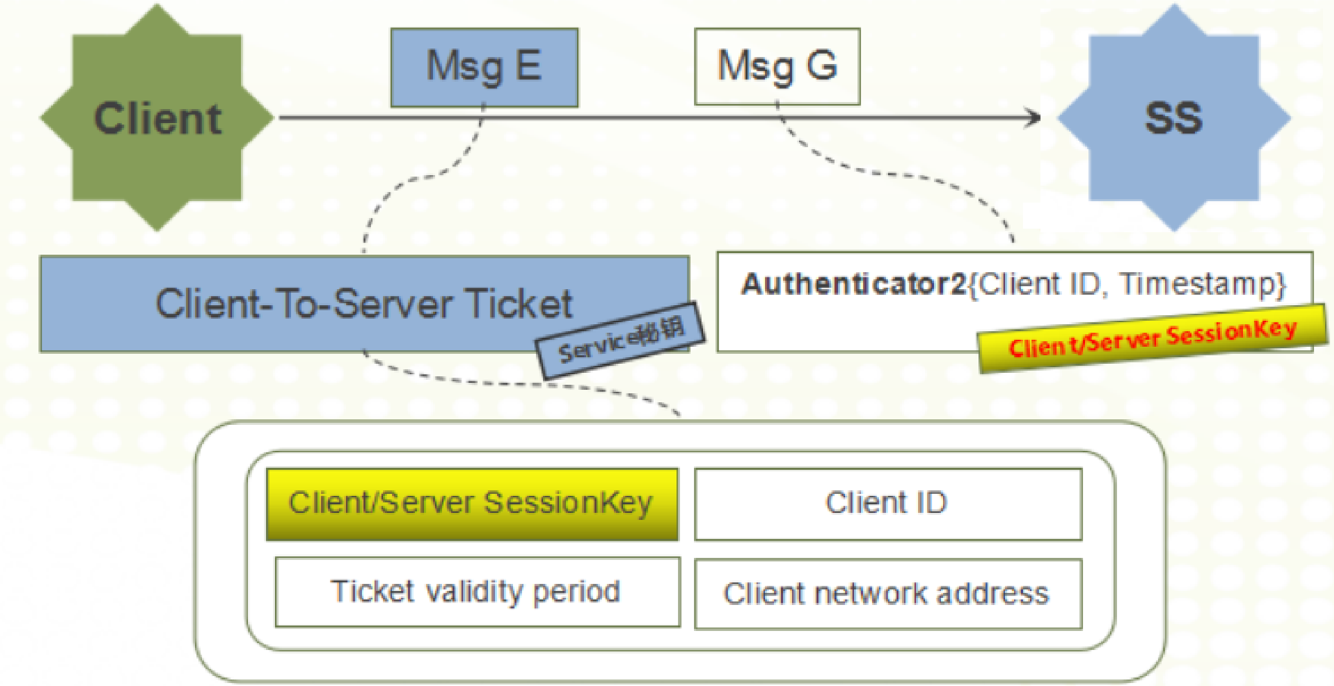
- AS authenticates Client (Alice)
- Msg A: Client/TGS SessionKey generated encypted by H(pwd)
  - Client can decrypt Client/TGS SessionKey
- Msg B: Client cannot decrypt

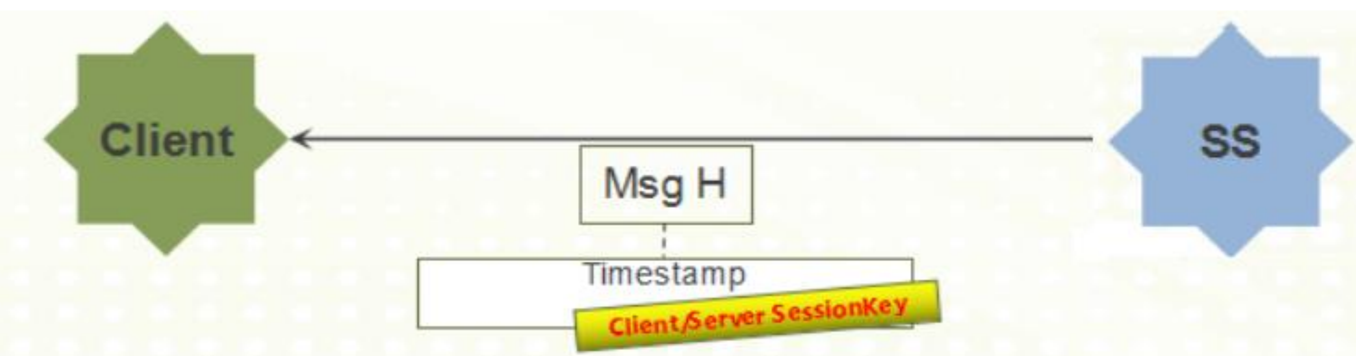
Step 3: Access grant



- Msg E: Client cannot decrypt

Step 4: Client-Server interaction





Case study: SPLICE/AS

1.	$C \rightarrow AC$	:	C, S, N_1
2.	$AC \rightarrow C$	:	$AC, \{AC, C, N_1, PKs\}_{SKac}$
3.	$C \rightarrow S$	:	$C, S, \{C, T, L, \{N_2\}_{PKs}\}_{SKc}$
4.	$S \rightarrow AC$	:	S, C, N_3
5.	$AC \rightarrow S$	:	$AC, \{AC, S, N_3, PKc\}_{SKac}$
6.	$S \rightarrow C$	:	$S, C, \{S, N_2+1\}_{PKc}$

- Objective: mutual authentication + establishing session key N_2
  - S: server, C: client, AC: authority of certification
  - AC knows public key of S & C; S & C know public key of AC

- The protocol involves 3 ways of authentication, what are they?
- When S is authenticated to C, and C to S?
- Why use encyption in 6, propose another way without encryption
- Why the protocol is not a key agreement protocol?

Case study: SPLICE/AS

1.	$C \rightarrow AC$	:	C, S, N_1
2.	$AC \rightarrow C$	:	$AC, \{AC, C, N_1, PKs\}_{SKac}$
3.	$C \rightarrow S$	:	$C, S, \{C, T, L, \{N_2\}_{PKs}\}_{SKc}$
4.	$S \rightarrow AC$	:	S, C, N_3
5.	$AC \rightarrow S$	:	$AC, \{AC, S, N_3, PKc\}_{SKac}$
6.	$S \rightarrow C$	:	$S, C, \{S, N_2+1\}_{PKc}$

- Perfect Forward Secrecy (PFS)
  - Breaking a long-term key does not break session keys before.
  - The protocol does not satisfy PFS, how to enforce this?

- Give an attack, where attacker can authenticate itself to S as C
  - How to counter the attack?
- Give an attack, where attacker can authenticate itself to C as S
  - How to counter the attack?

1.	$C \rightarrow X/AC$	:	C, S, N_1
1'.	$X/C \rightarrow AC$	:	C, X, N_1
2.	$AC \rightarrow C$	:	$AC, \{AC, C, N_1, PKx\}_{SKac}$
3.	$C \rightarrow X/S$	:	$C, S, \{C, T, L, \{N_2\}_{PKx}\}_{SKc}$
4.	$X \rightarrow AC$	:	X, C, N_3
5.	$AC \rightarrow X$	:	$AC, \{AC, X, N_3, PKc\}_{SKac}$
6.	$X/S \rightarrow C$	:	$S, C, \{S, N_2+1\}_{PKc}$

1.	$X \rightarrow AC$	:	X, S, N_1
2.	$AC \rightarrow X$	:	$AC, \{AC, X, N_1, S, PKS\}_{SKac}$
3.	$X/C \rightarrow S$	:	$C, S, \{C, T, L, \{N_2\}_{PKs}\}_{SKx}$
4.	$S \rightarrow X/AC$	:	S, C, N_3
4'.	$X/S \rightarrow AC$	:	S, X, N_3
5.	$AC \rightarrow S$	:	$AC, \{AC, S, N_3, PKx\}_{SKac}$
6.	$S \rightarrow X/C$	:	$S, C, \{S, N_2+1\}_{PKx}$

Case study: SPLICE/AS

1.	$C \rightarrow AC$	:	C, S, N_1
2.	$AC \rightarrow C$	:	$AC, \{AC, C, N_1, PKs\}_{SKac}$
3.	$C \rightarrow S$	:	$C, S, \{C, T, L, \{N_2\}_{PKs}\}_{SKc}$
4.	$S \rightarrow AC$	:	S, C, N_3
5.	$AC \rightarrow S$	:	$AC, \{AC, S, N_3, PKc\}_{SKac}$
6.	$S \rightarrow C$	:	$S, C, \{S, N_2+1\}_{PKc}$

- Give a Man-in-the-middle attack.
  - How to counter the attack?
- Show that the above attack is still feasible if ECB is used.

3.	$C \rightarrow X/S$	:	$C, S, \{C, T, L, \{N_2\}_{PKs}\}_{SKc}$
	3'. $X \rightarrow S$	:	$X, S, \{X, T, L, \{N_2\}_{PKs}\}_{SKx}$
	6'. $S \rightarrow X$	:	$S, X, \{S, N_2+1\}_{PKx}$
6.	$X/S \rightarrow C$	:	$S, C, \{S, N_2+1\}_{PKc}$

3.	$C \rightarrow X/S$	:	$C, S, \{C, T, L, \{C, N_2\}_{PKs}\}_{SKc}$
	3'. $X \rightarrow S$	:	$X, S, \{X, T, L, \{X, N_2\}_{PKs}\}_{SKx}$
	6'. $S \rightarrow X$	:	$S, X, \{S, N_2+1\}_{PKx}$
6.	$X/S \rightarrow C$	:	$S, C, \{S, N_2+1\}_{PKc}$

Random Number Generation

- Many crypto protocols require random numbers
  - Key generation
  - Authentication nonces

- How to generate random numbers?
  - How to generate truly random bits?
  - How to use cryptographic methods to stretch a little bit of true randomness into a large stream of pseudorandom values
    - indistinguishable from true random bits

What Can Go Wrong

```
unsigned char key[16];

srand(time(NULL));
for (i=0; i<16; i++)
    key[i] = rand() & 0xFF;
```

```
static unsigned int next = 0;
void srand(unsigned int seed) {
    next = seed;
}

/* RAND_MAX assumed to be 32767 */
int rand(void) {
    next = next * 1103515245 + 12345;
    return next % 32768;
}
```

- Seed highly predictable
  - time(NULL) returns the current time, in # seconds since 1/1/1970
  - only $3600 \times 24 \times 365 = 31{,}536{,}000 \approx 2^{25}$ seconds in a year
    - if I can guess the year? the month? the day?
- Output not very random
  - Last bit?
  - Predictable: depends only on last value
  - Only the last 15 bits matter, even last 8 bits

Real-world Examples

- X Windows “magic cookie” was generated using rand()
- Netscape SSL session keys used time & process ID as seed
- Kerberos
  - First discover to be similarly flawed
  - 4 yrs later, discovered flaw with memset()
- PGP used return value from read() to seed its PRNG
- On-line poker site used insecure PRNG to shuffle cards
- Debian Openssl generates predictable pseudorandom numbers

- Lessons learned
  - Seeds must be unpredictable
  - Algorithm for generating pseudorandom bits must be secure

Generating Pseudorandom Numbers

- True random number generator
  - TRNG
  - Generates bits distributed uniformly at random,
  - All outputs are equally likely, no patterns, correlations, etc.

- Cryptographically secure pseudorandom number generator
  - CS-PRNG
  - Taking a short true-random seed
  - Generating long sequence of bits
    - computationally indistinguishable from true random bits

CS-PRNG

- Cryptographically secure pseudorandom number generator
  - G: maps a seed to an output G(S)
    - E.g., G: $\{0,1\}^{128}$-> $\{0,1\}^{1000000}$
  - K: a random variable distributed uniformly at random in S
  - U: a random variable distributed uniformly at random in G(S)
  - G is secure if output G(K) is computationally indistinguishable from U

- Sample construction
  - Use the seed as a key k, and compute AES-CBC(k, 0)

TRNG

- TRNG should be random and unpredictable
- Good or bad choices?
  - IP addresses, Contents of network packets, Process IDs
  - High-speed clock
  - Soundcard, Keyboard input, Disk timings
    - Non-uniform

- How to convert non-uniform random sources into TRNG?
  - Use a cryptographic hash function,
    - Hash(x) truncated to n bits

Secret Sharing

- A trusted authority TA has a secret K
- Wants to split K into n shares S_1, ..., S_n,
- Distributing to n users U_1,...,U_n
  - Can reconstruct K from any t of the n shares
  - Any (t-1) shares reveal no information about K

- Such a scheme is called an (n,t) threshold secret sharing scheme

(n,n) Secret Sharing

- Suppose the secret K is an integer between 0 and M-1
- (n,n) threshold scheme:
  - Pick $S_1,...,S_{n-1}$ uniformly at random in [0,M-1]
  - Set $S_n = K - (S_1 + ... + S_{n-1}) \bmod M$

- How to reconstruct K?
- What happens if n-1 users get together

(n,t) Threshold Scheme

- Polynomials modulo prime p
  - Polynomials whose coefficients are elements mod p
  - E.g., $f(x) = y = x^2 + 2x + 4 \mod 5$
  - Uniquely determined by any n+1 distinct pairs (x_i, y_i)
    - Lagrange interpolation

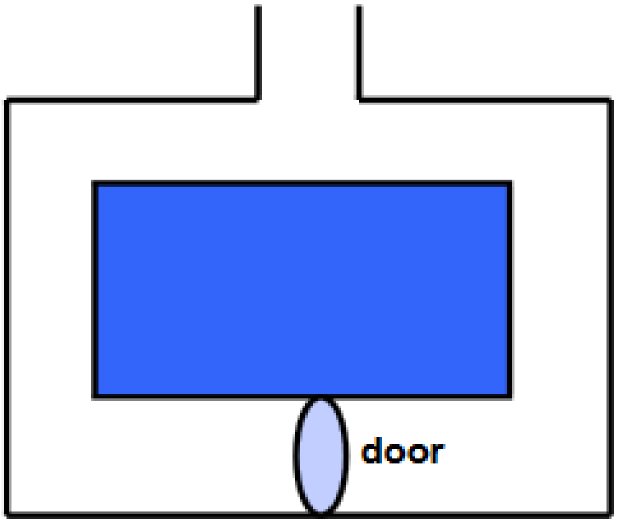
- To (n,t) threshold share secret K:
  - Pick a random polynomial f of degree t-1
    - $f(0) = K$
  - Share $s_i = f(i)$ for i = 1 to n
  - How to recover K?
  - How many shares do you need to recover K?
  - What happens if you have fewer shares than t?

Zero-knowledge Proof

- An interactive proof: prover proves to verifier he knows a secret without revealing it
  - Alice->Bob: I know solution to Q3 in hw 1,but I can’t tell you
  - Bob->Alice: tell me, or I don’t believe you
  - Alice->Bob: Zero-knowledge proof

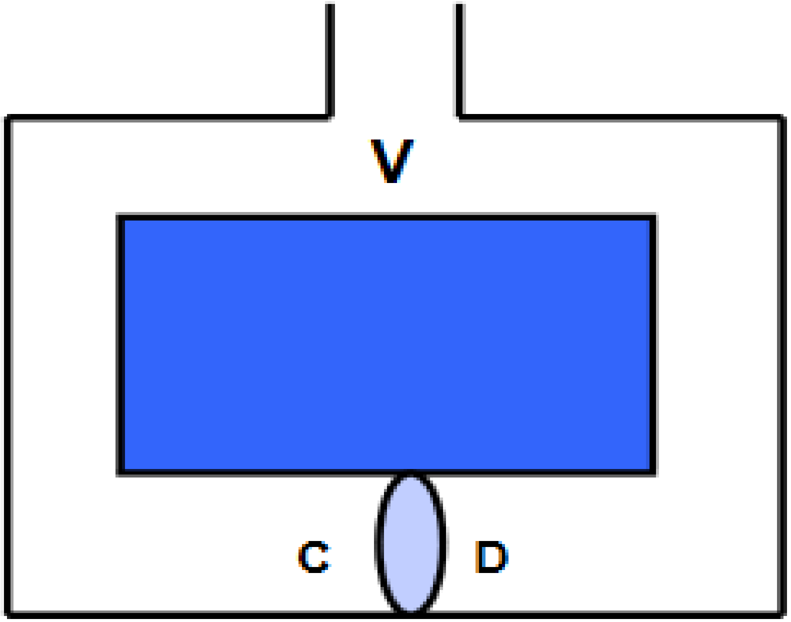
  - n=670592745=12345*54321 is not a ZKP that n is a prime

Example: Zero-knowledge Cave



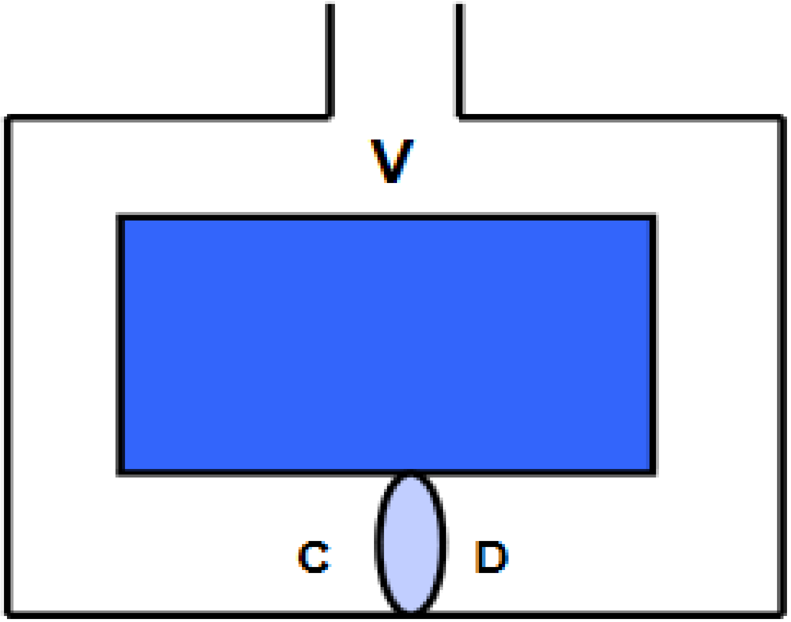
- Alice wants to prove to Bob that she has the key of the door
  - Without revealing it to Bob

Example: Zero-knowledge Cave



- Alice walks to either C or D
- Bob stands at V, calling either Left or Right
- Alice complies, using her key to open door if needed
- Alice & Bob repeats steps 1-3 for n times

Example: Zero-knowledge Cave



- What if Alice didn't know the key?
- What does Bob learn at the end of the proof?

How to prove knowing square root

- Finding square root mod N=pq is as hard as factoring
- A knows b s.t. $b^2 = y \mod pq$, she wishes to prove to B
  - A picks random r
  - $A \rightarrow B$: $s = r^2 \mod pq$
  - B flips coin
  - $B \rightarrow A$: coin flip
  - If heads
    - $A \rightarrow B$: $t = r \mod pq$
    - B verifies $t^2 \equiv s \mod pq$
  - If tails
  - $A \rightarrow B$: $t = rb \mod pq$
  - B verifies $t^2 \equiv sy \mod pq$
- What if A didn't know the square root?
- What did B learn after the proof?