Password hashing

CS 161: Computer Security

Prof. Raluca Ada Popa

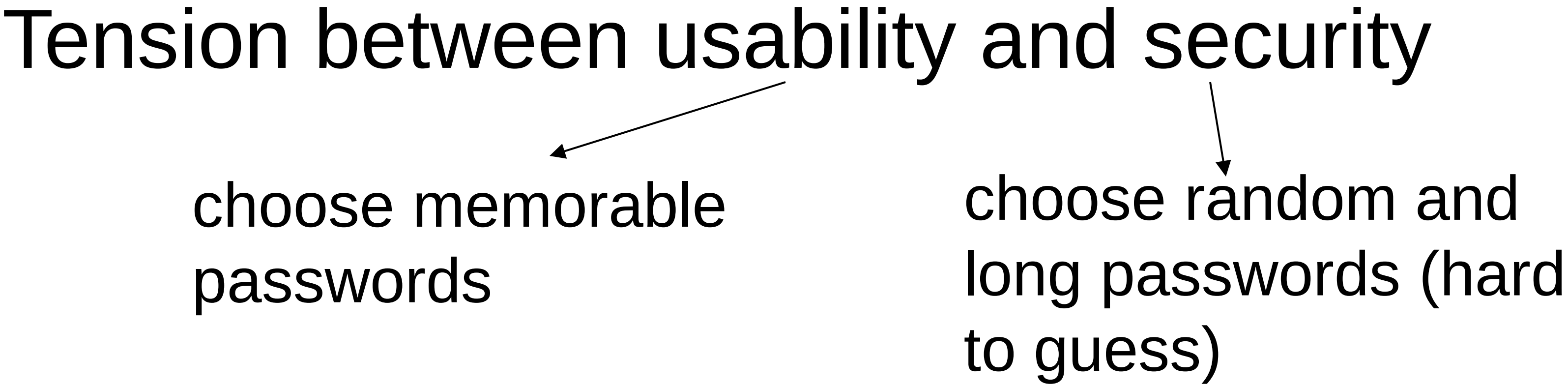
Feb 19, 2018

Announcement

Project 2 to be released Thursday

Midterm grades announced at end of week

Passwords



Tension between usability and security

choose memorable passwords

choose random and long passwords (hard to guess)

Attack mechanisms

- Online guessing attacks
  - Attacker tries to login by trying different user passwords in the live system
- Social engineering and phishing
  - Attacker fools user into revealing password
- Eavesdropping
  - Network attacker intercepts plaintext password on the connection
- Client-side malware
  - Key-logger/malware captures password when inserted and sends to attacker
- Server compromise
  - Attacker compromises server, reads storage and learns passwords

Defences/mitigations

Network eavesdropper:

- Encrypt traffic using SSL (will discuss later)

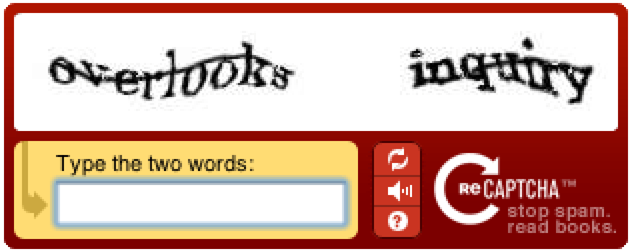
Client-side malware: hard to defend

- Intrusion detection mechanisms – detect malware when it is being inserted into the network
- Various security software (e.g., anti-virus)
- Use two-factor authentication

Mitigations for online-guessing attacks

- Rate-limiting
  - Impose limit on number of passwords attempts

- CAPTCHAs: to prevent automated password guessing



- Password requirements: length, capital letters, characters, etc.

Mitigations for server compromise

- Suppose attacker steals the database at the server including all password information
- Storing passwords in plaintext makes them easy to steal
- Further problem: users reuse passwords at different sites!

Don’t store passwords in plaintext at server!

Hashing passwords

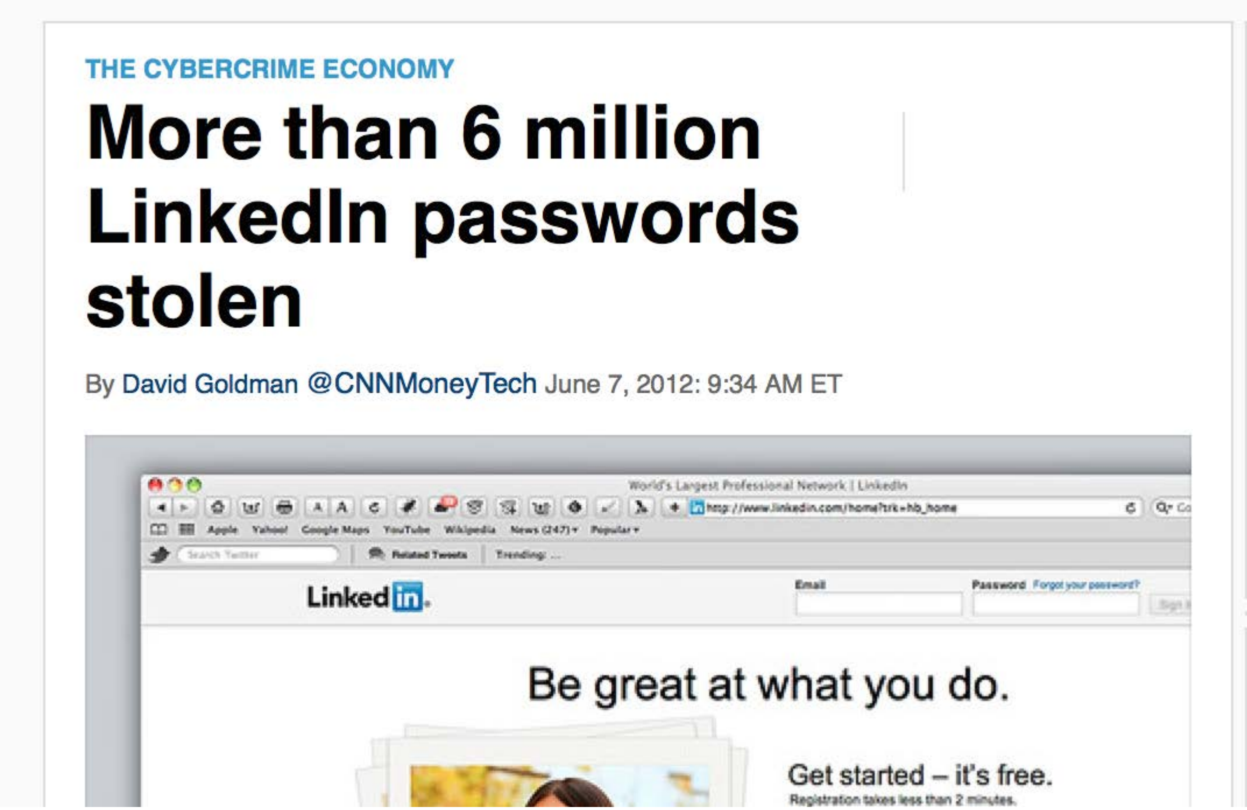
- Server stores hash(password) for each user using a cryptographic hash function
  - hash is a one-way function

username	**hash of password**
Alice	hash(Alice’s password)
Bob	hash(Bob’s password)

- When Alice logs in with password w (and provides w to server), server computes hash(w) and compares to Alice’s record

Password hashing: problems

- Offline password guessing
  - Dictionary attack: attacker tries all passwords against each hash(w)
  - Study shows that a dictionary of 2^{20} passwords can guess 50% of passwords
- Amortized password hashing
  - Idea: One brute force scan for all/many hashes
  - Build table (H(password), password) for all 2^{20} passwords
  - Crack 50% of passwords in this one pass



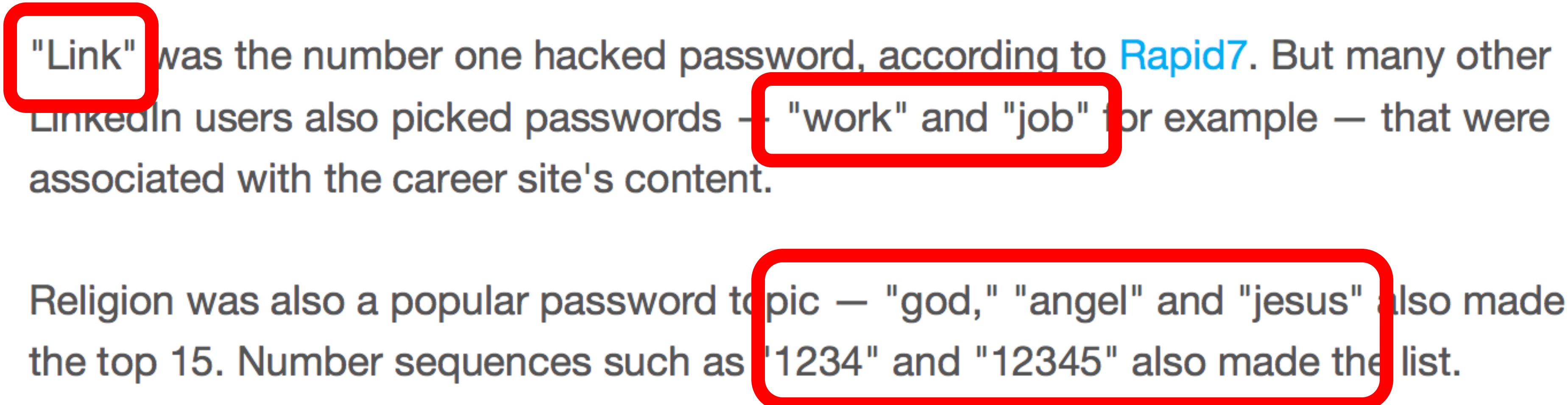
THE CYBERCRIME ECONOMY

More than 6 million LinkedIn passwords stolen

By David Goldman @CNNMoneyTech June 7, 2012: 9:34 AM ET

World's Largest Professional Network | LinkedIn
http://www.linkedin.com/home?trk=hb_home
Apple Yahoo! Google Maps YouTube Wikipedia News (247) Popular
Search Twitter
Related Tweets
Trending: ...
LinkedIn
Email
Password Forgot your password?
Be great at what you do.
Get started – it's free.
Registration takes less than 2 minutes.

LinkedIn was storing h(password)



"Link" was the number one hacked password, according to Rapid7. But many other LinkedIn users also picked passwords — "work" and "job" for example — that were associated with the career site's content.

Religion was also a popular password topic — "god," "angel" and "jesus" also made the top 15. Number sequences such as "1234" and "12345" also made the list.

Prevent amortized guessing attack

- Randomize hashes with salt
- Server stores (salt, hash(password, salt)), salt is random
- Two equal passwords have different hashes now
- Dictionary attack still possible, BUT need to do one brute force attack per hash now, not one brute force attack for many hashes at once

Salted hash example

username	salt	hash of password
Alice	235545235	hash(Alice’s password, 235545235)
Bob	678632523	hash(Bob’s password, 678632523)

Attacker tries to guess Alice’s password:

Computes table

‘aaaaaa’	hash(’aaaaaa’, 235545235),
‘aaaaab’	hash(’aaaaab’, 235545235),
…	
‘zzzzzzz’	hash(’zzzzzz’, 235545235)

This table is useless for Bob’s password because of different salt

Increase security further

- Would like to slow down attacker in doing a dictionary attack
- Use slow hashes = takes a while to compute the hash
- Define

  H(x) = hash(hash(hash(…hash(x))))

  use with x = password || salt

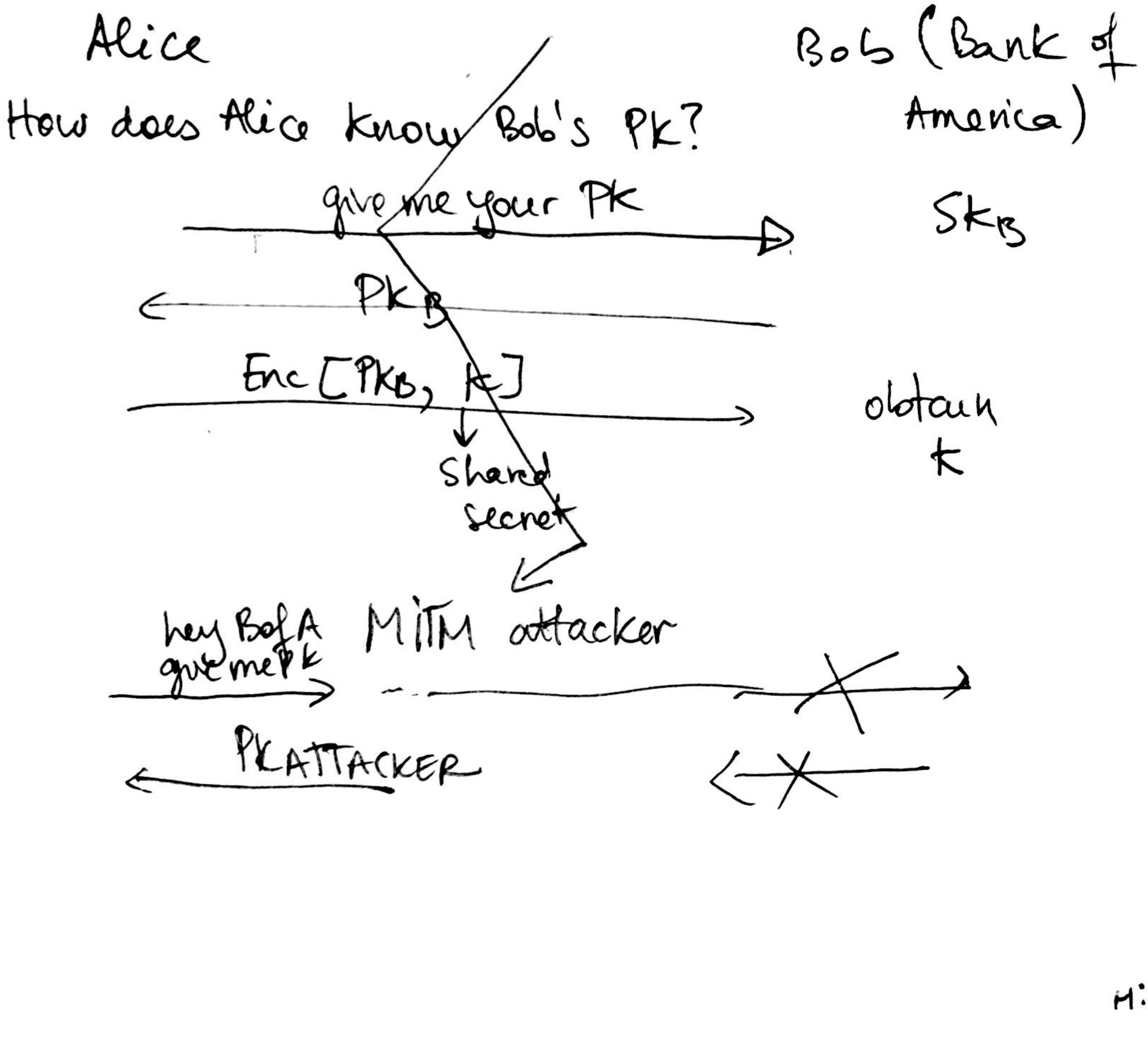
- Tension: time for user to authenticate & login vs attacker time
- If H is 1000 times slower and attack takes a day with H, attack now takes 3 years with F

Conclusions

- Do not store passwords in cleartext
- Store them hashed with salts, slower hash functions better

CS 161

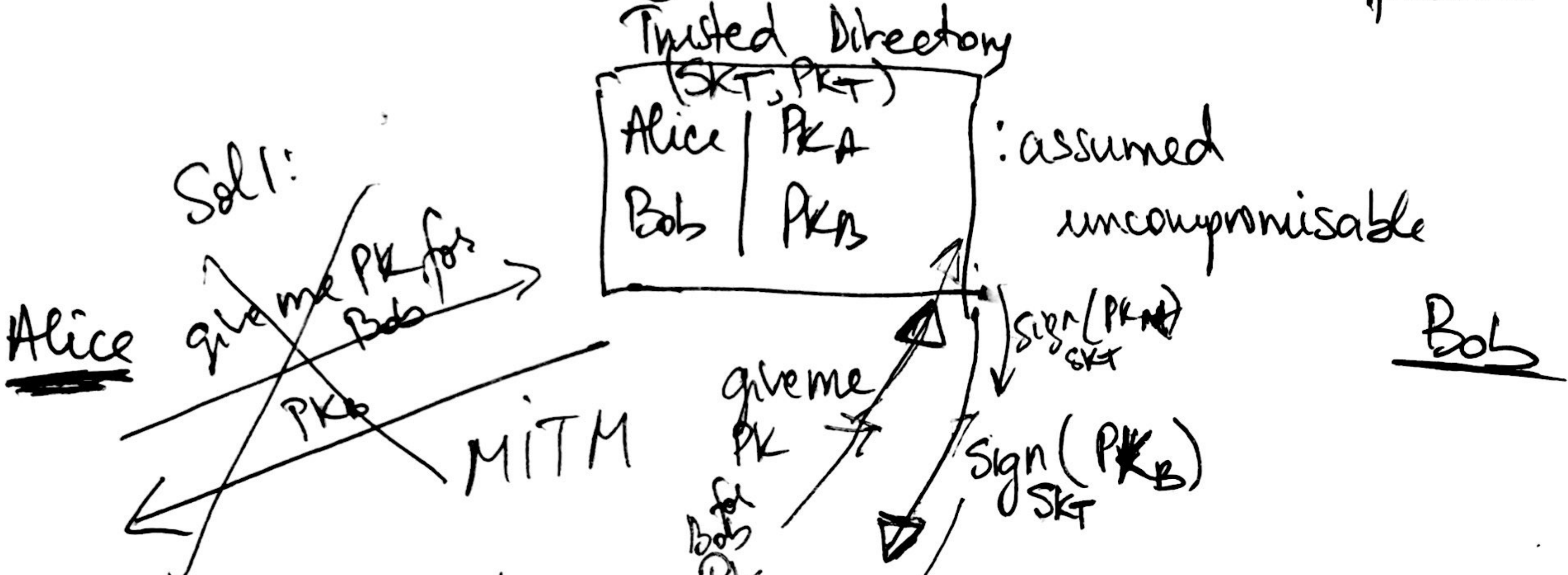
Key management


How does Alice know Bob's PK?

How can Alice obtain the PK of Bob securely

Public-key Infrastructure (PKI) = infrastructure (roles, policies, protocols) for managing PK and certificates

1) Trusted Directory Service



Alice	PK_A
Bob	PK_B

- everyone knows PK_T (Alice has PKT hardcoded in her browser)

Sol 2: TD answers with $sign_{SKT}(PK_B)$.

MITM attacker can return a Synat from TD for someo else

Problem: don't know if PK is Bob's

2) PKI Approach 2: Digital Certificates.

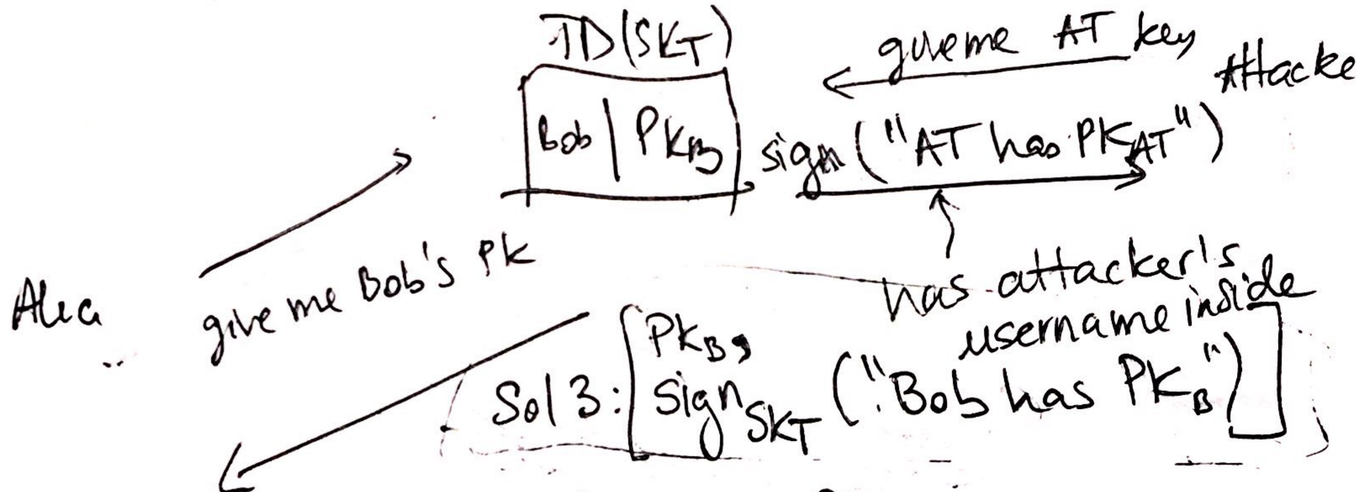
association between a username and their public as **certified** by some authority (e.g. Verisign)

CA = certificate authority (e.g. Verisign)

certificate = $\text{sign}_{SK_{CA}}(\text{username}, PK)$

Users verify certificates using PK_{CA} hardcoded in browsers

Ex: $\text{sign}_{SK_{Verisign}}(\text{Google IP:}, PK_{Google})$



- checks signature verifies using PK_T
- and Bob is in the signature

Suppose Bob's $\underline{SK_B}$ was compromised ⇒ Attacker has old SK_B

Bob generates new (PK_B', SK_B') and updates TD so it contains [Bob, PK_B'] ↑public

Problem with Sol 3: Attacker becomes MITM and replaces TD's response containing PK_B' with <u>old</u> response with PK_B <u>replay attack</u> (replaying old information)

Alice $\xrightarrow{\text{hey, Bob, give me PK}}$ Bob

$$\xleftarrow{PK_B,\ \underbrace{Sign_{SK_{CA}}(Bob, PK_B)}_{\text{can obtain certificate from anyone}}}$$

+ CA does not have to be online.

What if Bob changes PK? - add expiry to certificate

certificate: $Sign_{SK_{CA}}(\overset{\text{(name)}}{Bob}, PK_B, \underset{\text{April, 2018}}{\underline{expiry}})$

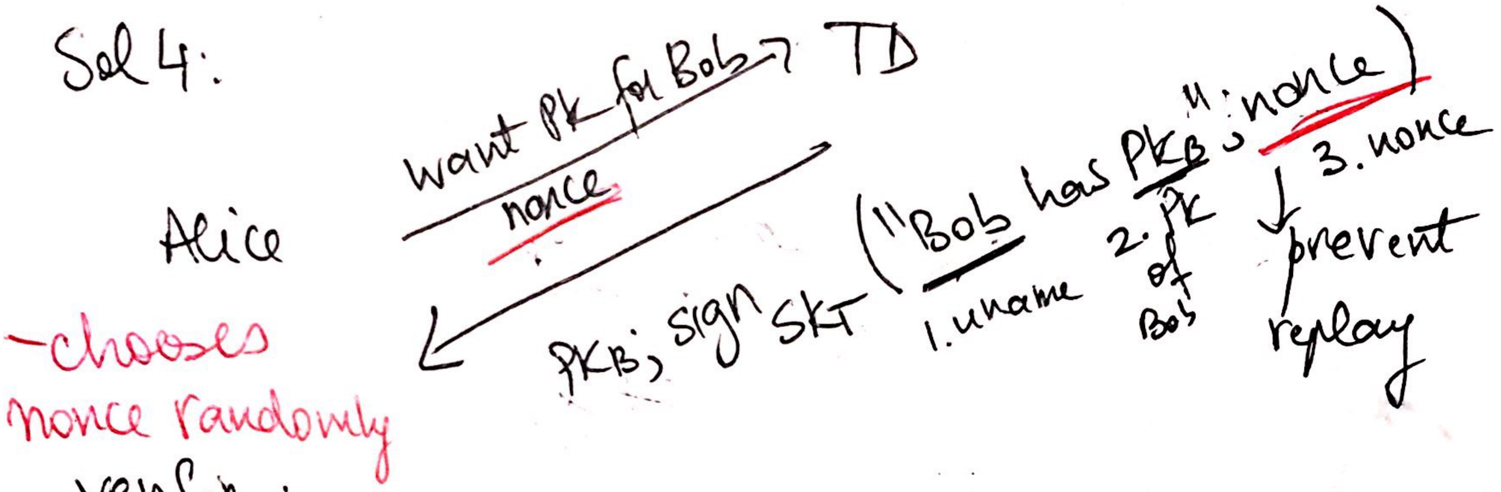
When expires, Bob needs to obtain new certificate from CA with new expiration date

Prevents replay only across expiration periods

If Bob's SK_B gets compromised before expiry, Attacker can impersonate Bob till then

Scanned by CamScanner

Sol 4:



Alice → TD: want PK for Bob, nonce

TD → Alice: PKB; sign SKT ("Bob has PKB", nonce)

verifies:

1) signature verifies using PK_T

2) signature has Bob's username

3) contains nonce Alice sent

Drawbacks: central point of attack and failure

TD has to be online always

Not scalable: has to know everyone's PK