Today in Cryptography (5830)

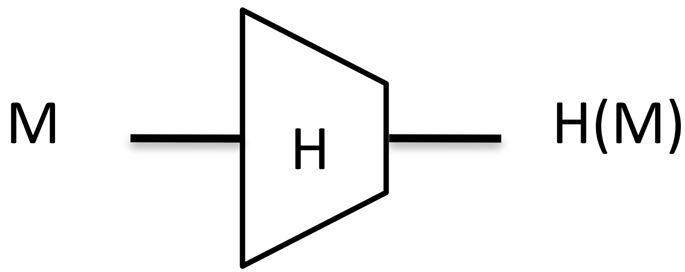
Hash functions
HMAC
Passwords and password-based key derivation

Where we are at

- Authenticated encryption
  - Symmetric encryption providing confidentiality and integrity
  - Security in face of active attackers
  - Uses message authentication codes as cryptographically strong error detection
    - We saw CBC-MAC, built from blockcipher
- Today: cryptographic hash functions
  - Used to build MACs, used many other places
  - “Swiss army knife” of cryptography

Cryptographic hash functions

A function H that maps arbitrary bit string to fixed length string of size n



MD5:	n = 128 bits
SHA-1:	n = 160 bits
SHA-256:	n = 256 bits

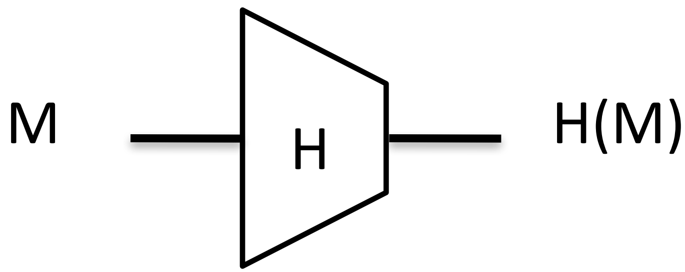
Many security goals asked of hash functions. Ideally, they behave as if they were a (public) random function.

Applications of hashing

- File comparison
- Digital signatures (coming up later)
- Password hashing
- Message authentication codes
- Key derivation

Cryptographic hash functions

A function H that maps arbitrary bit string to fixed length string of size n



MD5: n = 128 bits
SHA-1: n = 160 bits
SHA-256: n = 256 bits

Collision resistance:

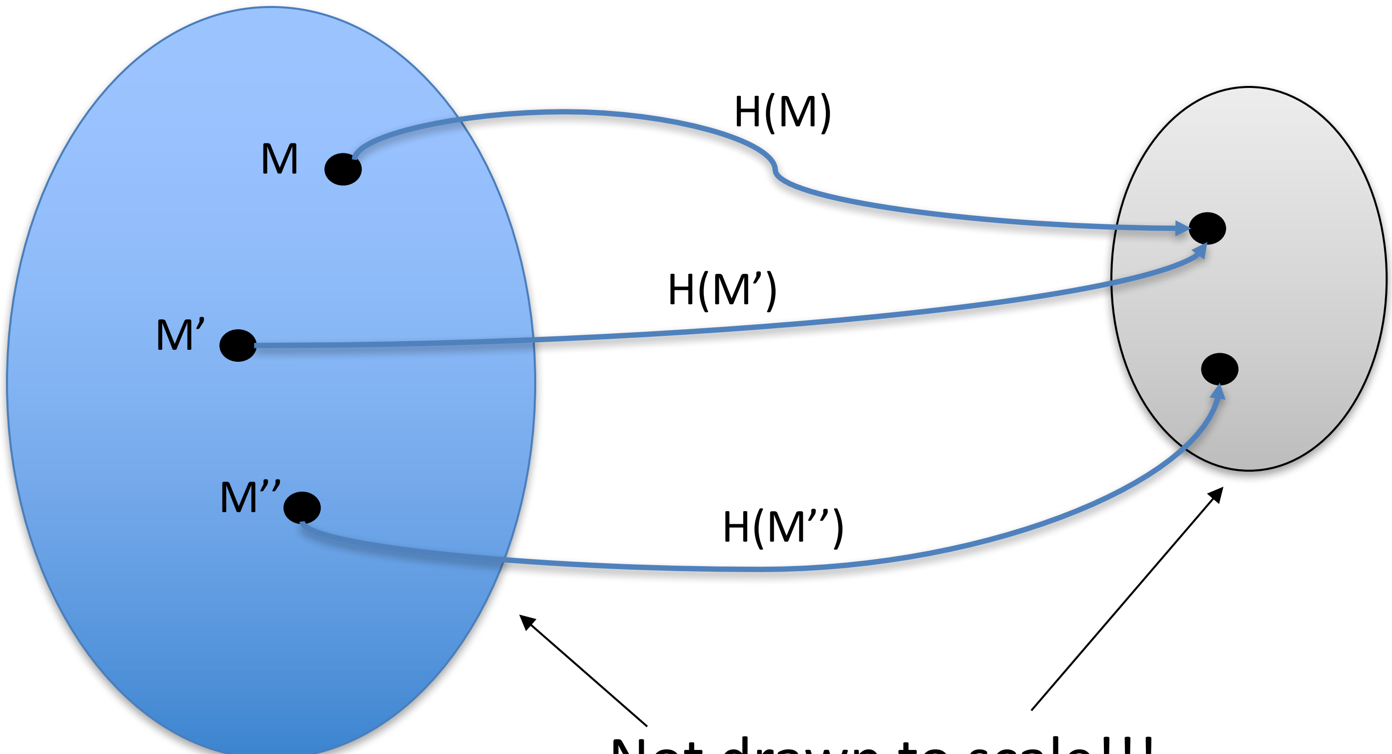
No computationally efficient adversary can find

$M \neq M'$ such that $H(M) = H(M')$

Collisions always exist

Domain (usually all strings up to some length)
SHA-1: up to length 2^{64}-1

Range $\{0,1\}^n$

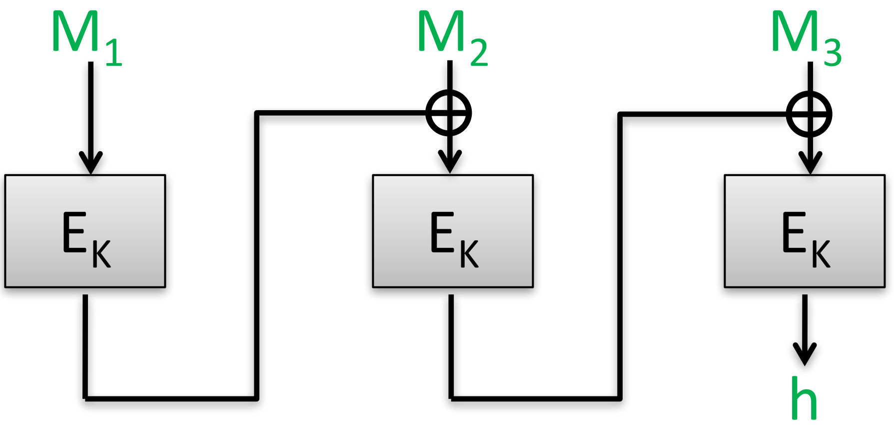


Not drawn to scale!!!

Pigeonhole principle:
size of domain larger than size of range implies there must be collisions

CBC-MAC as CR hash?

Key was secret in CBC-MAC.
But hash functions are publicly computable.
One idea is to use a random, **public K value known to attacker**



How do we *efficiently* find collisions?

<u>Adversary A(K):</u>
h <- CBC-MAC(0^n)
M_2 <- D_K(h) $\oplus$ E_K(1^n)
Return (0^n, 1^n||M_2)

Birthday attacks

- What is best possible security achievable by hash function with output length n bits?

- Answer: security is only achievable up to $2^{n/2}$ hash computations

The birthday problem

Choose q values $Y_1,\ldots,Y_q$ from $\{0,1\}^n$ at random. What is probability that two are the same?

Let $Coll_i$ be event that $Y_i = Y_j$ for some $j < i$

$$\begin{aligned}\Pr[Coll] &= \Pr[Coll_1 \vee \ldots \vee Coll_q] \\ &\leq \Pr[Coll_1] + \ldots + \Pr[Coll_q] \\ &= \frac{0}{2^n} + \frac{1}{2^n} + \frac{2}{2^n} + \ldots + \frac{q}{2^n} \\ &= \frac{q(q-1)}{2^n}\end{aligned}$$

Another proof shows that if $q \leq 2^{(n+1)/2}$

$$\Pr[Coll] \geq \frac{0.3 \cdot q(q-1)}{2^n}$$

The birthday attack

Let m be some length in domain of hash function H

<u>Adversary A:</u>

For i = 1 to q do:

X_i <-\$ $\{0,1\}^m$

h_i <- $H(X_i)$

If exists i,j s.t. $X_i \neq X_j$ and $h_i = h_j$ then

Return (X_i, X_j)

Return fail

Same # of domain points map to each range point

If H is *regular* then probability of success is exactly birthday probability

$$\Pr[\text{A finds collision}] \geq \frac{0.3 \cdot q(q-1)}{2^n}$$

Birthday attack run times

MD5:	$n = 128$ bits	2^{64}	MD5 computations
SHA-1:	$n = 160$ bits	2^{80}	SHA-1 computations
SHA-256:	$n = 256$ bits	2^{128}	SHA-2 computations

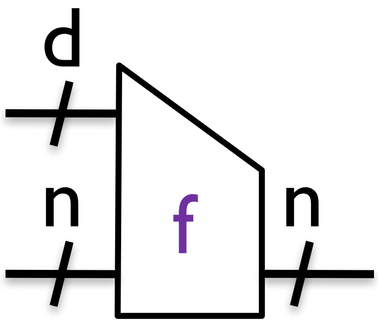
2^{64} too small by today's standards!

Bitcoin network computes about 2^{64} SHA-256 hashes ***per second***

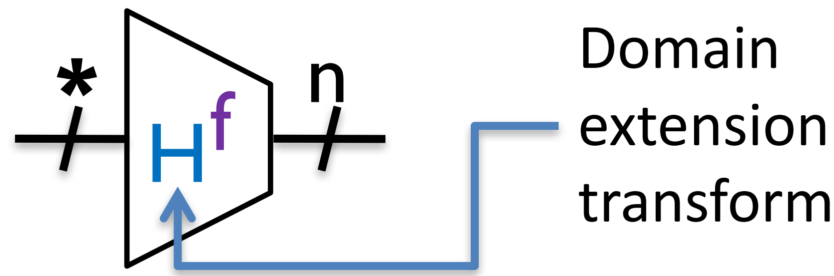
https://blockchain.info/charts/hash-rate

Two-step design for hash functions

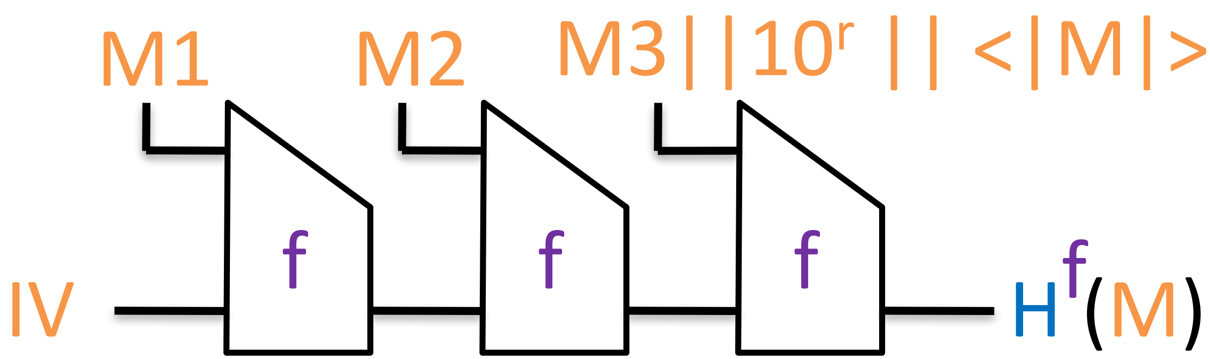
Compression Function



Hash Function



Domain extension called "Merkle-Damgard with strengthening"



Used in MD-x, SHA-1, SHA-256, …

IV is a fixed constant. Not randomly chosen.

Building compression functions

- Can build compression functions from suitable block ciphers

$$f(z,m) = E(m,z) \oplus z$$

Called Davies-Meyer construction

- Can use AES, but security too low. Why?
- SHA-1 uses custom E with k = d = 512 and n = 160
  - Message block length of SHA-1 is 512 bits

SHA-1 compression function

Expand 512-bit message into

$W_1,\ldots,W_{80}$ strings of length 32 bit values

(Think of this as "key schedule")

Chaining variable is 160 bits, 5 32-bit values

A , B , C , D , E

F(B,C,D) function that changes over rounds:

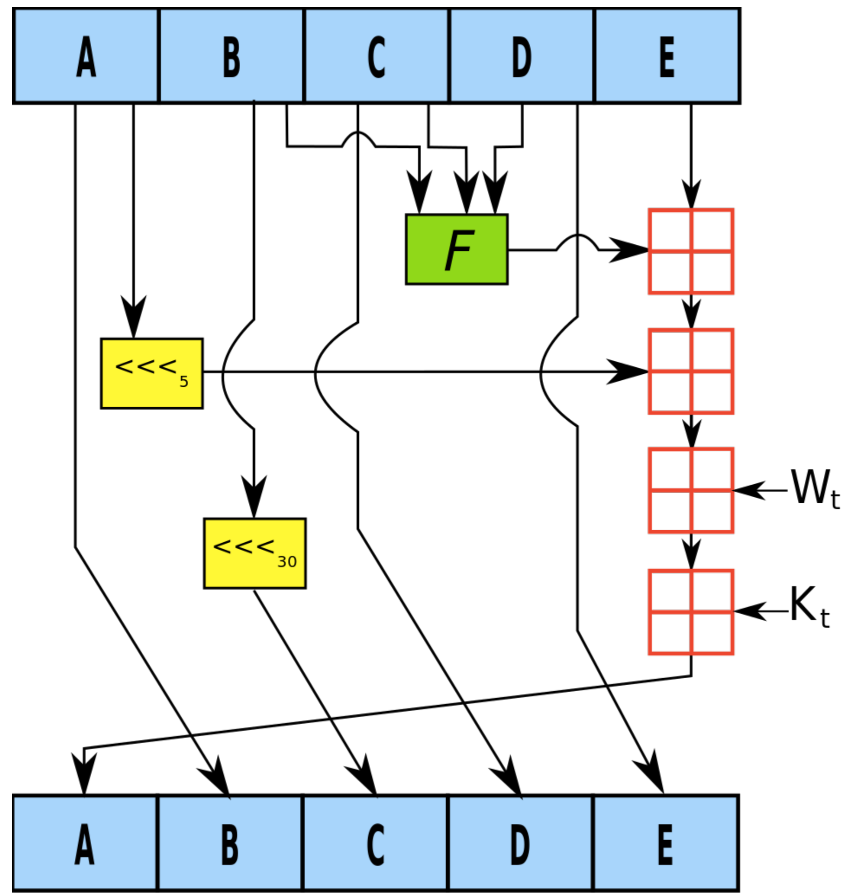
0-19: (B **and** C) **or** ((**not** B) **and** D)

20-39: B **xor** C **xor** D

40-59: (B **and** C) **or** (B **and** C) **or** (C **and** D)

60-79: B **xor** C **xor** D

Constants K_1 , … , K_{80} differ across rounds



Faster attacks than birthday?

- 2004: Full break of MD5 announced by Xiaoyun Wang and co-authors
  - MD5 is easy to break now. You can download programs to do it on your laptop
- 2005: Announced faster than 2^{80} attack against SHA-1 by Wang et al.
  - Not practical to run (2^{69} estimated cost)
- 2017: CWI and Google announce first demonstrated collision

SHAttered attack

Chosen prefix P. Find two pairs of message blocks
M_1, M_2 and M_1', M_2' such that for any suffix S:

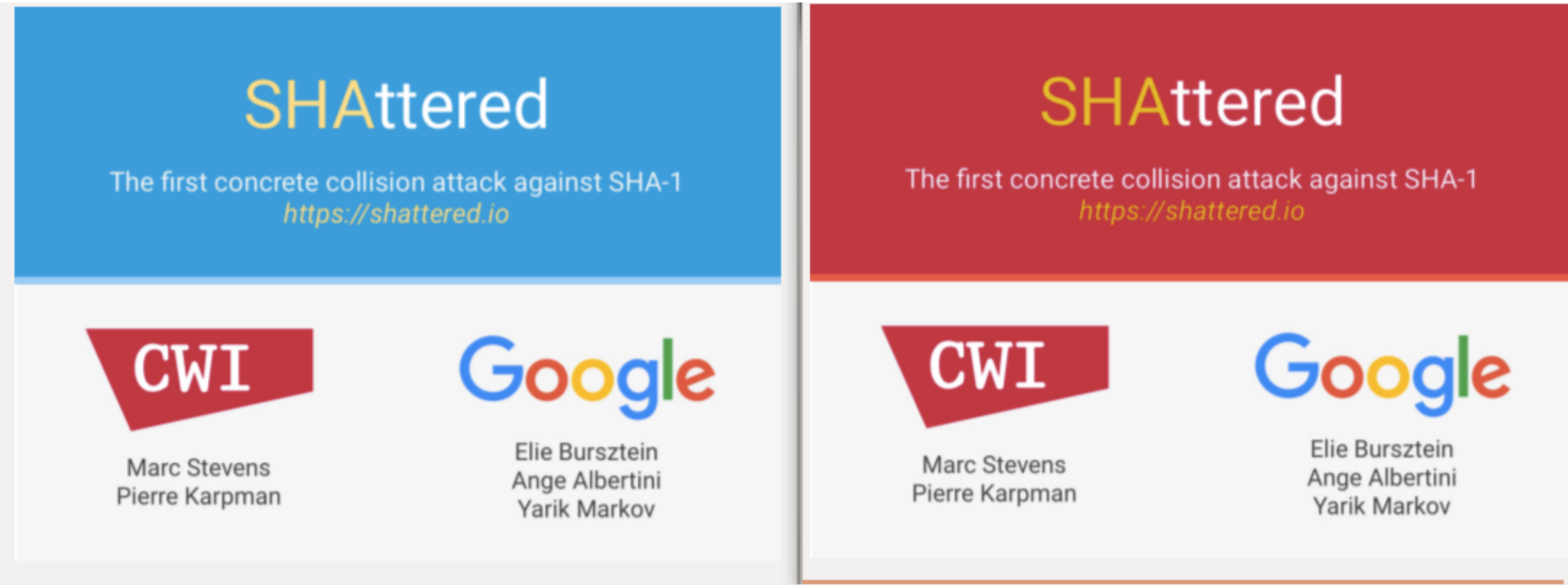
$$\text{SHA-1}(P \,||\, M_1 \,||\, M_2 \,||\, S) = \text{SHA-1}(P \,||\, M_1' \,||\, M_2' \,||\, S)$$

Referred to as a *identical-prefix collision attack*

How? Pick P, find M_1 and M_1' that form near-collision on chaining variable. Then complete collision by finding M_2 and M_2'

They show how to extend to build colliding PDF files

SHAttered attack



Required $2^{63.1}$ SHA-1 compression function applications
100,000x faster than birthday attack (2^{80})

Fallout of attack

SVN repositories can be broken (DoS attack)

- Checking in the two SHAttered PDFs corrupts repo

Linus Torvalds misunderstands security…

(to paraphrase) GIT’s ok because we can trust everyone

https://plus.google.com/+LinusTorvalds/posts/7tp2gYWQugL

Marc Stevens & Dan Shumow (Microsoft) developed *counter-cryptanalysis tool*

Way to detect if a particular file is one half of colliding pair

Deployed at several large companies

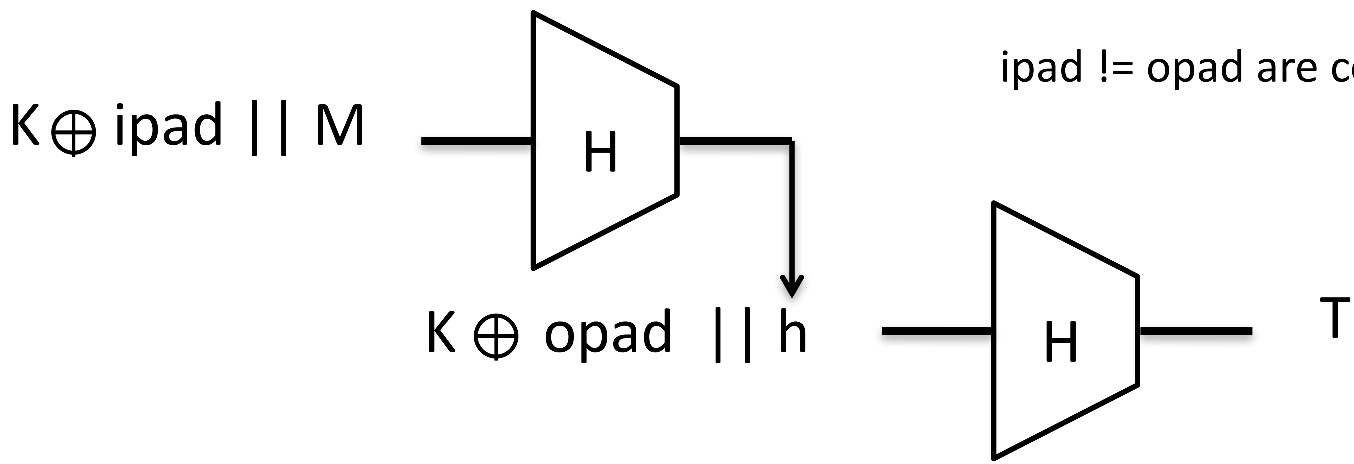
Ongoing migration away from SHA-1 to SHA-256 / SHA-3

Applications of hashing

- File comparison
- Digital signatures (coming up later)
- Password hashing
- For message authentication codes
- Key derivation

Building PRFs with hash functions: HMAC

Use a hash function H to build a MAC. K is a secret key



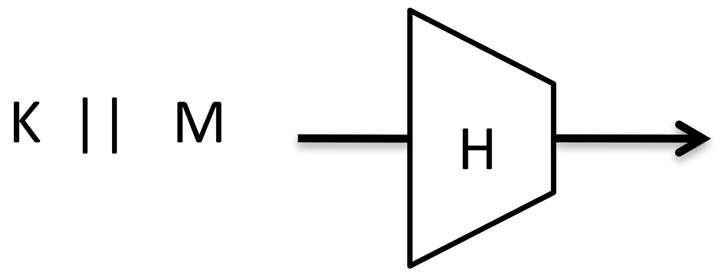
ipad != opad are constants

This is slight simplification, assuming |K| less than block length of H

HMAC-SHA-1, HMAC-SHA-256, etc.

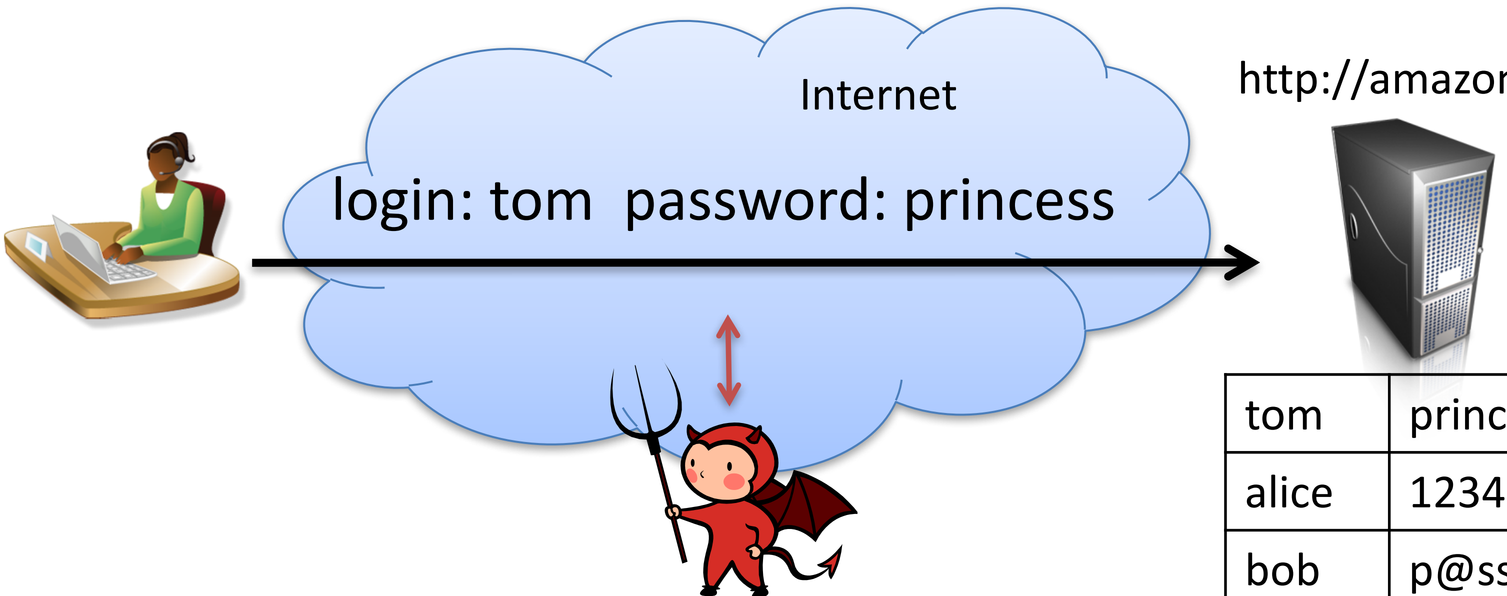
What's wrong with this PRF construction?

Assume H is a MD iterated hash function, define F(K,M) = H(K||M)



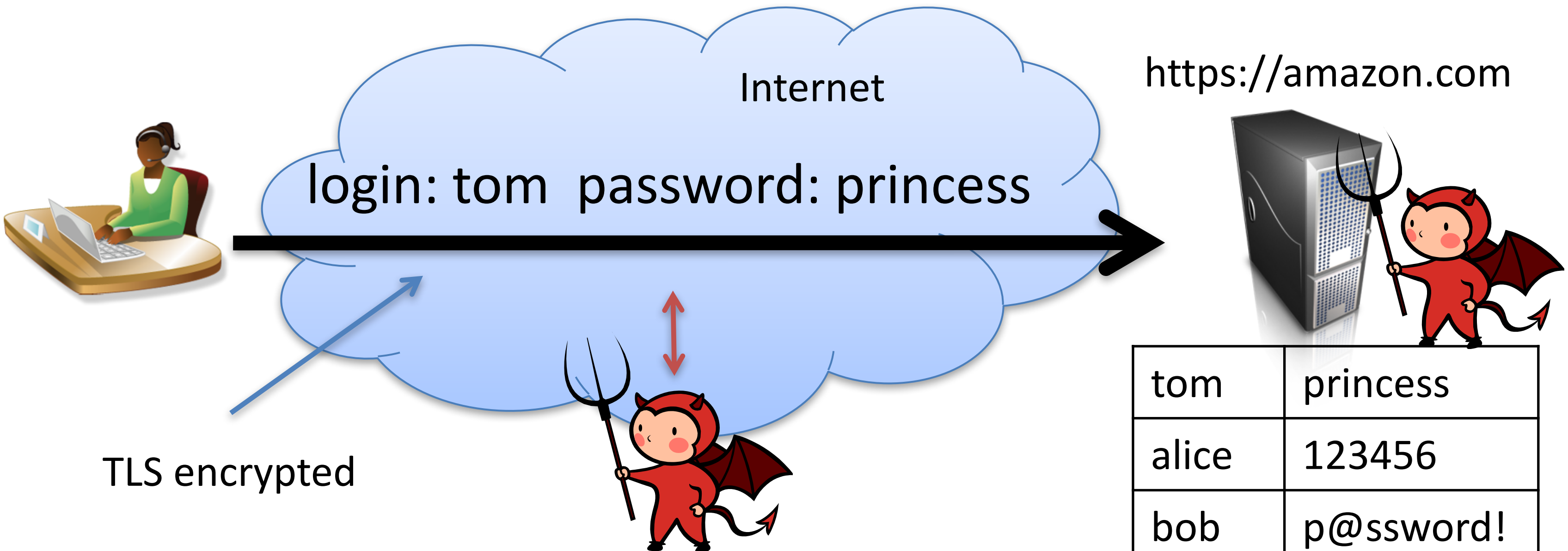
Length-extension attacks: Given F(K,M) can compute F(K,M||S) for attacker-chosen suffix S

Passwords



tom	princess
alice	123456
bob	p@ssword!

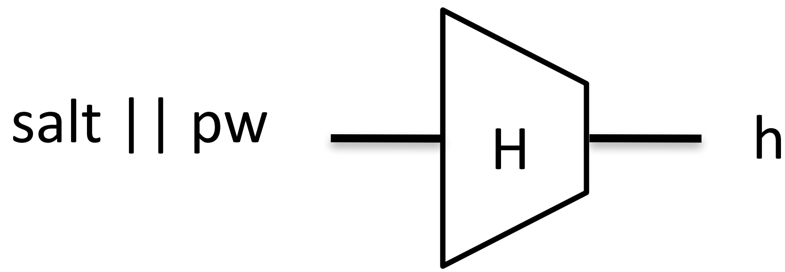
Passwords



tom	princess
alice	123456
bob	p@ssword!

Password hashing

Password hashing. Choose random salt and store (salt,h) where:



The idea: Attacker, given (salt,h), should not be able to recover pw

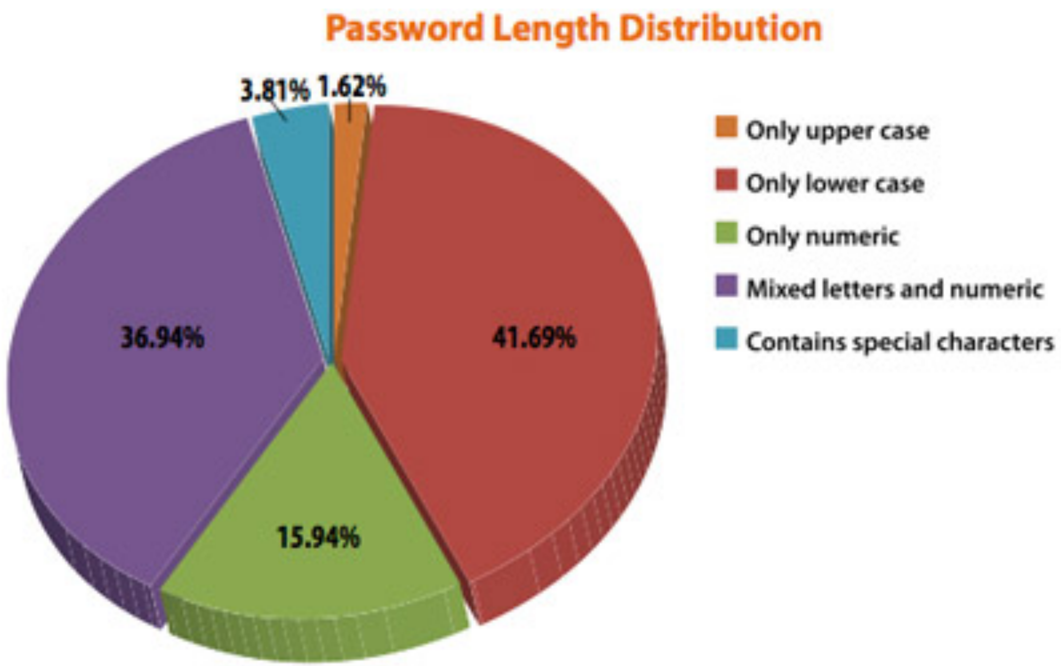
Or can they?

For each guess pw':
If H(salt||pw') = h then
Ret pw'

Rainbow tables speed this up in practice by way of precompution. Large salts make rainbow tables impractical

Rank	Password	Number of Users with Password (absolute)
1	123456	290731
2	12345	79078
3	123456789	76790
4	Password	61958
5	iloveyou	51622
6	princess	35231
7	rockyou	22588
8	1234567	21726
9	12345678	20553
10	abc123	17542

Rank	Password	Number of Users with Password (absolute)
11	Nicole	17168
12	Daniel	16409
13	babygirl	16094
14	monkey	15294
15	Jessica	15162
16	Lovely	14950
17	michael	14898
18	Ashley	14329
19	654321	13984
20	Qwerty	13856



From an Imperva study of released RockMe.com password database 2010

```
rist@seclab-laptop1:~/work/teaching/642-fall-2011/slides$ openssl speed sha1
Doing sha1 for 3s on 16 size blocks: 4109047 sha1's in 3.00s
Doing sha1 for 3s on 64 size blocks: 3108267 sha1's in 2.99s
Doing sha1 for 3s on 256 size blocks: 1755265 sha1's in 3.00s
Doing sha1 for 3s on 1024 size blocks: 636540 sha1's in 3.00s
Doing sha1 for 3s on 8192 size blocks: 93850 sha1's in 3.00s
OpenSSL 1.0.0d 8 Feb 2011
```

```
rist@seclab-laptop1:~/work/teaching/642-fall-2011/slides$ openssl speed aes-128-
cbc
Doing aes-128 cbc for 3s on 16 size blocks: 27022606 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 64 size blocks: 6828856 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 256 size blocks: 1653364 aes-128 cbc's in 3.00s
Doing aes-128 cbc for 3s on 1024 size blocks: 438909 aes-128 cbc's in 2.99s
Doing aes-128 cbc for 3s on 8192 size blocks: 54108 aes-128 cbc's in 3.00s
OpenSSL 1.0.0d 8 Feb 2011
```

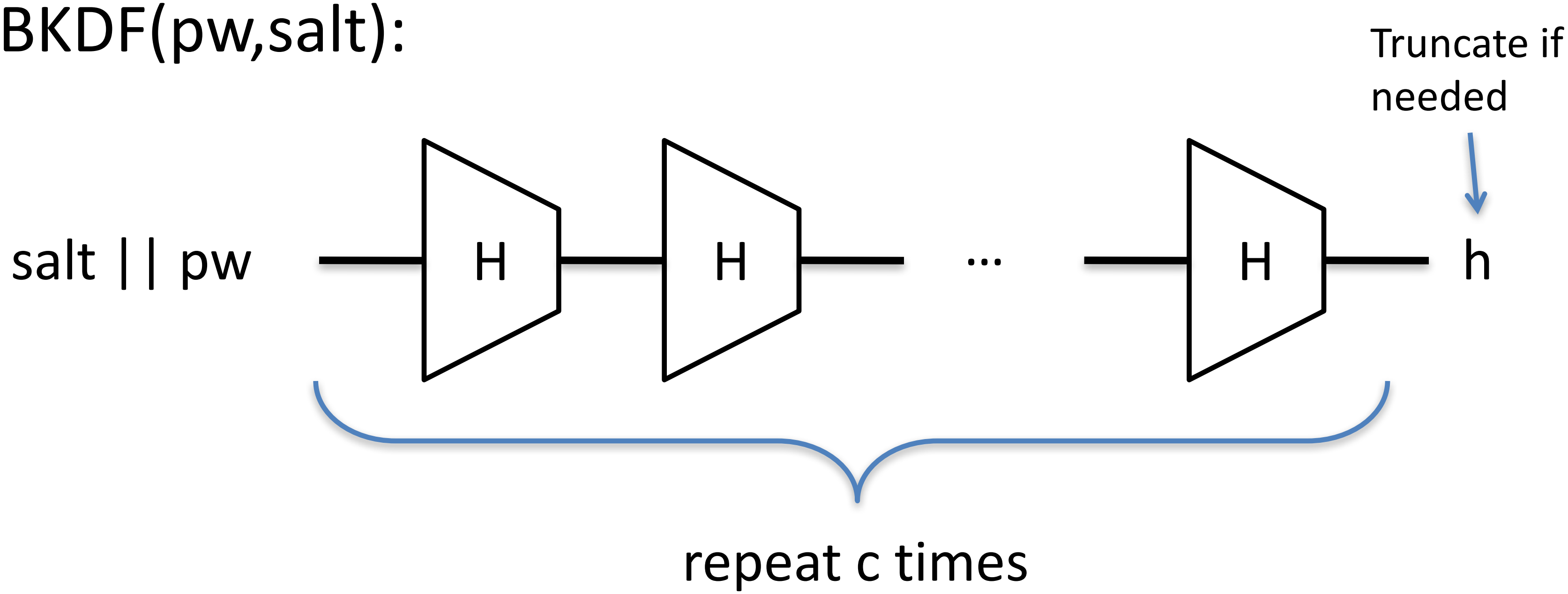
Say c = 4096. Generous back of envelope* suggests that in 1 second, can test 252 passwords and so a naïve brute-force:

6 numerical digits	10^6 = 1,000,000	~ 3968 seconds
6 lower case alphanumeric digits	36^6 = 2,176,782,336	~ 99 days
8 alphanumeric + 10 special symbols	72^8 = 722,204,136,308,736	~ 33million days

* I did the arithmetic…

Password-based Key Derivation (PBKDF)

PBKDF(pw,salt):



PKCS#5 standardizes PBKDF1 and PBKDF2, which are both hash-chain based.

Only slows down by a factor of c

scrypt, argon2: memory-hard hashing functions

Another application of PBKDFs: PW-based encryption

<u>Enc(pw,M):</u>
salt
K <- PBKDF(pw,salt)
C <- AEnc(K,M)
Return (salt,C)

<u>Dec(pw,salt||C):</u>
K <- PBKDF(pw,salt)
M <- ADec(K,C)
Return M

Here AEnc/ADec is an AE scheme
(e.g., CBC + HMAC)

Summary

- Hash functions
  - Used in a variety of applications
  - Core requirement collision resistance
- Birthday attacks break them in time $2^{n/2}$ for range size n bits
- Built from compression functions, which in turn can be viewed as block-cipher-based function
- Recent demonstration of SHA-1 collision