PASS: Strengthening and Democratizing Enterprise Password Hardening

Ari Juels
Jacobs Technion-Cornell Institute
Cornell Tech

with D. Akhawe (Dropbox). A. Athalye (MIT), R. Chatterjee (Cornell), A. Everspaugh (UWisc), T. Ristenpart (Cornell Tech), S. Scott (Royal Holloway)

Real World Cryptography, Stanford, 7 January 2016

CORNELL
TECH

Password breaches never go out of style

Adobe

130 million (ECB-encrypted) passwords
Oct. 2013

36 **million** passwords
August 2015

livingsocial

50 **million** passwords
April 2014

YAHOO!

273 million passwords
Jan. 2014

ebay

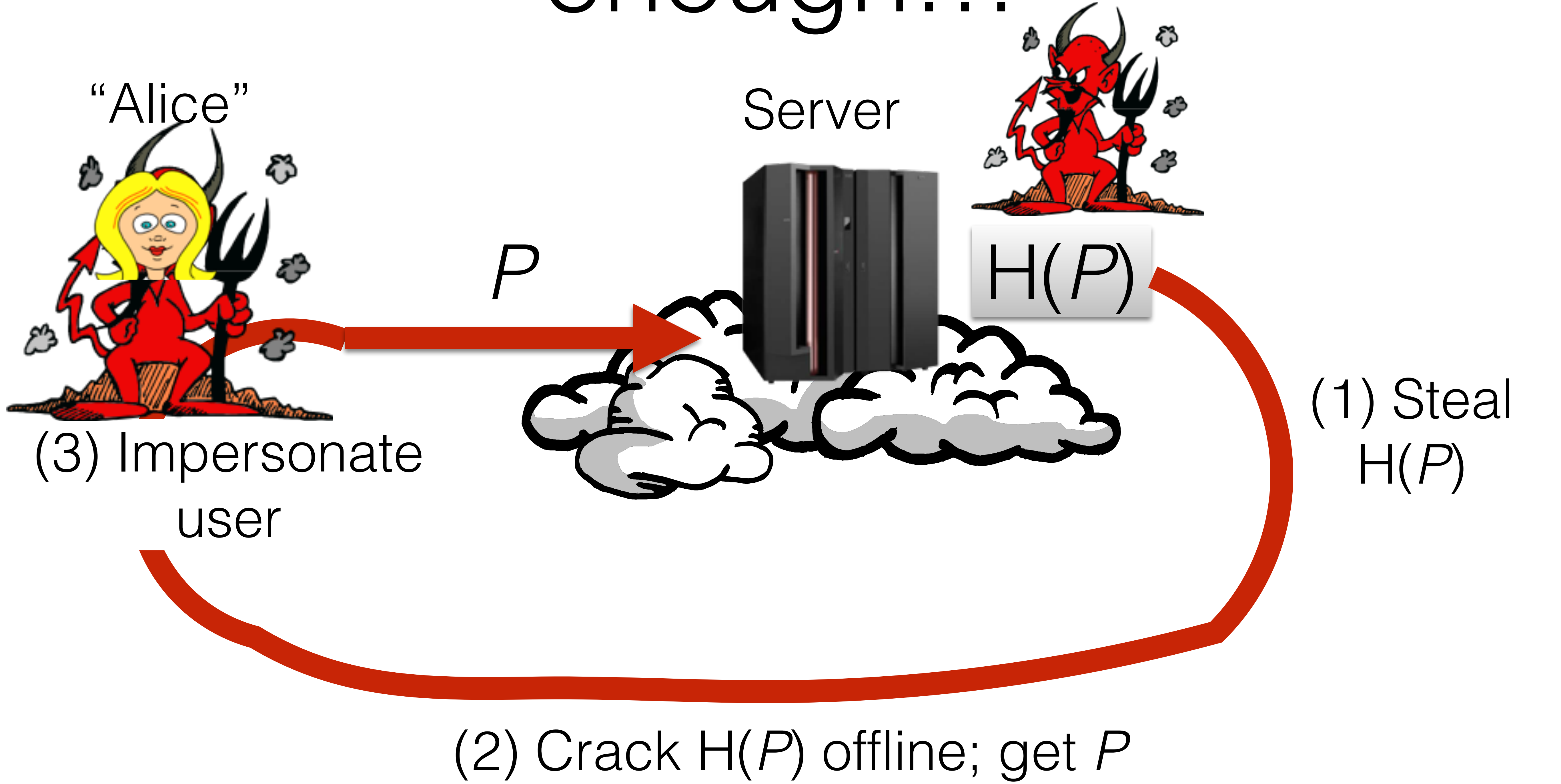
145 **million** passwords
May 2014

EVERNOTE

50 million passwords
March 2013

Plus last.fm, Twitter, eHarmony, etc., etc., etc.

Hashing often isn't enough…



Ashley Madison breach

- AM used salted bcrypt
  - Cost parameter 12
  - Very strong relative to common industry practice
  - Not strong enough to compensate for *weak* passwords
- Result of cracking sample of 4000 passwords…
- And for good measure AM left around a bunch of MD5 password hashes…

123456 202
password 105
12345 99
qwerty 32
12345678 31
ashley 28
baseball 27
abc123 27
696969 23
111111 21
football 20
fuckyou 20
madison 20
asshole 19

Source: http://www.pxdojo.net/2015/08/what-i-learned-from-cracking-4000.html

Even sophisticated organizations struggle to protect themselves

Can we:
(1) Create password-protection system better than industry norm and
(2) Can we democratize it?

PASS

Even sophisticated organizations struggle to protect themselves

Two major features of PASS:

(1) Password hardening protects against smash-and-grab password breaches

(2) Typo correctors safely correct (some) password typos

PASS

Password Hardening in PASS

The Facebook Password Onion

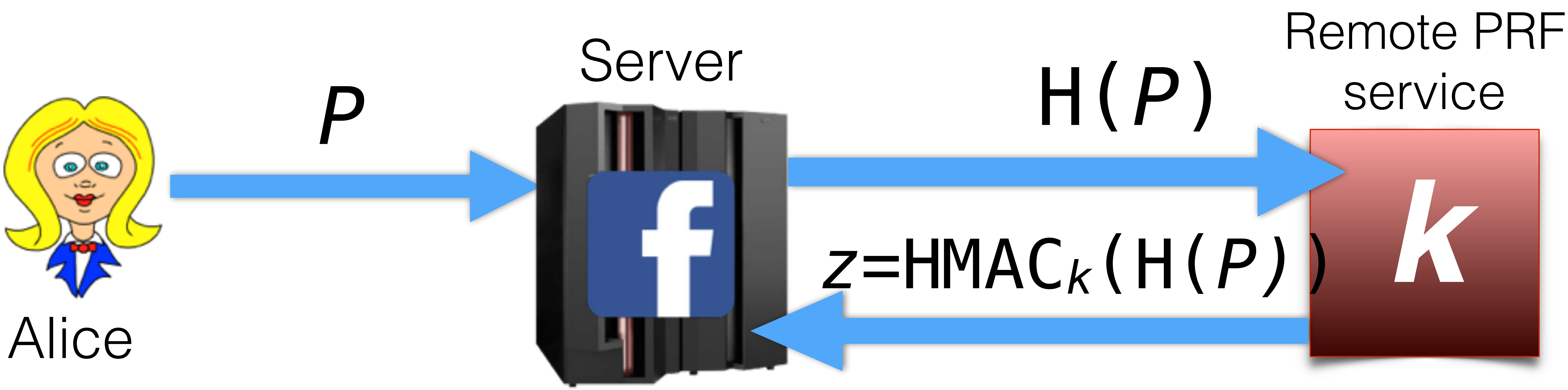
```
$cur  = ‘password’
$cur  = md5($cur)
$salt = randbytes(20)
$cur  = hmac_sha1($cur, $salt)
$cur  = remote_hmac_sha256($cur, $secret)
$cur  = scrypt($cur, $salt)
$cur  = hmac_sha256($cur, $salt)
```

From last year's RWC…

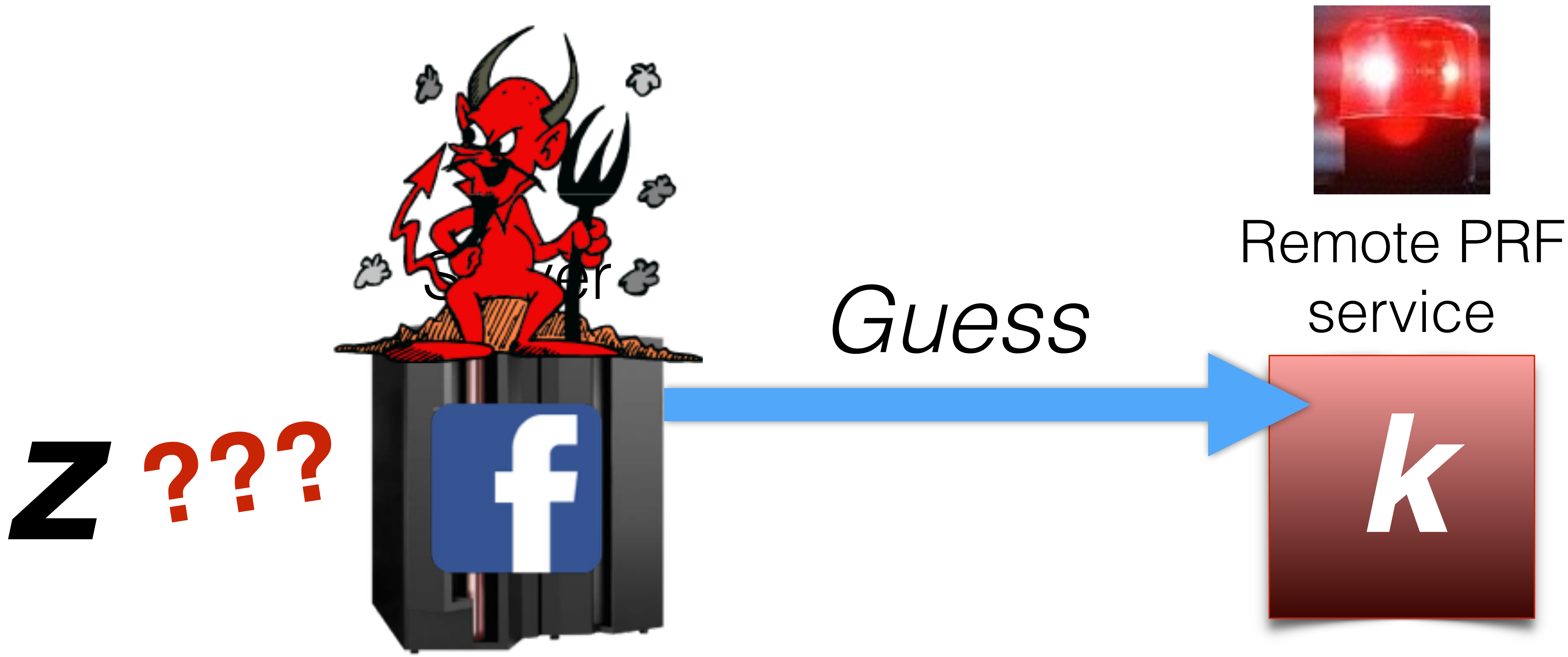
The Facebook Password Onion

```
$cur  = 'password'
$cur  = md5($cur)
$salt = randbytes(20)
$cur  = hmac_sha1($cur, $salt)
$cur  = remote_hmac_sha256($cur, $secret)
$cur  = scrypt($cur, $salt)
$cur  = hmac_sha256($cur, $salt)
```

Facebook approach



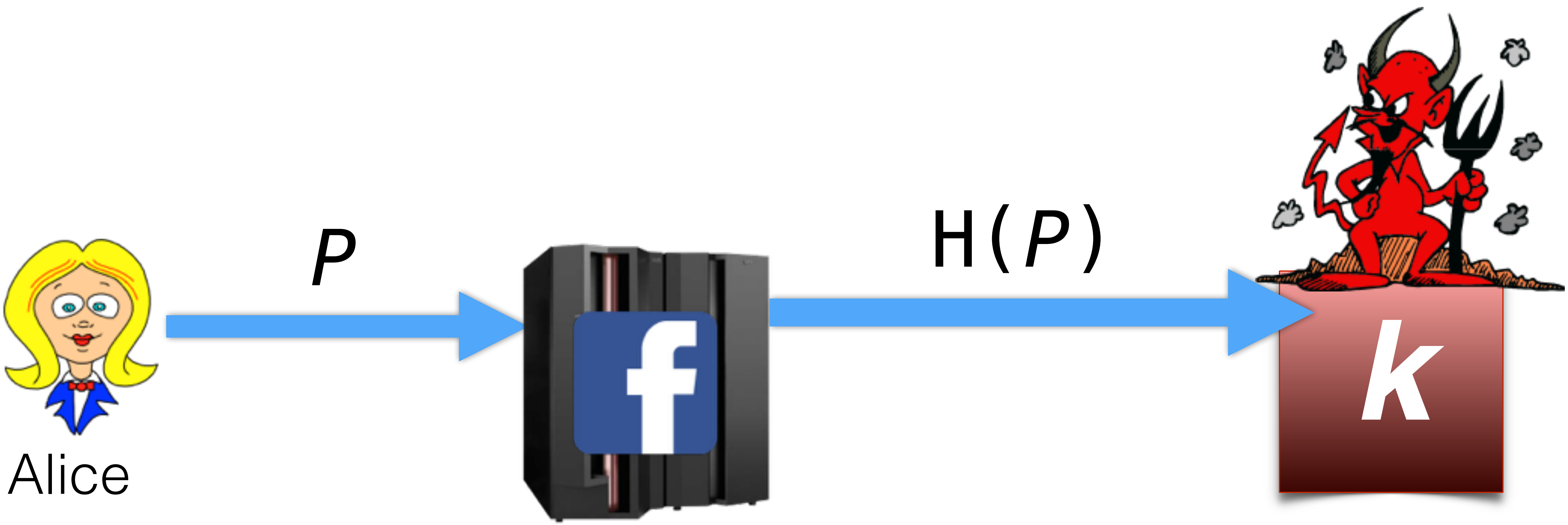
Facebook's remote hardening service



Turns *offline* attack into *online* attack

Facebook approach

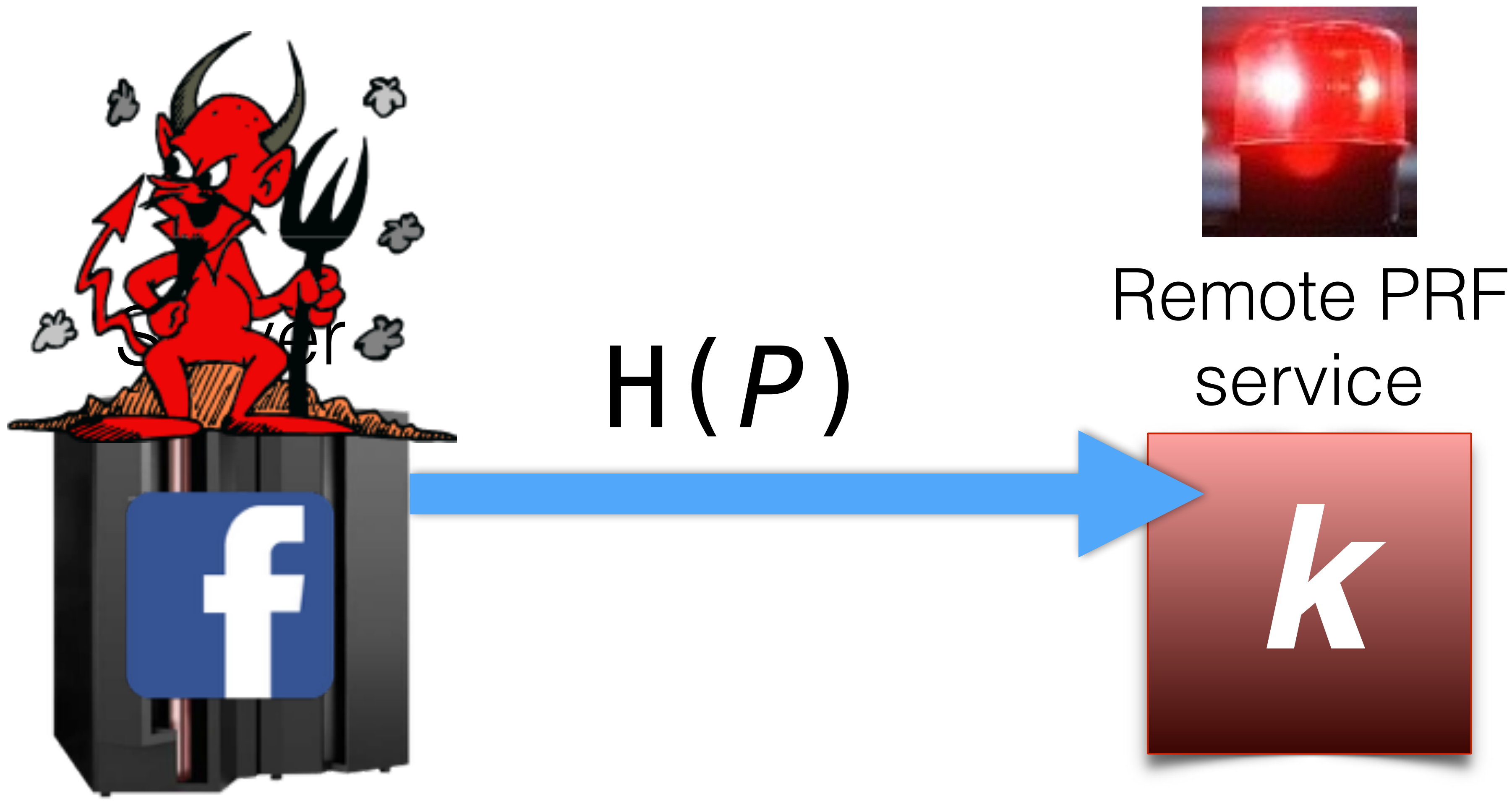
Drawback 1



(Hashed / HMACed) password exposed to PRF service!

Facebook approach

Drawback 2?



(Perhaps) not operating / alerting with per-user granularity

Facebook approach

Drawback 3

$+$ k'

z_1 = HMAC$_k$(H(P))

z_2 = HMAC$_k$(H(P))

z_3 = HMAC$_k$(H(P))

...

No support for periodic key rotation

The Facebook Password Onion

```
$cur  = ‘password’
$cur  = md5($cur)
$salt = randbytes(20)
$cur  = hmac_sha1($cur, $salt)
$cur  = remote_hmac_sha256($cur, $secret)
$cur  = scrypt($cur, $salt)
$cur  = hmac_sha256($cur, $salt)
$cur  = remote2_hmac_sha256($cur, $secret2)
$cur  = remote3_hmac_sha256($cur, $secret3)
…
$cur  = remotei_hmac_sha256($cur, $secreti)
```

PASS: PRF Service

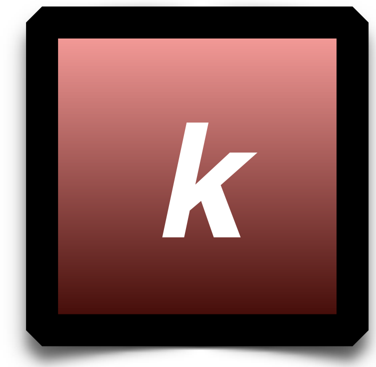
Hardens passwords à la Facebook, but also has:

1. *Blinding*: Conceals passwords from PRF service
2. *Graceful key rotation*: No code change (or service interruption)
3. *Fine-grained alerting*: Per-user monitoring / rate-limiting of PRF service requests

PASS: User registration

Password service

PRF service



user, P

t := random()

x := blind(P)

Blinded PW

(t, x)

User ID
for alerting /
throttling

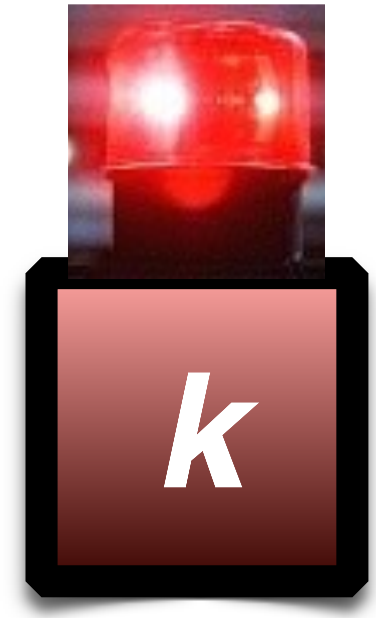
$y := \mathbf{F}_k(t, x)$

y

z := unblind(y)

store: (user, t, z)

PASS: Fine-grained monitoring



user,P

x:=blind(P)

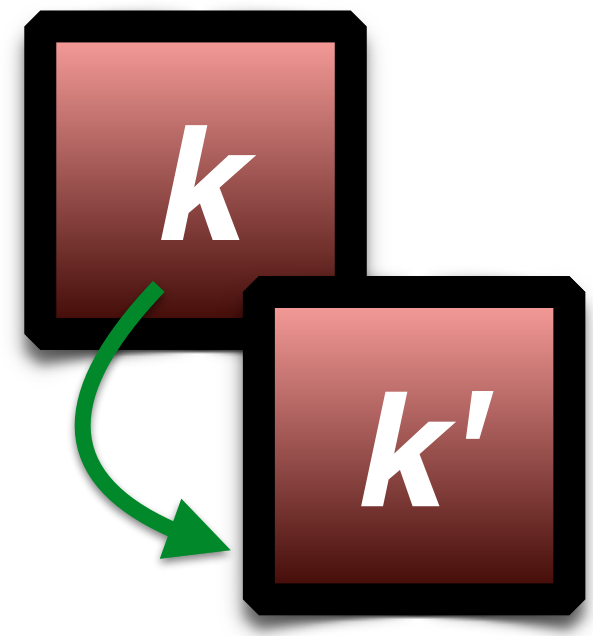
(t,x)

y := $\mathbf{F}_k(t,x)$

User identifier
t in clear

PASS: Key rotation

$\Delta_{k \to k'}$

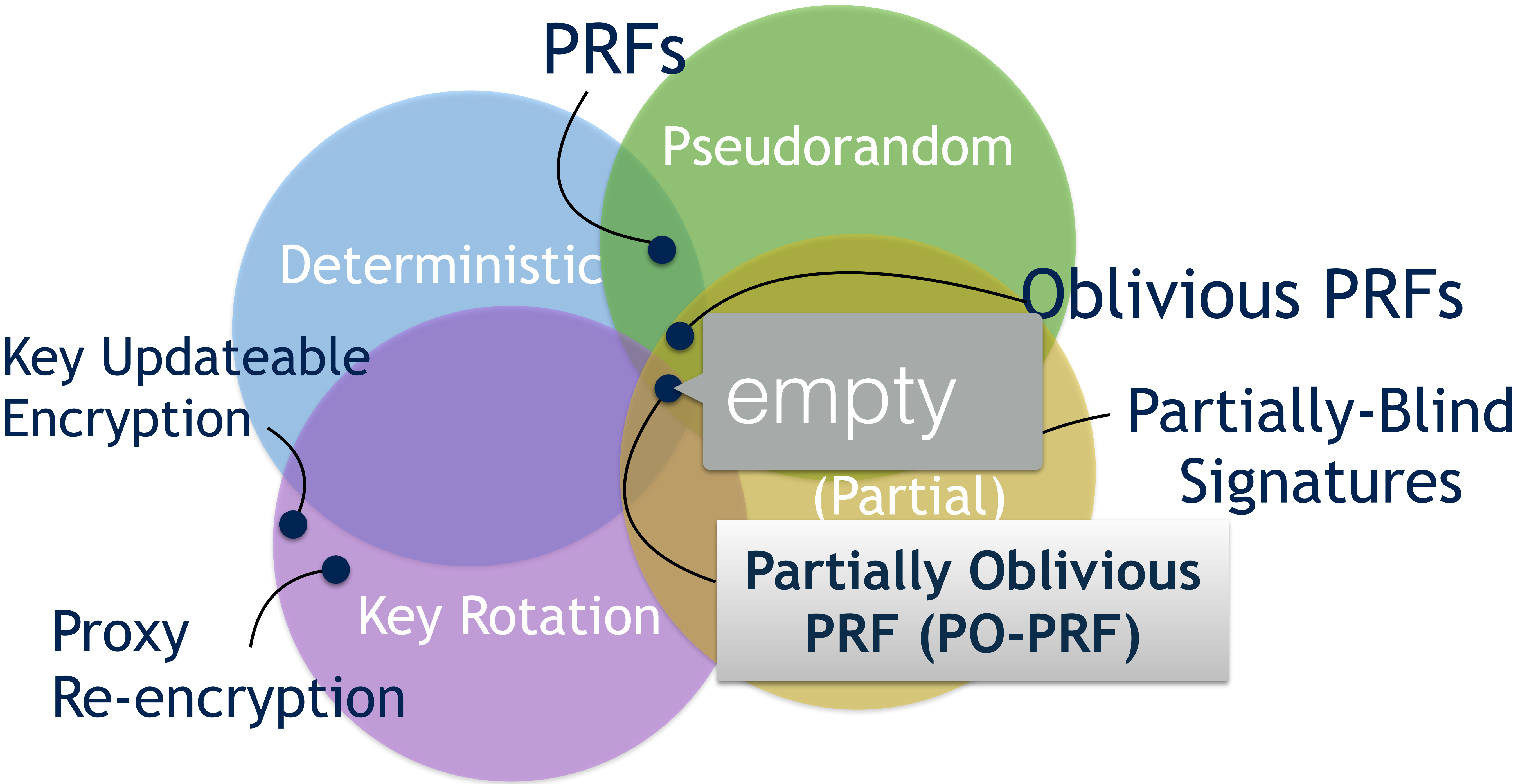


$z' \Leftarrow z$

(for all users)

update()

Existing crypto primitives insufficient

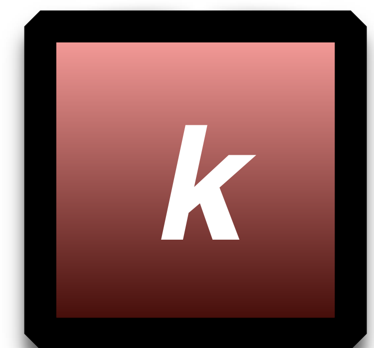


PO-PRF Construction

Bilinear Pairing

$e: G_1 \times G_2 \rightarrow G_T$

$e(a^x, b^y) = e(a,b)^{xy}$



t, x

$x := H(P)^r$

blind()

$F_k(t,x)$

y

$y := e(H(t),x)^k$

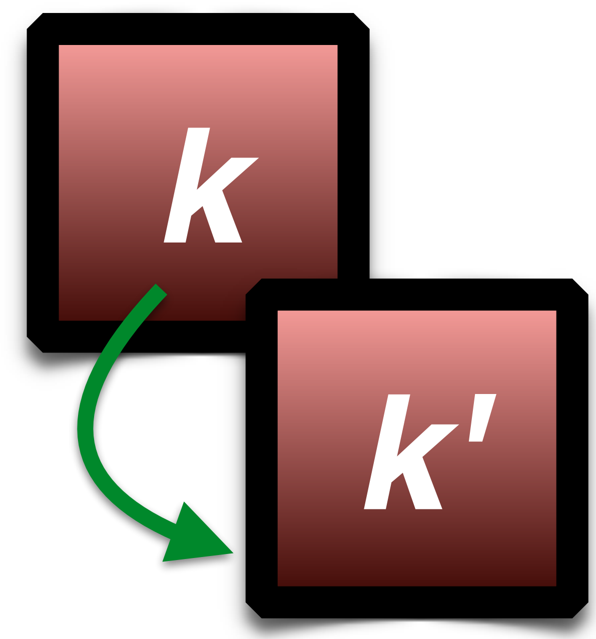
$z := y^{1/r} = e(H(t), H(P))^{k*r*1/r} = \boxed{e(H(t),H(P))^k}$

unblind()

Similar use of pairings: [Sakai, Ohgishi, Kasahara] [Boneh,Waters]

PASS: Key rotation

$\Delta_{k\to k'} = k'/k$



$$z' := z^{k'/k} = e(H(t),H(P))^{k*k'/k} = \boxed{e(H(t),H(P))^{k'}}$$

update()

PASS PRF service is easy to deploy

```
def verify(username, pass):
  (salt,check) = authTableLookup(username)
  digest = hashpass(salt, pass)
  return digest == check
  ppass = PASS.query(server, t, pass)
  digest = PASS.combine(ppass, digest)
```

Small change to code base
No impact on user experience

…and highly scalable

PRF Latency: 11.8ms (LAN) 96ms (WAN)

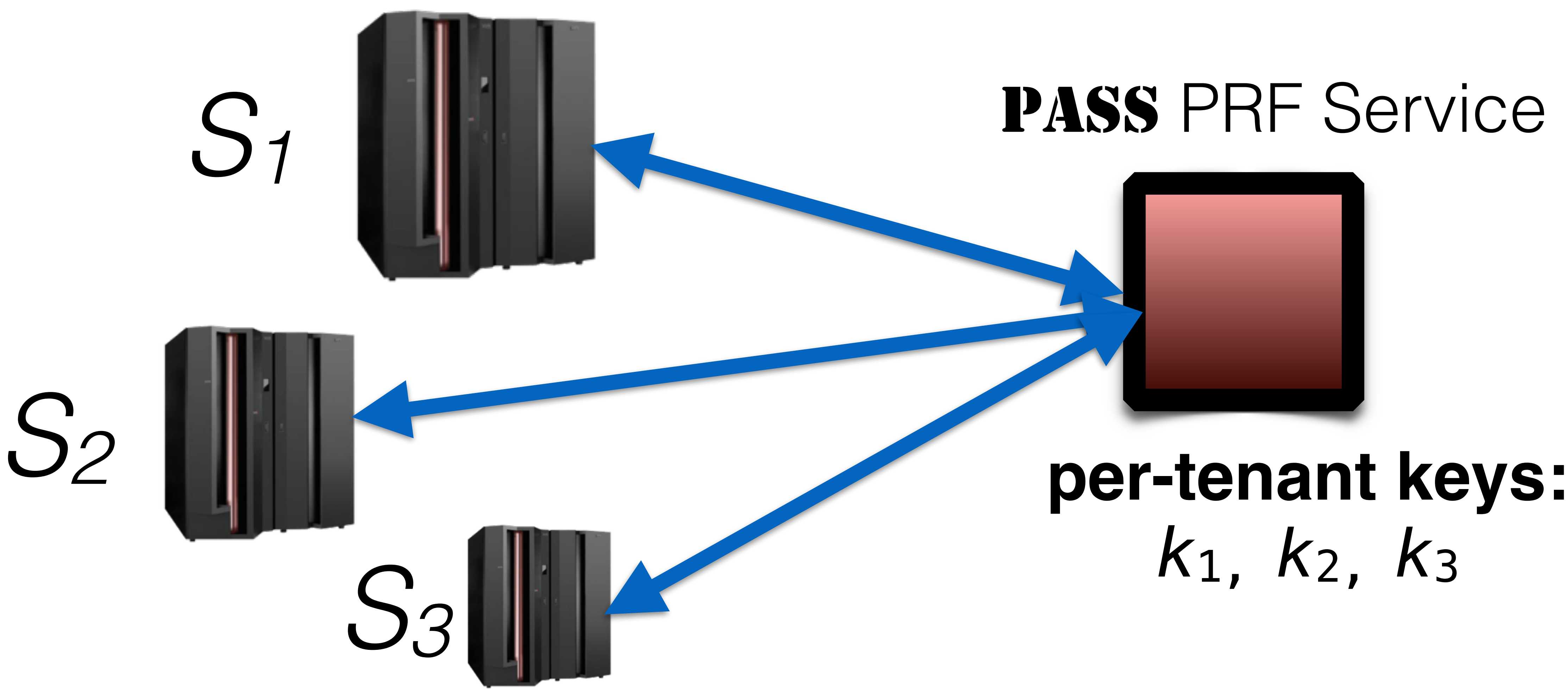
Throughput: 1350 connections/sec (8-core EC2 instance)

Within factor of 2 of TLS query for static page

PRF-Service Storage: One key!
(plus temporary rate-limiting state)

Multi-tenant service

Obliviousness means possibility of supporting multiple tenants / servers



…and good for many other password applications

File Encryption

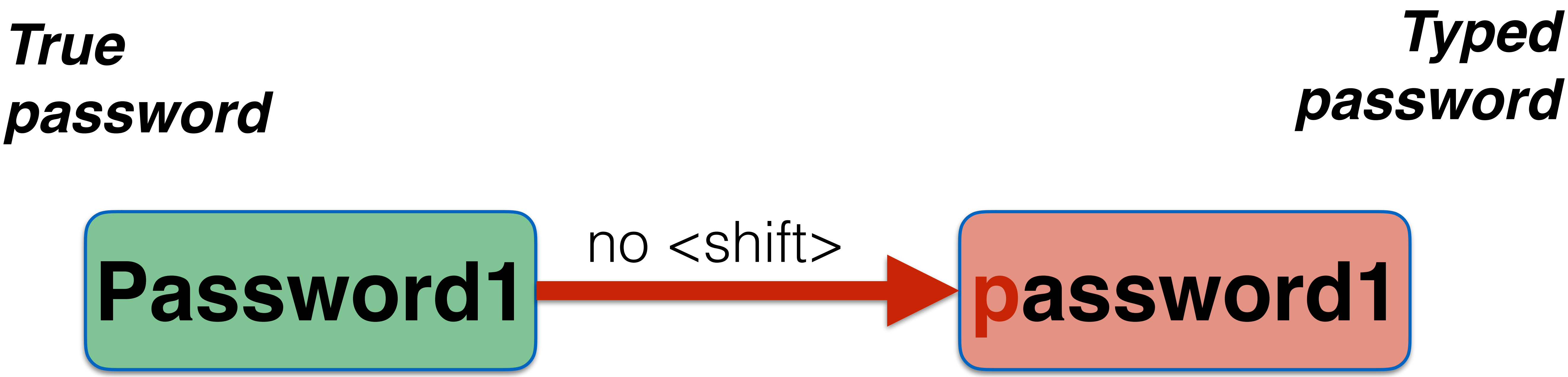
Bitcoin Brainwallet

Password managers

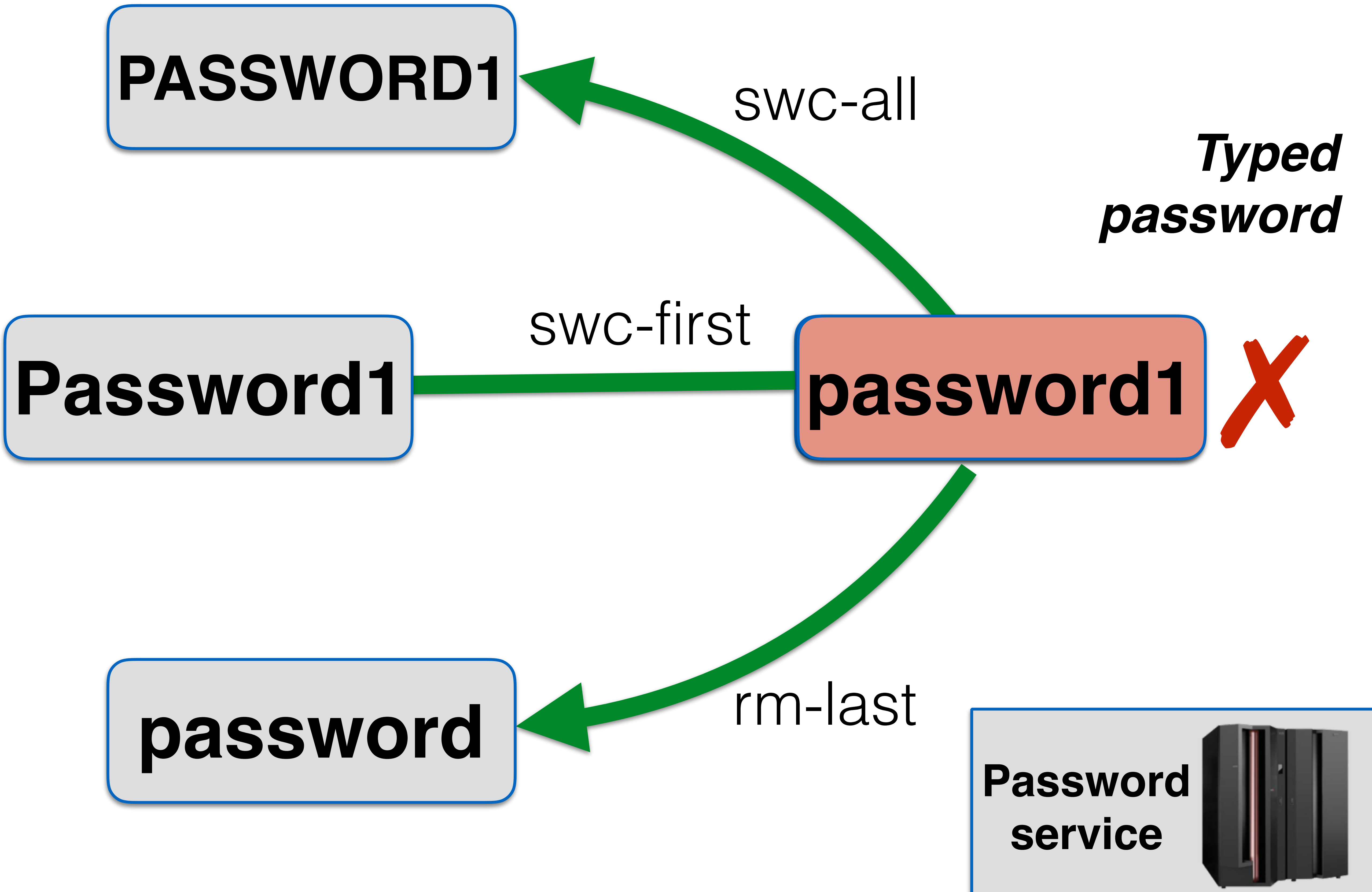
Message-locked encryption

Password Typo Correction in PASS

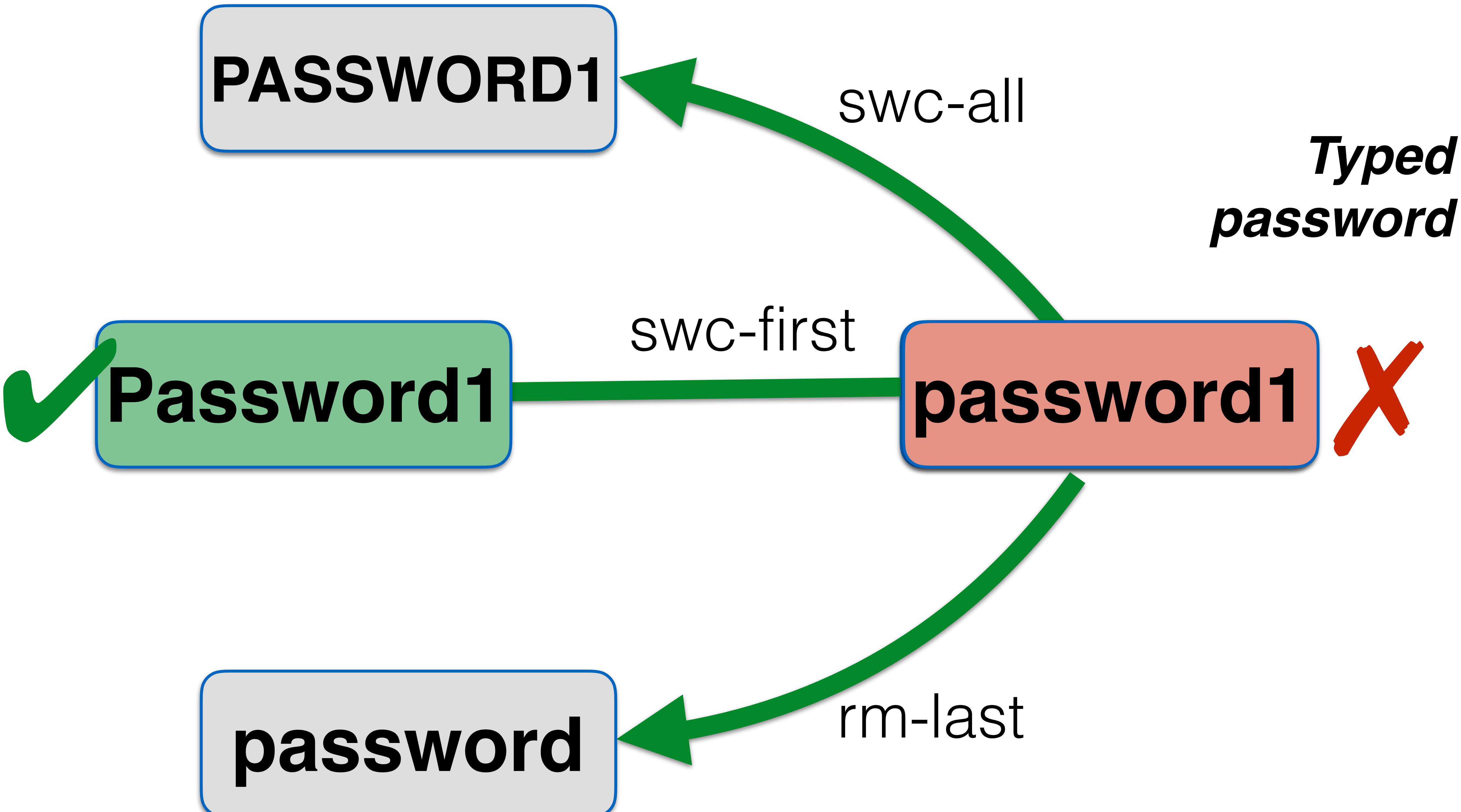
Password Typos



Why not try correctors?



Why not try correctors?



Password typo correctors: Industry practice

- Facebook, Vanguard, etc., doing some form of this
  - E.g., correcting CAPS LOCK
- Hue and cry

ZDNet

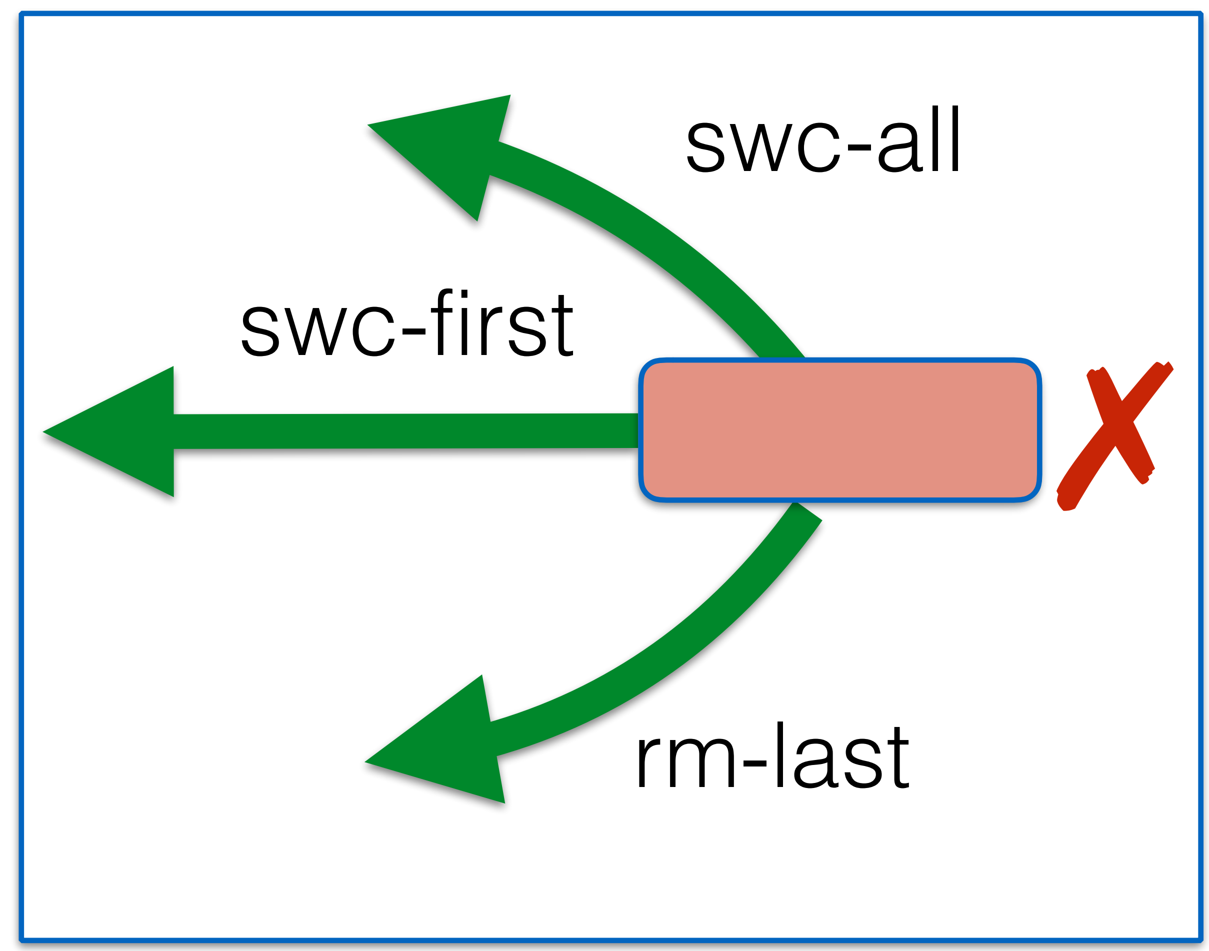
Facebook passwords are not case sensitive

If you have characters in your Facebook password, there's a second password that you can log in to the social network with.

- *c* correctors turns adversary's 1 password guess into (c+1) guesses
- ~~Increases attacker's guessing success by factor of c+1!~~ ✗

Experimental finding: A few correctors go a long way

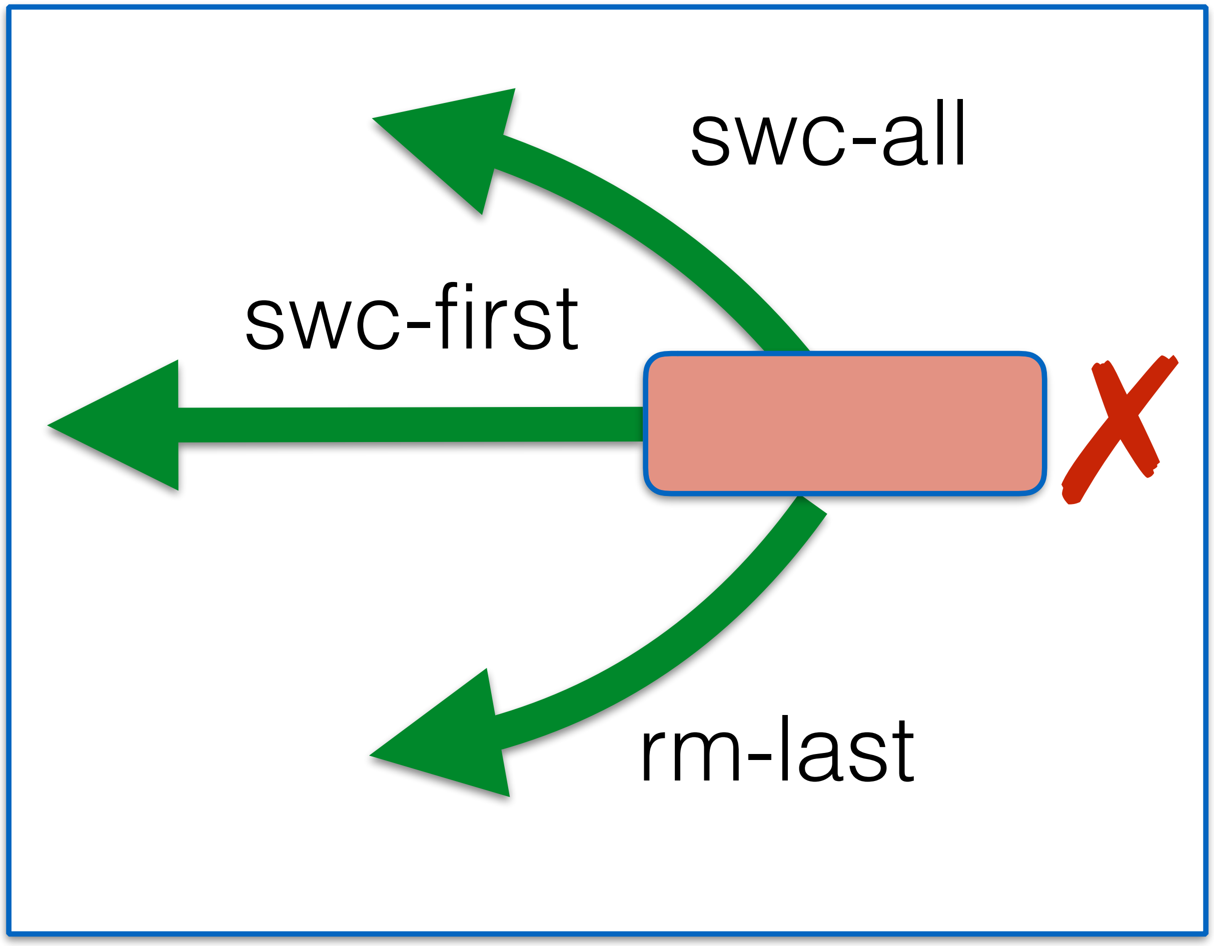
- Instrumented Dropbox for all users over 24-hour period
  - (No policy change)
- Set of three correctors:
  - C_{top3}= {swc-all, swc-first, rm-last}
- Key results:
  - Could correct 9% of failed password submissions
  - **3% of all users rejected but entered at least one password correctable by C_{top3}**



Users needlessly turned away from service!

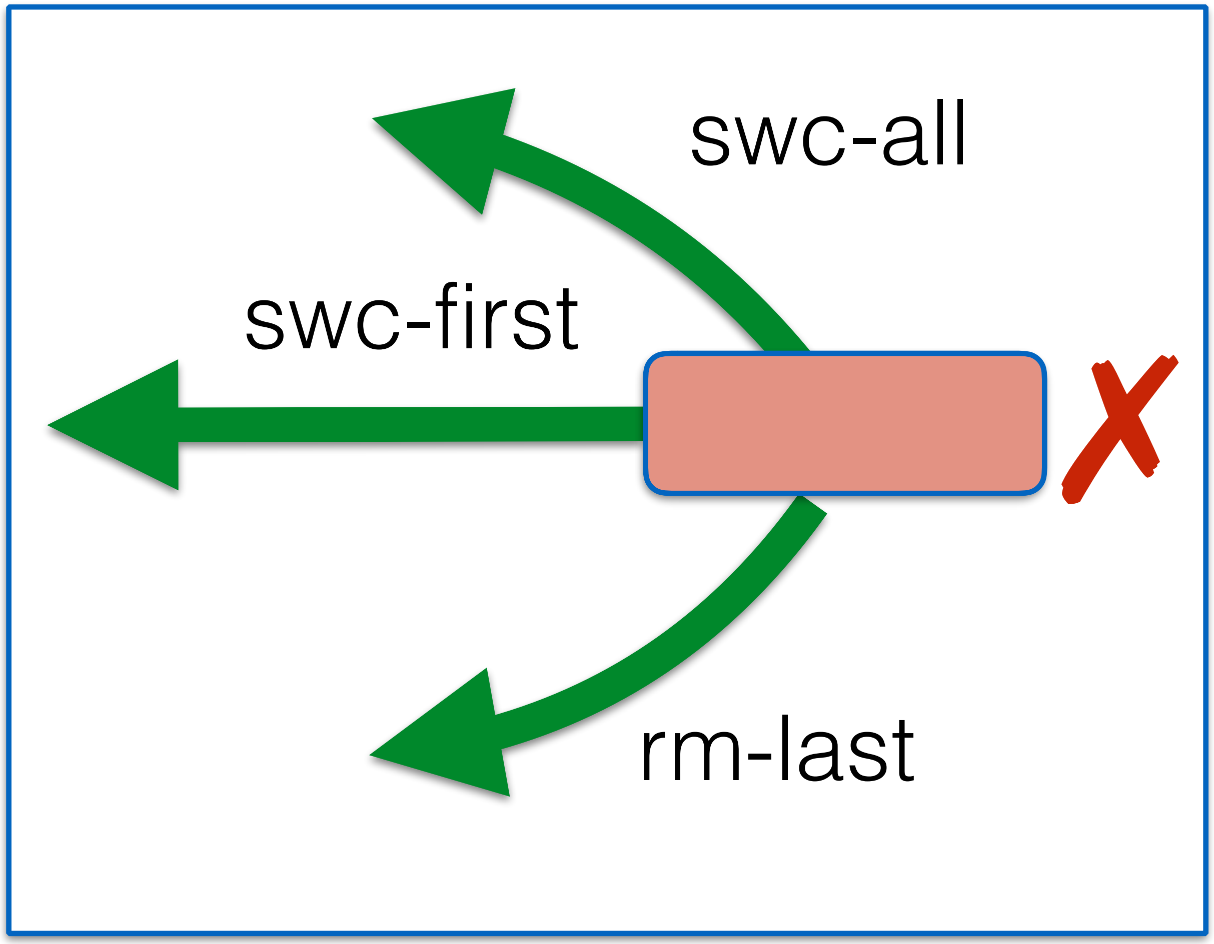
Another finding: Minimal security impact

- Analysis shows little security degradation for C_{top3}
  - Very pessimistic (1000 guesses): 9.54% ➜ 11.96% adv. success
  - Realistic analyses / scheme show *virtually no security loss*
- Intuition: Common passwords are lexicographically sparse
  - E.g., "password" is common, but "PASSWORD" isn't



Findings

- General "free corrections theorem" shows optimal strategy for correction with no security loss
  - Reasonable approximation possible
- **Conclusion: Typo correctors can be simple, effective, and safe for PASS!**



Summing up

- Enterprise password protections are broken
- **PASS**'s goal: improve best practice for passwords and democratize it
- **PASS** offers principled and practical:
  - Hardening of password databases
  - Typo correction
- Toward democratization:
  - Open-source (PRF)
  - Commercial offering in the works

To learn more about PASS

- Papers:
  - The Pythia PRF Service. A. Everspaugh, R. Chatterjee. S. Scott, A. Juels, and T. Ristenpart. USENIX Security. 2015.
  - pASSWORD tYPOS and How to Correct Them Securely. R. Chatterjee, A. Athalye, D. Akhawe, A. Juels, and T. Ristenpart. 2016. In submission.
- E-mail:

  - juels@cornell.edu
  - ristenpart@cornell.edu