Making Password Checking Systems Better

Tom Ristenpart

CORNELL TECH

<u>Covering joint work with:</u>
Anish Athayle, Devdatta Akawhe, Joseph Bonneau, Rahul Chatterjee, Adam Everspaugh, Ari Juels, Sam Scott

Password checking systems

tom, password1

Login
server

tom	password1
alice	123456
bob	p@ssword!

(plus hundreds of millions more)

Allow login if:

Attack detection mechanisms don't flag request

Password matches

Sometimes: second factor succeeds

Problems w/ password checking systems

tom, password1

Login server

tom	password1
alice	123456
bob	p@ssword!

People often enter wrong password:

- Typos
- Memory errors

Passwords databases must be protected:

- Server compromise
- Exfiltration attacks (e.g., SQL injection)

Widespread practice:

- Apply hashing w/ salts
- Hope slows down attacks enough

Today's talk

Pythia: moving beyond "hash & hope"

Harden hashes with off-system secret key using
partially oblivious pseudorandom function protocol

[Everspaugh, Chatterjee, Scott, Juels, R. – USENIX Security 2015]

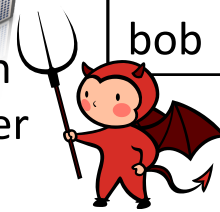
Typo-tolerant password checking

In-depth study of typos in user-chosen passwords
Show how to allow typos without harming security

[Chatterjee, Athayle, Akawhe, Juels, R. – Oakland 2016]

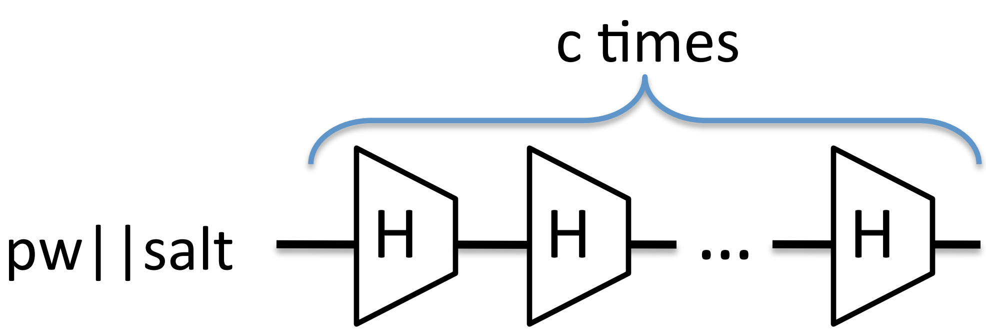
Password checking systems

tom, password1



tom	$salt_1$, $H^c(password1,salt_1)$
alice	$salt_2$, $H^c(123456,salt_2)$
bob	$salt_3$, $H^c(p@ssword!,salt_3)$

Websites should ***never*** store passwords directly,
they should be (at least) hashed with a salt (also stored)



Cryptographic hash function H
(H = SHA-256, SHA-512, etc.)

Common choice is c = 10,000

Better: scrypt, argon2

UNIX password hashing scheme, PKCS #5
Formal analyses: [Wagner, Goldberg 2000] [Bellare, R., Tessaro 2012]

Password database compromises

⋮	year	# stolen	% recovered	format
rockyou	2012	32.6 million	100%	plaintext (!)
LinkedIn	2012	117 million	90%	Unsalted SHA-1
Adobe®	2013	36 million	??	ECB encryption
Ashley Madison® Life is short. Have an affair.®	2015	36 million	33%	Salted bcrypt + MD5
⋮				

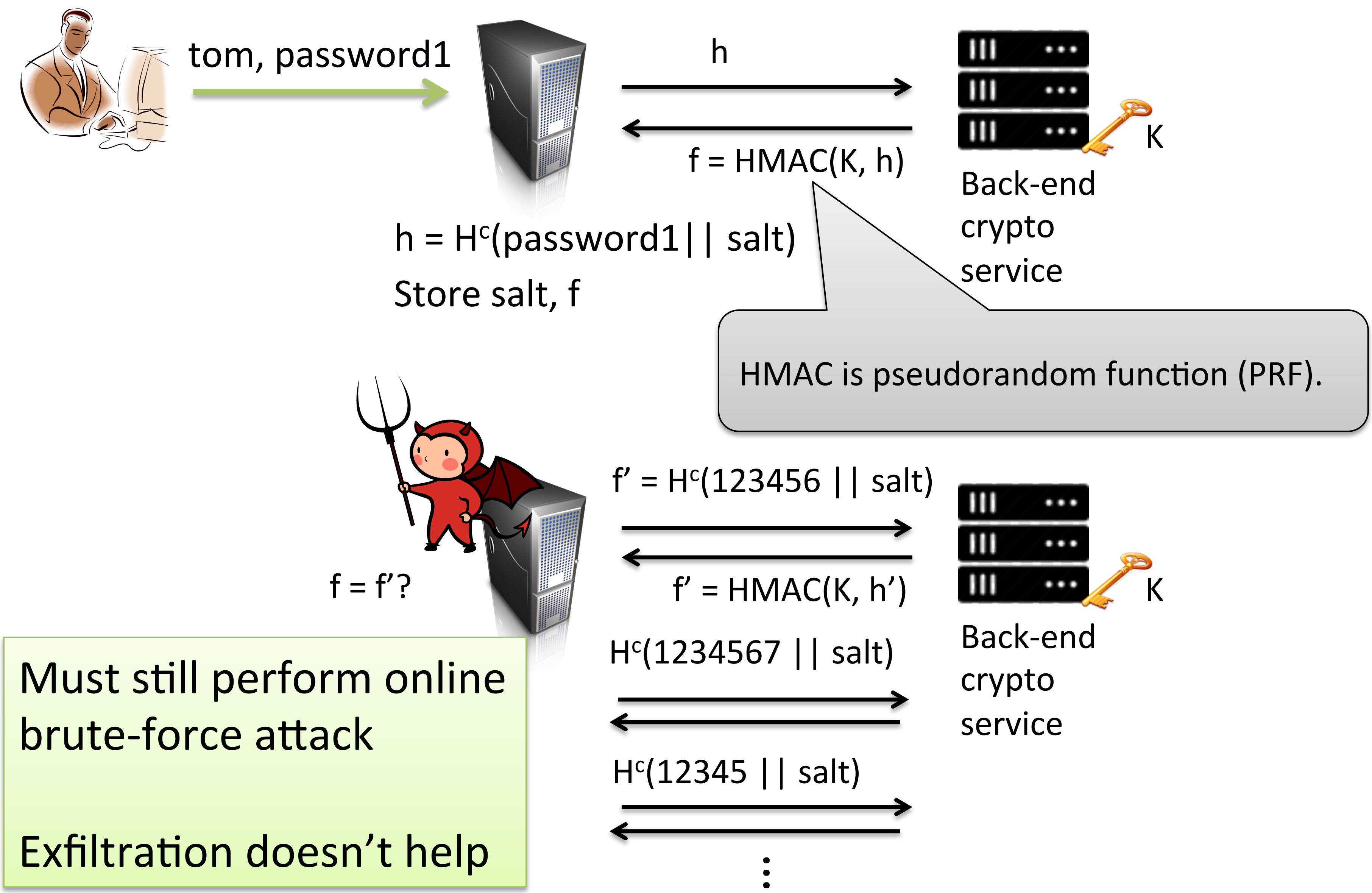
(1) Password protections often implemented incorrectly in practice

(2) Even in best case, hashing slows down but does not prevent offline brute-force cracking

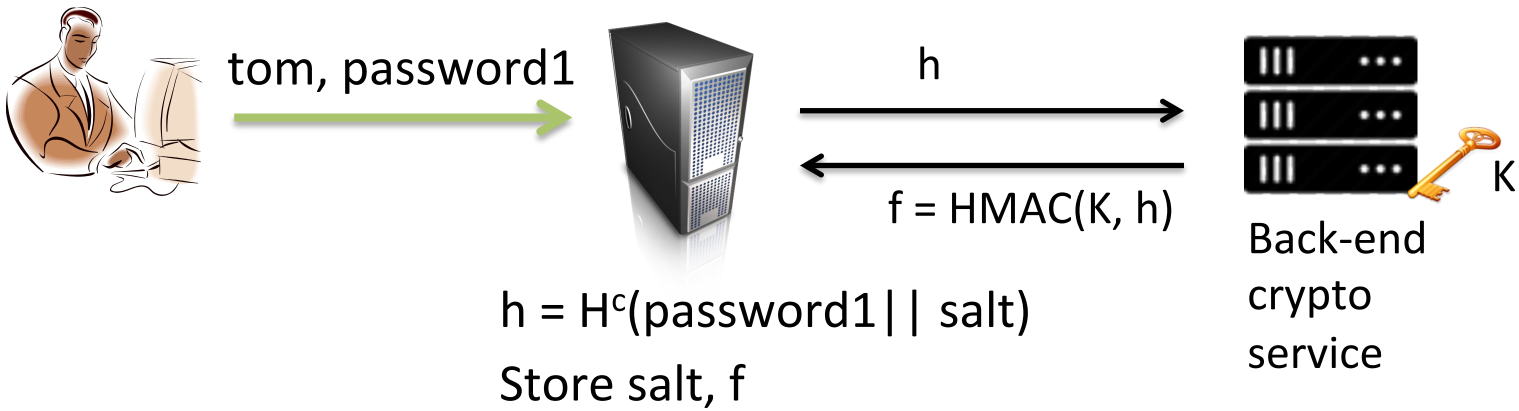
Facebook password onion

```
$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)
```

Strengthening password hash storage



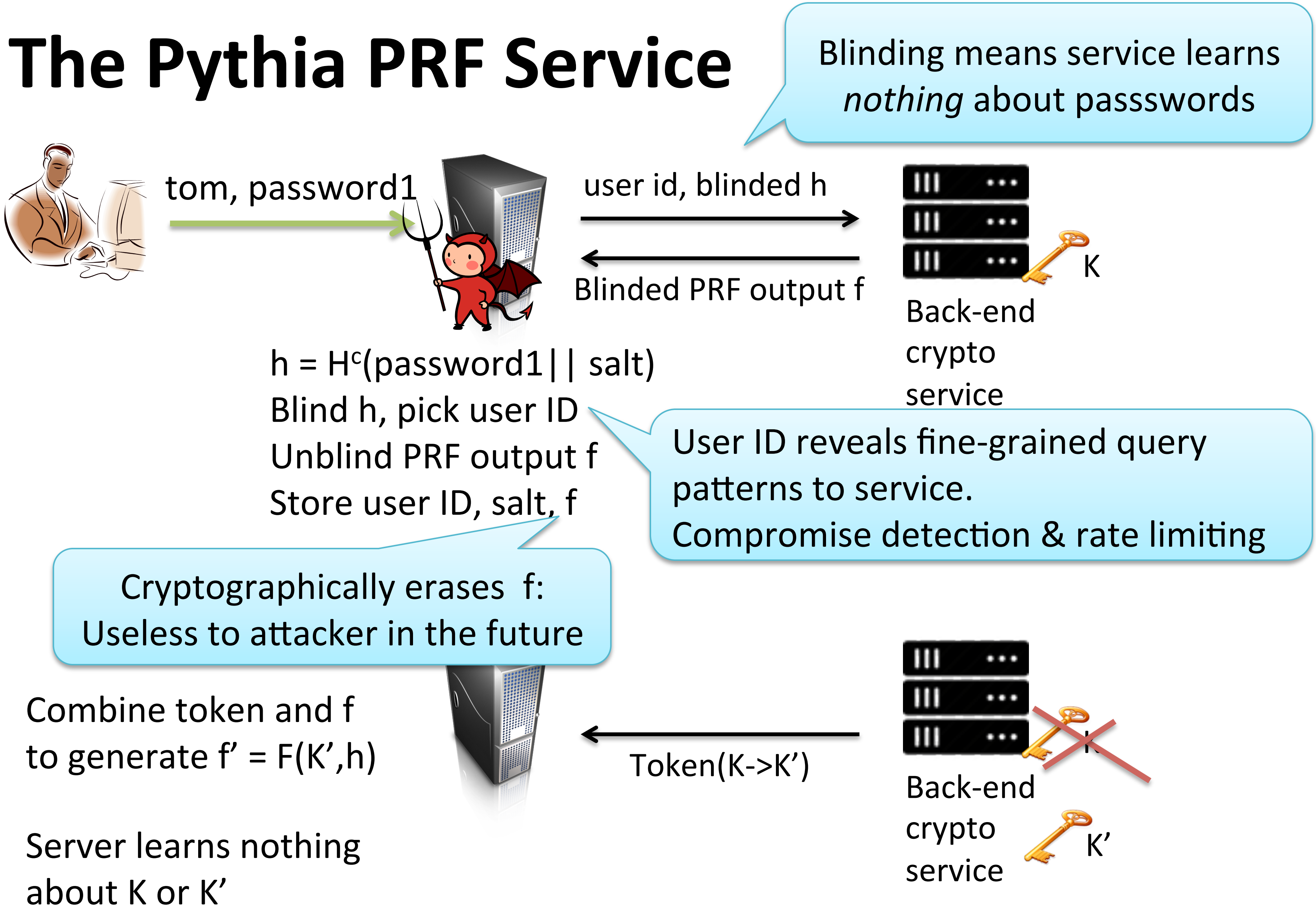
Strengthening password hash storage



Critical limitation: can't rotate K to a new secret K'

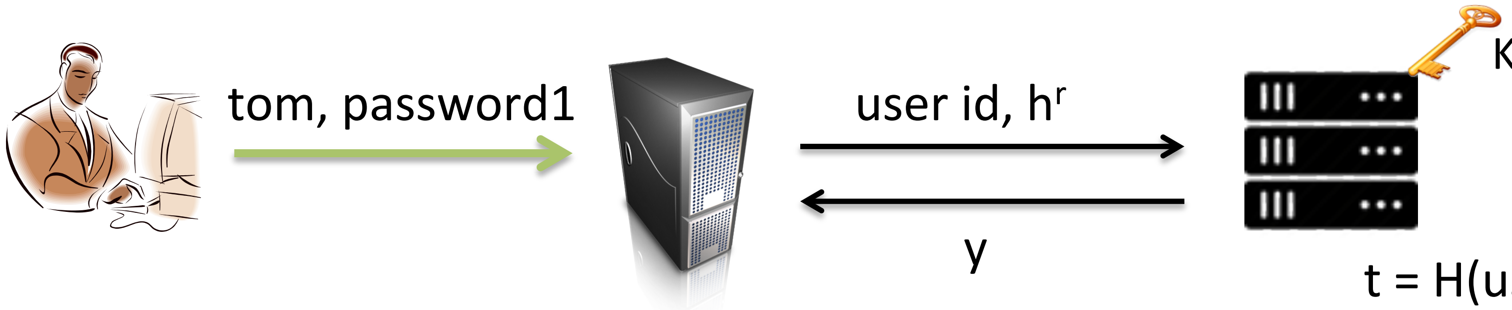
- Idea 1: Version database and update as users log in
  - *But doesn't update old hashes*
- Idea 2: Invalidate old hashes
  - *But requires password reset*
- Idea 3: Use secret-key encryption instead of PRF
  - *But requires sending keys to web server (or high bandwidth)*

The Pythia PRF Service



New crypto: partially-oblivious PRF

Groups G_1, G_2, G_T w/ bilinear pairing $e : G_1 \times G_2 \rightarrow G_T$ $\quad e(a^x, b^y) = c^{xy}$



$t = H(\text{user id})$

$y = e(t^K, h^r)$

$h = H^c(\text{password1} \,||\, \text{salt})$

Choose random r

$f = y^{1/r}$

Store user ID, salt, f

$$f = e(t^K, h^r)^{1/r} = e(t,h)^{Kr*1/r} = e(t,h)^K$$

- Pairing cryptographically binds user id with password hash
- Can add verifiability (proof that PRF properly applied)
- Key rotation straightforward: Token(K -> K') = K' / K
- Interesting formal security analysis (see paper)

The Pythia PRF Service

- Queries are fast despite pairings
  - PRF query: 11.8 ms (LAN) 96 ms (WAN)
- Parallelizable password onions
  - H^c and PRF query made in parallel (hides latency)
- Multi-tenant (theoretically: scales to 100 million login servers)
- Easy to deploy
  - Open-source reference implementation at
    http://pages.cs.wisc.edu/~ace/pythia.html

Today's talk

Pythia: moving beyond "hash & hope"

Harden hashes with off-system secret key using
partially oblivious pseudorandom function protocol

[Everspaugh, Chatterjee, Scott, Juels, R. – USENIX Security 2015]

Typo-tolerant password checking

In-depth study of typos in user-chosen passwords
Show how to allow typos without harming security

[Chatterjee, Athayle, Akawhe, Juels, R. – Oakland 2016]

Back to our big picture

tom, password1

Login server

tom	G_K(password1)
alice	G_K(123456)
bob	G_K(p@ssword!)

People often enter wrong password:
- Typos
- Memory errors

Passwords databases must be protected:
- Server compromise
- Exfiltration attacks (e.g., SQL injection)

Widespread practice:
- Apply hashing w/ salts
- Hope slows down attacks enough

Back to our big picture

tom, password1

Login server

tom	G_K(password1)
alice	G_K(123456)
bob	G_K(p@ssword!)

People often enter wrong password:

- Typos
- Memory errors

Users have hard time remembering (complex) passwords

[Ur et al. 2012] [Shay et al. 2012] [Mazurek et al. 2013] [Shay et al. 2014] [Bonneau, Schechter 2014]

Passwords can be difficult to enter without error (typo)

[Keith et al. 2007, 2009] [Shay et al. 2012]

Suggestions for error-correcting passphrases

[Bard 2007] [Jakobsson, Akavipat 2012] [Shay et al. 2012]

Facebook passwords are not case sensitive (update)

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

By Emil Protalinski for Friending Facebook | September 13, 2011 -- 12:26 GMT (05:26 PDT) | Topic: Security

password1 Password1 PASSWORD1

Typo-tolerant password checking:

Allow registered password or some typos of it

We focus on *relaxed* checkers

tom, Password1

tom	G_K(password1)
alice	G_K(123456)
bob	G_K(p@ssword!)

Apply typo corrector functions to incorrect submitted password:

Slow to compute G_K

	G_K(Password1)	✗
Apply caps lock corrector	G_K(pASSWORD1)	✗
Apply first case flip corrector	G_K(password1)	✓
…	…	

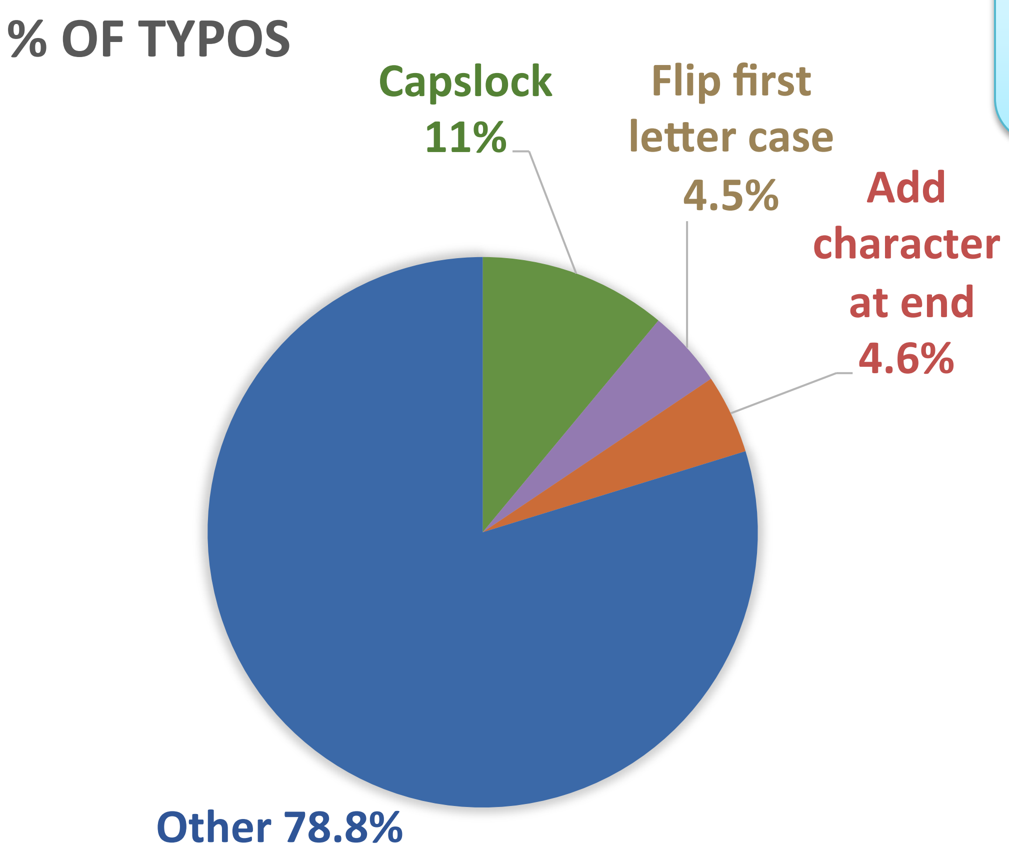
Can we find *small* but *useful* set of typo correctors?

Works with existing password hardening schemes
No change in what is stored

Mechanical Turk transcription study

100,000+ passwords typed by 4,300 workers

Top 3 account for 20% of typos



Impact of Top 3 typos in real world

Dropbox

Instrumented production login of Dropbox to quantify typos
NOTE: We did not admit login based on relaxed checker

24 hour period:

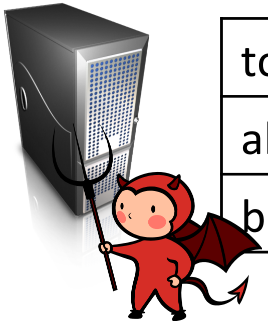
- **3% of all users** failed to login because one of top 3 typos
- 20% of users who made a typo would have saved at least 1 minute in logging into Dropbox if top 3 typos are corrected.

Allowing typos in password will add several person-months of login time every day.

Typo-tolerance would significantly improve usability of password-based login

Can it be secure?

Threat #1: Server compromise

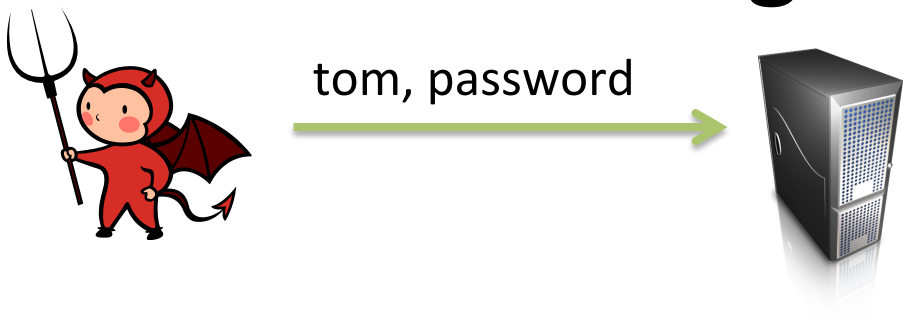


tom	G_K(password1)
alice	G_K(123456)
bob	G_K(p@ssword!)

No change to password DB

No change in security after compromise

Threat #2: Remote guessing attacks



tom	G_K(password1)
alice	G_K(123456)
bob	G_K(p@ssword!)

	G_K(password)	✗
Apply caps lock corrector	G_K(PASSWORD)	✗
Apply first case flip corrector	G_K(Password)	✗
Apply extra char corrector	G_K(passwor)	✗

Threat #2: Remote guessing attacks

tom, password

tom, iloveyou

⋮

tom	G_K(password1)
alice	G_K(123456)
bob	G_K(p@ssword!)

Server locks account after q failed attempts (e.g., q=10)

	G_K(iloveyou)	✗
Apply caps lock corrector	G_K(ILOVEYOU)	✗
Apply first case flip corrector	G_K(Iloveyou)	✗
Apply extra char corrector	G_K(iloveyo)	✗

Up to 4 passwords checked at cost of 1 query

=>

~~Attack success increases by 4x~~

Attack simulation using password leaks

Adversary knows:

Distribution of passwords, and the set of correctors ()

Exact checking

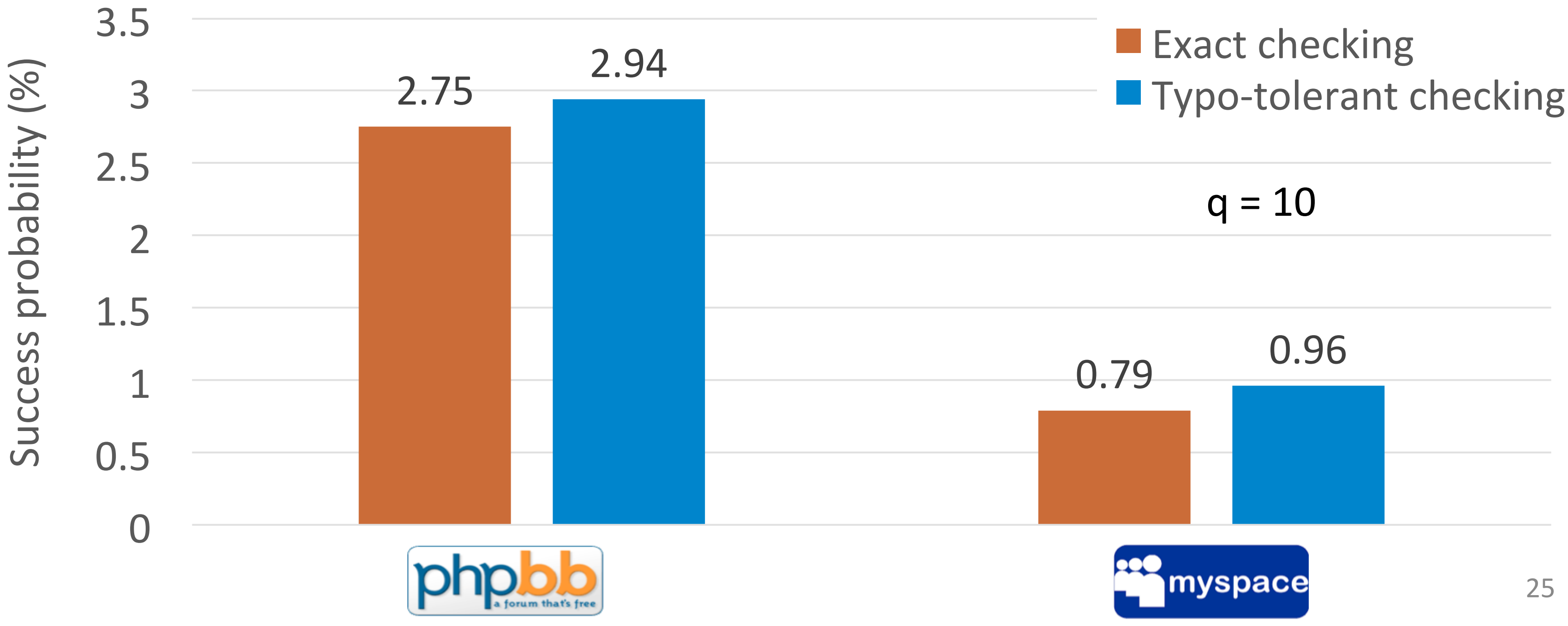
Query most probable q passwords

Typo-tolerant checking

Query q passwords that maximizes success

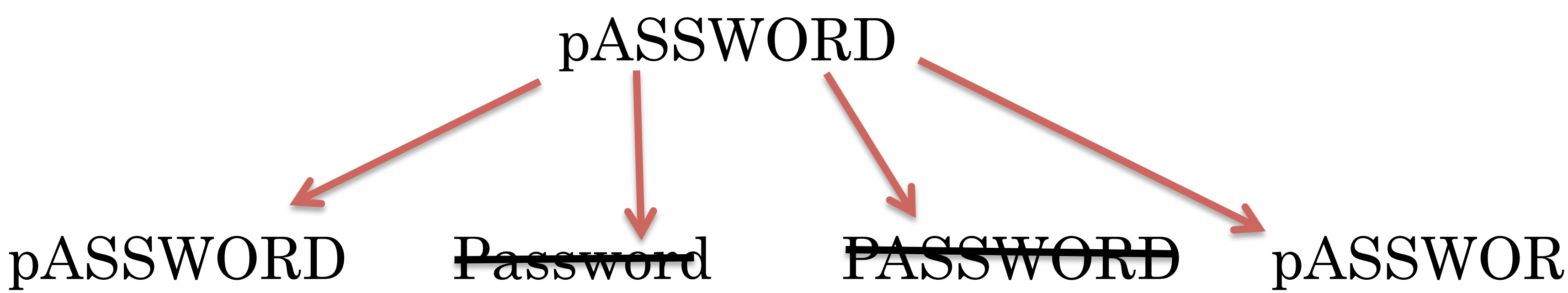
NP-complete problem.

Compute using greedy approximation



Security-sensitive typo tolerance

Don't check a correction if the resulting password is too popular.



Free Corrections Theorem:

For any password distribution, set of correctors, and query budget q, there exists a typo-tolerant checking scheme with no loss in security

Security-sensitive typo tolerance

Assume distribution over passwords and order them in decreasing probability:

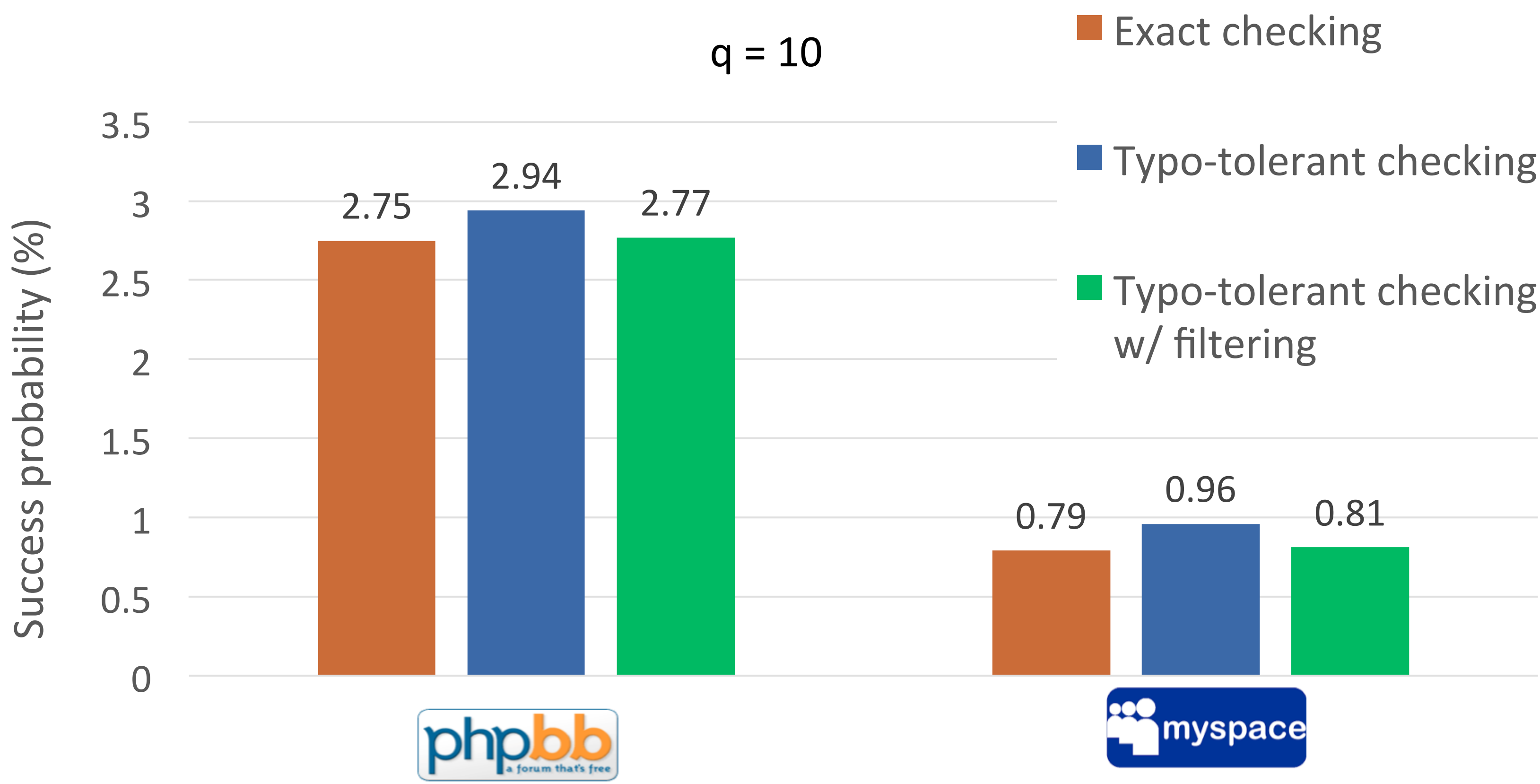
$$pw_1 \quad pw_2 \; \dots \; pw_q \quad pw_{q+1} \quad pw_{q+2} \quad pw_{q+3} \; \dots$$

<u>Construction:</u>
For any password, check as many typos as one can while ensuring correctness and that $\Sigma_{\text{pw corrected}}$ Pr[pw] $\leq$ Pr[pw_q]

Ensures optimal adversarial strategy is to query $pw_1,\dots,pw_q$ against typo-tolerant checker. Same as for strict checker

Checkers w/ heuristic filtering

Use password leak rockyou to estimate distribution



Typo-tolerance can enhance user experience without degrading security in practice

Today's talk

Pythia: moving beyond "hash & hope"

Harden hashes with off-system secret key using
partially oblivious pseudorandom function protocol

[Everspaugh, Chatterjee, Scott, Juels, R. – USENIX Security 2015]

Typo-tolerant password checking

In-depth study of typos in user-chosen passwords
Show how to allow typos without harming security

[Chatterjee, Athayle, Akawhe, Juels, R. – Oakland 2016]