

# Engineering Robust Server Software Security

Significant portions based on slides from  
Micah Sherr @ Georgetown

# Security

- First major topic: **security**
  - What do we mean by security?

# Security

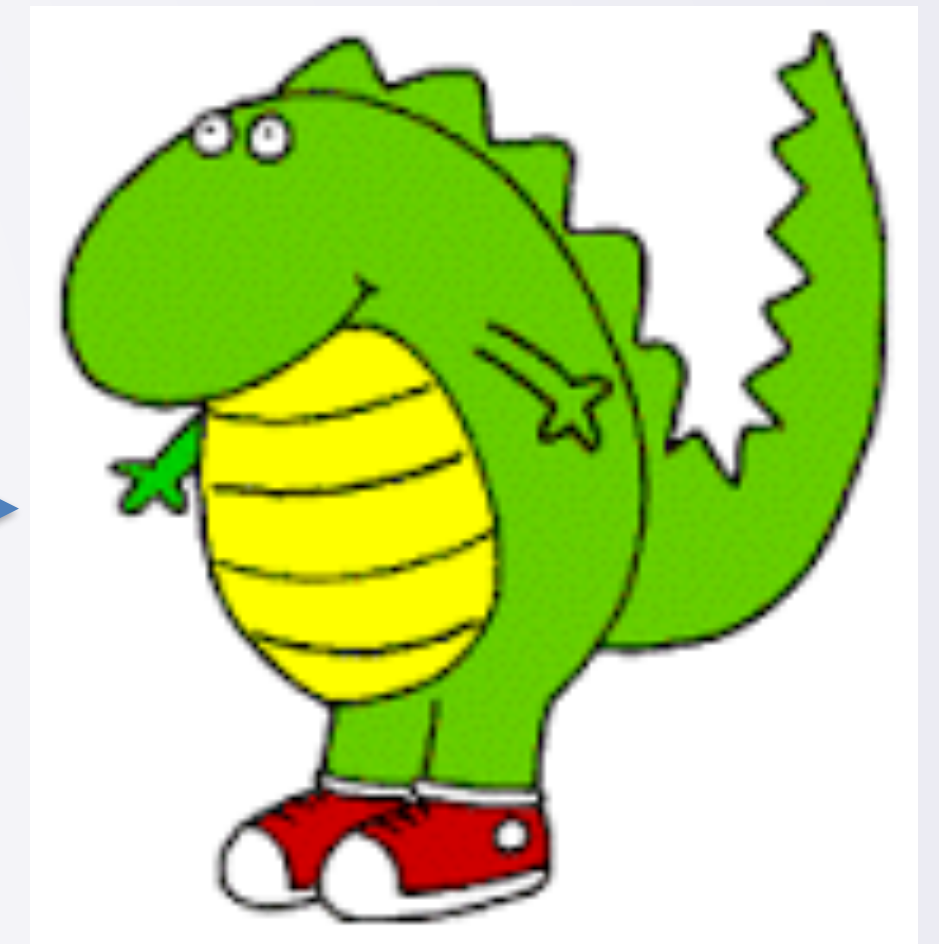
- First major topic: **security**
  - What do we mean by security?
- **Confidentiality**: things are kept secret
- **Integrity**: things cannot be tampered with
- **Authentication**: things can figure out that you are who you claim to be
- **Availability**: things are useable (for users who are supposed to be able to)

# Confidentiality: Example 1



Alice

Leftover Food in HH 218



Bob



Eve



# Confidentiality: Example 1

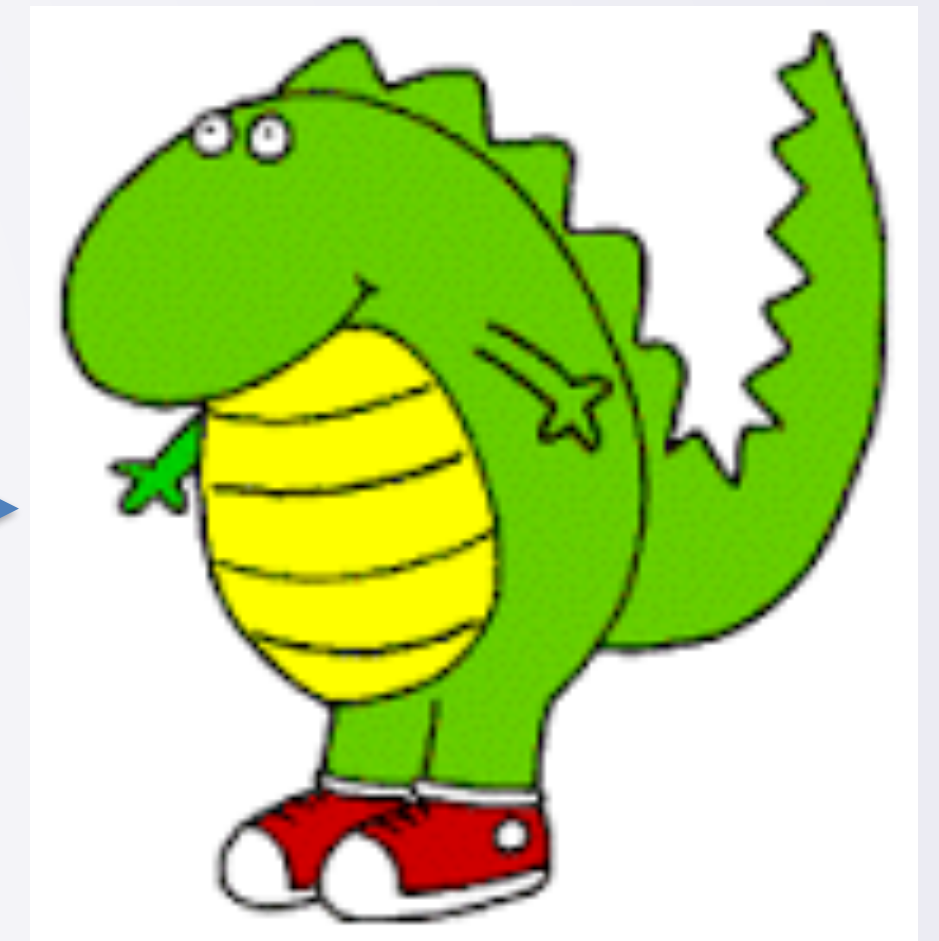
$f(\text{Leftover Food in HH 218})$   
= Al481manj417a@#1naL

$f^{-1}(\text{Al481manj417a@#1naL})$   
= Leftover Food in HH 218



Alice

Al481manj417a@#1naL



Bob



Eve

??

# Confidentiality: Example 2

```
[eve@linux] $ cat ~alice/secret.txt  
ls: /home/alice/secret.txt : Permission denied  
[eve@linux] $
```





# Integrity: Example 1



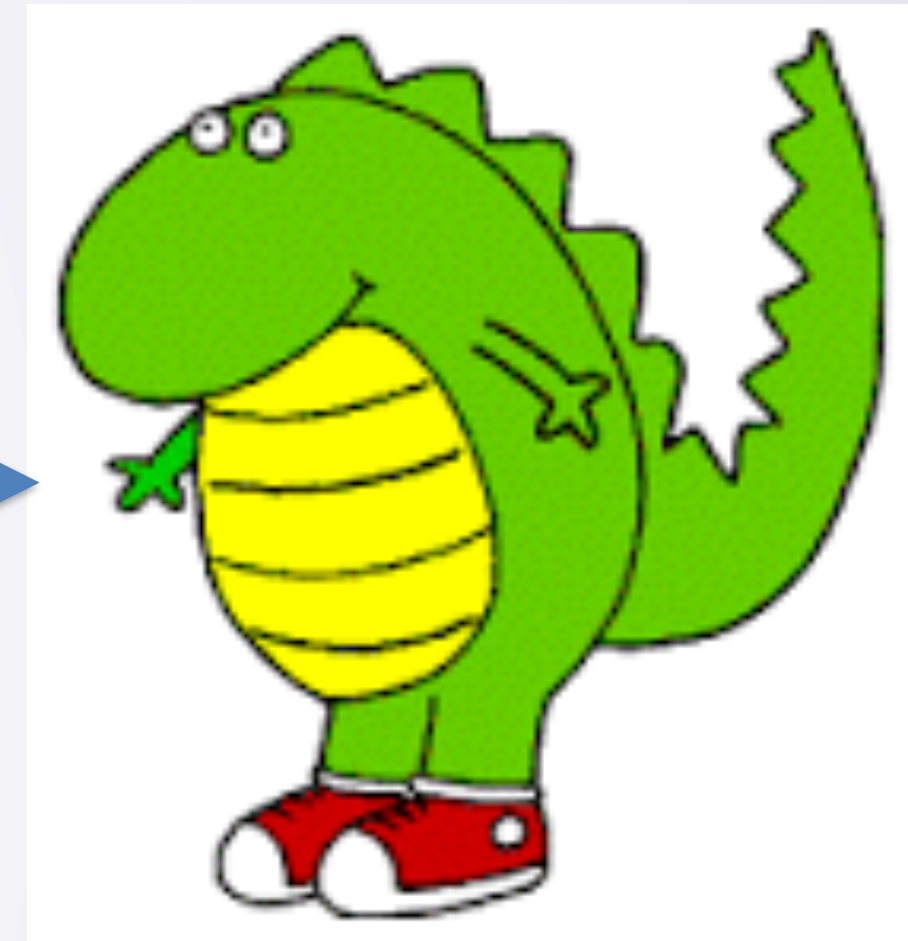
Alice

Please send \$1000  
to account 123456  
Thanks, Alice



Eve

Please send \$1000  
to account 467129  
Thanks, Alice

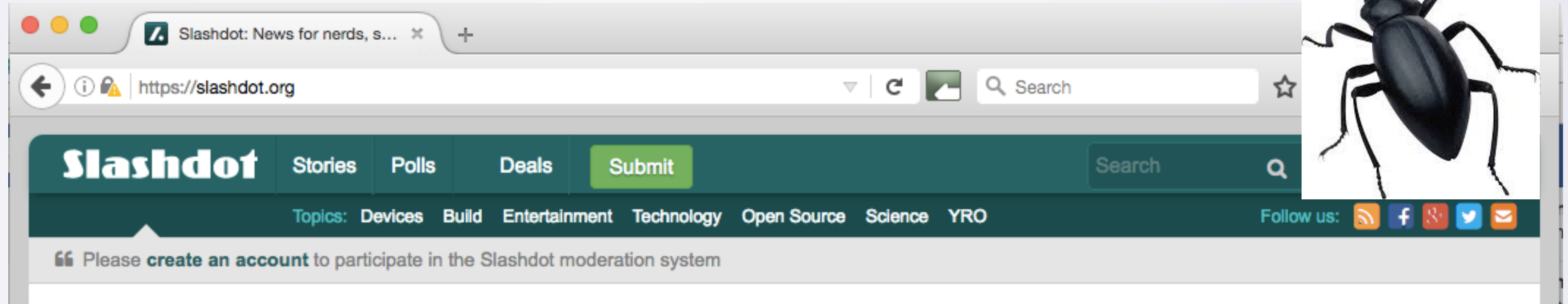


Bob

# Integrity: Example 2



Alice



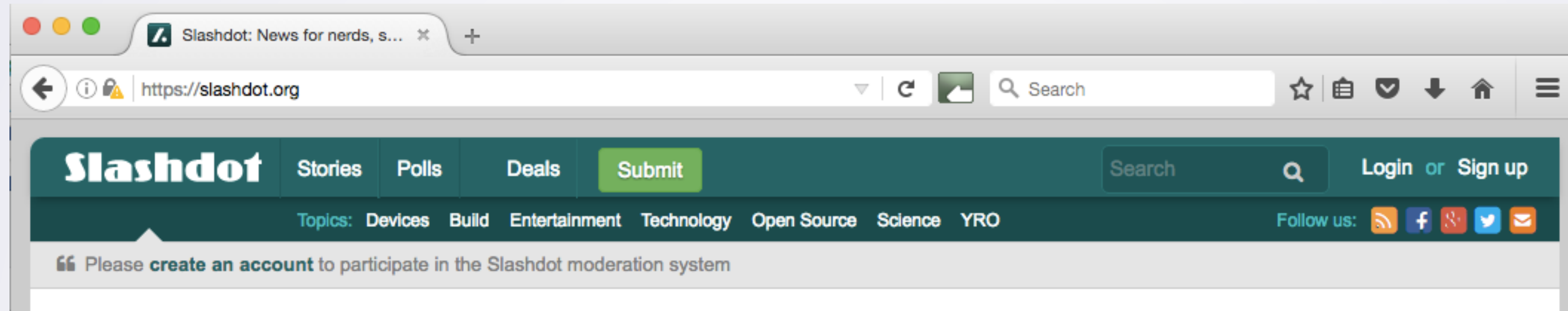
Eve's awesome software services! 20% off. This week only!



Eve



# Security Difficulty: Hard To Detect Compromises



Alice

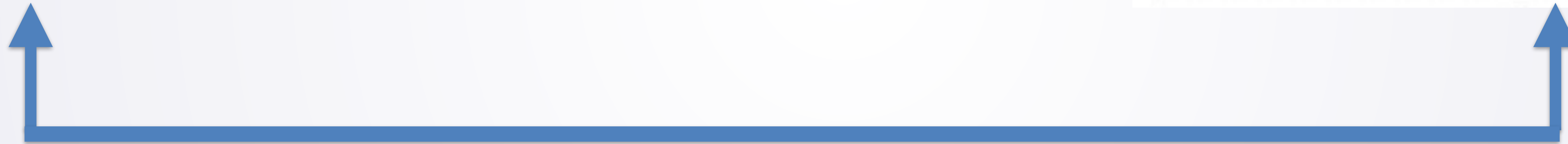
Is my browser hacked??



# Confidentiality + Integrity

Physical attacks:

- End of semester: Tami will talk about SGX



# Authentication

Hi I'm Alice

Please buy 1000 shares  
of foobar corp with the  
money in my account



Wait... Are you really Alice?  
...Prove it!



# Authentication

- Most common: **password**
  - How good are your passwords?
- Two factor:
  - Call/text/whatever device you have
- Cryptographic keys
  - ssh keys

# Speaking of Authentication...



Alice

username: alice  
password: m\$1iKa^PQ1t#aRn7



- What would happen if Alice sent her password cleartext?

# Speaking of Authentication...



Alice

username: alice  
password: m\$1iKa^PQ1t#aRn7



Eve

Now I know Alice's password!



# Speaking of Authentication...



Alice

(encrypted username/password)



- Ok, so Alice encrypts username/password...

# Speaking of Authentication...



Alice

(encrypted username/password)



Username	Password
alice	m\$1iKa^PQ1t#aRn7
bob	lWi8@P(~qtY3oR
eve	,\$ql)Pq>4iQldV7

- VERY BAD: DO NOT DO
  - Server stores password in plaintext

# Speaking of Authentication...



Alice

(encrypted username/password)



Username	Password
alice	m\$1iKa^PQ1t#aRn7
bob	lWi8@P(~qtY3oR
eve	,\$ql)Pq>4iQldV7

- **VERY BAD: DO NOT DO**
  - Server stores password in plain



Sweet! Now I have everyone's password... Maybe they use them on other sites too!....



# Speaking of Authentication...



Alice

(encrypted username/password)



Username	PasswdHash
alice	A1C399F501AB5432
bob	F09485ACB154A
eve	154FABC9F523C0

- Better, but still wrong
  - Server stores hashes: computes hash(password) + compares to table

# Speaking of Authentication...



Alice

(encrypted username/password)



Username	PasswdHash
alice	A1C399F501AB5432
bob	F09485ACB154A
eve	154FABC9F523C0

Now I have hashes... What can I do with them?

- Better, but still wrong
  - Server stores hashes: computes hash(password) + compares to table



# Eve: Breaks Hashes (All at Once)

```
Len = 1
while (1) {
    for each string s of length Len{
        int h = hash(s)
        users = mapOfStolenData.lookup(h);
        if (users is not empty) {
            print users + "password =" + s
        }
    }
    Length ++;
}
```





# Difficulty to Crack?

- Generally 95 possible characters
- Number of possible strings for a given length:

Length	Num Strings ( $95^L$ )	Time if 500K hash/sec
4	81 M	2 minutes
6	735 B	17 days
8	6E+15	400 years
10	6E+19	3M years

Is 8 characters safe?



# Difficulty to Crack?

- Generally 95 possible characters
- Number of possible strings for a given length:

Length	Num Strings ( $95^L$ )	Time if 500K hash/sec	Time if 350B hash/sec
4	81 M	2 minutes	< 1 sec
6	735 B	17 days	2 seconds
8	6E+15	400 years	5 hours
10	6E+19	3M years	5 years

I just bought  
25 GPUs..



# Password Cracking

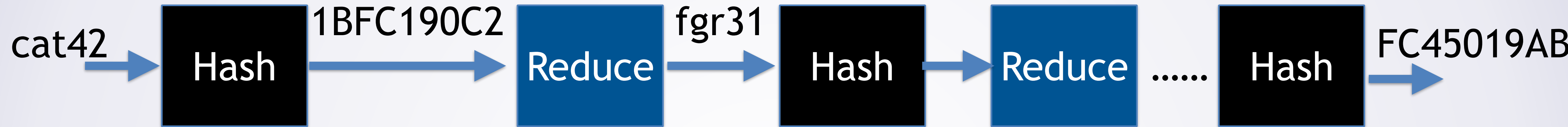
- That analysis is for
  - Every possible password (every combination)
  - Gets ALL stolen hashes at once
    - 10,000 users
    - 5 hours: every password of length 8 (thousands of passwords)
- Can probably speed up by using common passwords
  - password
  - 1234
  - ...



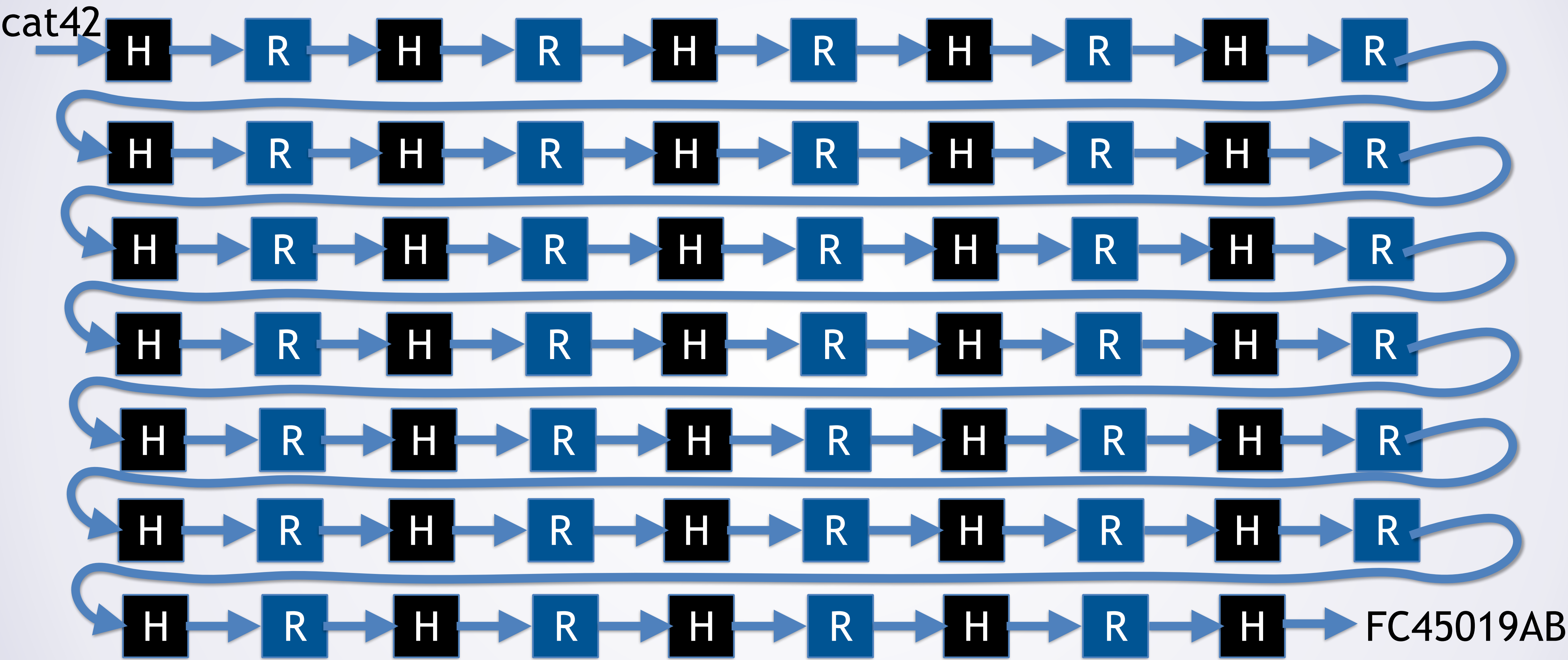
# Password Cracking (cont'd)

- Can also trade **time** for **space**
  - Execution time/memory (or disk) requirement tradeoff
- Option 1 build map hash  $\rightarrow$  password (e.g., before stealing hashes)
  - $95^6 * 16 \text{ bytes/entry} \approx 10 \text{ TB}$  [HDD: costs about \$500]
  - $95^8 * 16 \text{ bytes/entry} \approx 96,540 \text{ TB}$  [seems expensive]
- Option in the middle?
  - Pre-compute some things
  - Make attack faster
  - Do not require so much storage?

# Rainbow Tables

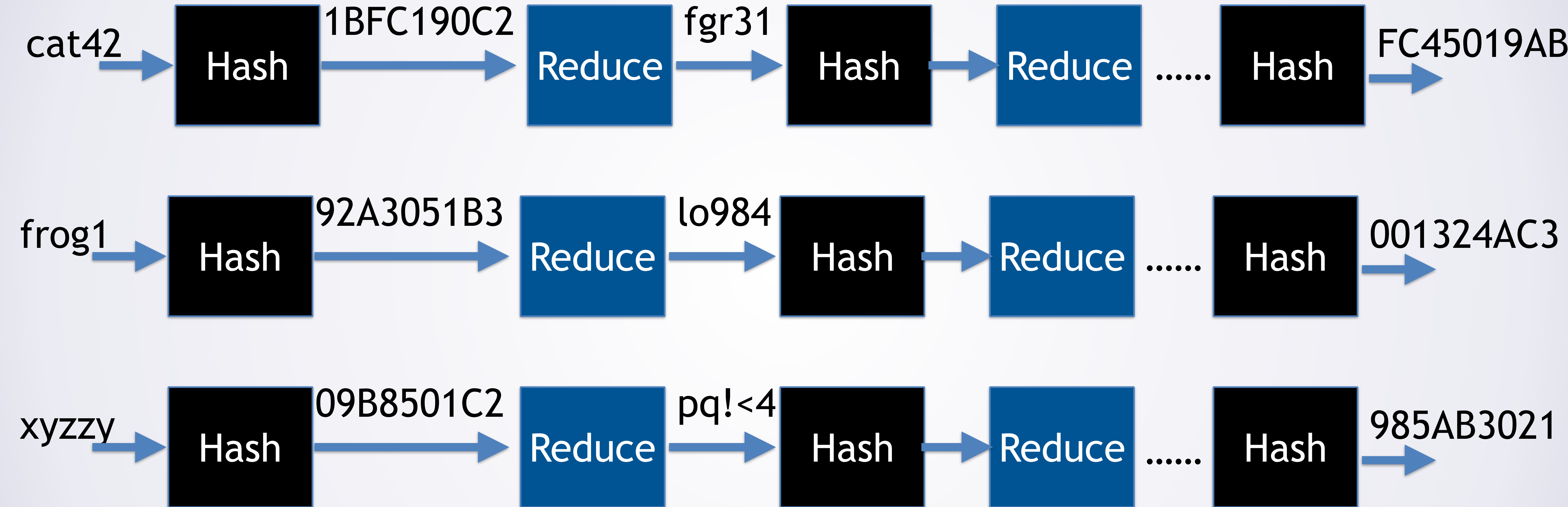


# Rainbow Tables





# Rainbow Tables



# Rainbow Tables

cat42	FC45019AB
frog1	001324AC3
xyzzz	985AB3021

# Rainbow Tables

## Rainbow Table

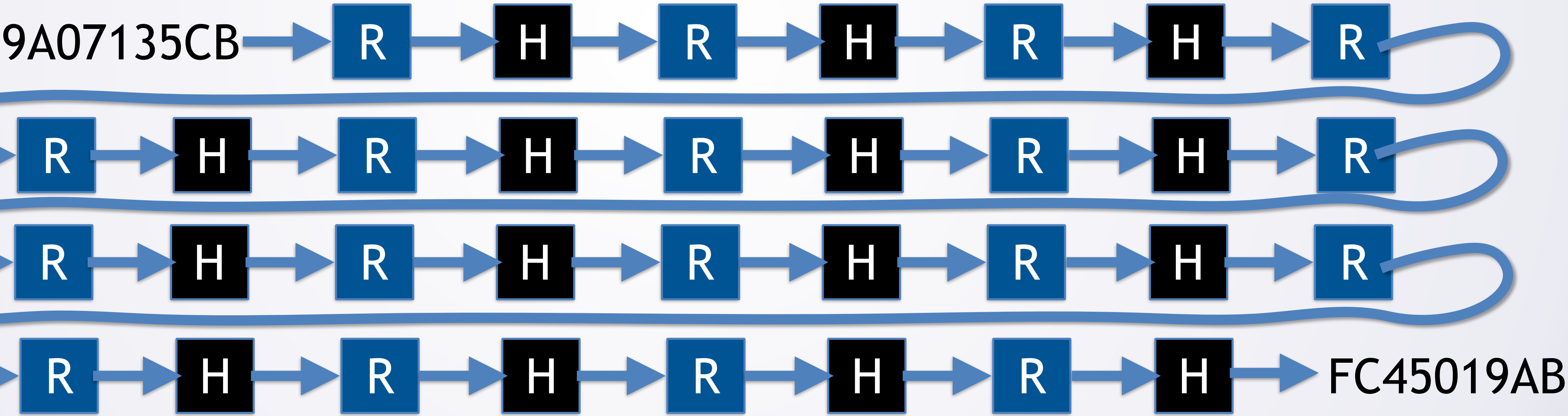
cat42	FC45019AB
frog1	001324AC3
xyzyz	985AB3021

in table?

No: keep going

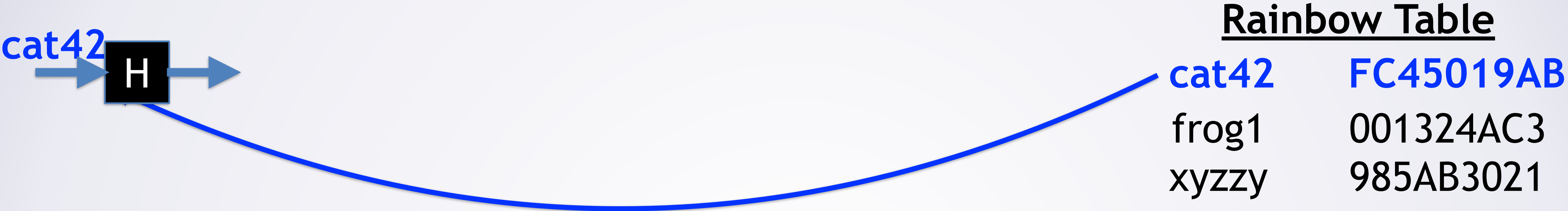
Hashed Password:

157645A39

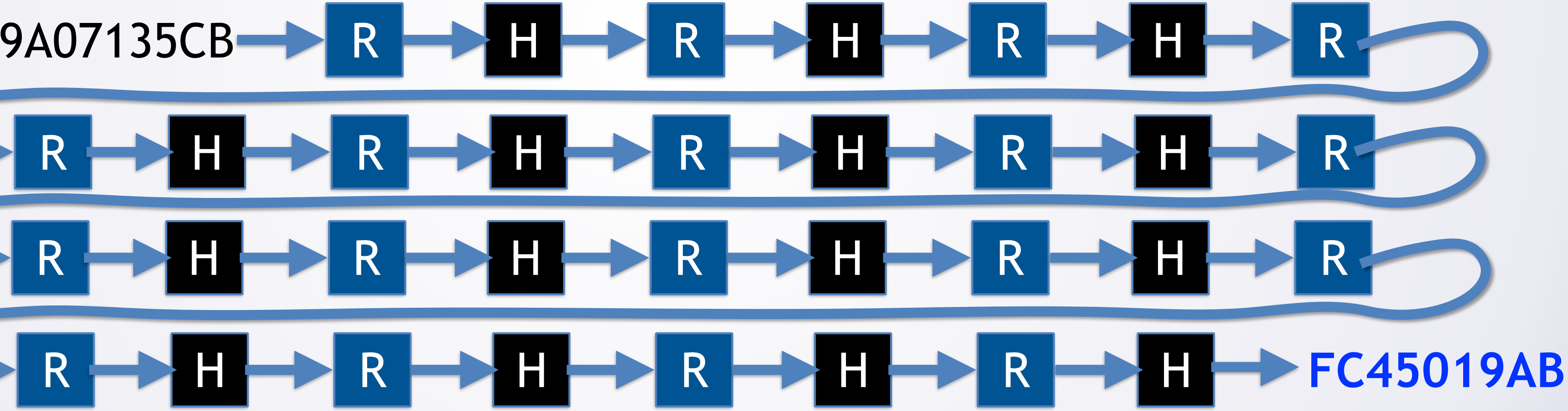




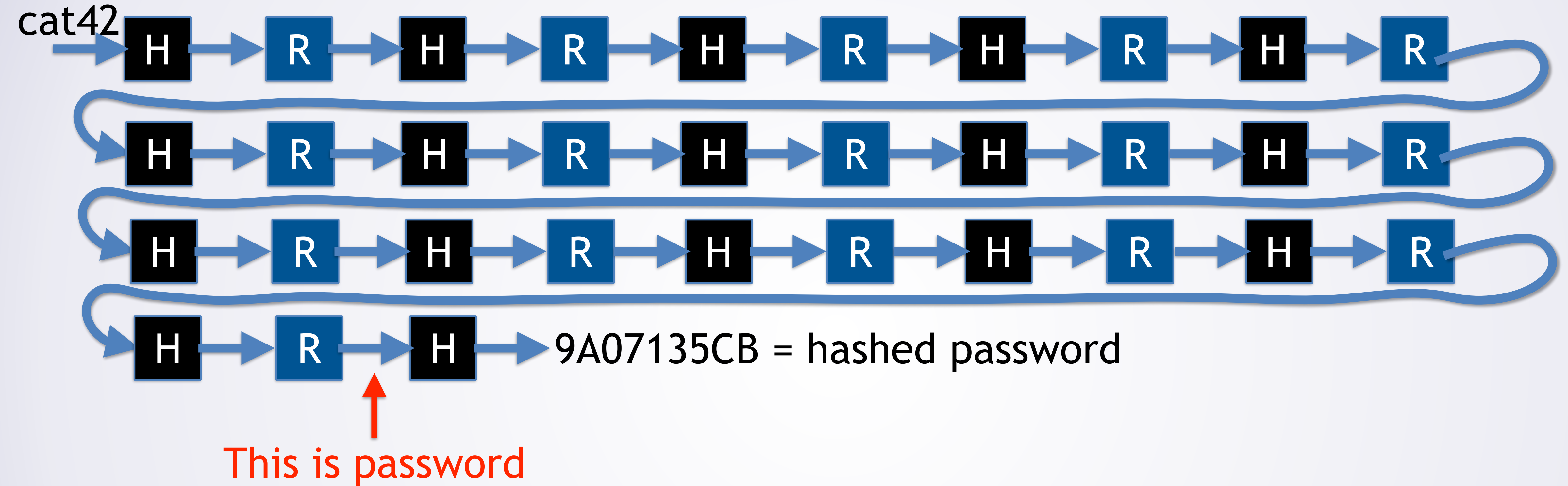
# Rainbow Tables



Hashed Password:



# Rainbow Tables



# Space vs Time

- Full map
  - $95^6 * 16 \text{ bytes/entry} \approx 10 \text{ TB}$  [HDD: costs about \$500]
  - $95^8 * 16 \text{ bytes/entry} \approx 96,540 \text{ TB}$  [seems expensive]
- Rainbow table (w/ 1B hashes/chain):
  - $95^6 * 16 \text{ bytes/entry} \approx 10 \text{ KB}$  [fits in L1 cache]
  - $95^8 * 16 \text{ bytes/entry} \approx 96 \text{ MB}$  [fits in RAM]
  - $95^{10} * 16 \text{ bytes/entry} \approx 830 \text{ GB}$  [cheap hard disk]

**Important Lesson: HASHING IS NOT ENOUGH**



# Speaking of Authentication...



Alice

(encrypted username/password)



Username	PasswdHash	Salt
alice	A1C399F501AB5432	1A45FB9C072BC90A
bob	F09485ACB154A	9841ABCD416790
eve	154FABC9F523C0	FAB981230CDBEA

- **Correct (assuming we get everything else right)**
  - Server stores hashes + **salt**: computes hash(password, **salt**)

# Speaking of Authentication...

Username	PasswdHash	Salt
alice	A1C399F501AB5432	1A45FB9C072BC9
bob	F09485ACB154A	9841ABCD416790
eve	154FABC9F523C0	FAB981230CDBEA



To crack Alice's password: try various combinations of strings (s)  
hash (s, 0x1A45FB9C072BC9)

To crack Bob's password: try various combinations of strings (s)  
hash (s, 0x9841ABCD416790)

# What Does Salt Get Us?

- Pre-computation is ineffective
  - Build a map for each possible salt?
    - 64 bit salt-> will take forever + be huge
  - Rainbow tables?
    - Still need rainbow table for each salt
    - Expensive to build/store
- Crack each user's password separately
  - Rather than in parallel
  - Slowdown factor of number of users



# What Does Salt Get Us?

Now this analysis is for **ONE** user's password (not all at once)  
Multiply by number of user's to do them all...



Length	Num Strings (95 <sup>L</sup> )	Time if 500K hash/sec	Time if 350B hash/sec
4	81 M	2 minutes	< 1 sec
6	735 B	17 days	2 seconds
8	6E+15	400 years	5 hours
10	6E+19	3M years	5 years

# Speaking of Authentication...

- **Correct (assuming we get everything else right)**
  - Server stores hashes + **salt**: computes `hash(password, salt)`

# Speaking of Authentication...

- What else do we need to get right?
  - Sufficiently long ( $\geq 64$  bits), **random** salt
  - Secure hash: **SHA-2 or SHA-3**
    - NOT **MD5, SHA-0, or SHA-1**
  - Use **key stretching** algorithm
    - E.g., PBKDF2 (also popular: bcrypt)



# Key Stretching

- Do we want hashing algorithm to be **slow** or **fast**?

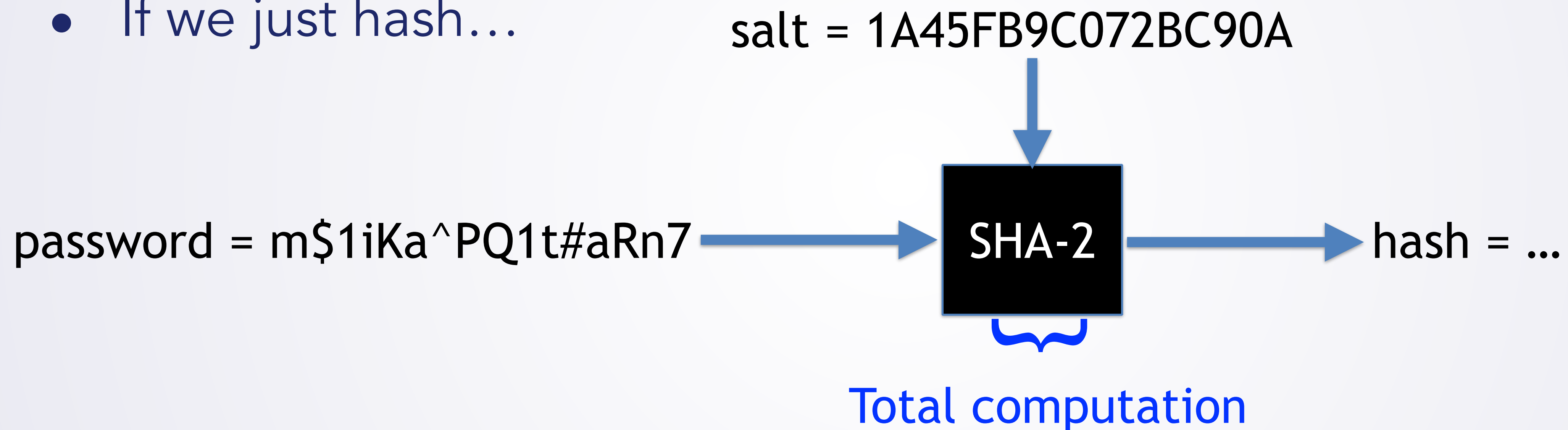
Length	Num Strings ( $95^L$ )	Time if 500K hash/sec	Time if 350B hash/sec
4	81 M	2 minutes	< 1 sec
6	735 B	17 days	2 seconds
8	6E+15	400 years	5 hours
10	6E+19	3M years	5 years

Slow :)

Fast :(

# Key Stretching

- Do we want hashing algorithm to be **slow** or **fast**?
  - Want attackers to have to spend more work to break hashes
- If we just hash...



# Key Stretching

salt = 1A45FB9C072BC90A50C30000

- PKBDF2

password = m\$1iKa^PQ1t#aRn7

password = m\$1iKa^PQ1t#aRn7

password = m\$1iKa^PQ1t#aRn7

password = m\$1iKa^PQ1t#aRn7

SHA-2

SHA-2

SHA-2

SHA-2

XOR

XOR

XOR

Total computation



# Speaking of Authentication...

- What else do we need to get right?
  - Sufficiently long ( $\geq 64$  bits), **random** salt
  - Secure hash: **SHA-2 or SHA-3**
    - NOT **MD5, SHA-0, or SHA-1**
  - Use **key stretching** algorithm
    - E.g., PBKDF2 (also popular: bcrypt)
  - Do NOT screw up and weaken things:
    - <https://arstechnica.com/security/2015/09/once-seen-as-bulletproof-11-million-ashley-madison-passwords-already-cracked/>
- Do NOT try to invent things yourself!

# Use Libraries That Do It Right

- Hashing in C/C++?
  - Use libssl
- Hashing in python?
  - Use hashlib: `hashlib.pbkdf2_hmac('sha512', pwd, salt, itrs)`
- Authentication in Django?
  - <https://docs.djangoproject.com/en/1.10/topics/auth/passwords/>

# Wrap Up

- Intro to security:
  - Confidentiality
  - Integrity
  - Authentication
    - Much discuss of password safety
  - Availability
- Next time:
  - Cryptography (Tami)