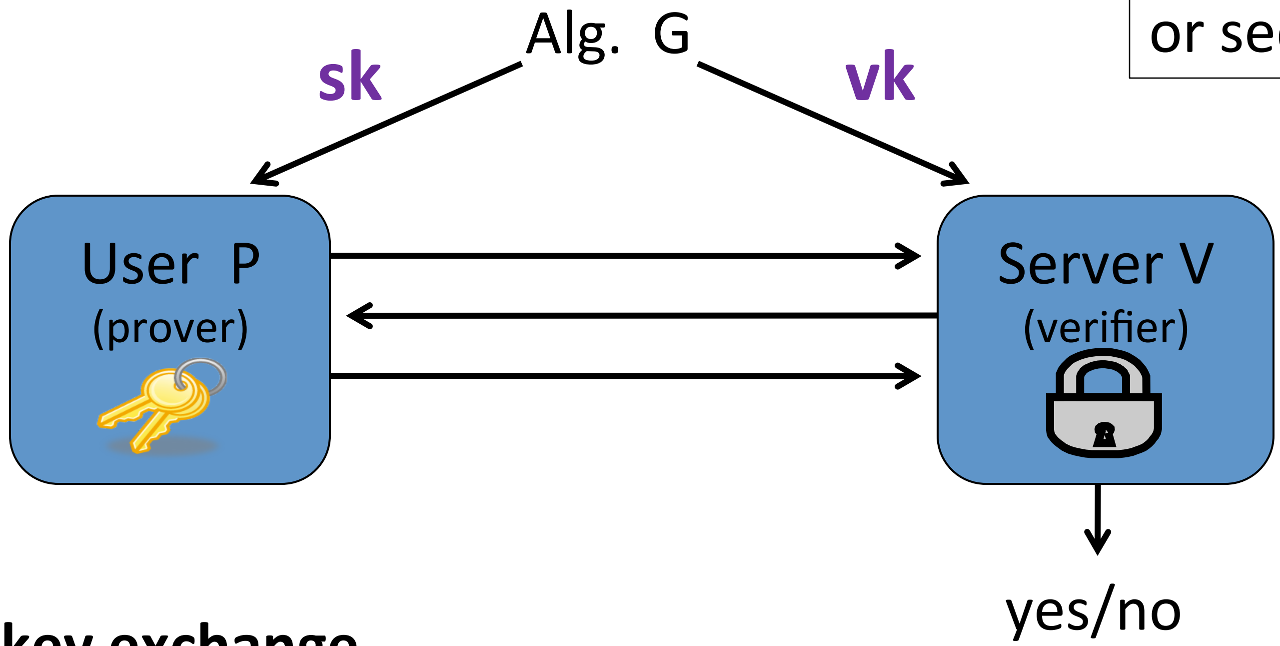
CS255: Dan Boneh

Identification Protocols

Authenticating users

The Setup

vk either public or secret



no key exchange

Applications

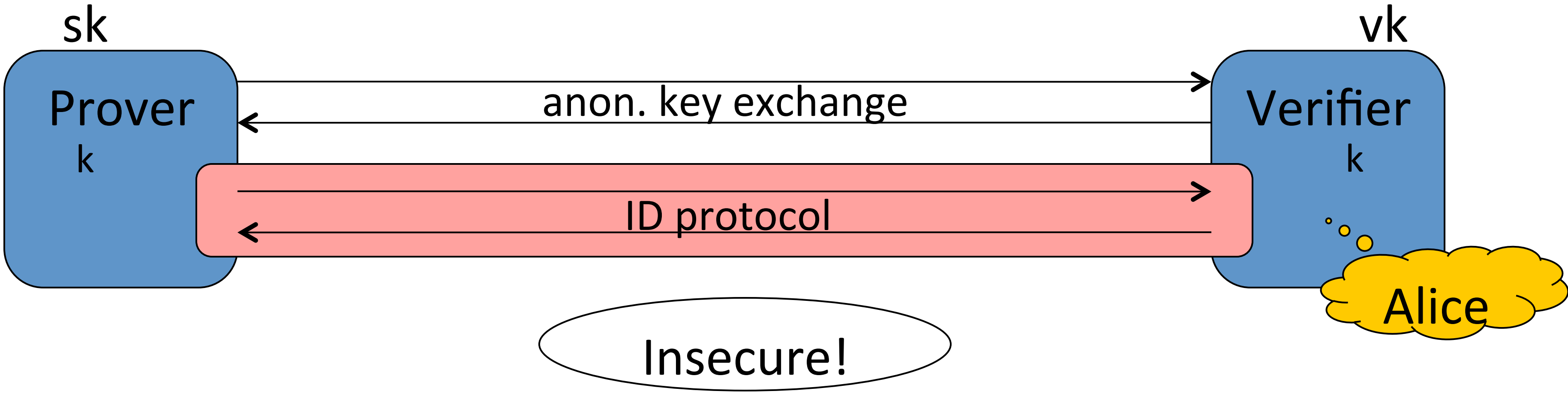
– Physical locks: (friend-or-foe)
  - Wireless car entry system (e.g. KeeLoq)
  - Opening an office door or a garage door

– Login at a bank ATM or a desktop computer

– Login to a remote web site once key-exchange with one-sided authentication completes (e.g. SSL)

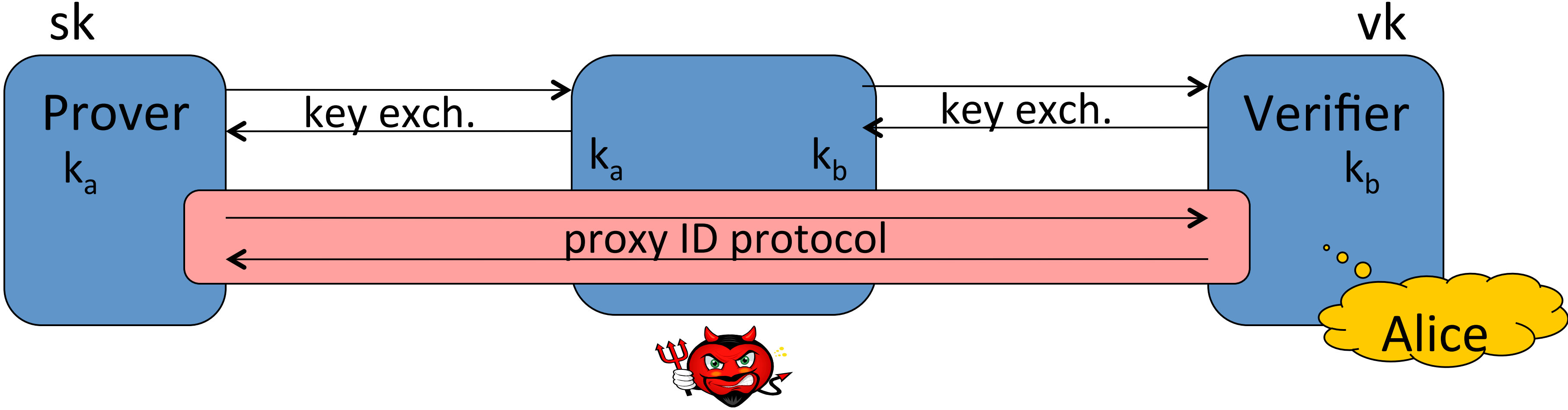
ID Protocols: how not to use

- 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

Identification Protocols

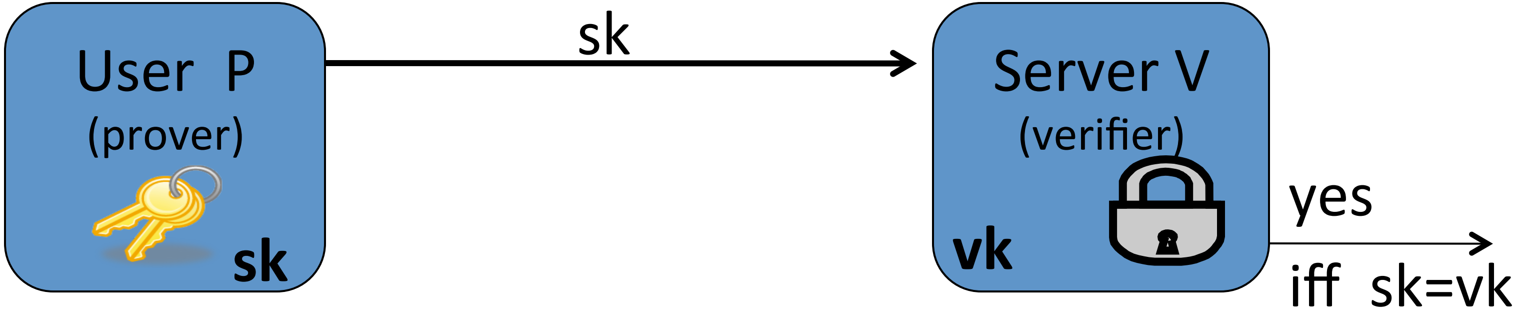
Security against direct attacks

(password systems)

Basic Password Protocol (incorrect version)

- **PWD**: finite set of passwords

- Algorithm G (setup):
  - choose pw ← 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}
…	…

A (small) sample of server-side password breaches

2012: **Linked-in**: 6 million passwords (hashed, **unsalted**)

2013:

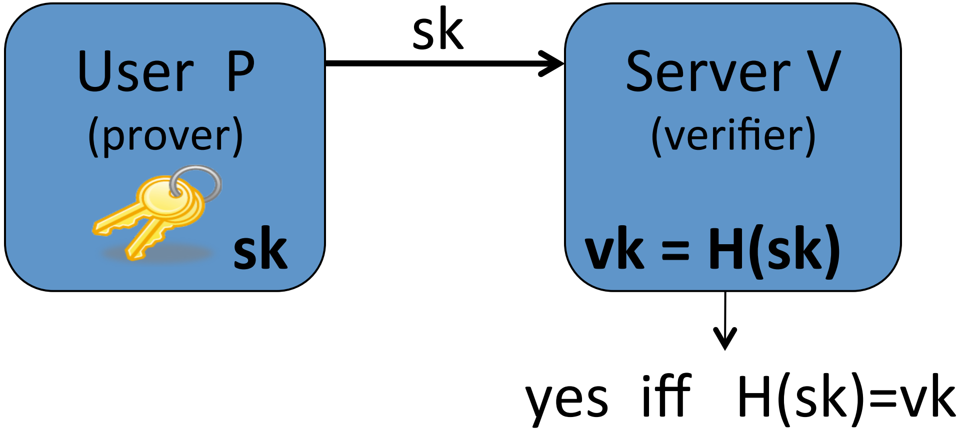
- **Twitter**: 250,000 passwords (hashed, salted)
- **LivingSocial**: 50 million records
  Names, emails, DOB, passwords (hashed, salted)
- **Evernote**: 50 million records
  usernames, emails, hashed passwords
- **Adobe**: 38 million records (http://xkcd.com/1286/)
  email addrs., **password hints**, and **encrypted** passwords

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)"

example: SHA-256, PBKDF2



password file on server

Alice	$H(pw_A)$
Bob	$H(pw_B)$
…	…

Weak password choice

Users frequently choose weak passwords: (adobe list, 2013)

Password:	123456	123456789	password	adobe123	12345678	qwerty	1234567
Fraction of users:	5%	1.1%	0.9%	0.5%	0.5%	0.5%	0.3%

Total: 8.8%

A common occurrence

- Example: the Rockyou password list, 2009 (6 most common pwds)

  123456, 12345, Password, iloveyou, princess, abc123

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

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 ∈ 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 a dictionary attack on each password !

Preventing Batch Dictionary Attacks

Public salt:

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

password database

id	salt	hash (PBKDF2)
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|**)

Further Slowing Down Dictionary Attacks

Slow hash function H: (say 0.1 sec. to hash pw)

- Example: H(pw) = SHA1(SHA1(… SHA1(pw, S_A) …))
- Unnoticeable to user, but makes offline dictionary attack harder
- Use PBKDF2: tunable # iterations

Alice	$\mathbf{S_A}$	H(pw_A , S_A , $\mathbf{r_A}$)
Bob	$\mathbf{S_B}$	H(pw_B , S_B , $\mathbf{r_B}$)
…	…	…

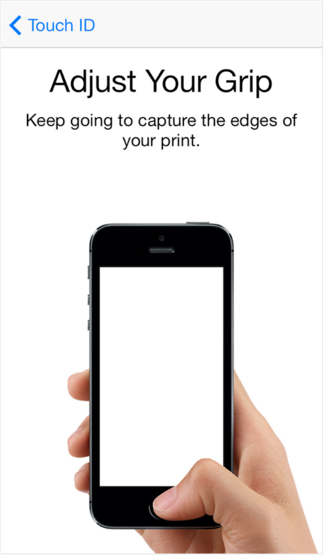
<u>**Secret salts**</u>:

- When setting pwd choose short random r (8 bits)
- When verifying pw for A, try all values of r_A. 128 times slow down on average.
- 256 times slow down for attacker

Strengthening User Authentication

One option: biometrics:

- Fingerprints, retina, facial recognition, …
- Benefit: hard to forget



note: CCC'13

<u>Problems</u>:

- Biometrics are not generally secret
- Cannot be changed, unlike passwords

$\Rightarrow$ Should primarily be used as a second factor authentication

The Common Password Problem

Users tend to use the same password at many sites

- Password at a high security site can be exposed by a break-in at a low security site

Standard defense: (PwdHash)

- Client side software that converts a common password pw into a unique site password

$$pw' \leftarrow H(pw, \text{user-id}, \text{server-id})$$

pw' is sent to server

Identification 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

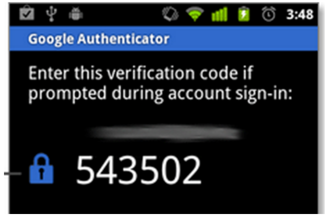
A protocol is "secure against eavesdropping" if no efficient adversary can win this game

The password protocol is clearly insecure !

2^{nd} factor OTP authentication (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]

Google authenticator

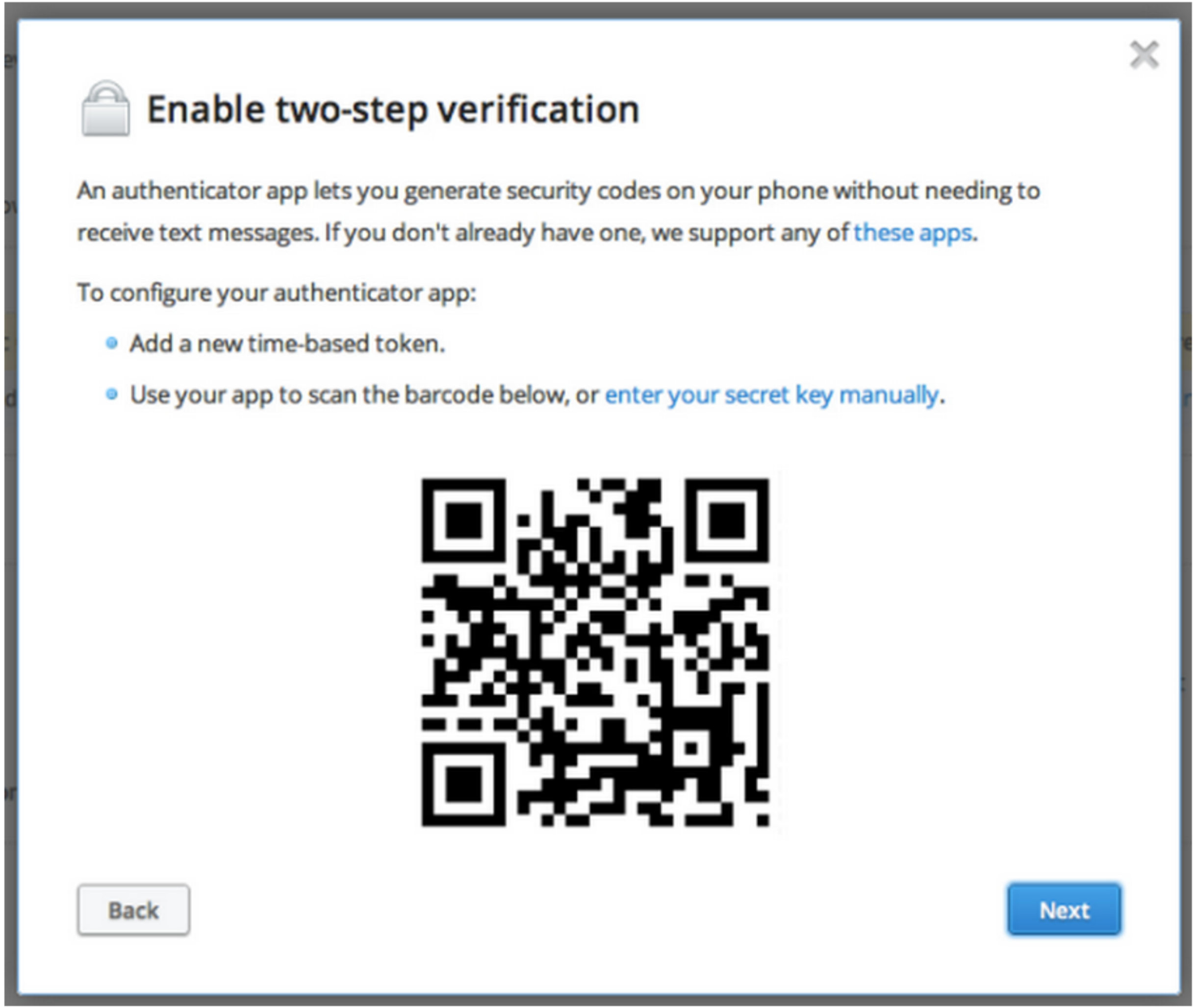
- 6-digit timed one-time passwords (TOTP) based on [RFC 6238]
- Wide web-site adoption:
  - Evernote, Dropbox, WordPress, outlook.com, …

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 lockout


Enable two-step verification
An authenticator app lets you generate security codes on your phone without needing to receive text messages. If you don't already have one, we support any of these apps.
To configure your authenticator app:
Add a new time-based token.
Use your app to scan the barcode below, or enter your secret key manually.
Back
Next

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 not-secret?

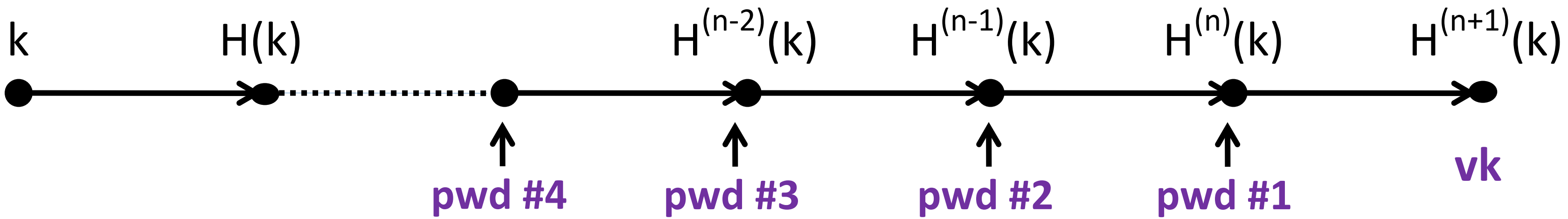
The S/Key system (public vk, stateful)

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

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

TOTP vs. S/Key

S/Key:

- **public** vk,   **limited** number of authentications
- Long one-time passwords (128 bits)

TOTP:

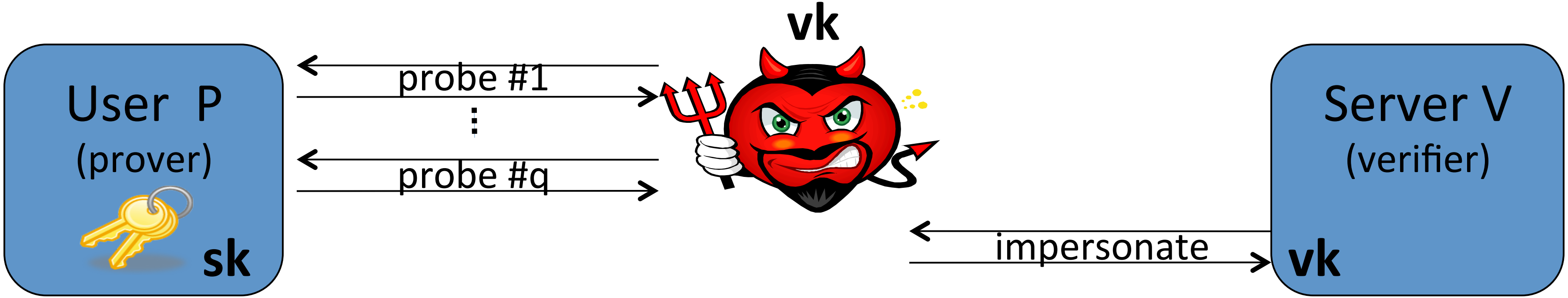
- **secret** vk,   **unlimited** number of authentications
- Short one-time passwords (6 digits, i.e. 20 bits)

Identification Protocols

Security against active attacks

(challenge-response protocols)

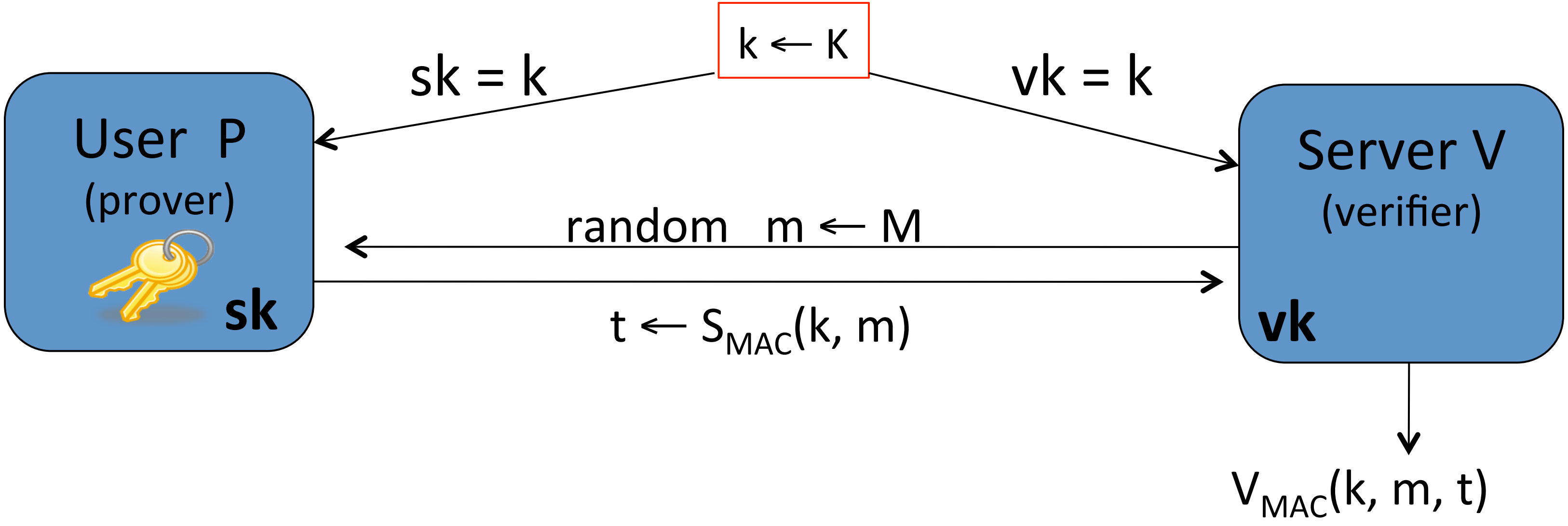
Active Attacks



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

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:

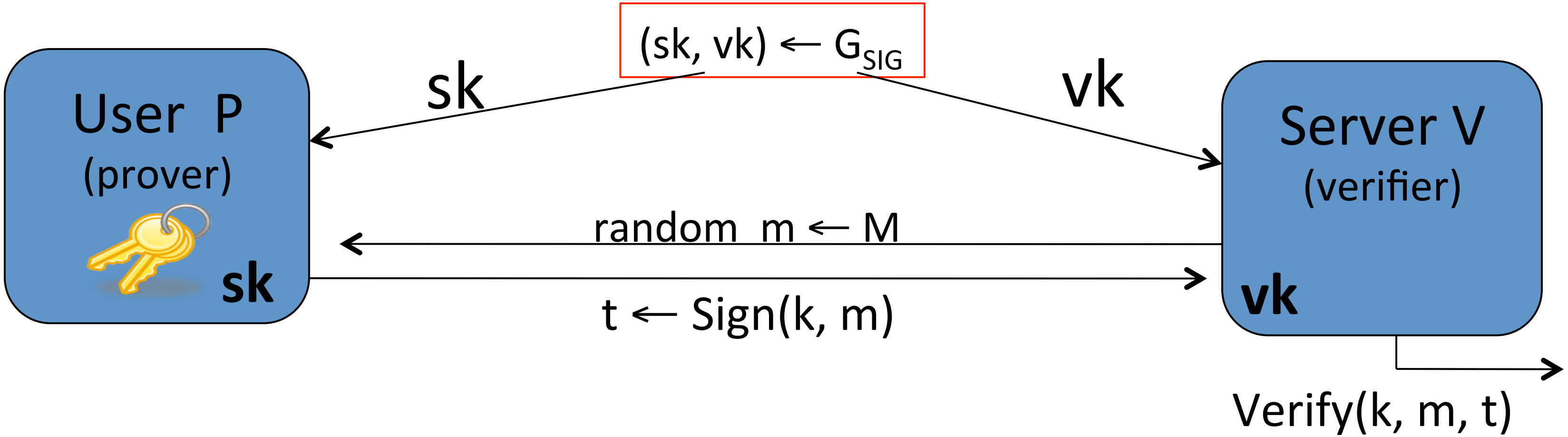
  Given [m , S_{MAC}(pw, m)] eavesdropper can try all pw $\in$ Dict to recover pw

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