Security

Lecture III

Introduction to Authentication Schemes

Lecturer

Dr. Seyed Reza Kamel

Password authentication

- Basic idea
  - User has a secret password
  - System checks password to authenticate user

Password authentication

- Basic idea
  - User has a secret password
  - System checks password to authenticate user
- Issues
  - How is password stored?
  - How does system check password?
  - How easy is it to guess a password?

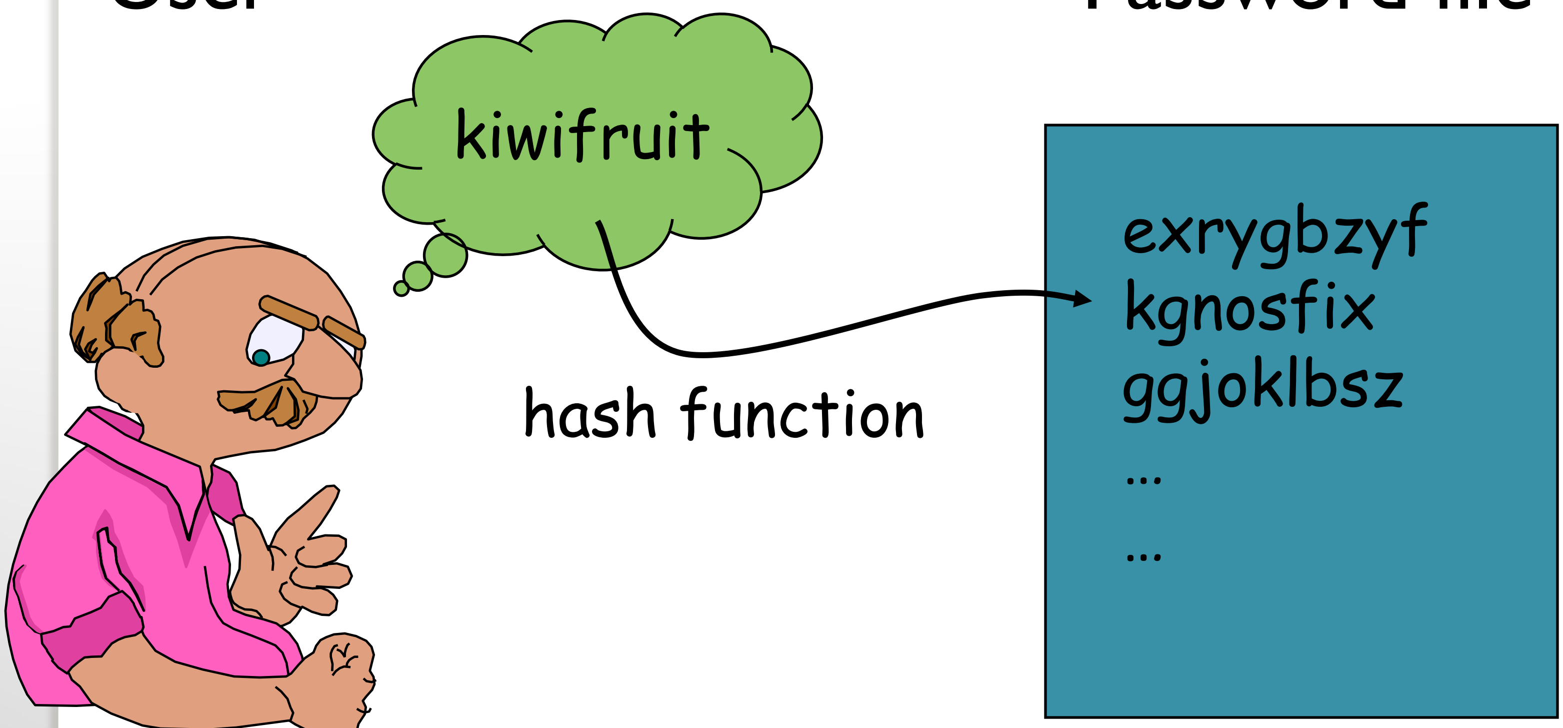
Password authentication

- Basic idea
  - User has a secret password
  - System checks password to authenticate user
- Issues
  - How is password stored?
  - How does system check password?
  - How easy is it to guess a password?
    - Difficult to keep password file secret, so best if it is hard to guess password even if you have the password file

Basic password scheme

User

Password file



Basic password scheme

- Hash function $h : \text{strings} \rightarrow \text{strings}$
  - Given **h(password)**, hard to find password
  - No known algorithm better than trial and error
- User password stored as h(password)

Basic password scheme

- Hash function $h : \text{strings} \rightarrow \text{strings}$
  - Given **h(password)**, hard to find password
  - No known algorithm better than trial and error
- User password stored as h(password)
- When user enters password
  - System computes h(password)
  - Compares with entry in password file
- No passwords stored on disk

Unix Password System (Example)

- Hash function is 25xDES
  - 25 rounds of DES-variant encryptions

Unix Password System (Example)

- Hash function is 25xDES
  - 25 rounds of DES-variant encryptions
- Password file is publicly readable
  - Other information in password file …

Unix Password System (Example)

- Hash function is 25xDES
  - 25 rounds of DES-variant encryptions
- Password file is publicly readable
  - Other information in password file …
- Any user can try "dictionary attack"
  - User looks at password file
  - Computes hash(word) for every word in dictionary

Unix Password System (Example)

- Hash function is 25xDES
  - 25 rounds of DES-variant encryptions
- Password file is publicly readable
  - Other information in password file …
- Any user can try "dictionary attack"
  - User looks at password file
  - Computes hash(word) for every word in dictionary
- "Salt" makes dictionary attack harder

Unix Password System (Example)

- Password line

  walt:fURfuu4.4hY0U:129:129:Belgers:/home/walt:/bin/csh

Unix Password System (Example)

- Password line

walt:fURfuu4.4hY0U:129:129:Belgers:/home/walt:/bin/csh

UserName

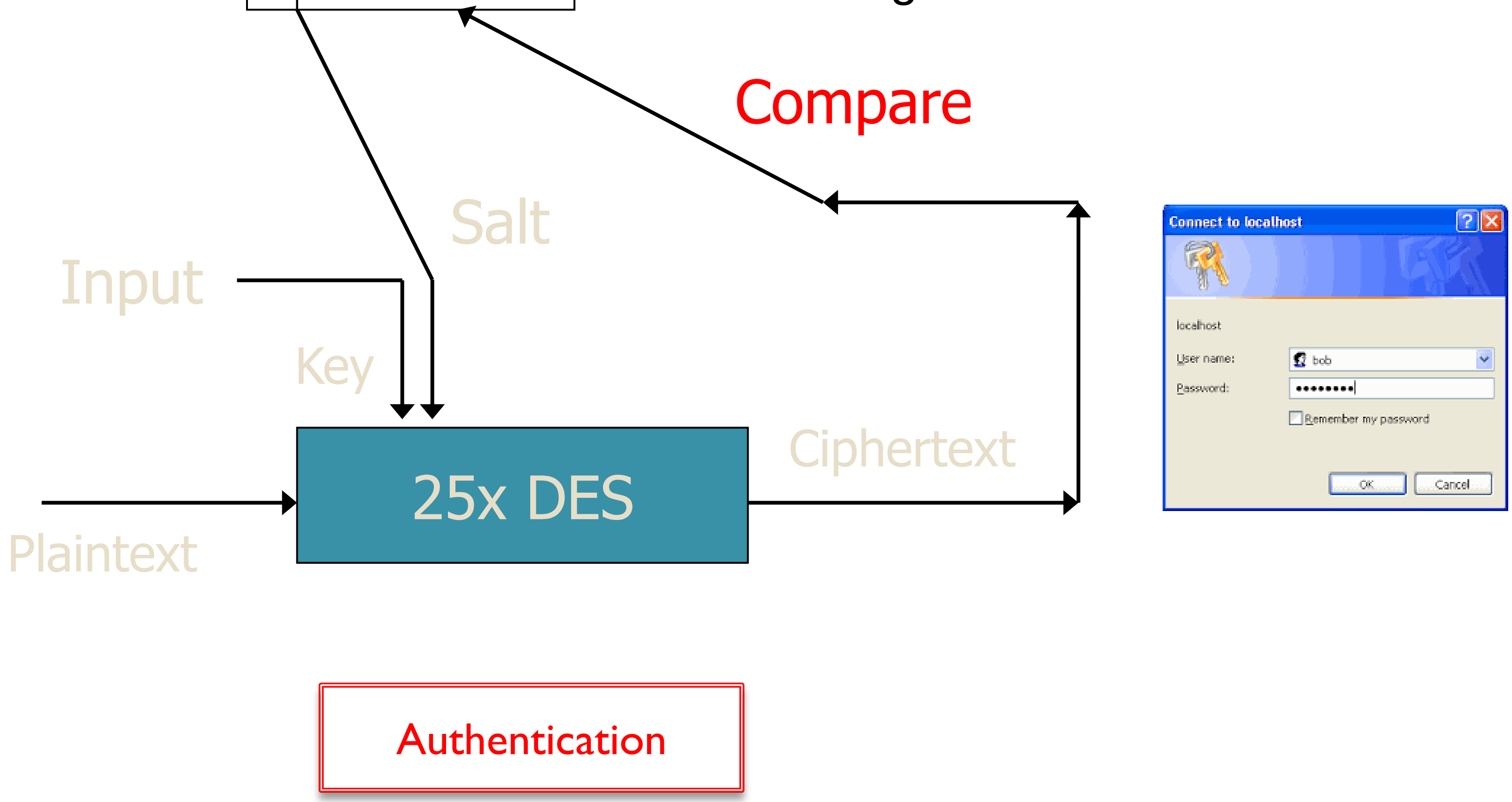
SALT

Password Section

Unix Password System (Example)

- Password line

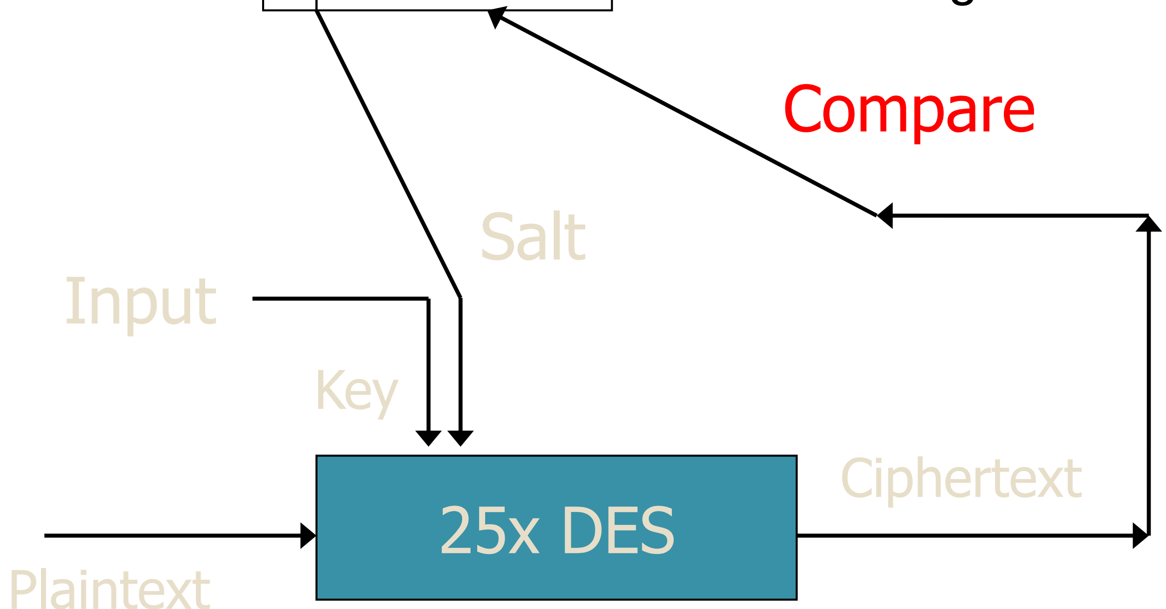
walt:fURfuu4.4hY0U:129:129:Belgers:/home/walt:/bin/csh



Unix Password System (Example)

- Password line

walt:fURfuu4.4hY0U:129:129:Belgers:/home/walt:/bin/csh

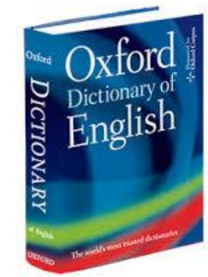


When password is set, salt is chosen randomly

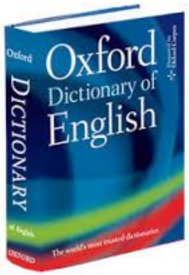
12-bit salt slows dictionary attack by factor of 2^{12}

Dictionary Attack – some numbers

- If passwords had meaning…



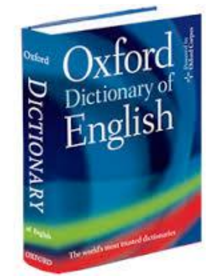
Dictionary Attack – some numbers



- If passwords had meaning…
- Typical password dictionary
  - 1,000,000 entries of common passwords
    - people's names, common pet names, and ordinary words
  - Suppose you generate and analyze 10 guesses per second
    - This may be reasonable for a web site; offline is *much* faster
  - Dictionary attack in at most 100,000 seconds = 28 hours, or 14 hours on average

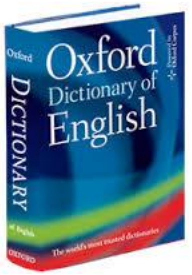
Dictionary Attack – some numbers

- If passwords were random…



Dictionary Attack – some numbers

- If passwords were random…



  - Assume six-character password
    - Upper- and lowercase letters (a-z, A-Z),
    - Digits (0-9),
    - 32 punctuation characters (: , . # …)

  - 689,869,781,056 password combinations

  - Exhaustive search requires 1,093 years on average

Advantages of salt

Advantages of salt

- Without salt
  - Same hash functions on all machines
    - Compute hash of all common strings once
    - Compare hash file with all known password files
- With salt
  - One password hashed 2^{12} different ways
    - Pre-compute hash file?
      - Need much larger file to cover all common strings
    - Dictionary attack on known password file
      - For each salt found in file, try all common strings

Challenge-response Authentication

Goal: Bob wants Alice to "prove" her identity to him

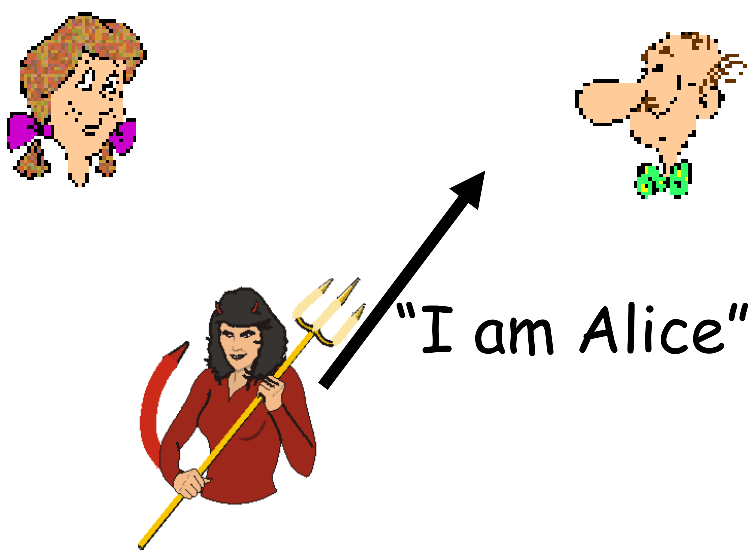
Protocol ap1.0: Alice says "I am Alice"

Failure scenario??

Authentication

Goal: Bob wants Alice to "prove" her identity to him

Protocol ap1.0: Alice says "I am Alice"



in a network,
Bob can not "see"
Alice, so Trudy simply
declares
herself to be Alice

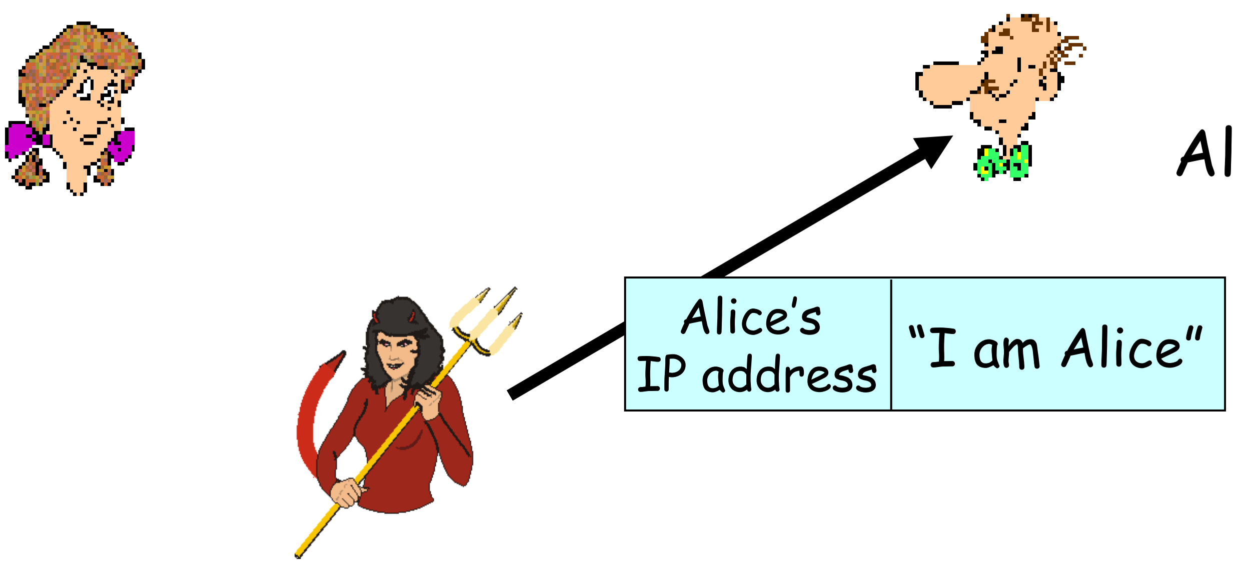
Authentication: another try

<u>Protocol ap2.0:</u> Alice says "I am Alice" in an IP packet containing her source IP address

Failure scenario??

Authentication: another try

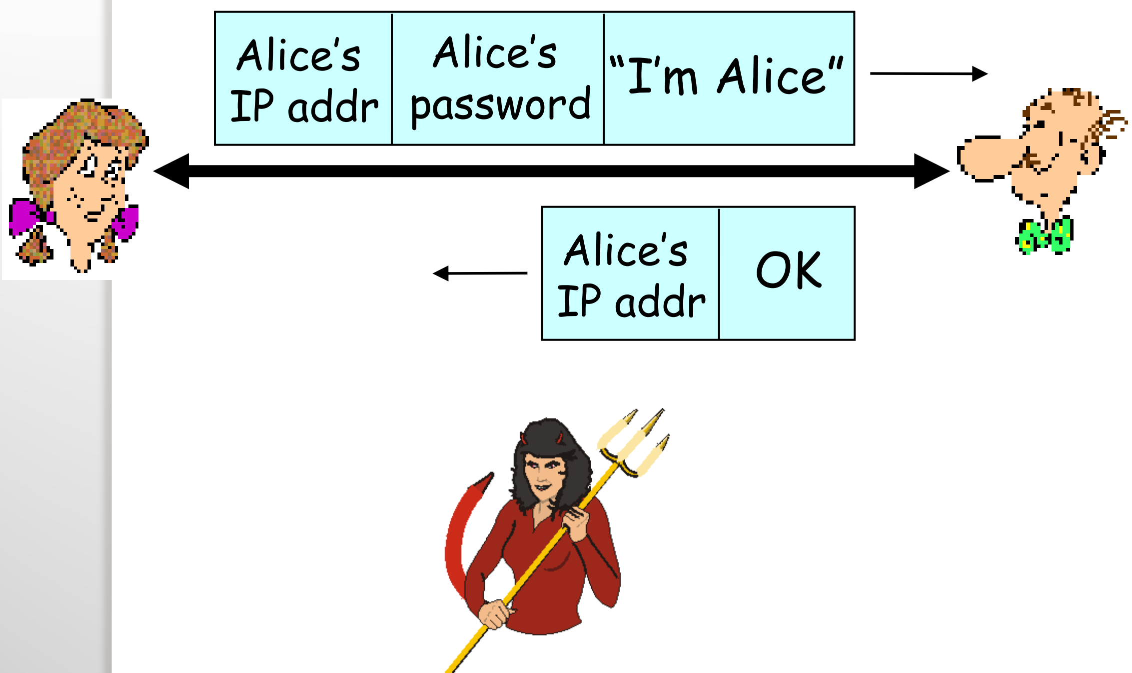
Protocol ap2.0: Alice says "I am Alice" in an IP packet containing her source IP address



Trudy can create a packet "spoofing" Alice's address

Authentication: another try

Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.

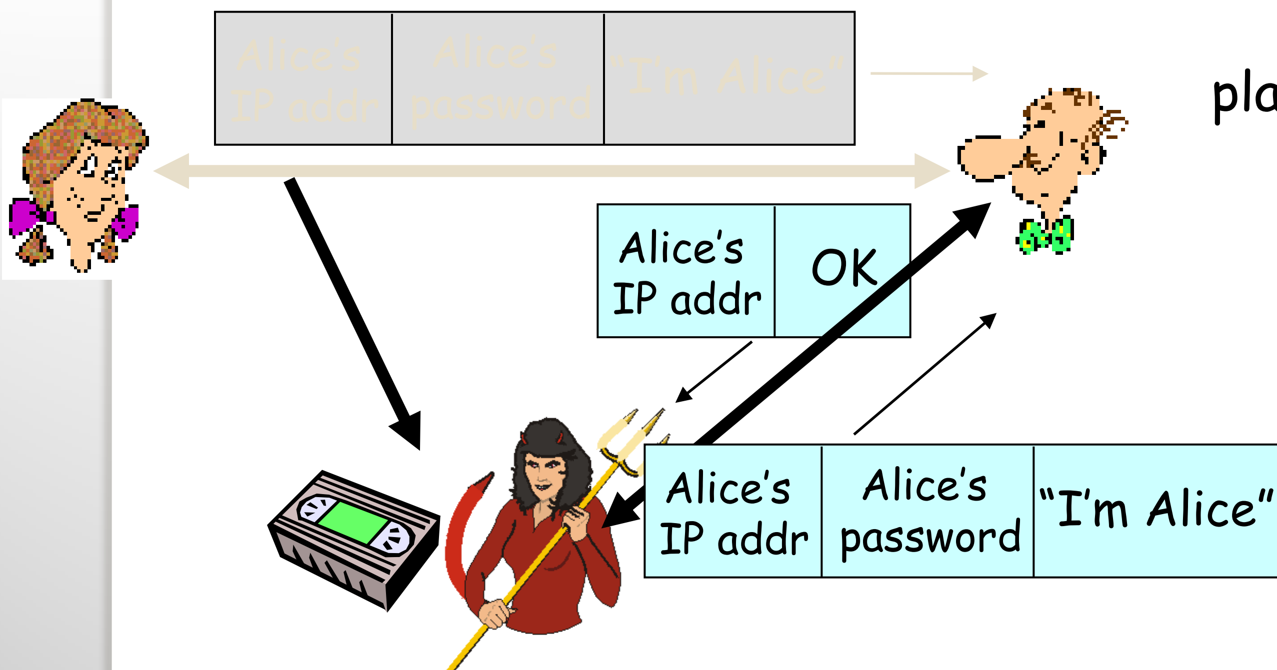


Failure scenario??

Authentication: another try

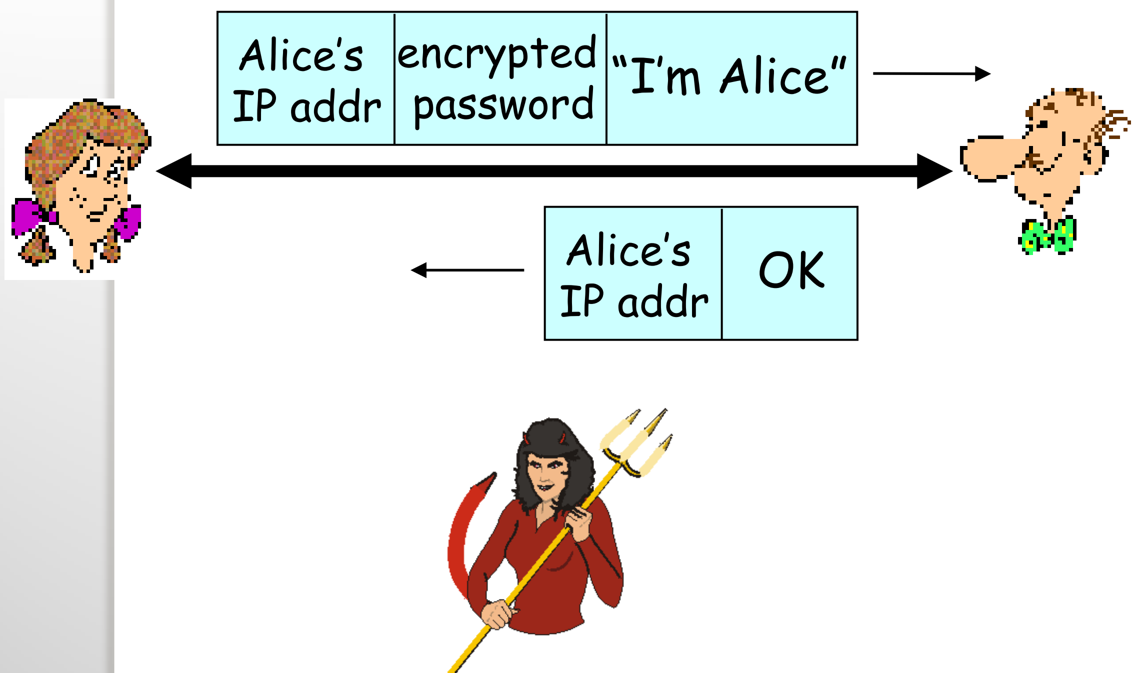
Protocol ap3.0: Alice says "I am Alice" and sends her secret password to "prove" it.

playback attack: Trudy records Alice's packet and later plays it back to Bob



Authentication: yet another try

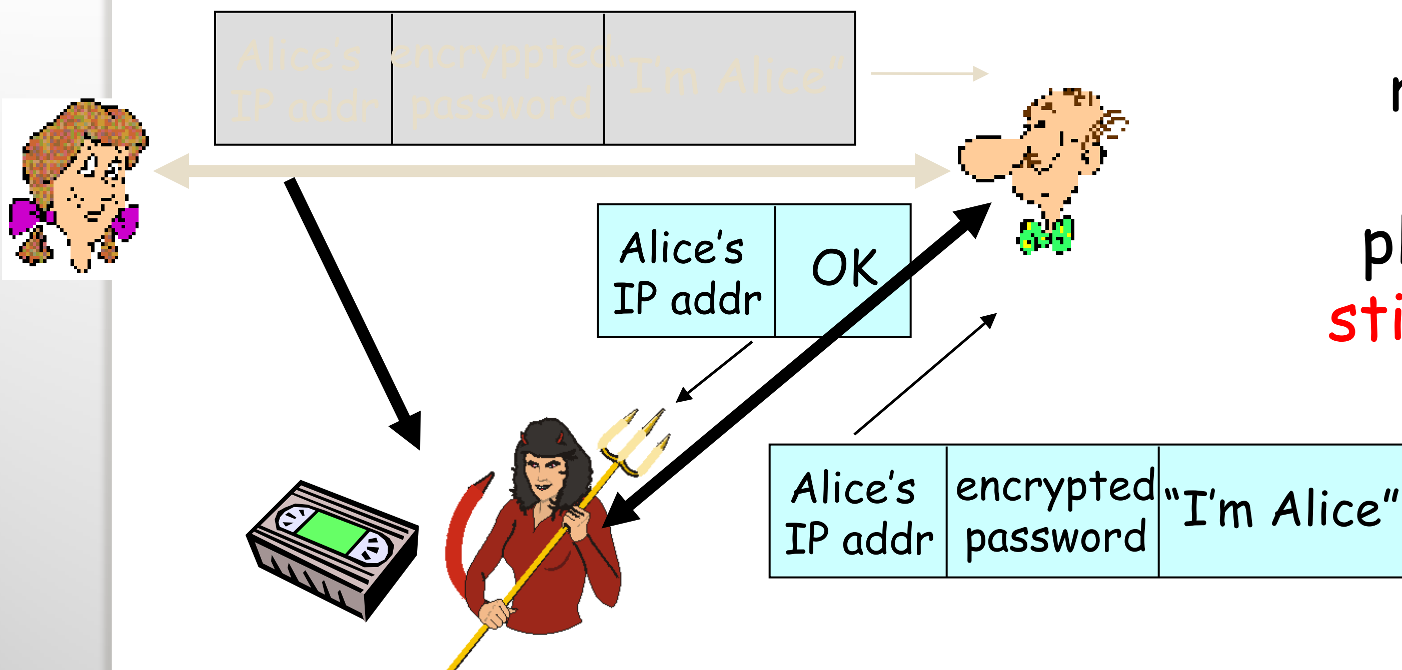
Protocol ap3.1: Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



Failure scenario??

Authentication: another try

<u>Protocol ap3.1:</u> Alice says "I am Alice" and sends her *encrypted* secret password to "prove" it.



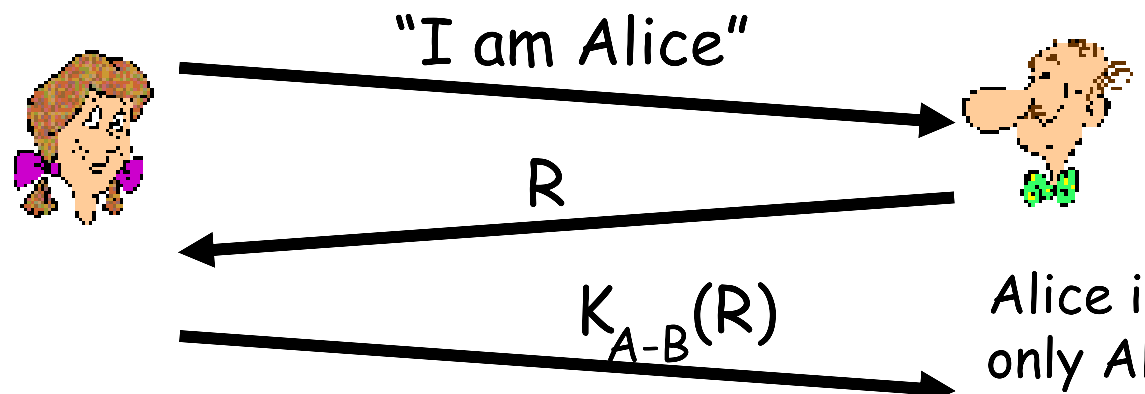
record and playback still works!

Authentication: yet another try

Goal: avoid playback attack

Nonce: number (R) used only *once –in-a-lifetime*

ap4.0: to prove Alice "live", Bob sends Alice nonce, R. Alice must return R, encrypted with shared secret key



Alice is live, and only Alice knows key to encrypt nonce, so it must be Alice!

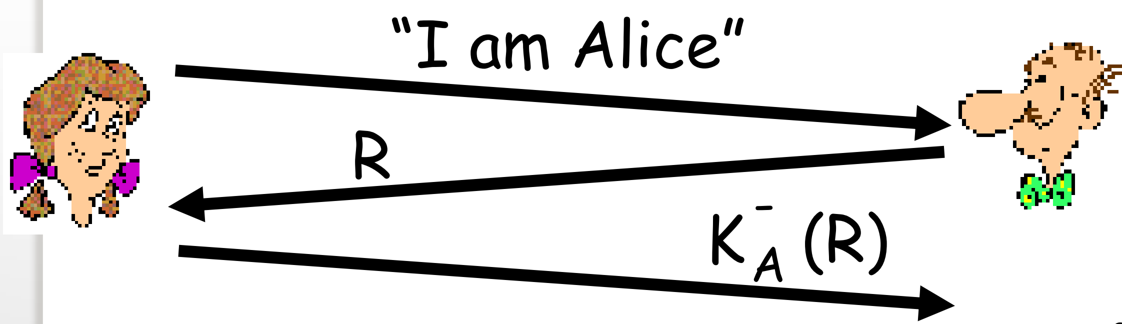
Failures, drawbacks?

Authentication: ap5.0

ap4.0 doesn't protect against <u>server database reading</u>

can we authenticate using public key techniques?

<u>ap5.0:</u> use nonce, public key cryptography



Bob computes

$K_A^+(K_A^-(R)) = R$

and knows only Alice could have the private key, that encrypted R such that

$K_A^+(K_A^-(R)) = R$

Trusted Intermediaries

Symmetric key problem:

- How do two entities establish shared secret key over network?

Trusted Intermediaries

<u>Symmetric key problem:</u>

- How do two entities establish shared secret key over network?

Solution:

- trusted key distribution center <u>(KDC) </u>acting as intermediary between entities

Trusted Intermediaries

Public key problem:

- When Alice obtains Bob's public key (from web site, e-mail, diskette), how does she know it is Bob's public key, not Trudy's?

Trusted Intermediaries

Public key problem:

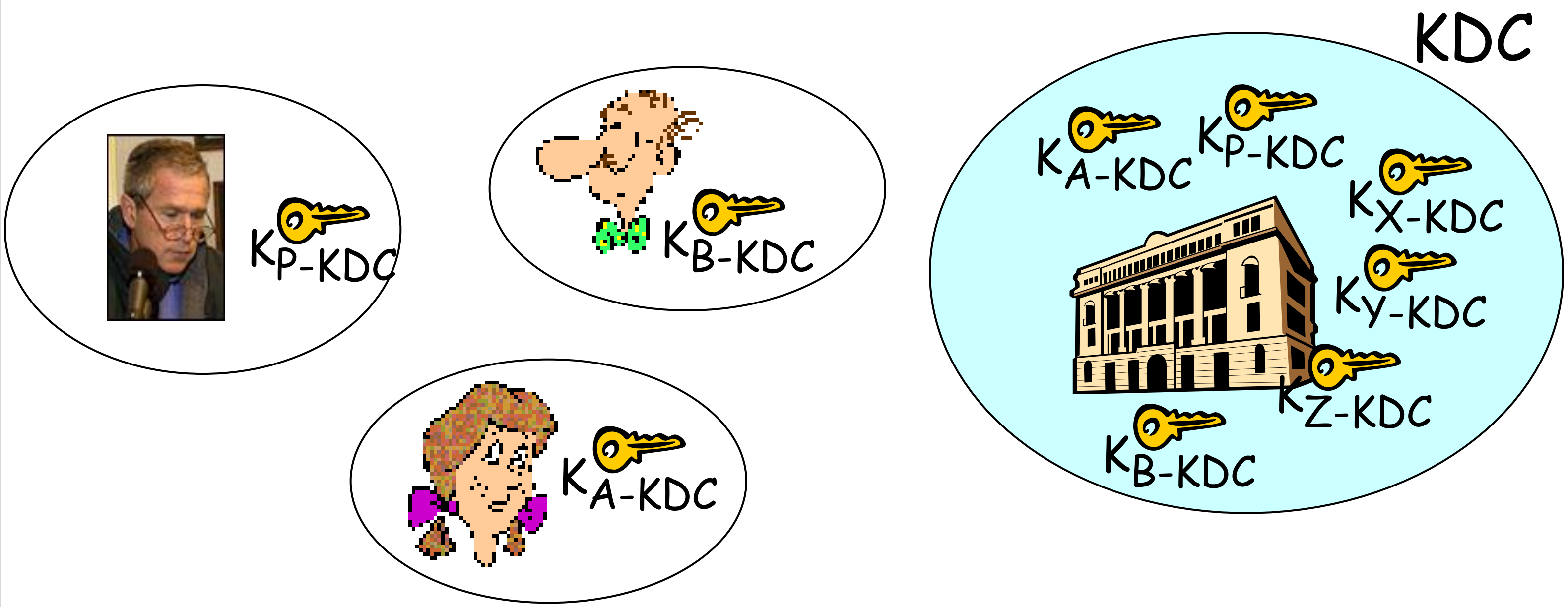
- When Alice obtains Bob's public key (from web site, e-mail, diskette), how does she know it is Bob's public key, not Trudy's?

Solution:

- trusted certification authority (CA)

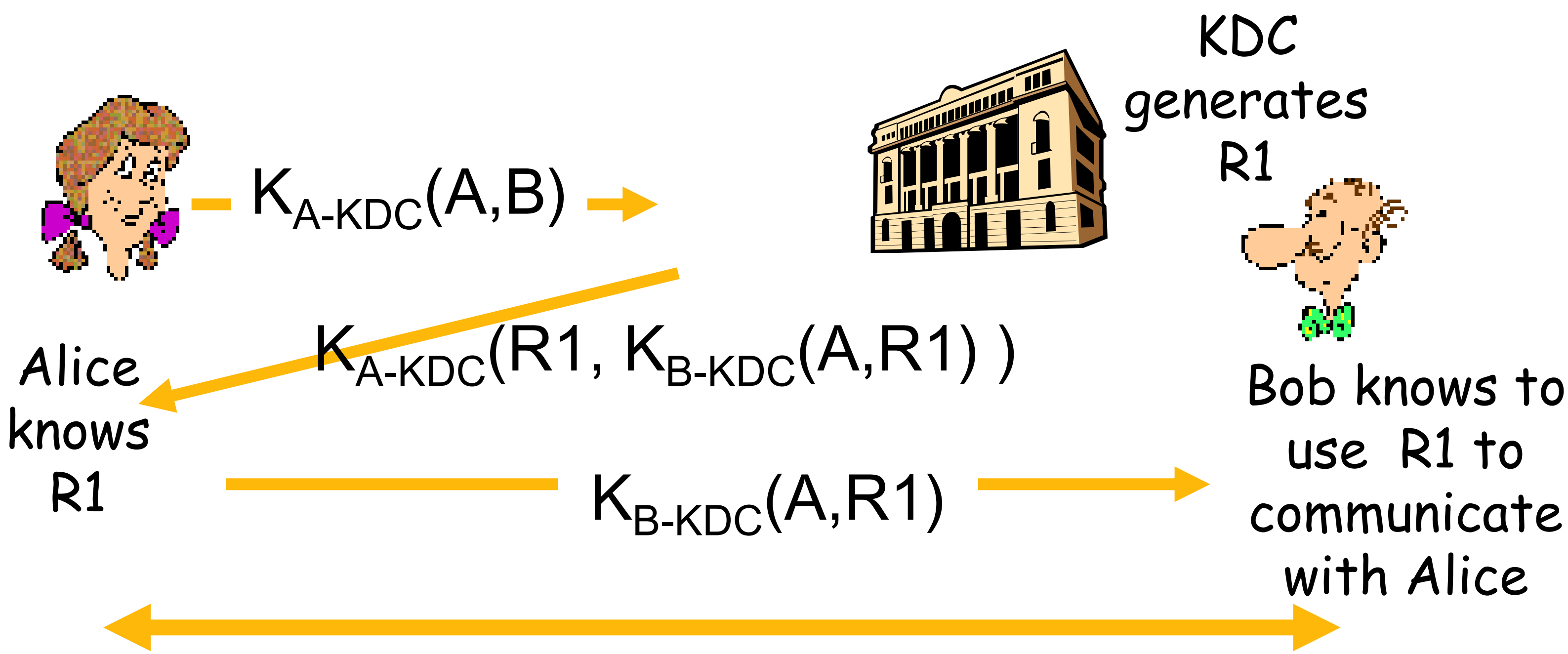
Key Distribution Center (KDC)

- Alice, Bob need shared symmetric key.
- KDC: server shares different secret key with *each* registered user (many users)
- Alice, Bob know own symmetric keys, $K_{A\text{-}KDC}$ $K_{B\text{-}KDC}$, for communicating with KDC.



Key Distribution Center (KDC)

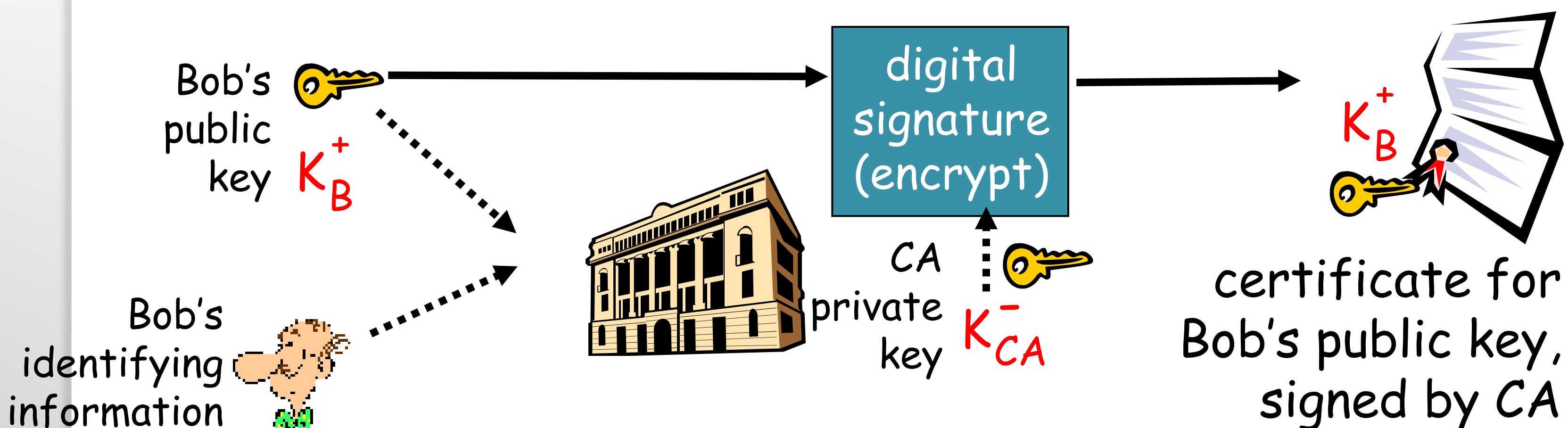
Q: How does KDC allow Bob, Alice to determine shared symmetric secret key to communicate with each other?



Alice and Bob communicate: using R1 as *session key* for shared symmetric encryption

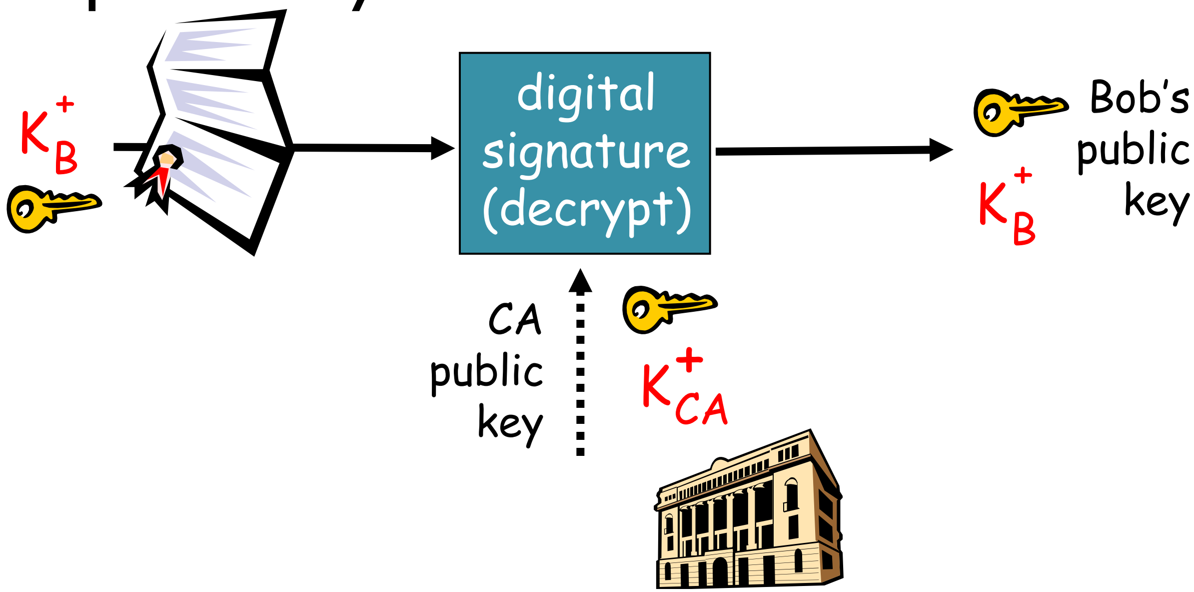
Certification Authorities

- Certification authority (CA): binds public key to particular entity E
- E registers its public key with CA
  - E provides "proof of identity" to CA
  - CA creates certificate binding E to its public key.
  - Certificate containing E's public key digitally signed by CA – CA says "this is E's public key"



Certification Authorities

- When Alice wants Bob's public key:
  - gets Bob's certificate (Bob or elsewhere)
  - apply CA's public key to Bob's certificate, get Bob's public key



Single KDC/CA

- Problems
  - Single administration trusted by all principals
  - Single point of failure
  - Scalability
- Solutions: break into multiple domains
  - Each domain has a trusted administration

Multiple KDC/CA Domains

Secret keys:

- KDCs share pairwise key
- topology of KDC: tree with shortcuts

Public keys:

- cross-certification of CAs
- example: Alice with CA_A, Boris with CA_B
  - Alice gets CA_B's certificate (public key p_1), signed by CA_A
  - Alice gets Boris' certificate (its public key p_2), signed by CA_B (p_1)

با تشکر
پرسش و پاسخ