

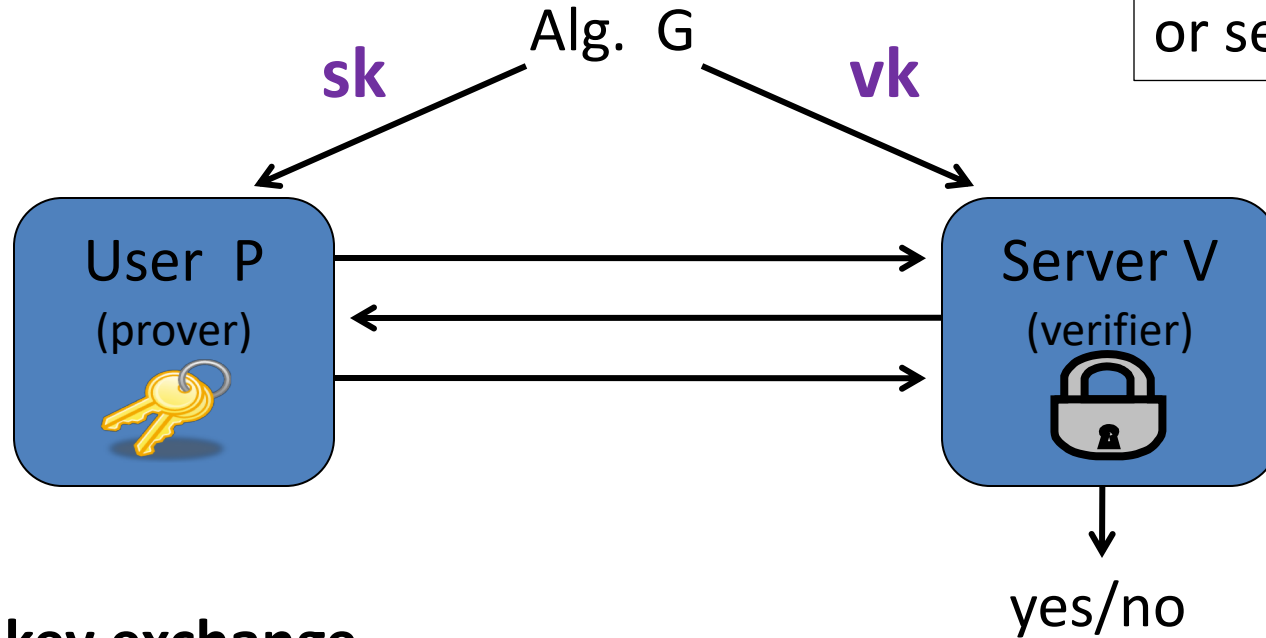


ID protocols

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

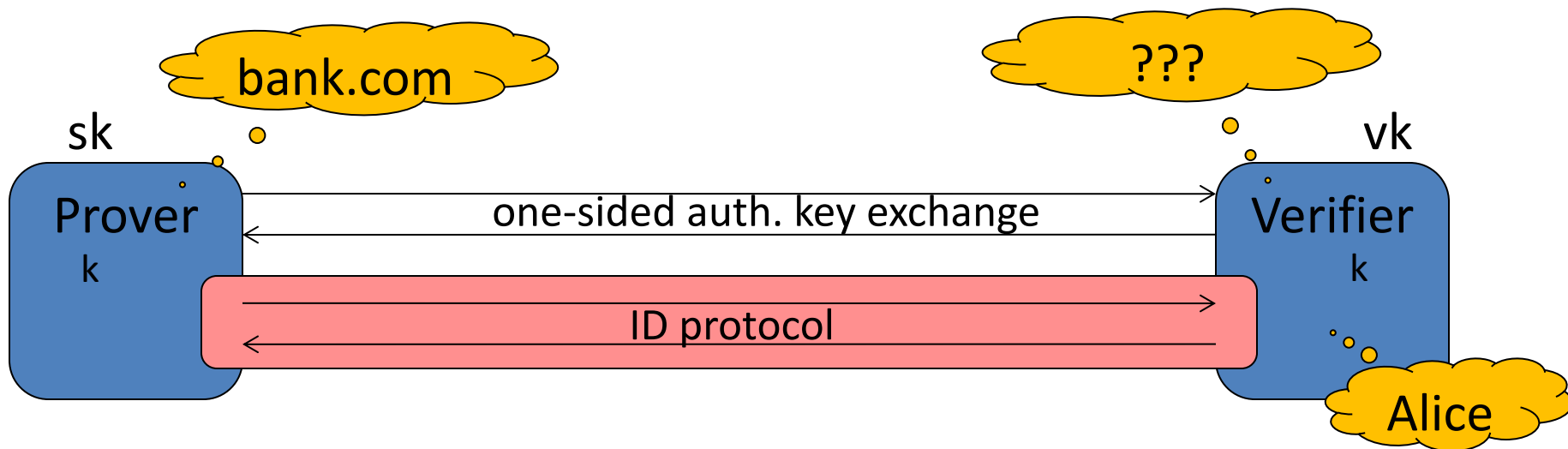


- Login at a bank ATM or a desktop computer



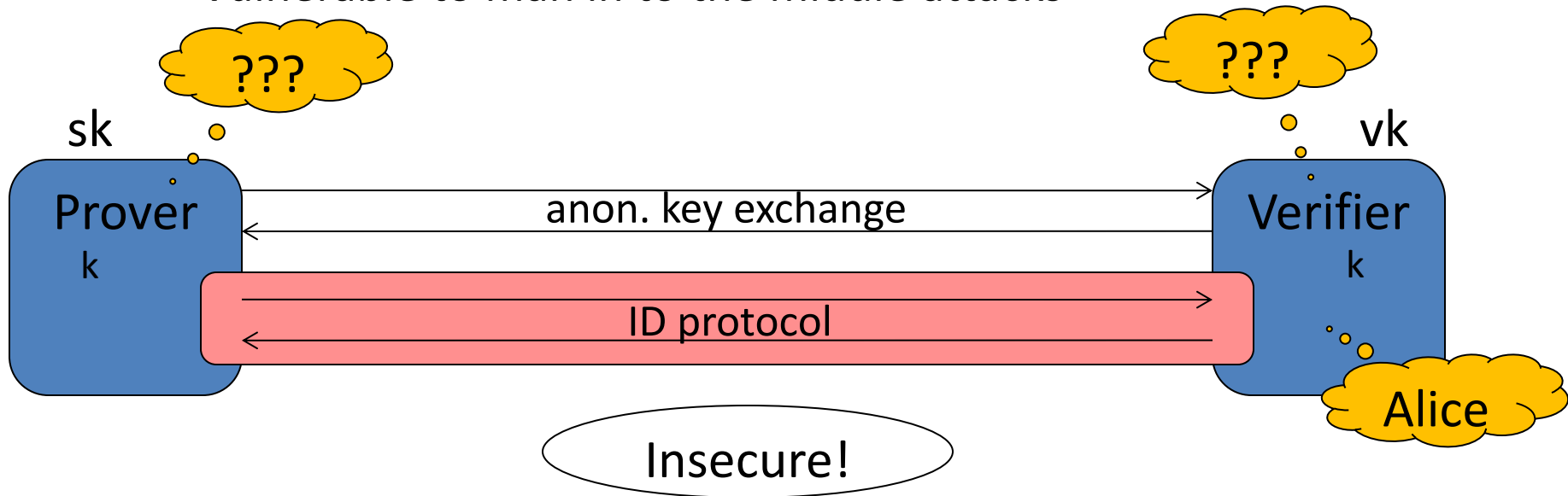
Applications: Internet

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



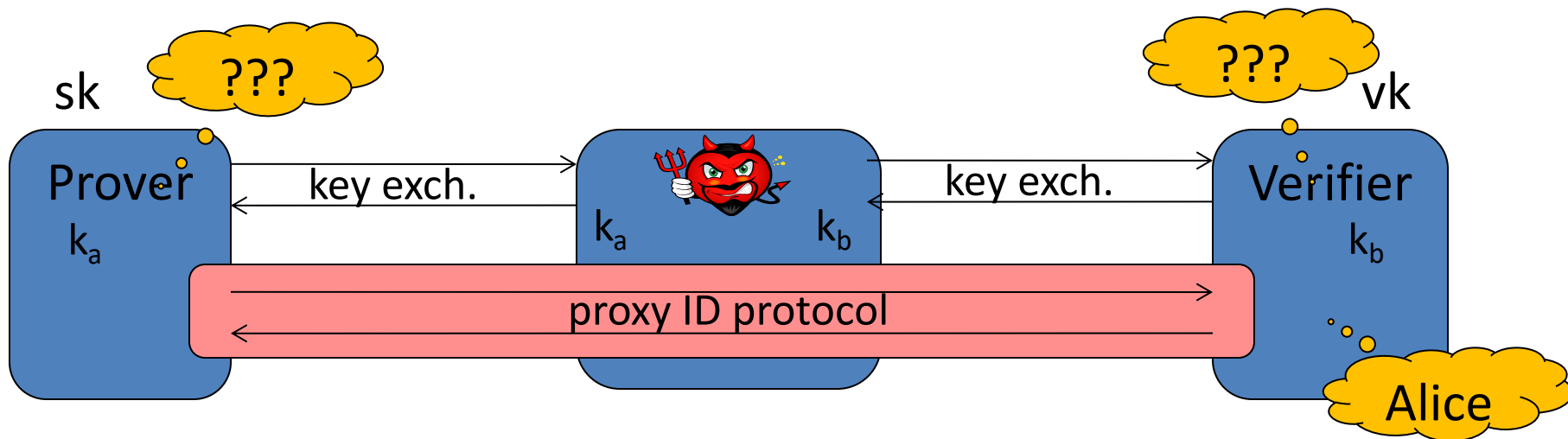
ID Protocols: how not to use

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



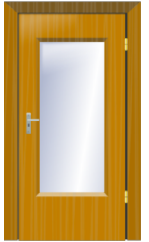
ID Protocols: how not to use

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



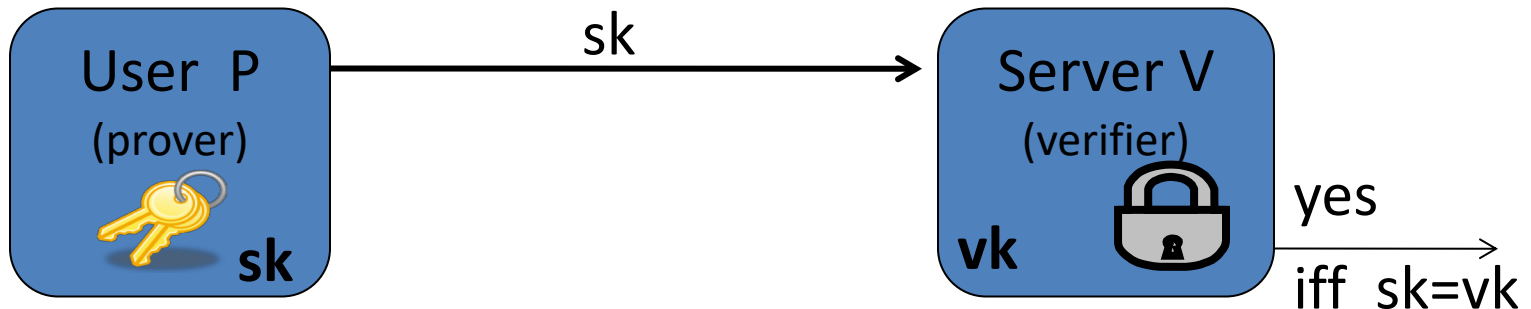


ID protocols

Direct attacks

Basic Password Protocol (incorrect version)

- **PWD**: finite set of passwords
- Algorithm G (KeyGen):
 - choose $pw \leftarrow \text{PWD}$. 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!

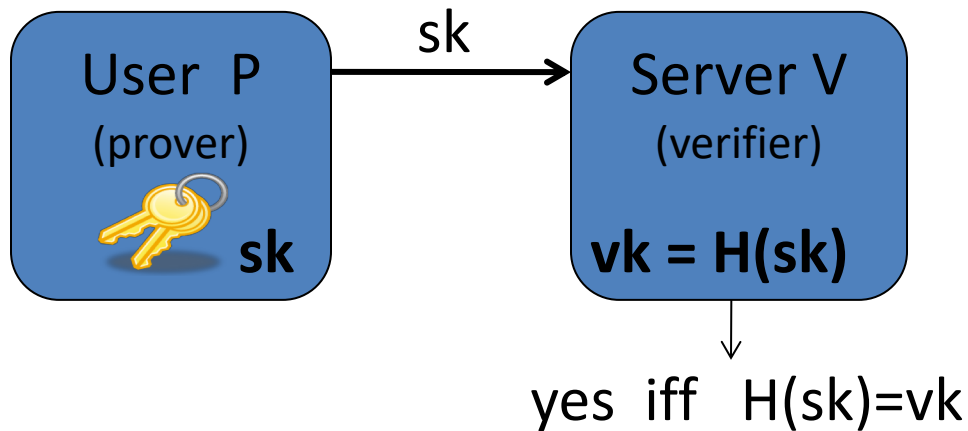
password file on server

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

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

- 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 all 7-letter passwords in a few minutes
- Scan through 360,000,000 guesses in few seconds
 \Rightarrow 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: $\text{SHA1}(\text{pwd})$)

Alice	$H(\text{pw}_A)$
Bob	$H(\text{pw}_B)$
...	...

Batch dict. attack:

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

Total time: $O(|\text{Dict}| + |F|)$ [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 $H(\text{pw}, S_A) = h_A$

id	S	h
Alice	S_A	$H(\text{pw}_A, S_A)$
Bob	S_B	$H(\text{pw}_B, S_B)$
...

Recommended salt length, $n = 64$ bits

- Attacker must re-hash dictionary for each user

Batch attack time is now: $O(|\mathbf{Dict}| \times |\mathbf{F}|)$

How to hash a password?

Linked-in: **SHA1** hashed (**unsalted**) passwords

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

How to hash?

PBKDF2, bcrypt: slow hash functions

- Slowness by “iterating” a crypto hash function like SHA256
Example: $H(\text{pw}) = \text{SHA256}(\text{SHA256}(\dots \text{SHA256}(\text{pw}, S_A) \dots))$
- 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

⇒ attacker can do dictionary attack much faster
than 1000 evals/sec.

How to hash: a better approach

Scrypt: a slow hash function AND need lots of memory to evaluate
⇒ custom hardware not much faster than commodity CPU

Problem: memory access pattern depends on input password
⇒ local attacker can learn memory access pattern
for a given password
⇒ 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

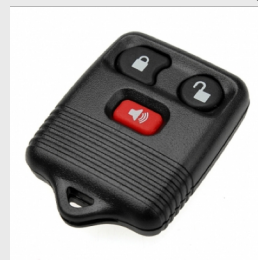
(one-time password systems)

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



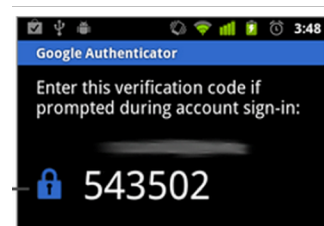
A protocol is “secure against eavesdropping” if no efficient adversary can win this game

The password protocol is clearly insecure !

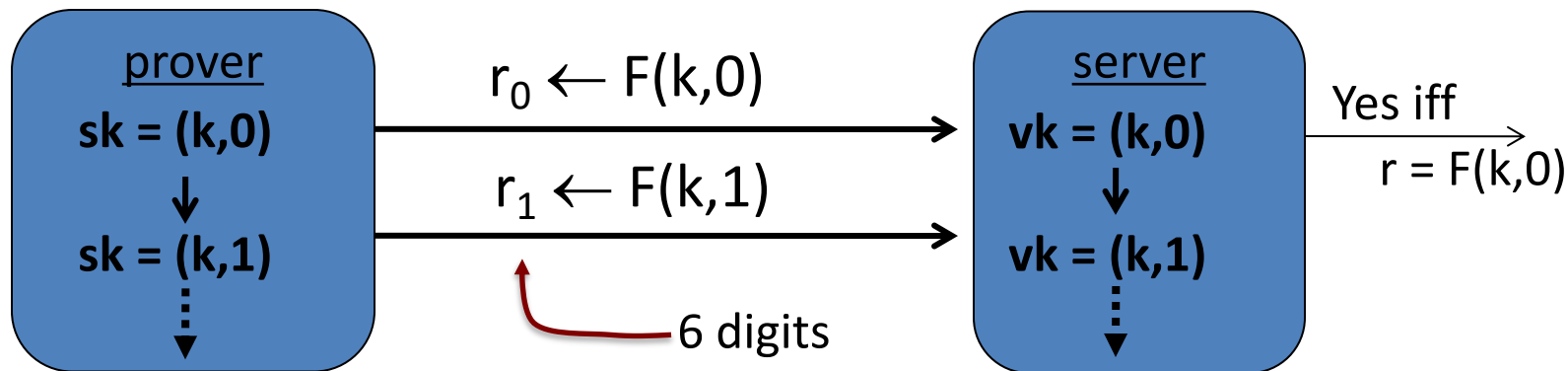
One-time passwords (secret vk, stateful)

Setup (algorithm G):

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



Identification:



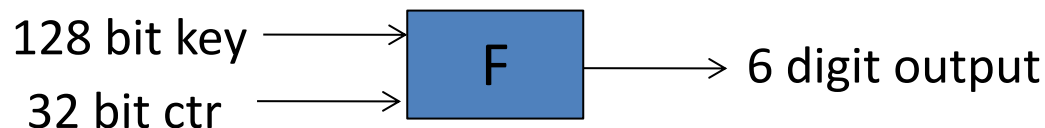
often, time-based updates: $r \leftarrow F(k, \text{time})$ [stateless]

The SecurID system

(secret vk, stateful)

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

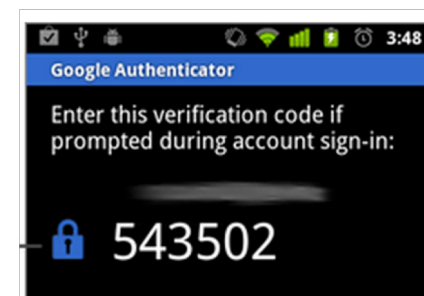
RSA SecurID uses AES-128:



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

- Time based: every 60 seconds
- User action: every button press

Both systems allow for skew in the counter value



Google authenticator

- 6-digit timed one-time passwords (TOTP) based on [RFC 6238]
- Wide web-site adoption:
 - Evernote, Dropbox, WordPress, outlook.com, ...

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)

Danger: password reset upon user logout

Server compromise exposes secrets

March 2011:

- RSA announced servers attacked, secret keys stolen
⇒ enabled SecurID user impersonation

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

The S/Key system

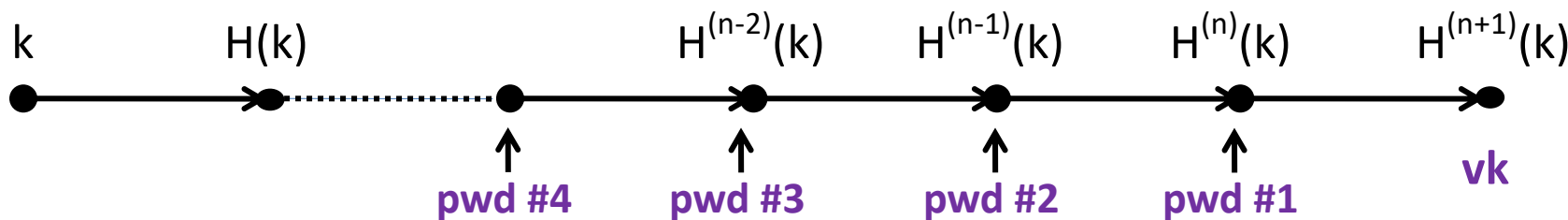
(public vk, stateful)

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

Algorithm G: (setup)

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

Identification:



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($vk=H^{(i+1)}(k)$): 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:

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

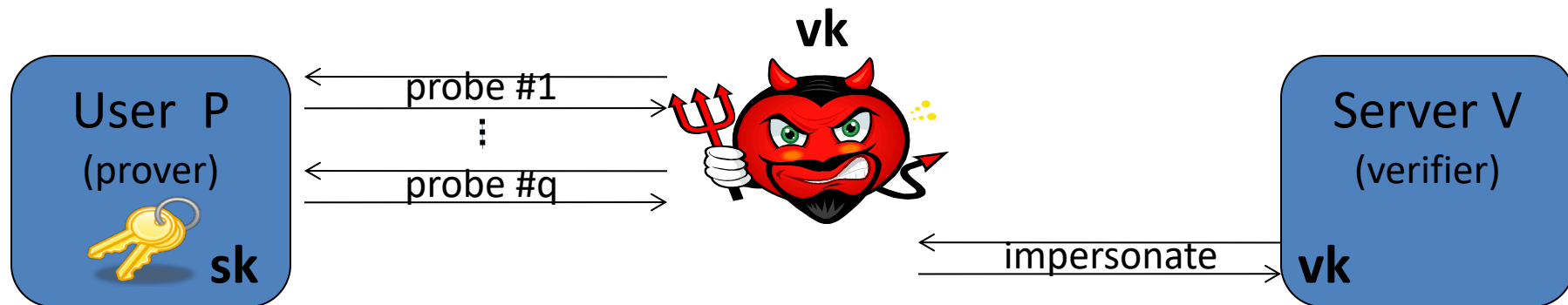


ID protocols

Security against
active attacks

(challenge-response protocols)

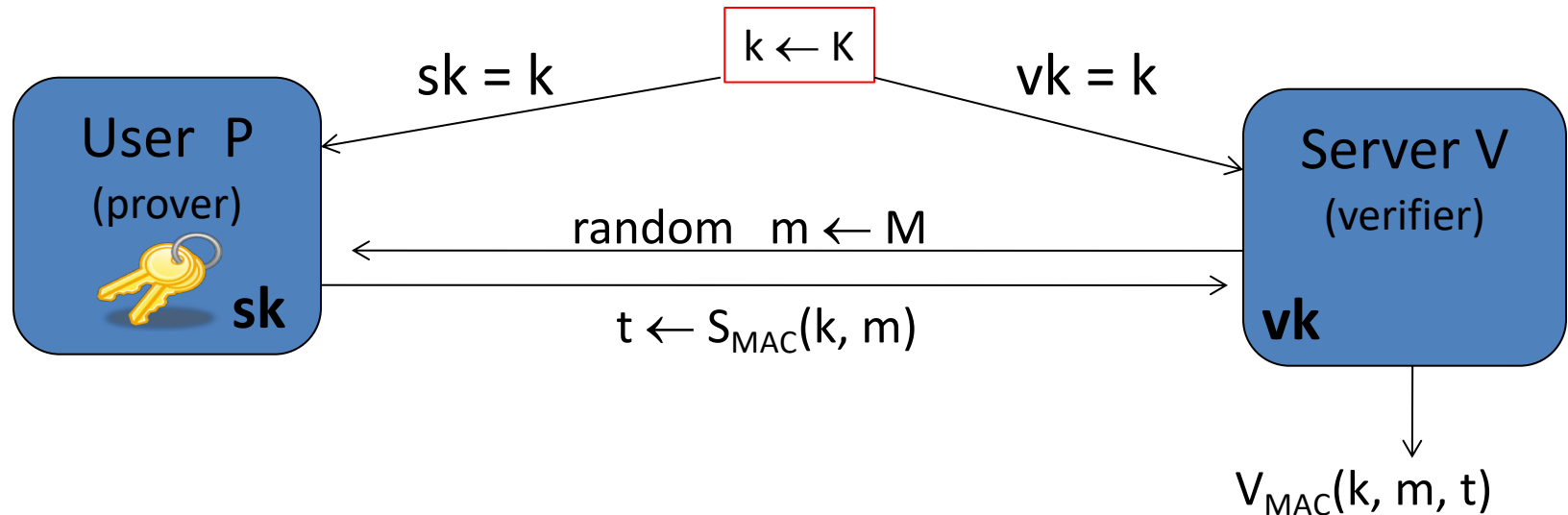
Active Attacks



- 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

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:

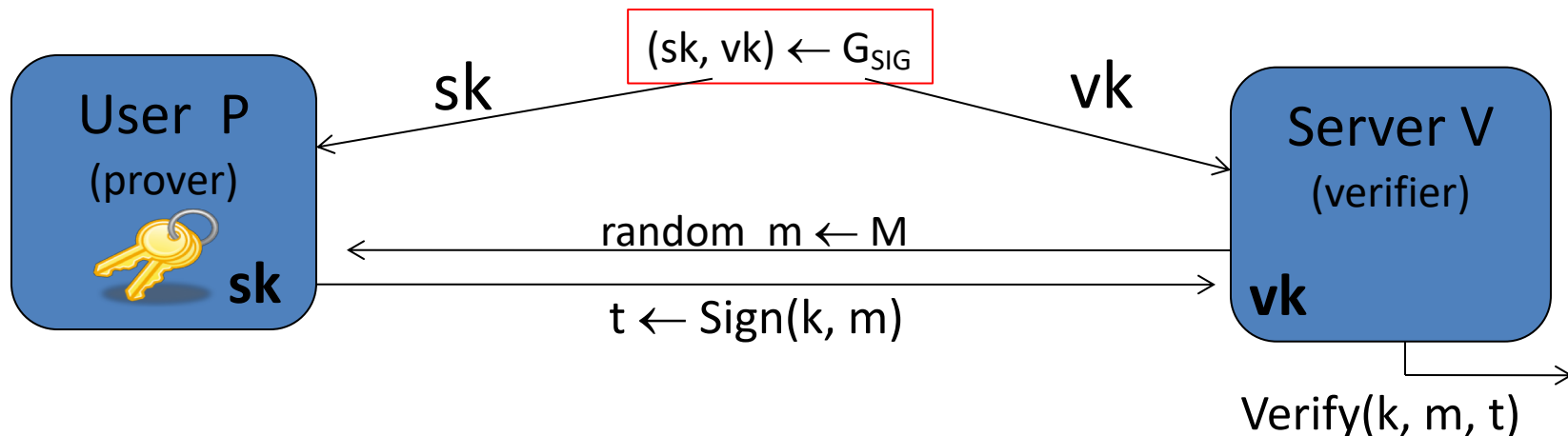
- Both m and t can be short
- CryptoCard: 8 chars each



Sig-based Challenge Response

(public vk)

Replace MAC with a digital signature:



“Thm”: Protocol is secure against active attacks (**public vk**),
provided $(G_{SIG}, Sign, Verify)$ is a secure digital sig.

but t is long (≥ 20 bytes)

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