Course Business

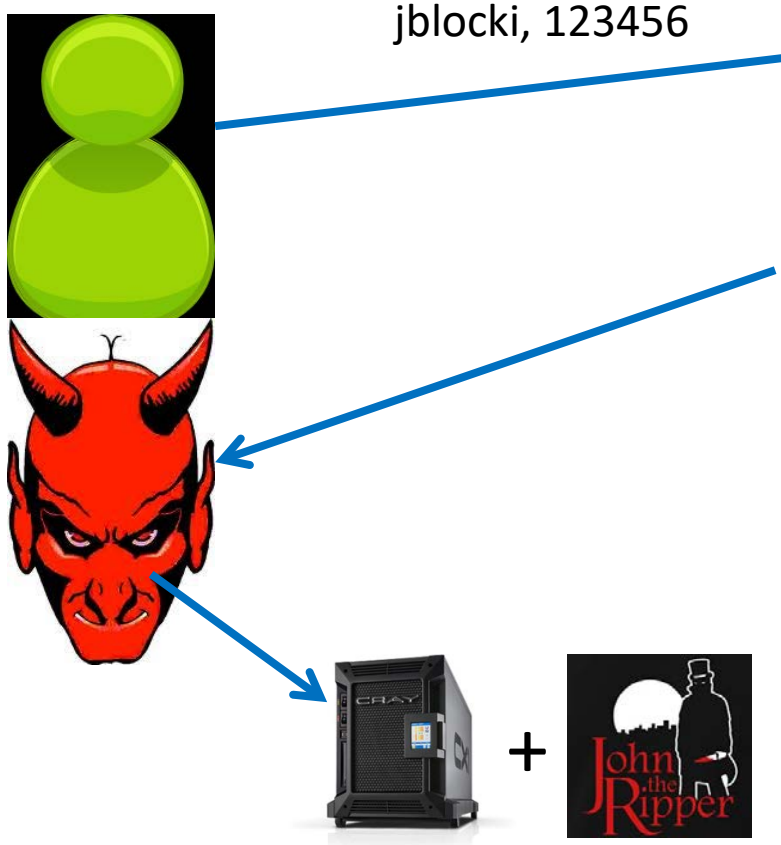
- Homework due today
- Final Exam Review on Monday, April 24th
  - Practice Final Exam Solutions Released Monday
- Final Exam on Monday, May 1st (in this classroom)
  - Adib will proctor
- I am traveling April 25-May 3rd
  - Will still be available by e-mail to answer questions
- Guest Lectures on April 26 and 28
  - Secret Sharing --- Prof. Kate (April 26)
  - TBD --- Prof. Spafford (April 28)

Cryptography
CS 555

Topic 40: Password Hashing and Authentication

Slide Credit: Slides about Pythia PRF Service from Tom Ristenpart

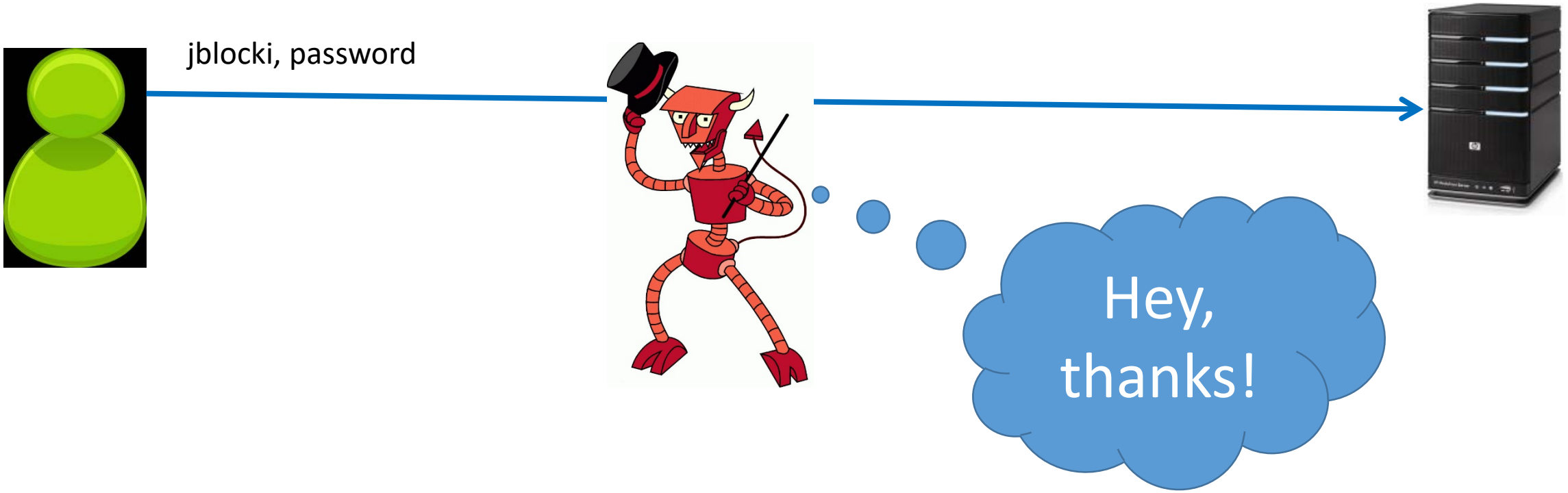
Password Storage



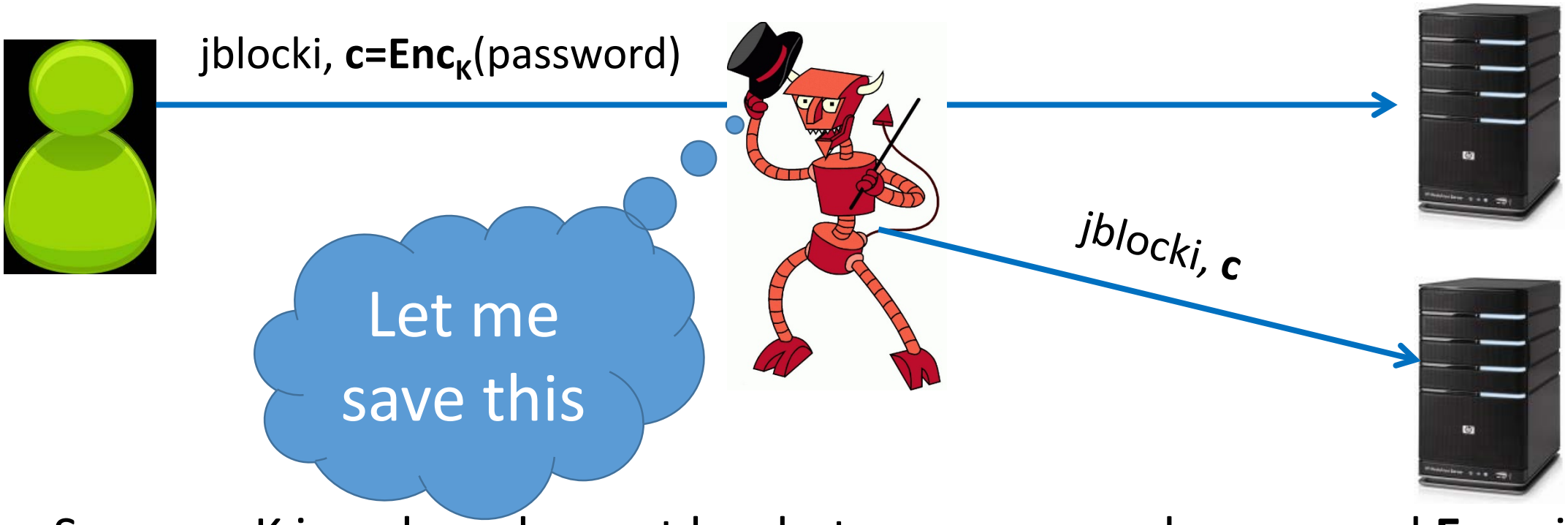
Username	Salt	Hash
jblocki	89d978034a3f6	85e23cfe0021f584e3db87aa72630a9a2345c062

SHA1(12345689d978034a3f6)=85e23cfe0021f584e3db87aa72630a9a2345c062

How to send the password?



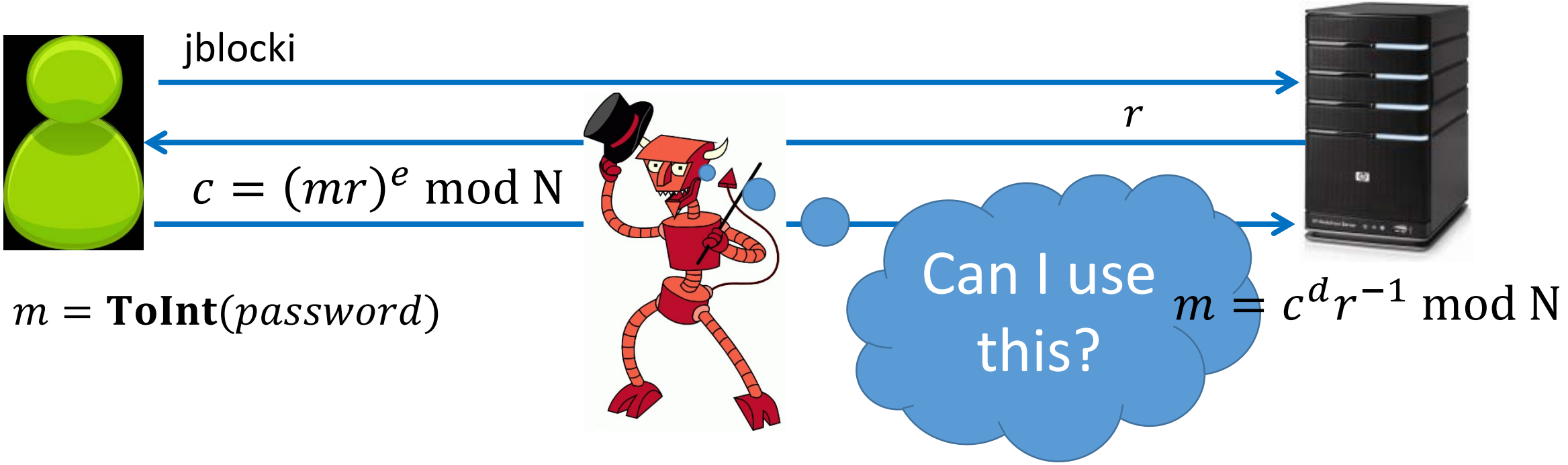
How to send the password?



- Suppose K is a shared secret key between user and server and **Enc$_K$** is CCA-Secure
- Unlikely that user and server share secret key anyway…

How to send the password?

RSA key: (N,e,d)

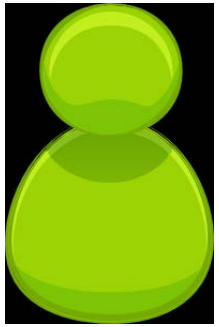


VerifyPasswordGuess(guess)

1. Let $\mathrm{m}' \leftarrow \mathbf{ToInt}(guess)$
2. Check if $c == (m'r)^e$

How to send the password?

(pk,sk)



Jblocki, **c**, **Enc**$_K$(password)

$(c, k) = \mathbf{Encaps}_{pk}(R)$

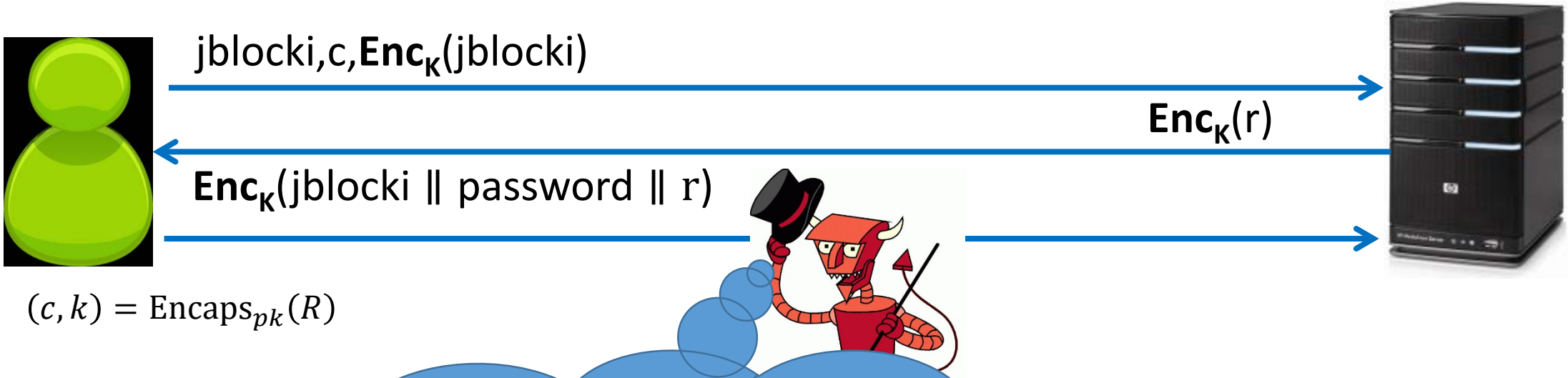
K=$\mathrm{Decaps}_{sk}(c)$

Can I use this?

jblocki, **c**, **Enc**$_K$(password)

joe, **c**, **Enc**$_K$(password)

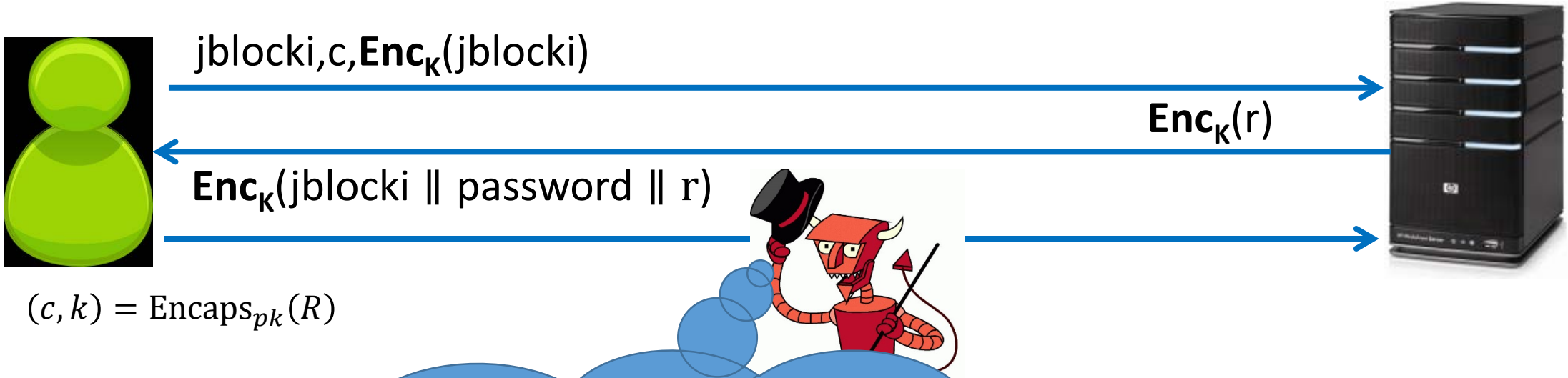
How to send the password?



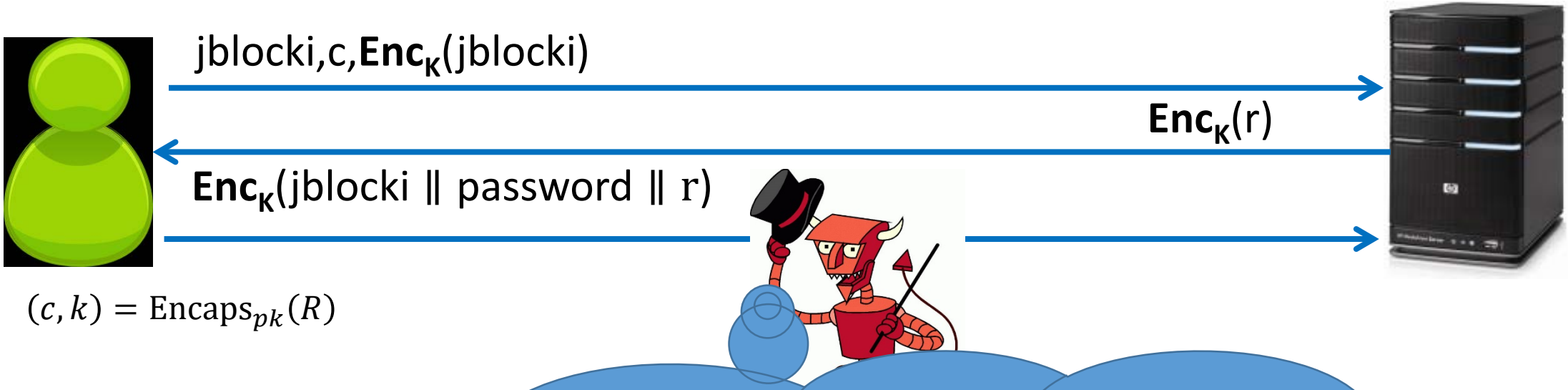
$(c, k) = \text{Encaps}_{pk}(R)$

Curses! The Encaps/Enc are CCA-Secure and the ciphertext is bound to the username/session

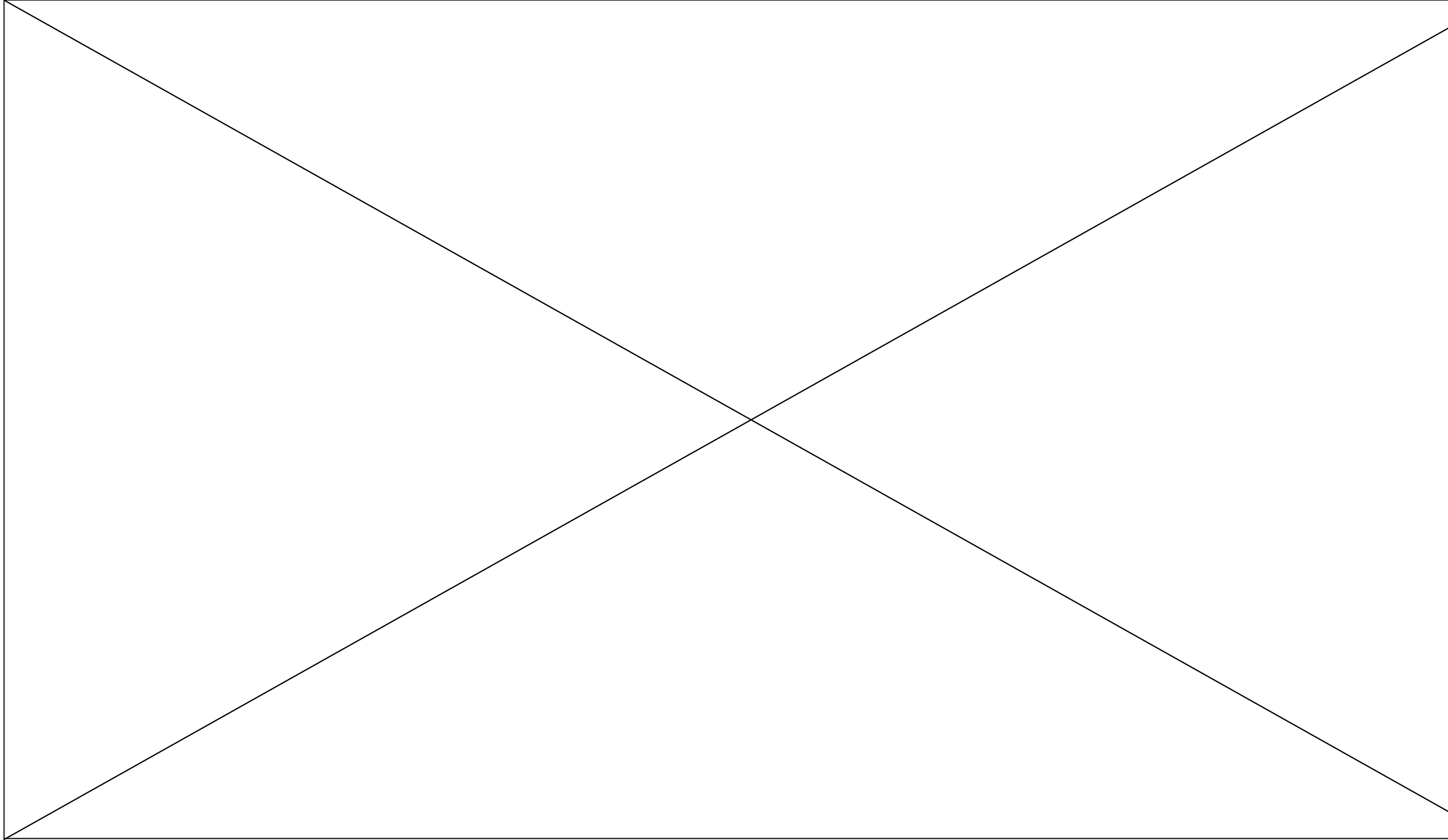
How to send the password?



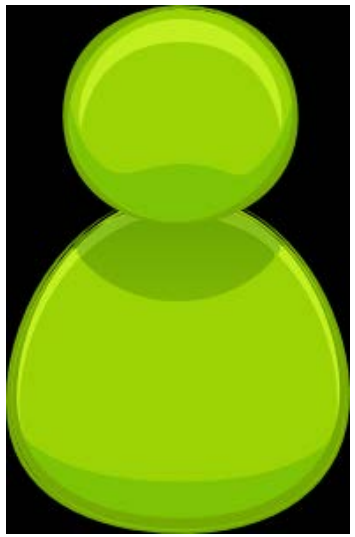
How to send the password?



|**Enc**$_K$(jblocki ‖ password ‖ r)|
might reveal length of password
See http://www.ringroadbug.com/



How to send the password?

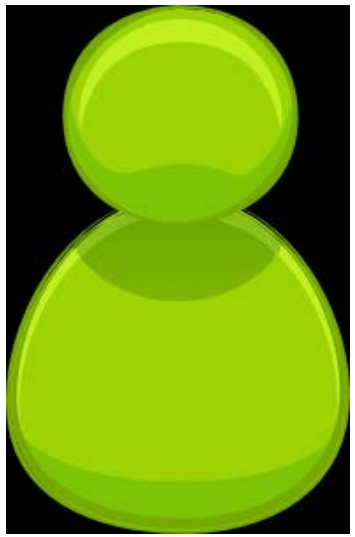


TLS handshake

$\mathbf{Enc}_K$(jblocki ‖ password)

Authenticated Encryption (AES-GCM)
(still need to worry about padding!)

How to send the password?



TLS handshake

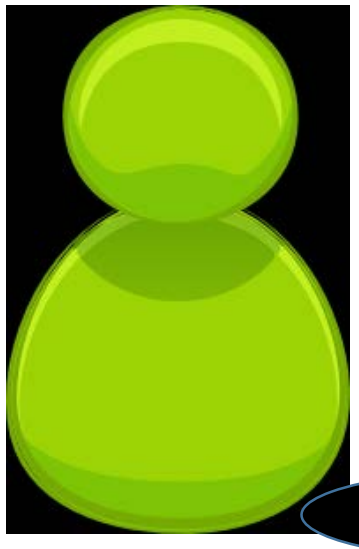
$\mathbf{Enc_K}$(jblocki ‖ Pad(password))

Authenticated Encryption (AES-GCM)

Pad(password)

1. $\ell \leftarrow Length(password)$
2. Find k,r s.t $\ell = 50k + r$
3. Return $password \parallel x^{50-r}$ (e.g., x is special character disallowed in passwords)

How to send the password?



TLS handshake

$\mathbf{Enc}_K$(jblocki ‖ h=H(jblocki ‖ password))

Authenticated Encryption (AES-GCM)

Advantage if server doesn't salt or hash passwords
See http://plaintextoffenders.com/

Username	Salt	Hash
jblocki	89d978034a3f6	85e23cfe0021f584e3db87aa72630a9a2345c062

MHF(h,89d978034a3f6)=85e23cfe0021f584e3db87aa72630a9a2345c062

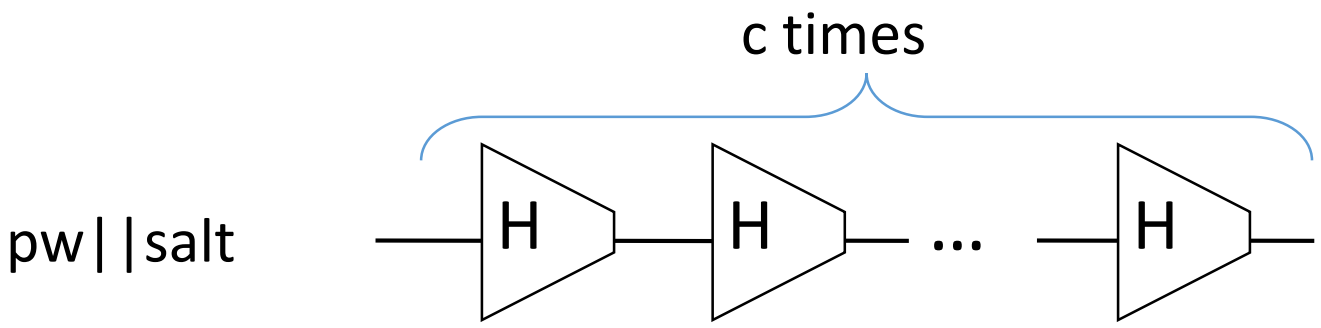
Password checking systems

tom, password1

Login server

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]

AshleyMadison hack: 36 million user hashes

Salts + Passwords hashed using bcrypt with c = 2^{12} = 4096

4,007 cracked directly with trivial approach

290729 123456
79076 12345
76789 123456789
59462 password
49952 iloveyou
33291 princess
…

Password cracker

List of possible passwords in order of their likelihood
Recompute hash and check

Examples: Hashcat, Johntheripper, academic projects

AshleyMadison hack: 36 million user hashes

Salts + Passwords hashed using bcrypt with c = 2^{12} = 4096

4,007 cracked directly with trivial approach

CynoSure analysis: **11 million** hashes cracked

>630,000 people used usernames as passwords

MD5 hashes left lying around accidentally

http://cynosureprime.blogspot.com/2015/09/csp-our-take-on-cracked-am-passwords.html

AshleyMadison in good company

⋮

rockyou

32.6 million leaked (2012)
32.6 million recovered (plaintext!)

LinkedIn

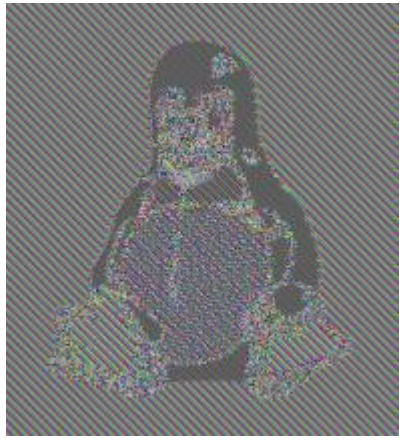
6.5 million leaked (2012)
5.85 million recovered in 2 weeks

YAHOO!

442,832 leaked (2012)
442,832 recovered

Adobe

36 million accounts leaked (2013)
Encrypted, but with ECB mode



⋮

(1) Password protections often implemented incorrectly in practice

(2) Even in best case, hashing slows down but does not prevent password recovery

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

tom, password1

h

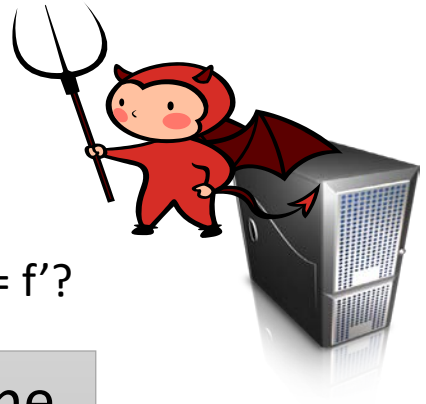
f = HMAC(K, h)

Back-end crypto service

$h = H^c(password1 || salt)$

Store salt, f

HMAC is pseudorandom function (PRF).



f = f'?

$f' = H^c(123456 || salt)$

f' = HMAC(K, h')

Back-end crypto service

$H^c(1234567 || salt)$

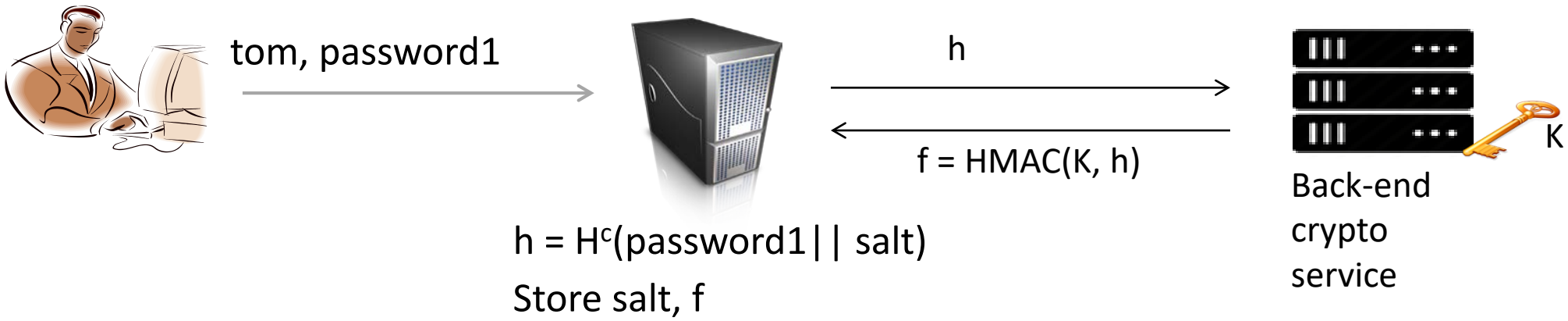
$H^c(12345 || salt)$

⋮

Must still perform online brute-force attack

Exfiltration doesn't help

Strengthening password hash storage



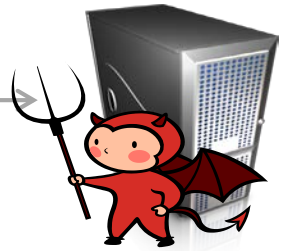
Critical limitation: can't rotate K to a new secret K'

- 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

Blinding means service learns *nothing* about passwords

tom, password1



user id, blinded h

Blinded PRF output f

Back-end crypto service

$h = H^c(password1 || salt)$
Blind h, pick user ID
Unblind PRF output f
Store user ID, salt, f

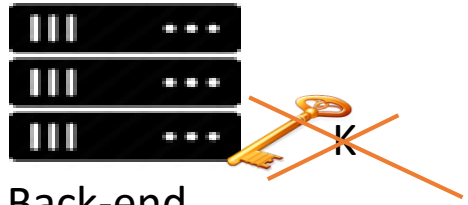
User ID reveals fine-grained query patterns to service.
Compromise detection & rate limiting

Cryptographically erases f:
Useless to attacker in the future

Combine token and f
to generate f' = F(K',h)

Server learns nothing about K or K'

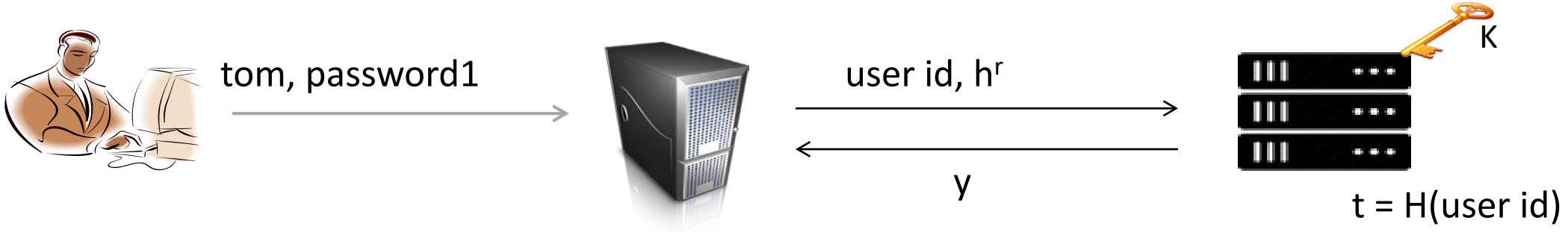
Token(K->K')



Back-end crypto service

New crypto: partially-oblivious PRF

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



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

- 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

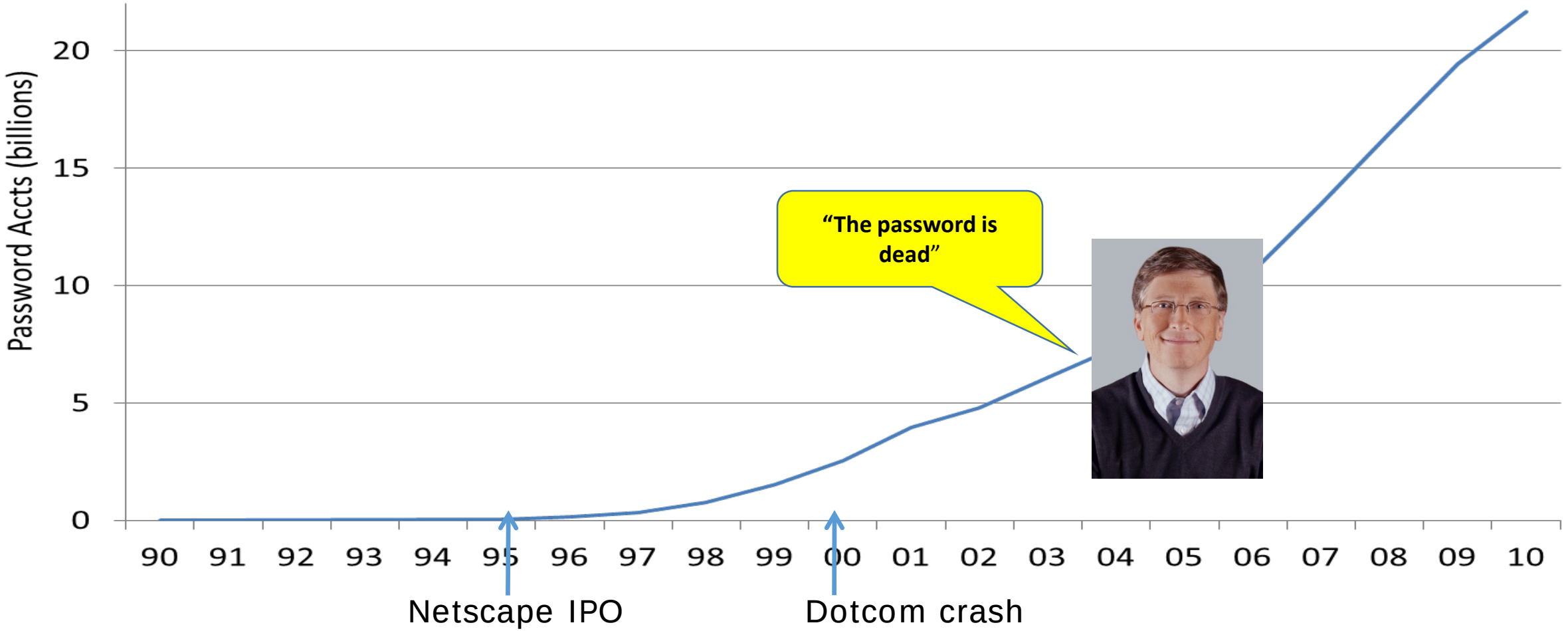
- Queries are fast despite pairings
  - PRF query: 11.8 ms (LAN) 96 ms (WAN)
- Parallelizable password onions
  - H^c and PRF query made in parallel (hides latency)
- Multi-tenant (theoretically: scales to 100 million servers)
- Easy to deploy
  - Open-source reference implementation at
    http://pages.cs.wisc.edu/~ace/pythia.html

Next Class

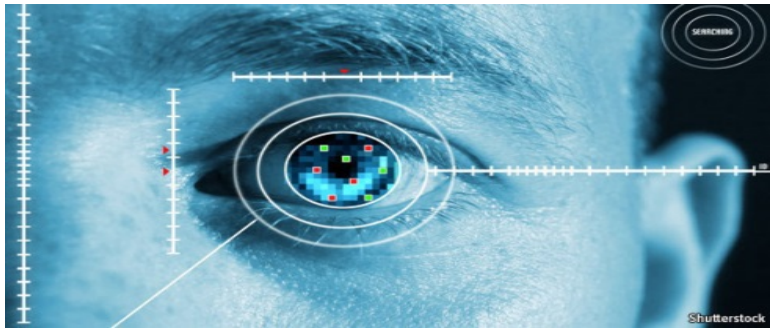
- Review for Final Exam
- **Recommendation:** Try to complete the practice final exam over the weekend

Passwords vs time: Look how far we've come

Source: Cormac's estimate

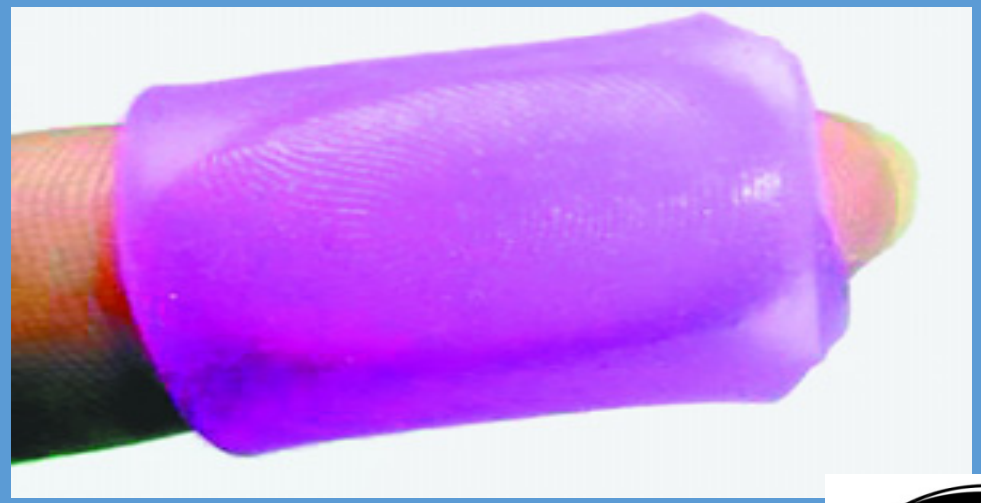


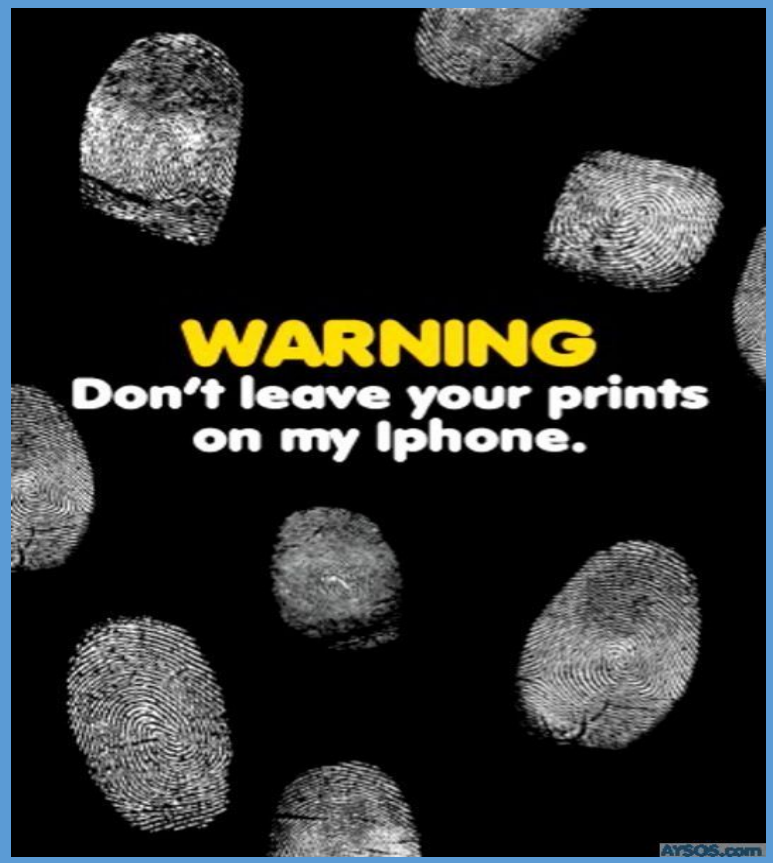
Biometrics



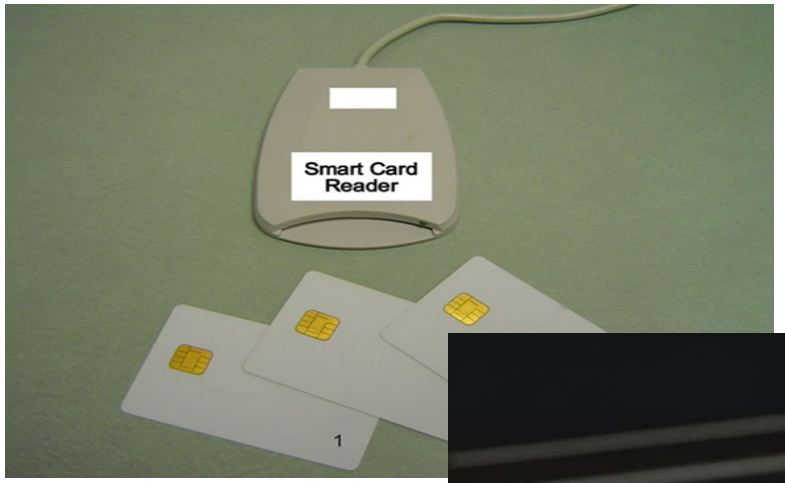
Biometrics

Challenges: Revoke? Secrecy?





Hardware Tokens

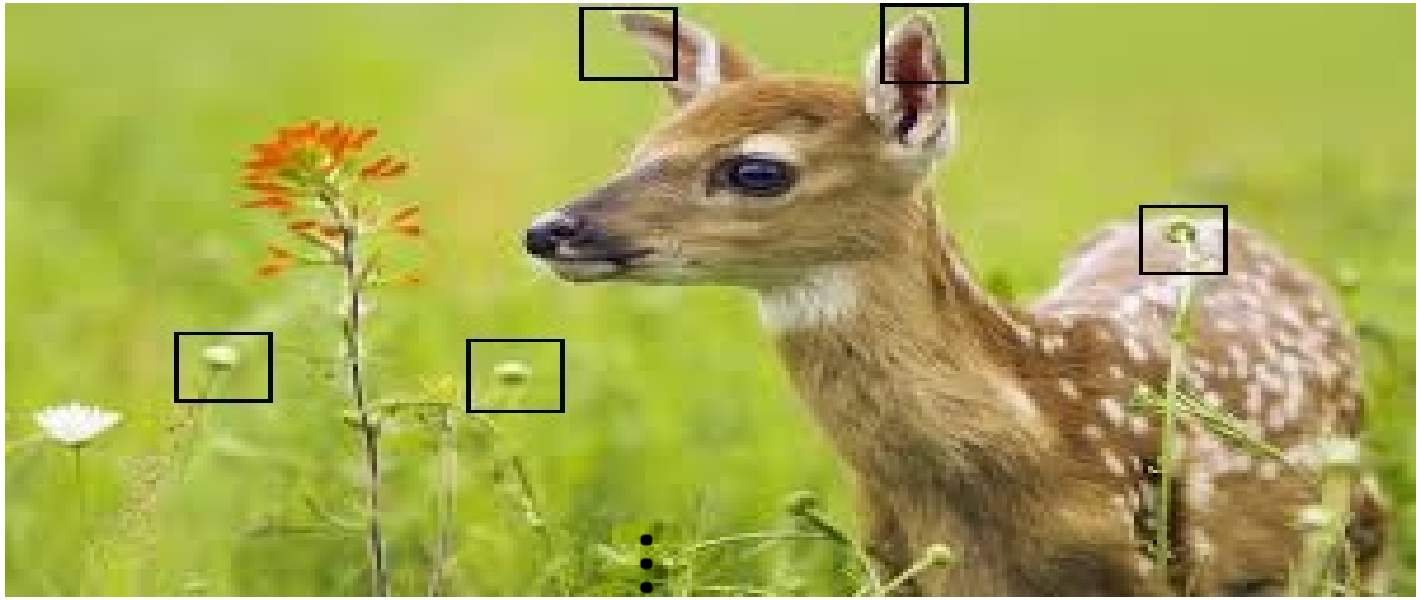


Hardware Tokens

Challenge: $$$ + more stuff to carry around

Graphical Passwords

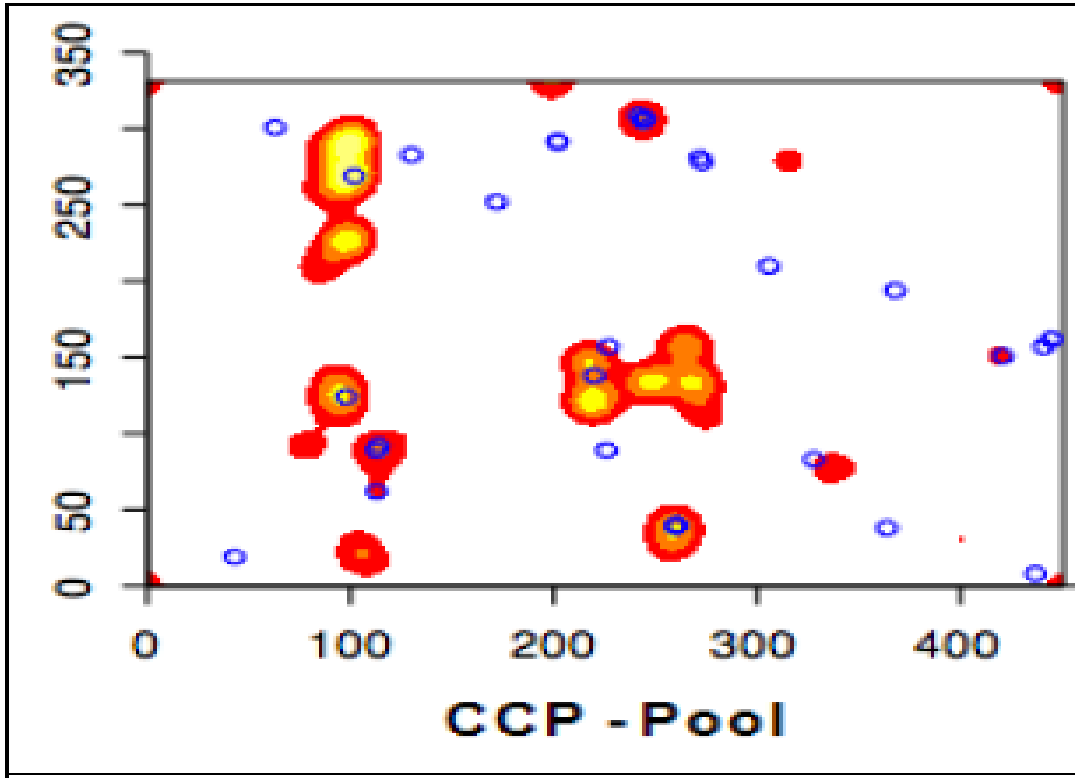
- Examples:
  - Passfaces, Cued Click Points, Windows 8



Graphical Passwords

Challenge: Multiple Passwords

Graphical Passwords: Hotspots



Graphical Passwords: Hotspots

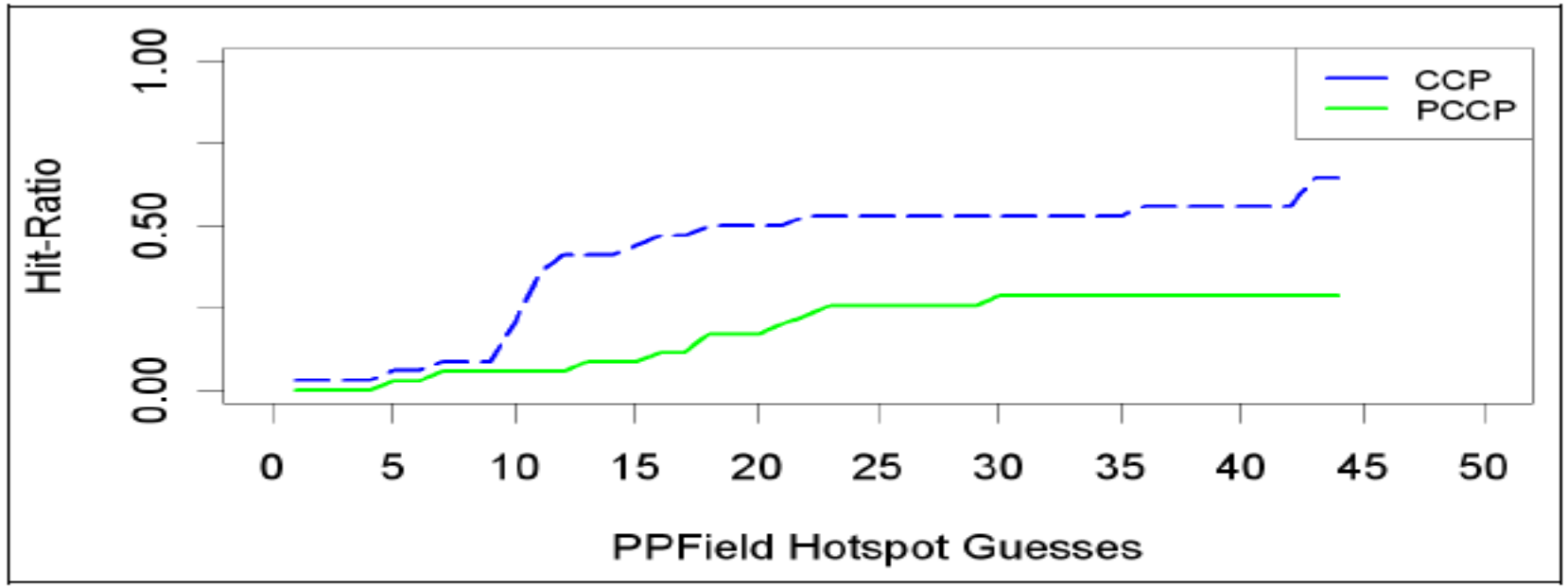


Figure 7: Individual click-points "guessable" using hotspots from the PassPoints-field study on the Pool image

Password Managers

- Password Management Software

LastPass ****

The Last Password You'll Ever Need.

Related Work

Challenge: Single point of failure

References

- **Depth-Robust Graphs and Their Cumulative Memory Complexity**. with Joel Alwen and Krzysztof Pietrzak. EUROCRYPT 2017 (to appear). [ePrint]
- **On the Computational Complexity of Minimal Cumulative Cost Graph Pebbling**. with Samson Zhou. (Working Paper). [arXiv]
- **Towards Practical Attacks on Argon2i and Balloon Hashing**. with Joel Alwen. EuroS&P 2017 (to appear). [ePrint]
- **Efficiently Computing Data Independent Memory Hard Functions**. with Joel Alwen. CRYPTO 2016. [Full Version]