CS255: identification protocols

Announcements:

- HW#4 is out on the course web site
- Last lecture: guest lecture by Jennifer Granick, ACLU

Quick recap

Signatures:

- From trapdoor functions (such as RSA)
- From CRH (one-time sigs ⇒ many-time sigs, good for software updates)
- From discrete-log: next week

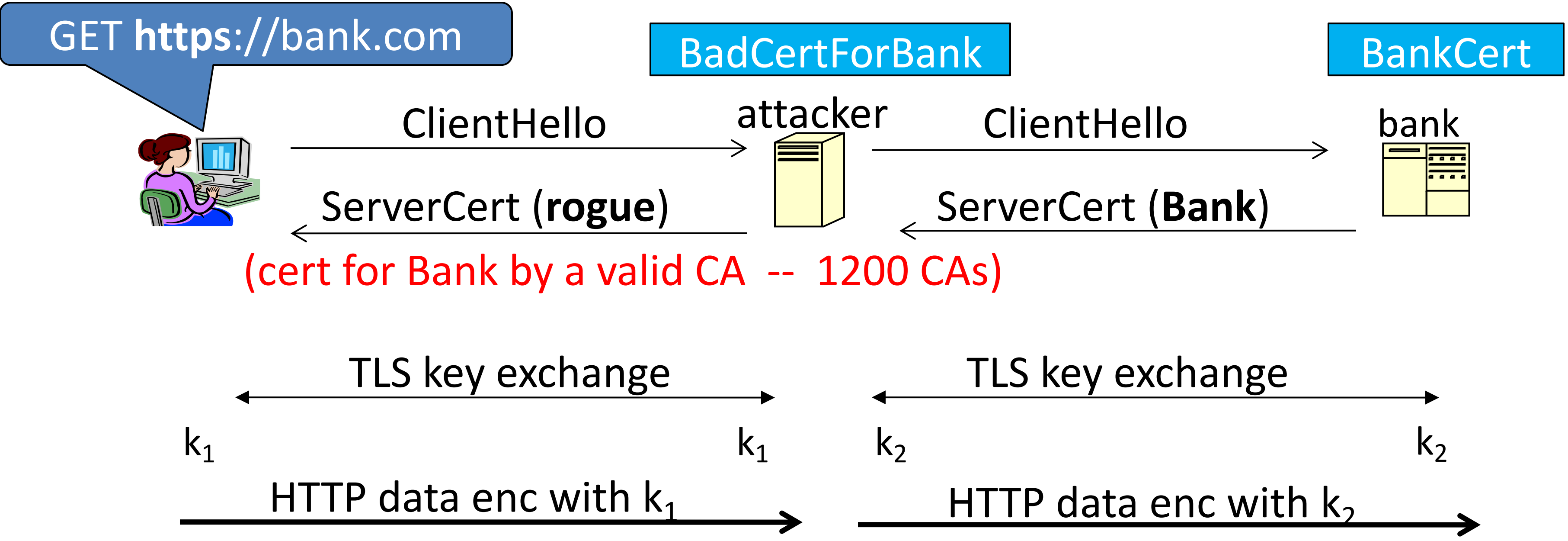
Certificates: bind a public key to an identity

[issuer-id, subject-id, PK, validity-period, serial #, …] + [CA sig]

Revocation methods: expiration and CRLset (list of revoked serial #s)

What if a CA incorrectly issues a cert to an adversary?

Certificate wrong issuance: the problem



Person-in-the-middle attack:
attacker sees all traffic, server cannot detect

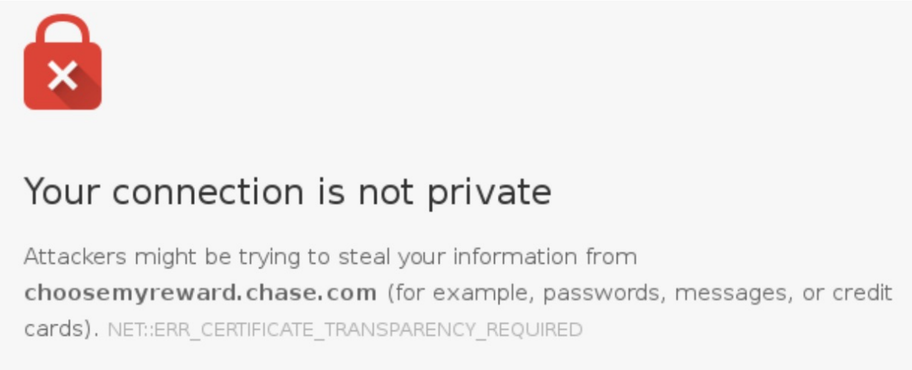
A defense: cert transparency (CT)

Idea: CA’s must push all certs. they issued to a public log

- Browser will only use a cert if it is published on (two) log servers
- Server attaches to certificate a signed statement from log (SCT)
- Companies can scan logs to look for invalid issuance (service by CA)

April 30, 2018:

- **CT required by chrome.**
  Otherwise, cert is rejected.



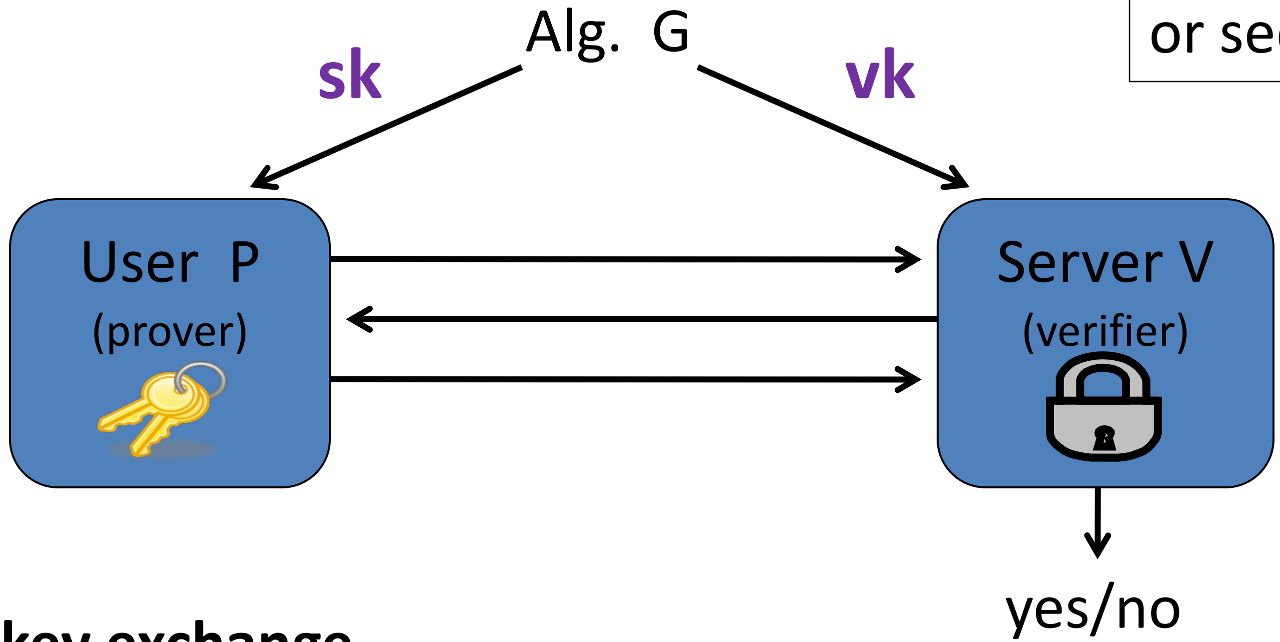
Part 3: Done with crypto primitives, moving on to protocols.

ID protocols

Overview

The Setup

vk either public
or secret



no key exchange

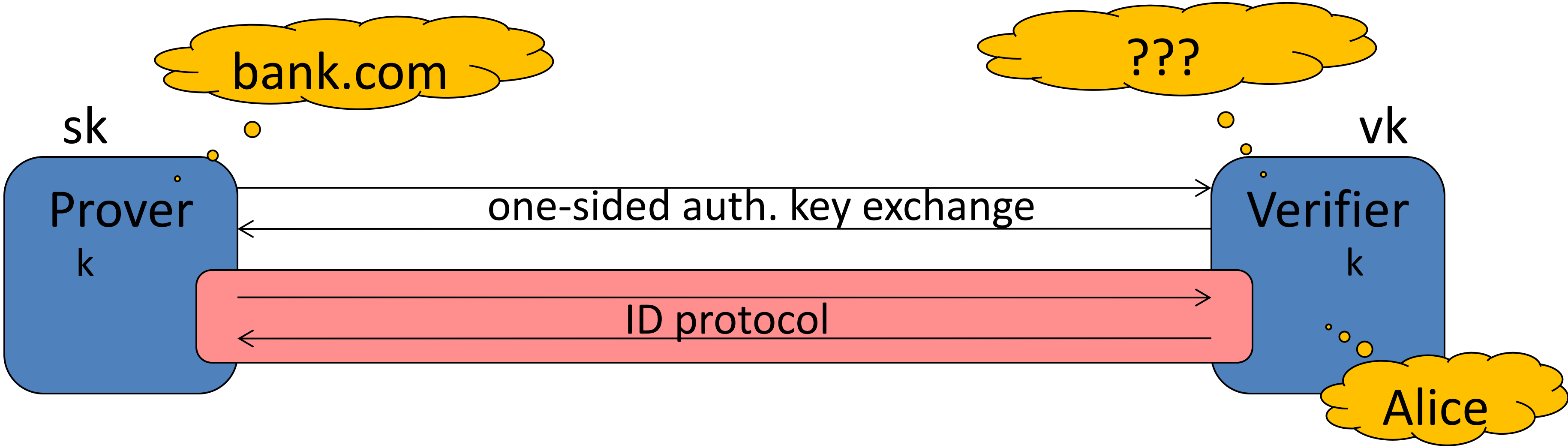
Applications: physical world

– Physical locks: (friend-or-foe)
  - Wireless car entry system
  - Opening an office door

– Login at a bank ATM or a desktop computer

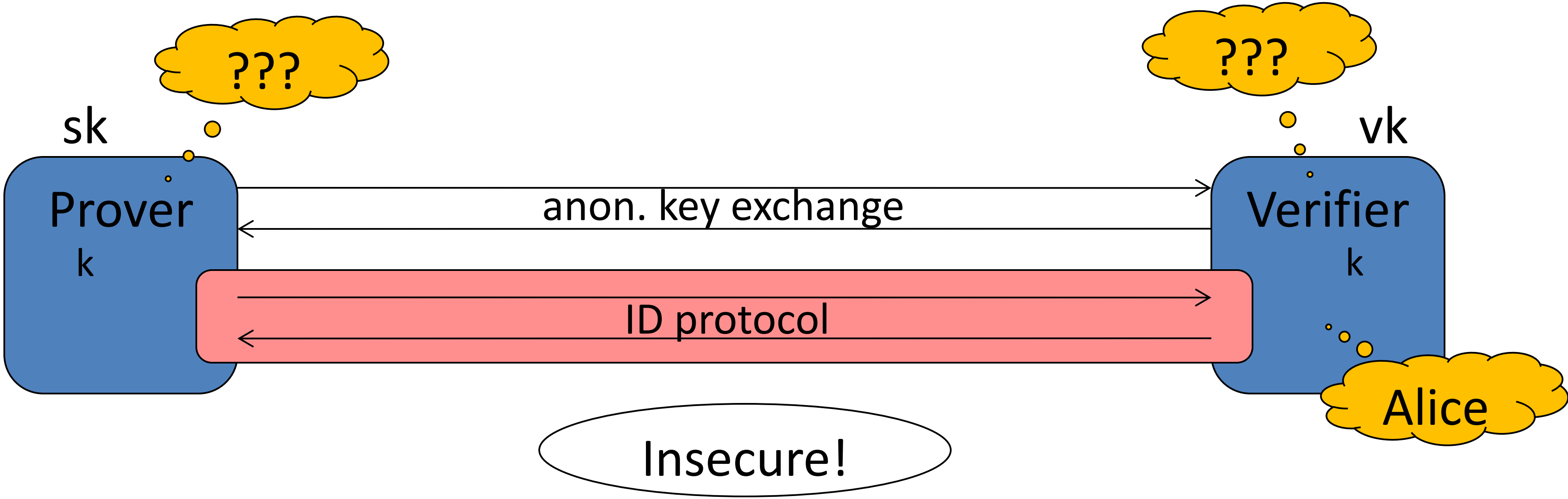
Applications: Internet

Login to a remote web site after a key-exchange with one-sided authentication (e.g. HTTPS)



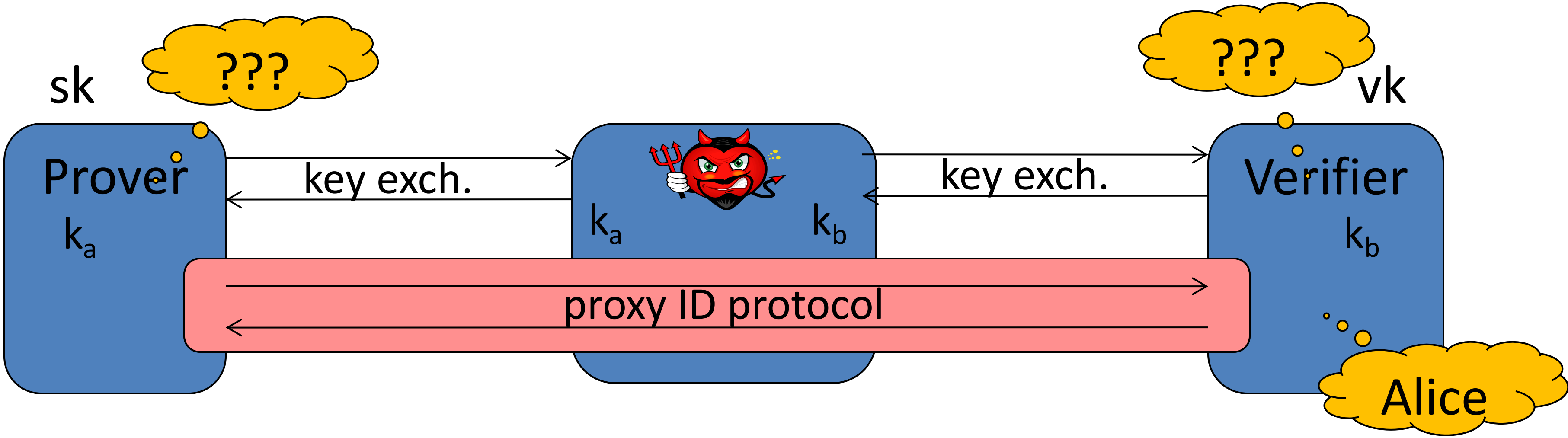
ID Protocols: how not to use

- ID protocol do not establish a secure session between Alice and Bob !!
  - Not even when combined with anonymous key exch.
  - Vulnerable to man in to the middle attacks



ID Protocols: how not to use

- ID protocol do not set up a secure session between Alice and Bob !!
  - Not even when combined with anonymous key exch.
  - Vulnerable to man in to the middle attack



ID Protocols: Security Models

1. **Direct Attacker**: impersonates prover with no additional information (other than vk)
   – Door lock

2. **Eavesdropping attacker**: impersonates prover after eavesdropping on a few conversations between prover and verifier
   – Wireless car entry system

3. **Active attacker**: interrogates prover and then attempts to impersonate prover
   – Fake ATM in shopping mall

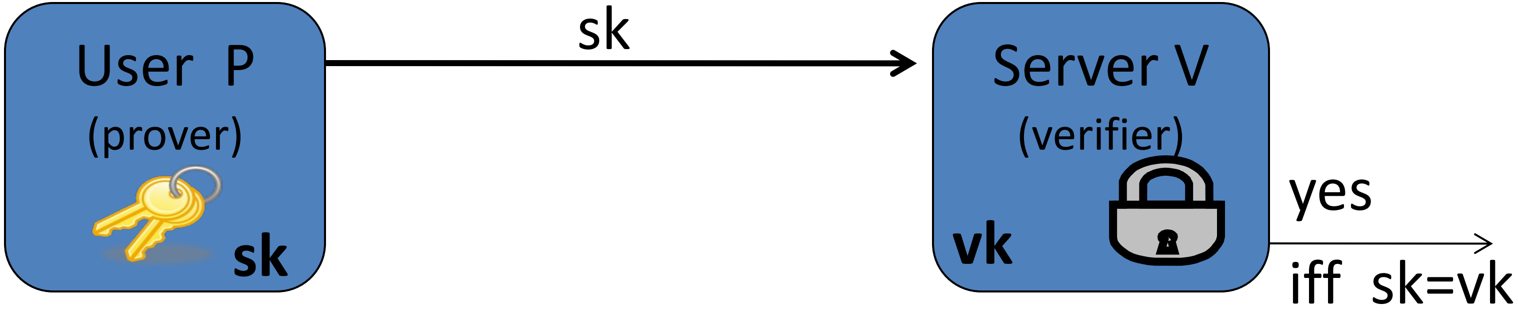
ID protocols

Direct attacks

Basic Password Protocol (incorrect version)

- **PWD**: finite set of passwords

- Algorithm G (KeyGen):
  - choose pw $\leftarrow$ PWD. output sk = vk = pw.



Basic Password Protocol (incorrect version)

Problem: vk must be kept secret

- Compromise of server exposes all passwords
- Never store passwords in the clear!

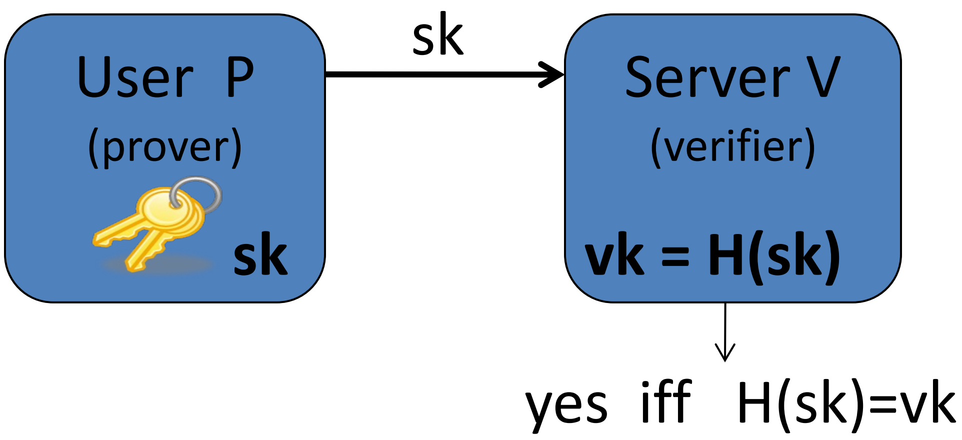
password file on server

Alice	pw_{alice}
Bob	pw_{bob}
…	…

Basic Password Protocol: version 1

H: one-way hash function from PWD to X

- "Given H(x) it is difficult to find y such that H(y)=H(x)"



password file on server

Alice	$H(pw_A)$
Bob	$H(pw_B)$
…	…

Problem: Weak Password Choice

Users frequently choose weak passwords:

(SplashData, 2018, from more than 5 million passwords leaked on the Internet)

1. 123456
2. password
3. 123456789
4. 12345678
5. 12345
6. 111111
7. 1234567
8. sunshine
9. qwerty
10. iloveyou

Dictionary of 360,000,000 words covers about 25% of user passwords

Note: Google password checker

- The 25 top passwords on the list cover more than 10% of users
- Nearly 3% of people use the worst password, 123456.

Online dictionary attack: attacker has a list of usernames.
For each username the attacker tries the password '123456'.

- Success after 33 tries on average (!)

Can be mitigated by e.g., IP-based rate limiting

Offline Dictionary Attacks

Suppose attacker obtains a **single** vk = H(pw) from server

- **Offline** attack: hash all words in Dict until a word w is found such that H(w) = vk
- Time O(|Dict|) per password

Off the shelf tools (e.g. John the ripper):

- Scan through <u>**all**</u> 7-letter passwords in a few minutes
- Scan through 360,000,000 guesses in few seconds

⇒ will recover 23% of passwords

Batch Offline Dictionary Attacks

Suppose attacker steals **entire** pwd file F

- Obtains hashed pwds for **all** users
- Example (2012): Linkedin (6M: SHA1(pwd))

Alice	$H(pw_A)$
Bob	$H(pw_B)$
…	…

Batch dict. attack:

- For each $w \in$ Dict: test if $H(w)$ appears in F (using fast look-up)

Total time: **O(|Dict| + |F|)** [Linkedin: 6 days, 90% of pwds. recovered]

Much better than attacking each password individually !

Preventing Batch Dictionary Attacks

Public salt:

- When setting password, pick a random n-bit salt S
- When verifying pw for A, test if **$H(pw, S_A) = h_A$**

id	S	h
Alice	S_A	$H(pw_A\ ,\ S_A)$
Bob	S_B	$H(pw_B\ ,\ S_B)$
…	…	…

Recommended salt length, n = 64 bits

- Attacker must re-hash dictionary for each user

Batch attack time is now: O(**|Dict| × |F|**)

How to hash a password?

<u>Linked-in</u>: **SHA1** hashed (**unsalted**) passwords

$\Rightarrow$ 6 days, 90% of passwords recovered by exhaustive search

The problem: SHA1 is too fast …

attacker can try all words in a large dictionary

To hash passwords:

- Use a **<u>keyed</u>** hash function (e.g., HMAC) where key stored in HSM
- In addition: use a **<u>slow</u>**, **<u>space-hard</u>** function

How to hash?

<u>PBKDF2</u>, <u>bcrypt</u>: slow hash functions

- Slowness by "iterating" a crypto hash function like SHA256

  Example: H(pw) = SHA256(SHA256(… SHA256(pw, S_A) …))

- Number of iterations: set for 1000 evals/sec
- Unnoticeable to user, but makes offline dictionary attack harder

Problem: custom hardware (ASIC) can evaluate hash function 50,000x faster than a commodity CPU

$\Rightarrow$ attacker can do dictionary attack much faster than 1000 evals/sec.

How to hash: a better approach

<u>Scrypt</u>: a slow hash function AND need lots of memory to evaluate

$\Rightarrow$ custom hardware not much faster than commodity CPU

Problem: memory access pattern depends on input password

$\Rightarrow$ local attacker can learn memory access pattern for a given password

$\Rightarrow$ eliminates need for memory in an offline dictionary attack

Is there a space-hard function where time is independent of pwd?

- Password hashing competition (2015): **Argon2i** (also Balloon)

ID protocols

Security against eavesdropping attacks

(one-time password systems)

Eavesdropping Security Model

Adversary is given:

- Server's vk, and
- the transcript of several interactions between honest prover and verifier. (example: remote car unlock)

adv. goal is to impersonate prover to verifier

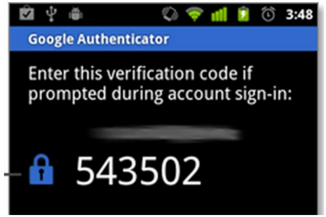
A protocol is "secure against eavesdropping" if no efficient adversary can win this game

The password protocol is clearly insecure !

One-time passwords (secret vk, stateful)

Setup (algorithm G):

- Choose random key **k**
- Output **sk = (k,0)** ; **vk = (k,0)**



Identification:

prover

sk = (k,0)

↓

sk = (k,1)

⋮

$r_0 \leftarrow F(k,0)$ →

$r_1 \leftarrow F(k,1)$ →

6 digits

server

vk = (k,0)

↓

vk = (k,1)

⋮

Yes iff $r = F(k,0)$

often, time-based updates: $r \leftarrow F(k, \text{time})$ [stateless]

The SecurID system (secret vk, stateful)

"Thm": if F is a secure PRF then protocol is secure against eavesdropping

RSA SecurID uses AES-128:

128 bit key → F → 6 digit output
32 bit ctr → F

Advancing state: sk $\leftarrow$ (k, i+1)

- Time based: every 60 seconds (TOTP)
- User action: every button press

Both systems allow for skew in the counter value

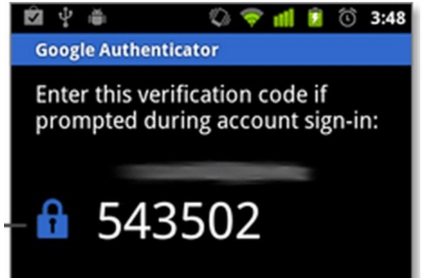
TOTP: Google authenticator

6-digit timed one-time passwords (TOTP) based on [RFC 6238]

To enable TOTP for a user: web site presents QR code with embedded data:

otpauth://totp/Example:**alice@dropbox.com**?
secret=**JBSWY3DPEHPK3PXP** & issuer=Example

Subsequent user logins require user to present TOTP



Server compromise exposes secrets

March 2011:

- RSA announced servers attacked, secret keys stolen

  $\Rightarrow$ enabled SecurID user impersonation

Is there an ID protocol where server key vk is public?

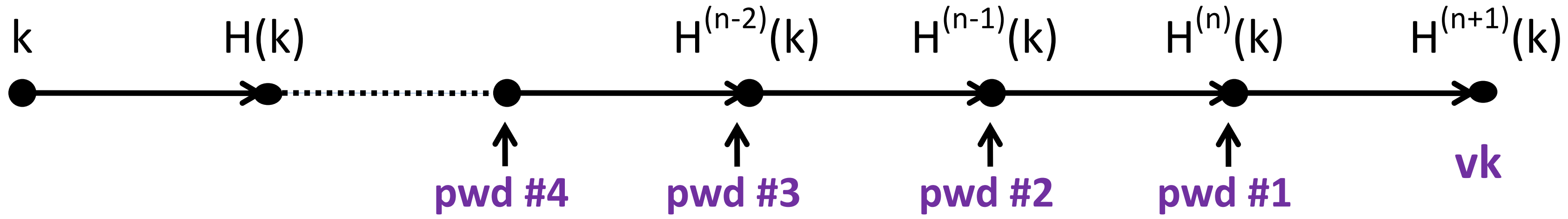
The S/Key system (public vk, stateful)

Notation: $H^{(n)}(x) = \underbrace{H(H(\ldots H(x)\ldots))}_{\text{n times}}$

<u>Algorithm G:</u> (setup)

- Choose random key $k \leftarrow K$
- Output **sk = (k,n)** ; **vk = $\mathbf{H^{(n+1)}(k)}$**

<u>Identification</u>:



The S/Key system (public vk, stateful)

Identification (in detail):

- Prover ($sk=(k,i)$): send $t \leftarrow H^{(i)}(k)$; set $sk \leftarrow (k,i-1)$
- Verifier($vk=H^{(i+1)}(k)$, t): if $H(t)=vk$ then $vk \leftarrow t$, output "yes"

Notes: vk can be made public;
but need to generate new sk after n logins ($n \approx 10^6$)

"Thm": S/Key_n is secure against eavesdropping (public vk)
provided H is one-way on n-iterates

SecurID vs. S/Key

S/Key:

- **public** vk, **limited** number of authentications
- Long authenticator t (e.g., 80 bits)

SecurID / TOTP:

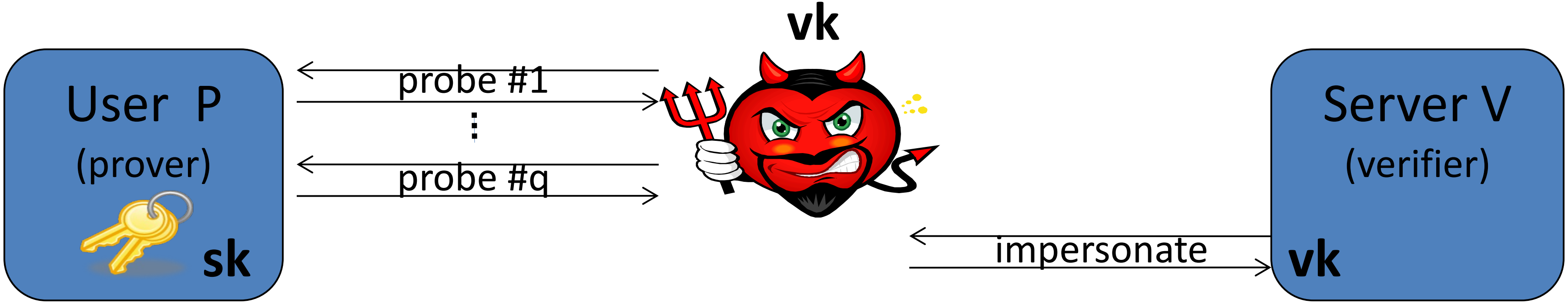
- **secret** vk, **unlimited** number of authentications
- Short authenticator (6 digits)

ID protocols

Security against active attacks

(challenge-response protocols)

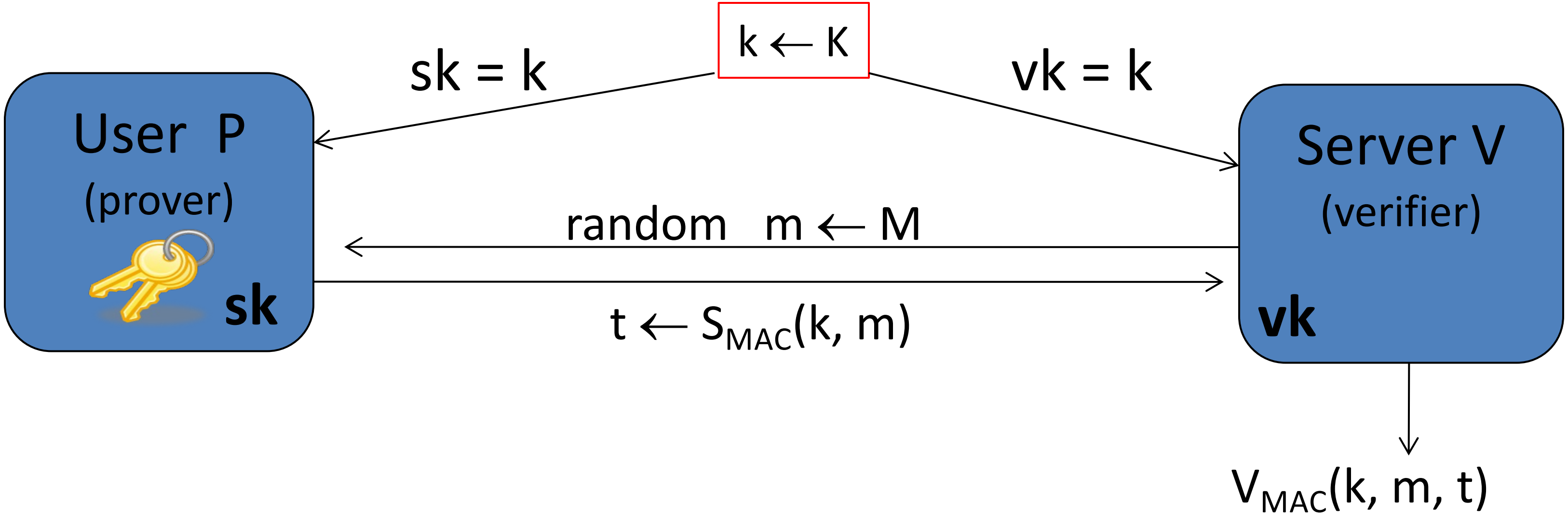
Active Attacks



- Offline fake ATM: interacts with user; later tries to impersonate user to real ATM
- Offline phishing: phishing site interacts with user; later authenticates to real site

All protocols so far are vulnerable

MAC-based Challenge Response (secret vk)



"Thm": protocol is secure against active attacks (secret vk),
provided (S_{MAC}, V_{MAC}) is a secure MAC and $|M| \geq 2^{128}$

MAC-based Challenge Response

Problems:

- vk must be kept secret on server
- dictionary attack when k is a human pwd:

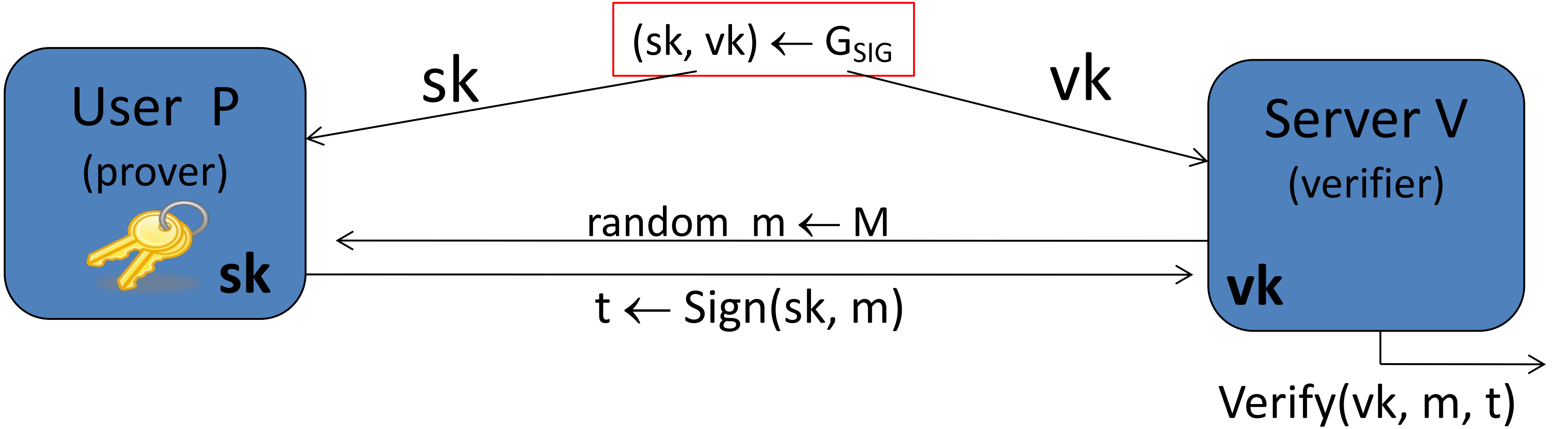
  Given [m , S_{MAC}(pw, m)] eavesdropper can
  try all pw $\in$ Dict to recover pw

Main benefit:

- Both m and t can be short
- CryptoCard: 8 chars each

Sig-based Challenge Response (public vk)

Replace MAC with a digital signature:



"Thm": Protocol is secure against active attacks **(public vk),** provided $(G_{SIG}, \text{Sign}, \text{Verify})$ is a secure digital sig. and $|M| \geq 2^{128}$

but t is long (≥20 bytes)

Signature-based Challenge Response in the real world

The Universal Second Factor (U2F) Standard

(and WebAuthn)

Goals:

- **Browser malware cannot steal user credentials**
- U2F should not enable tracking users across sites
- U2F uses counters to defend against token cloning

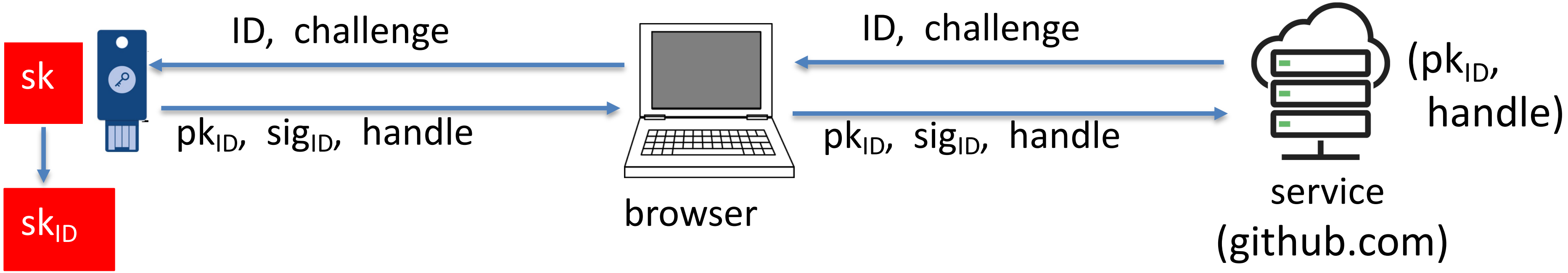
U2F token

browser

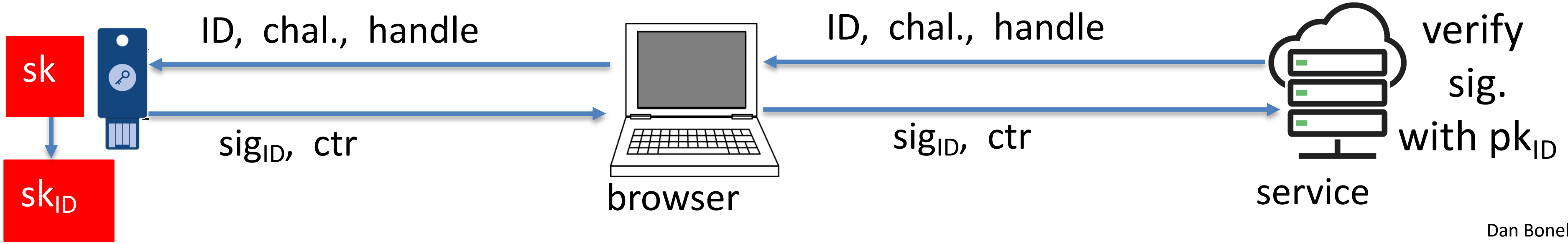
service (github.com)

The U2F protocol: two parts (simplified)

Device registration:

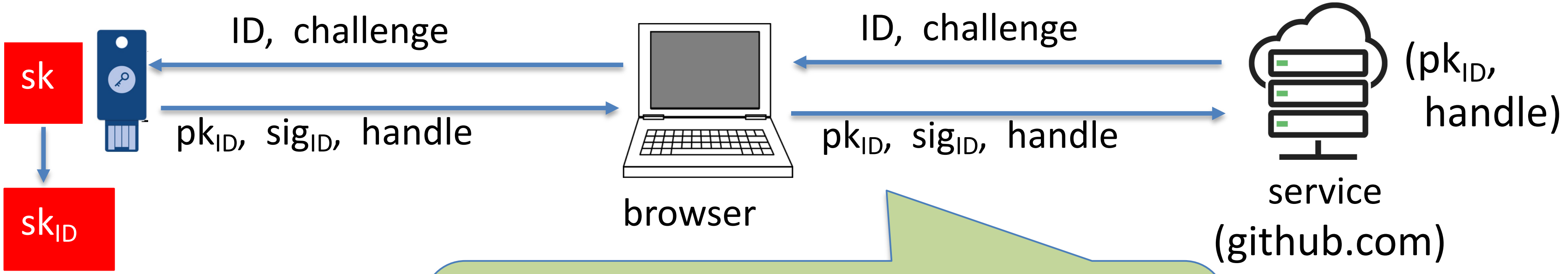


Authentication:



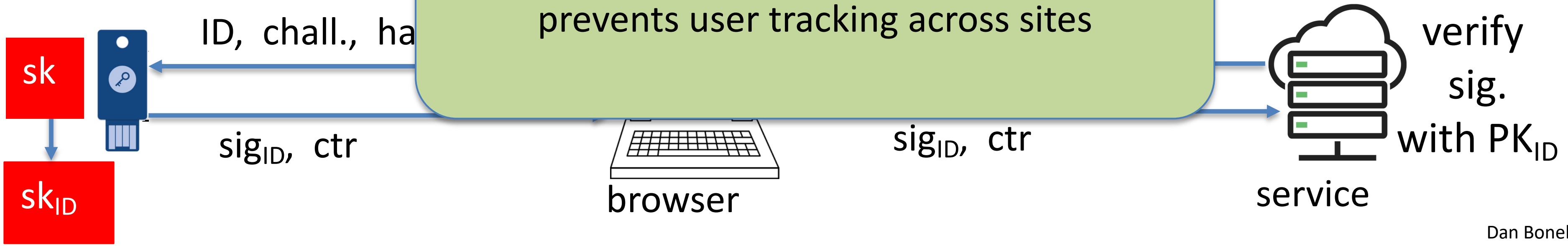
The U2F protocol: two parts (simplified)

Device registration:



Unlinkable pk_{ID} per site
prevents user tracking across sites

Authentication:



Summary

ID protocols: useful in settings where adversary cannot interact with prover during impersonation attempt

Three security models:

- **Direct**: passwords (properly salted and hashed)
- **Eavesdropping attacks**: One time passwords
  - SecurID: secret vk, unbounded logins
  - S/Key: public vk, bounded logins
- **Active attacks**: challenge-response

THE END