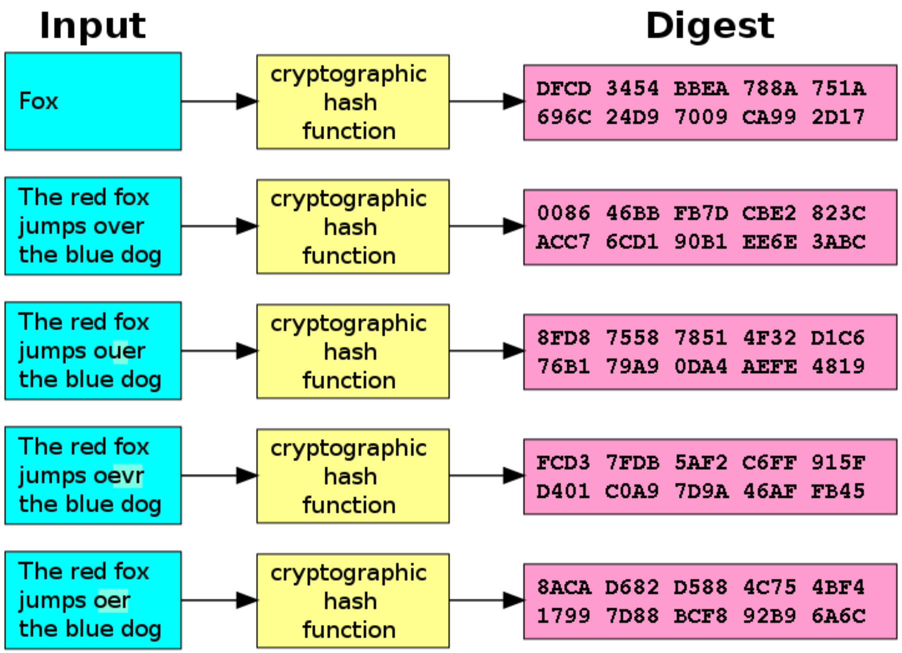
Secure hashing, authentication



Input

Digest

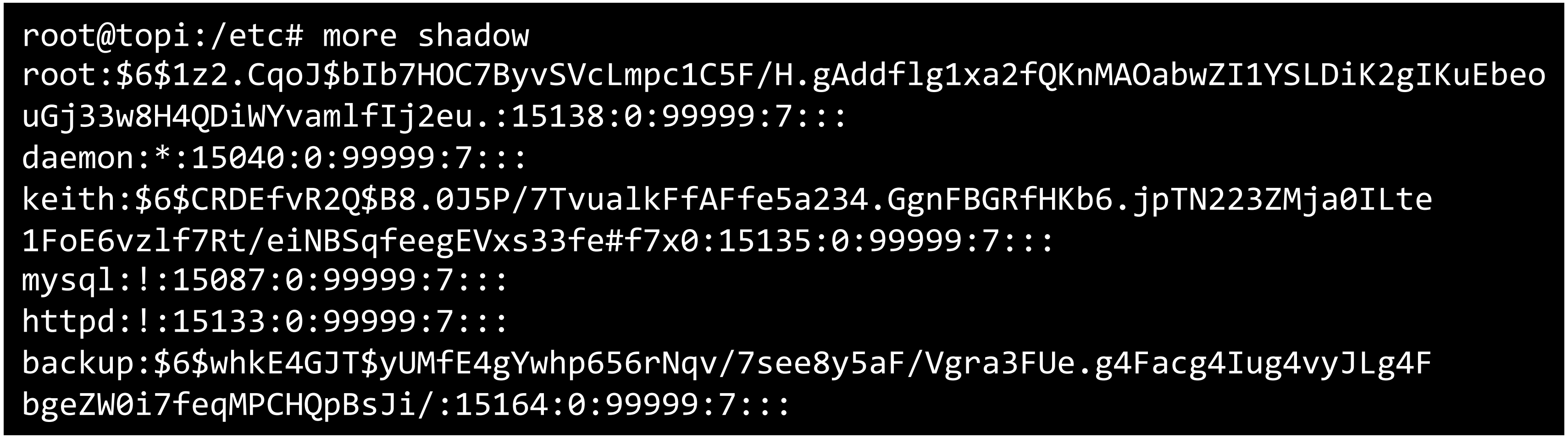
Fox → cryptographic hash function → DFCD 3454 BBEA 788A 751A 696C 24D9 7009 CA99 2D17

The red fox jumps over the blue dog → cryptographic hash function → 0086 46BB FB7D CBE2 823C ACC7 6CD1 90B1 EE6E 3ABC

The red fox jumps ouer the blue dog → cryptographic hash function → 8FD8 7558 7851 4F32 D1C6 76B1 79A9 0DA4 AEFE 4819

The red fox jumps oevr the blue dog → cryptographic hash function → FCD3 7FDB 5AF2 C6FF 915F D401 C0A9 7D9A 46AF FB45

The red fox jumps oer the blue dog → cryptographic hash function → 8ACA D682 D588 4C75 4BF4 1799 7D88 BCF8 92B9 6A6C



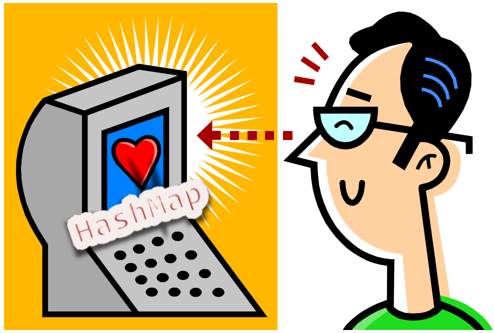
```
root@topi:/etc# more shadow
root:$6$1z2.CqoJ$bIb7HOC7ByvSVcLmpc1C5F/H.gAddflg1xa2fQKnMAOabwZI1YSLDiK2gIKuEbeo
uGj33w8H4QDiWYvamlfIj2eu.:15138:0:99999:7:::
daemon:*:15040:0:99999:7:::
keith:$6$CRDEfvR2Q$B8.0J5P/7TvualkFfAFfe5a234.GgnFBGRfHKb6.jpTN223ZMja0ILte
1FoE6vzlf7Rt/eiNBSqfeegEVxs33fe#f7x0:15135:0:99999:7:::
mysql:!:15087:0:99999:7:::
httpd:!:15133:0:99999:7:::
backup:$6$whkE4GJT$yUMfE4gYwhp656rNqv/7see8y5aF/Vgra3FUe.g4Facg4Iug4vyJLg4F
bgeZW0i7feqMPCHQpBsJi/:15164:0:99999:7:::
```

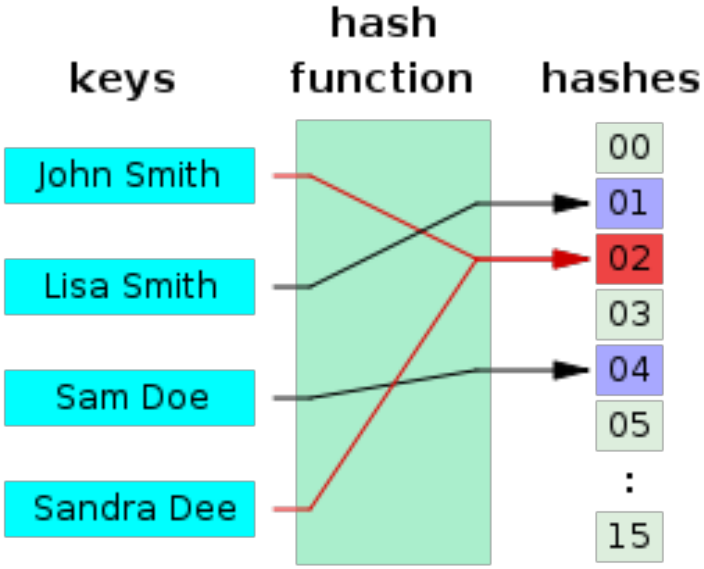
Overview

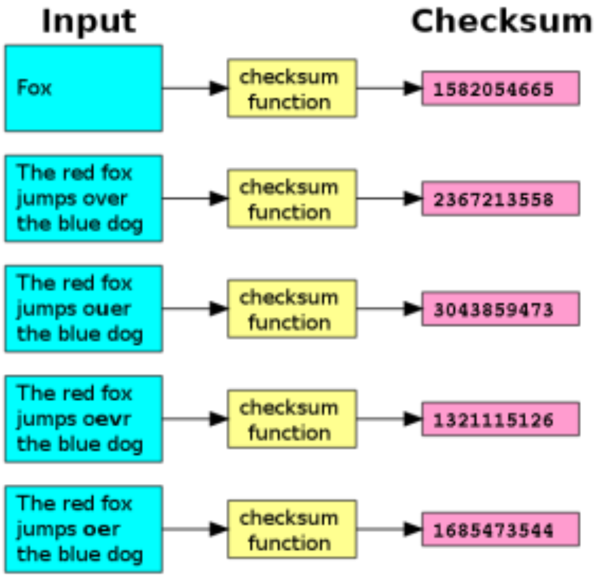
- Hash functions
  - Normal vs. cryptographic hash
  - Uses
  - Important properties
  - How they work, current choices
- Authentication
  - Passwords
    - Store as a hash
    - Salting
  - Tokens
  - Biometrics

Hash functions

- Normal hash functions:
  - Key: large data set of variable length
  - Value: small data set of fixed length
  - Examples:
    - Error checking: checksum, CRC
    - Constant time data structures: Java HashMap

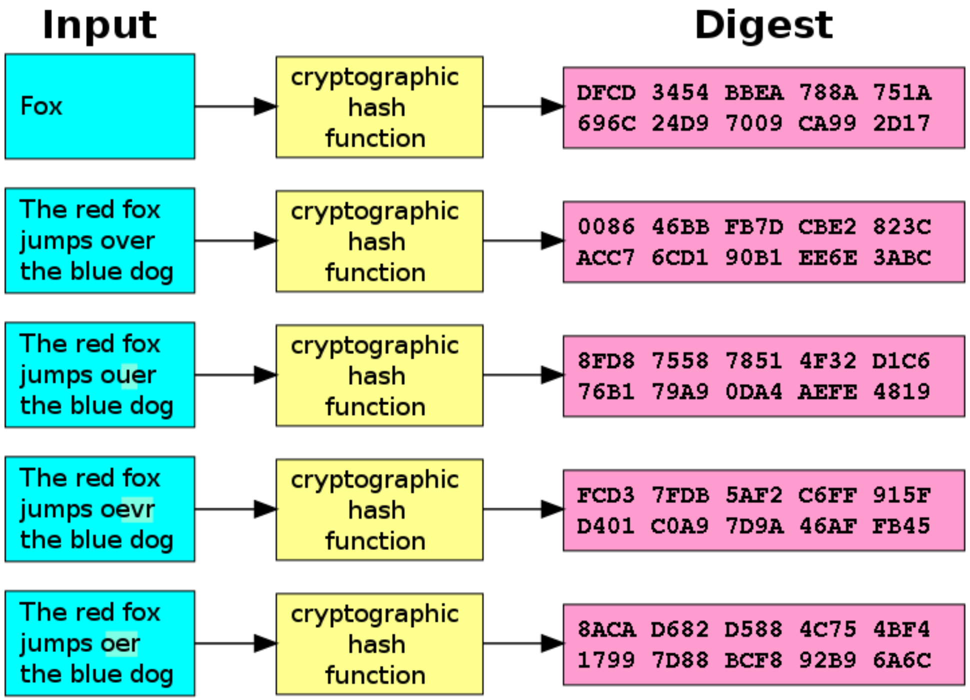






Cryptographic hash functions

- Secure hash functions:
  - Hash H(x) easy to compute for x
  - Arbitrary length input → fixed length output
  - Output: message digest, fingerprint
  - One-way: given h=H(x) intractable to find x
    - MD5 (128 bits)
    - SHA-1 (160 bits)
    - SHA-256 (256 bits)
    - SHA-512 (512 bits)


Input
Digest
Fox
cryptographic hash function
DFCD 3454 BBEA 788A 751A
696C 24D9 7009 CA99 2D17
The red fox jumps over the blue dog
cryptographic hash function
0086 46BB FB7D CBE2 823C
ACC7 6CD1 90B1 EE6E 3ABC
The red fox jumps ouer the blue dog
cryptographic hash function
8FD8 7558 7851 4F32 D1C6
76B1 79A9 0DA4 AEFE 4819
The red fox jumps oevr the blue dog
cryptographic hash function
FCD3 7FDB 5AF2 C6FF 915F
D401 C0A9 7D9A 46AF FB45
The red fox jumps oer the blue dog
cryptographic hash function
8ACA D682 D588 4C75 4BF4
1799 7D88 BCF8 92B9 6A6C

Secure hash uses

- User authentication
  - e.g. Store hash instead of actual password
- Message authenticity
  - e.g. Digital signing of email messages
- Intrusion detection
  - e.g. Detect if important files changed
- Compact file identifiers
  - e.g. Git

Secure hashes, important properties

- First pre-image resistant
  - For a hash h, intractable to find x s.t. H(x) = h
  - Hard to invert
  - Why do we need this?
    - If easy to invert, not useful for secure applications
    - Hacker could recover your actual password from database on a compromised server

Secure hashes, important properties

- Second pre-image resistant
  - Weak collision resistant
  - For input x, intractable to find y ≠ x s.t. H(y) = H(x)
  - Difficult to find a second input that hashes to the same thing as a given first input

Secure hashes, important properties

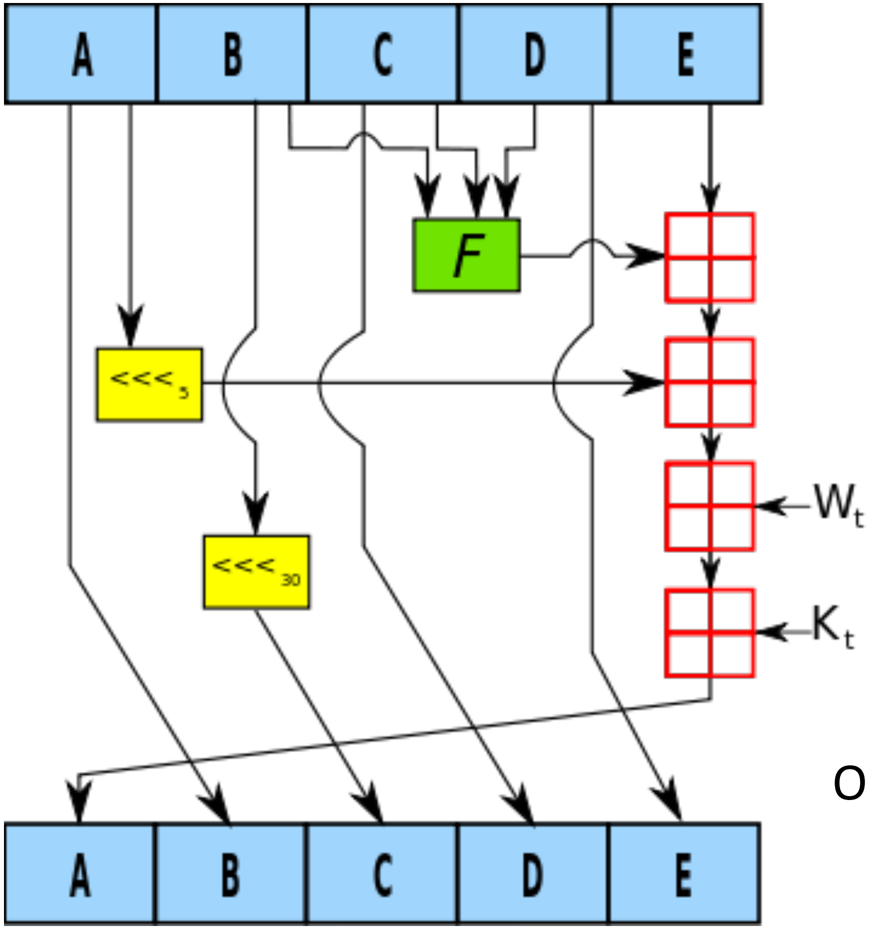
- Strong collision resistant
  - Intractable to find any pair (x, y) s.t. H(x) = H(y)
  - Difficult to find any two inputs with same hash
  - Why do we need this?
    - Attacker computes 2 messages: x, y s.t. H(x) = H(y)
    - Gives message x to Alice to hash and sign
      - e.g. Invoice with a total of $50
    - Attacker replaces x with y
      - e.g. Invoice with a total of $5000
    - Attack can claim Alice signed the larger one

History of secure hashes

- 1991: MD5 by Ron Rivest, 128-bit
- 1993: SHA-0 by NSA
- 1995: SHA-1, revised to fix weakness, 160-bit
- 1996: Attack found on MD5
- 2001: SHA-2 family, 224/256/384/512 bits
- 2005: Attack found on SHA-1
- 2010: NIST, federal agencies must use SHA-2
- 2012: SHA-3, NIST selects Keccak "ket-chak"

http://blogs.technet.com/b/srd/archive/2012/06/06/more-information-about-the-digital-certificates-used-to-sign-the-flame-malware.aspx

http://www.win.tue.nl/hashclash/rogue-ca/



One round within SHA-1 :

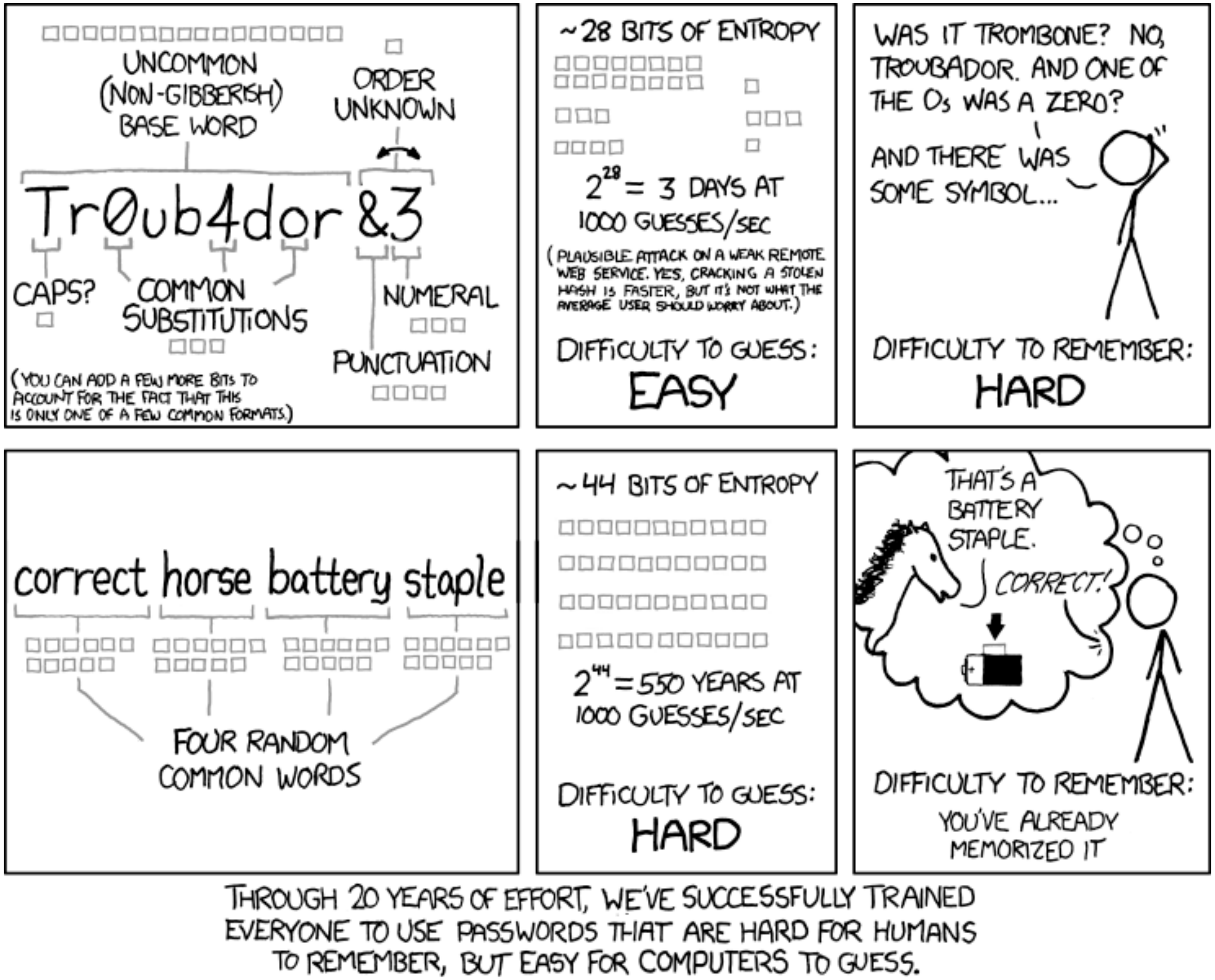
- A, B, C, D and E are 32-bit words of state
- F is a nonlinear function that varies
- $<<<_n$ a left bit rotation by n places
- n varies for each operation
- W_t expanded message word of round t
- K_t round constant of round t
- Box with plus: addition modulo 2^{32}

Authentication

- Proving your identify
  - Something you know: password, PIN, pet's name
  - Something you possess: a key, smart card
  - Something you are: fingerprints, retina, face
  - Something you do: voice print, handwriting, typing rhythm
- Means of authentication
  - Password
  - Token-based
  - Biometric

Password authentication

- Users choose some secret password
  - Differing levels of required complexity/annoyance


UNCOMMON (NON-GIBBERISH) BASE WORD
ORDER UNKNOWN
Tr0ub4dor &3
CAPS?
COMMON SUBSTITUTIONS
NUMERAL
PUNCTUATION
(YOU CAN ADD A FEW MORE BITS TO ACCOUNT FOR THE FACT THAT THIS IS ONLY ONE OF A FEW COMMON FORMATS.)
~28 BITS OF ENTROPY
2^28 = 3 DAYS AT 1000 GUESSES/SEC
(PLAUSIBLE ATTACK ON A WEAK REMOTE WEB SERVICE. YES, CRACKING A STOLEN HASH IS FASTER, BUT IT'S NOT WHAT THE AVERAGE USER SHOULD WORRY ABOUT.)
DIFFICULTY TO GUESS: EASY
WAS IT TROMBONE? NO, TROUBADOR. AND ONE OF THE Os WAS A ZERO?
AND THERE WAS SOME SYMBOL...
DIFFICULTY TO REMEMBER: HARD
correct horse battery staple
FOUR RANDOM COMMON WORDS
~44 BITS OF ENTROPY
2^44 = 550 YEARS AT 1000 GUESSES/SEC
DIFFICULTY TO GUESS: HARD
THAT'S A BATTERY STAPLE.
CORRECT!
DIFFICULTY TO REMEMBER: YOU'VE ALREADY MEMORIZED IT
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

https://xkcd.com/936/

Password storage

- User ID and password
  - Must be stored somewhere, e.g. `/etc/passwd`
    - Normally hash of password, not plaintext or encrypted
  - Shadow password file, e.g. `/etc/shadow`
    - Reachable only by privileged users

```
% more passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
work:x:1000:1000::/home/work:/bin/sh
mysql:x:104:110:MySQL Server,,,:/nonexistent:/bin/false
httpd:x:1001:1001::/home/httpd:/bin/sh
backup:x:1005:1005::/home/backup:/bin/sh
```

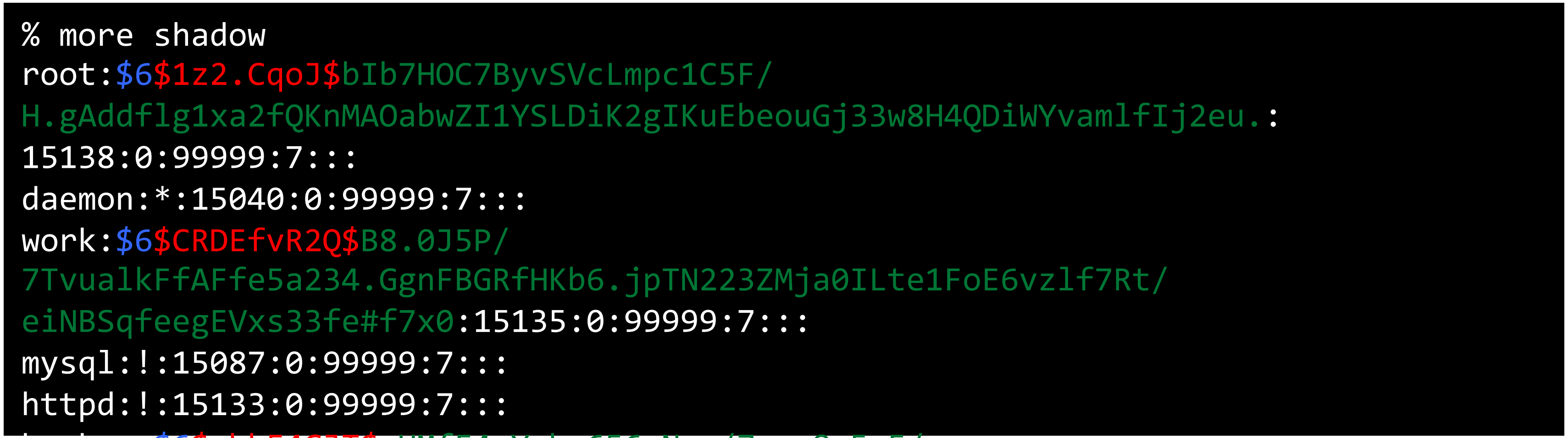
```
% more shadow
root:$6$1z2.CqoJ$bIb7HOC7ByvSVcLmpc1C5F/H.gAddflg1xa2fQKnMAOabwZI1YSLDiK2gIKuEbeo
uGj33w8H4QDiWYvamlfIj2eu.:15138:0:99999:7:::
daemon:*:15040:0:99999:7:::
work:$6$CRDEfvR2Q$B8.0J5P/7TvualkFfAFfe5a234.GgnFBGRfHKb6.jpTN223ZMja0ILte
1FoE6vzlf7Rt/eiNBSqfeegEVxs33fe#f7x0:15135:0:99999:7:::
mysql:!:15087:0:99999:7:::
httpd:!:15133:0:99999:7:::
backup:$6$whkE4GJT$yUMfE4gYwhp656rNqv/7see8y5aF/Vgra3FUe.g4Facg4Iug4vyJLg4F
bgeZW0i7feqMPCHQpBsJi/:15164:0:99999:7:::
```

Attacking passwords

- If password hashes compromised, attacker:
  - Knows any users with same password
  - Can tell if user has same password on multiple systems (if systems use same hash function)
  - Can use an offline dictionary attack
- Dictionary attack:
  - Precompute hash value for:
    - All sequences of a given (short) length
    - Common words
  - Check for match against hash in password file

Salt

- Salting passwords
  - On account creation, assign salt value
    - Timestamp, random value, ...
  - Salt stored unencrypted, associated with user ID
  - Hash computed from salt plus user's password
  - Makes dictionary attack much more expensive



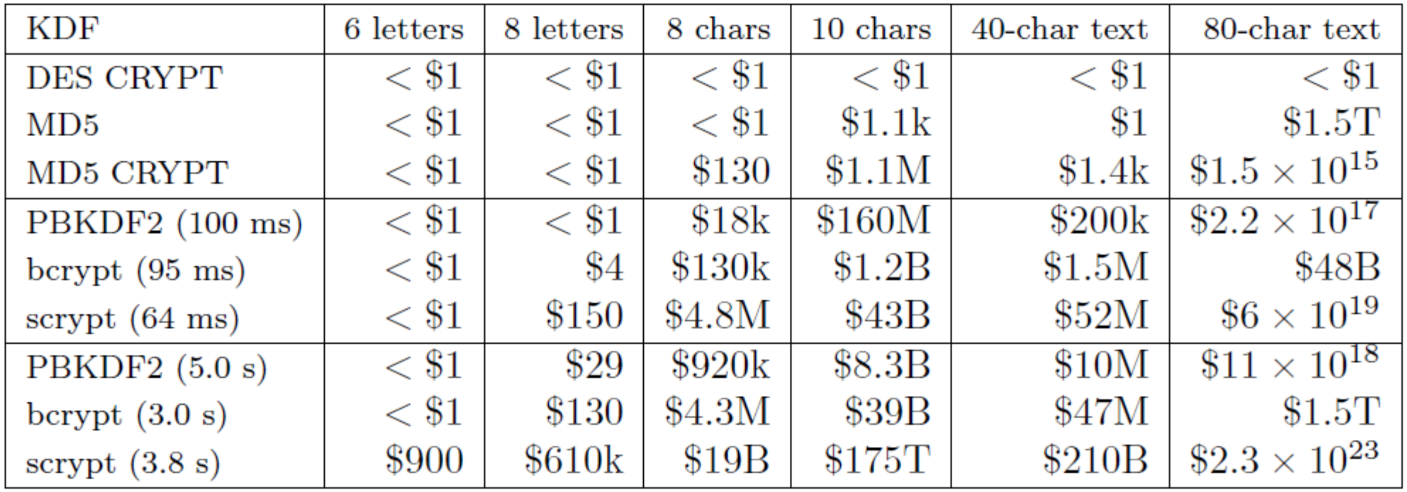
```
% more shadow
root:$6$1z2.CqoJ$bIb7HOC7ByvSVcLmpc1C5F/
H.gAddflg1xa2fQKnMAOabwZI1YSLDiK2gIKuEbeouGj33w8H4QDiWYvamlfIj2eu.:
15138:0:99999:7:::
daemon:*:15040:0:99999:7:::
work:$6$CRDEfvR2Q$B8.0J5P/
7TvualkFfAFfe5a234.GgnFBGRfHKb6.jpTN223ZMja0ILte1FoE6vzlf7Rt/
eiNBSqfeegEVxs33fe#f7x0:15135:0:99999:7:::
mysql:!:15087:0:99999:7:::
httpd:!:15133:0:99999:7:::
       $6$whkE4GJT$yUMfE4gYwhp656rNqv/7see8y5aF/
Vgra3FUe.g4Fa.g4Iug4vyJLg4FbgeZW0i7FeqHPCHQpB5Ji/
```

$6 = SHA-512, random salt value, SHA-512 of salt + password

Key stretching (strengthing)

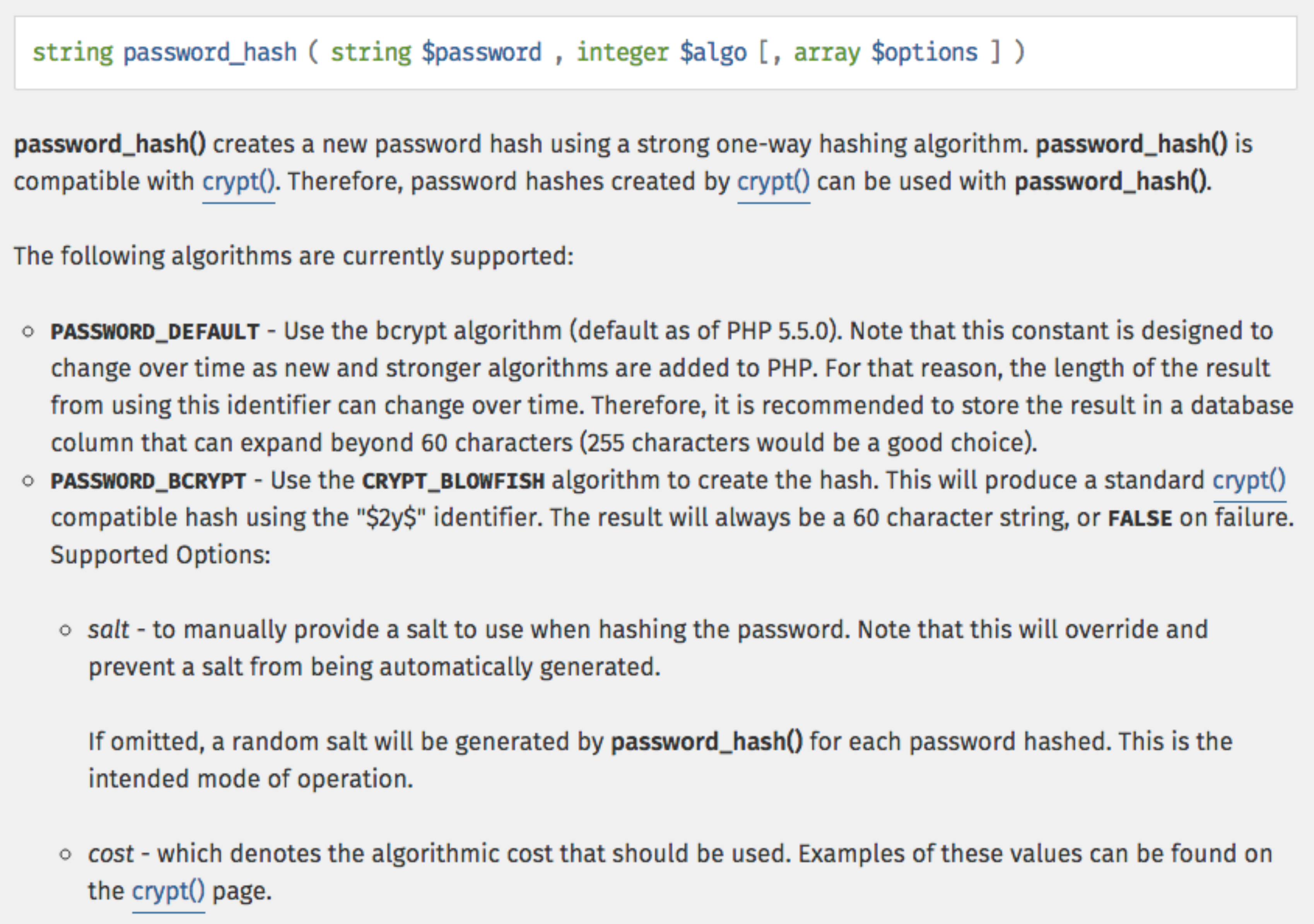
- Slow down the hashing
  - Users can wait a bit, slows brute-force attacks
    - e.g. UNIX CRYPT, iterated DES 25 times
  - hash(intermediate hashes, password, salt)
    - $x_0 = 0$
    - $x_i = h(x_{i-1} + \text{password} + \text{salt})$ for $i = 1, \dots, r$

TABLE 1. Estimated cost of hardware to crack a password in 1 year.



KDF	6 letters	8 letters	8 chars	10 chars	40-char text	80-char text
DES CRYPT	< $1	< $1	< $1	< $1	< $1	< $1
MD5	< $1	< $1	< $1	$1.1k	$1	$1.5T
MD5 CRYPT	< $1	< $1	$130	$1.1M	$1.4k	$1.5 × 10^{15}
PBKDF2 (100 ms)	< $1	< $1	$18k	$160M	$200k	$2.2 × 10^{17}
bcrypt (95 ms)	< $1	$4	$130k	$1.2B	$1.5M	$48B
scrypt (64 ms)	< $1	$150	$4.8M	$43B	$52M	$6 × 10^{19}
PBKDF2 (5.0 s)	< $1	$29	$920k	$8.3B	$10M	$11 × 10^{18}
bcrypt (3.0 s)	< $1	$130	$4.3M	$39B	$47M	$1.5T
scrypt (3.8 s)	$900	$610k	$19B	$175T	$210B	$2.3 × 10^{23}

http://www.tarsnap.com/scrypt/scrypt.pdf



```
string password_hash ( string $password , integer $algo [, array $options ] )
```

password_hash() creates a new password hash using a strong one-way hashing algorithm. **password_hash()** is compatible with crypt(). Therefore, password hashes created by crypt() can be used with **password_hash()**.

The following algorithms are currently supported:

- **PASSWORD_DEFAULT** - Use the bcrypt algorithm (default as of PHP 5.5.0). Note that this constant is designed to change over time as new and stronger algorithms are added to PHP. For that reason, the length of the result from using this identifier can change over time. Therefore, it is recommended to store the result in a database column that can expand beyond 60 characters (255 characters would be a good choice).
- **PASSWORD_BCRYPT** - Use the **CRYPT_BLOWFISH** algorithm to create the hash. This will produce a standard crypt() compatible hash using the "$2y$" identifier. The result will always be a 60 character string, or **FALSE** on failure.
  Supported Options:

  - *salt* - to manually provide a salt to use when hashing the password. Note that this will override and prevent a salt from being automatically generated.

    If omitted, a random salt will be generated by **password_hash()** for each password hashed. This is the intended mode of operation.

  - *cost* - which denotes the algorithmic cost that should be used. Examples of these values can be found on the crypt() page.

http://php.net/manual/en/function.password-hash.php

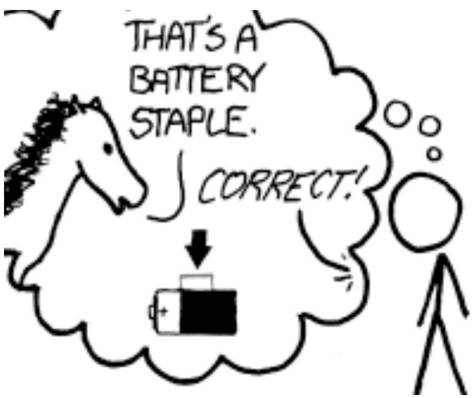
http://php.net/manual/en/function.crypt.php

Improving password security

- Reactive password checker:
  - System attacks itself, revokes guessed passwords
    - But system has to do an expensive amount of work
- Proactive password checker:
  - Users selects a candidate password
  - System checks if allowable
  - Hopefully guide users to secure choice
    - Without *too* much annoyance
  - But different systems have:
    - Different min/max lengths, allowed symbols, case rules, number rules, ...

Improving password security

- User education
  - – Encourage/force longer more complex passwords
    - e.g. Users often mistakenly believe reversing word makes password unguessable
  - – Use first letter of personal phrase
    - "My dog's first name is Rex" -> "MdfniR"
  - – Use random collection of words
    - "correcthorsebatterystaple"

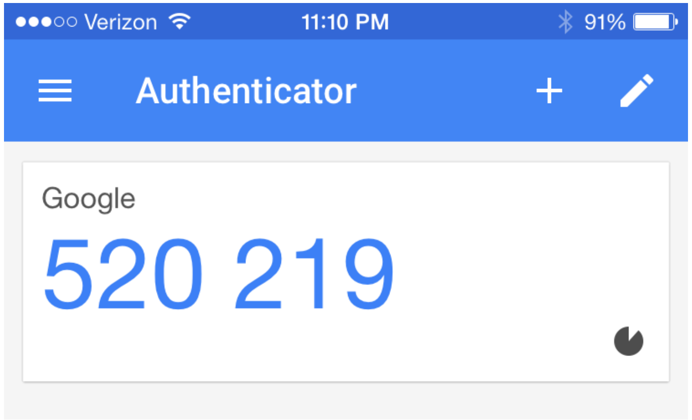


- Computer-generated passwords
  - – Normally low acceptance, users write them down
  - – Generate pronounceable syllables, FIPS PUB 181

Token-based / two-factor

- Require possession of an object
  - Unique ID based
    - Magnetic strip
    - Embedded microprocessor
    - e.g. ATM card, mobile phone

- Often in combination with user knowledge
  - e.g. ATM PIN, password


●●●○○ Verizon 11:10 PM 91%

Authenticator

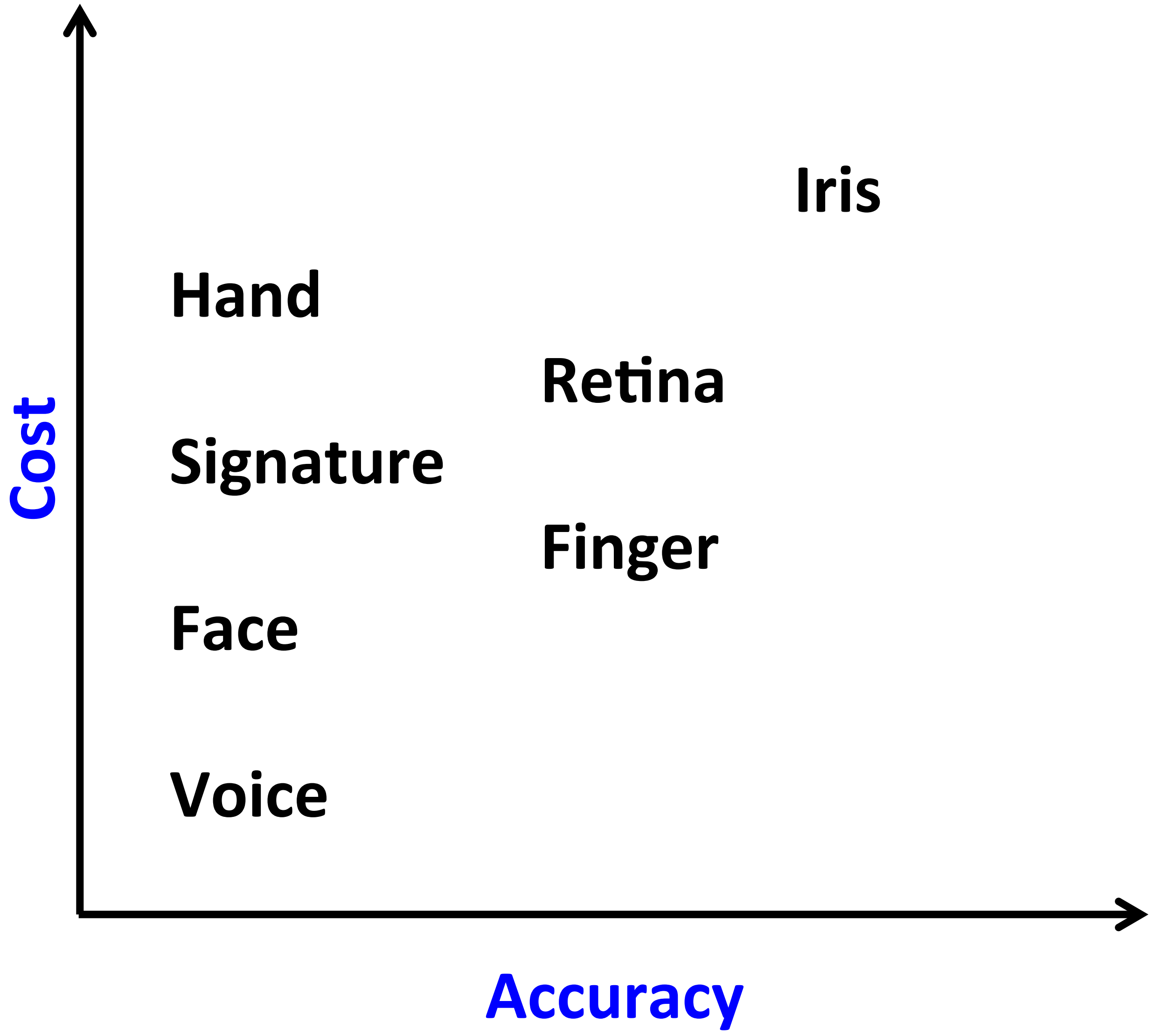
Google

520 219

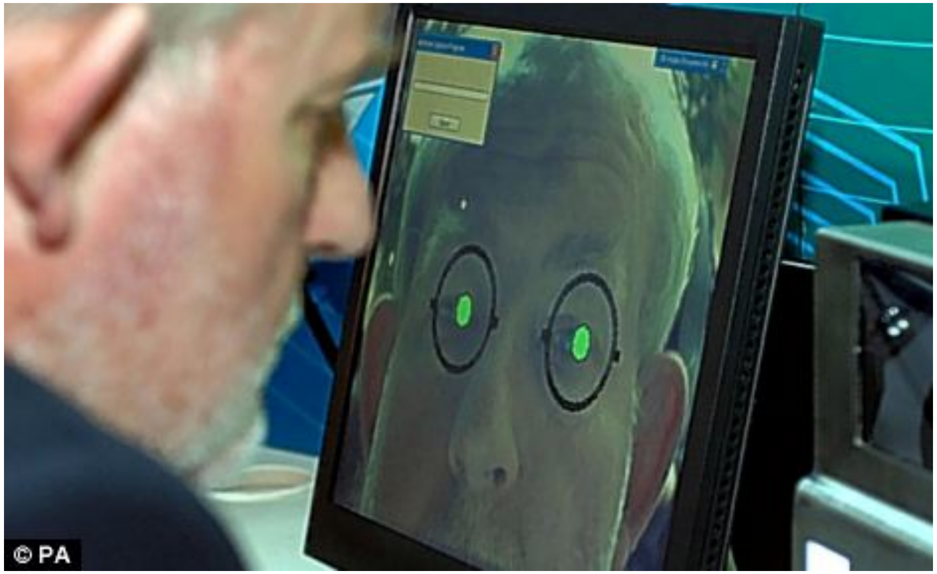
Biometric authentication

- Pattern recognition based
  - Facial recognition: location of facial features
  - Fingerprints: ridges and furrows on fingertip
  - Hand geometry: shape, length, width of fingers
  - Retinal: veins beneath retinal surface
  - Iris: structure of the iris
  - Signature: style of handwriting
  - Voice: patterns in speech signal
- Verification: proving you are who you say
- Identification: find out who you are
  - e.g. Avoid entering username

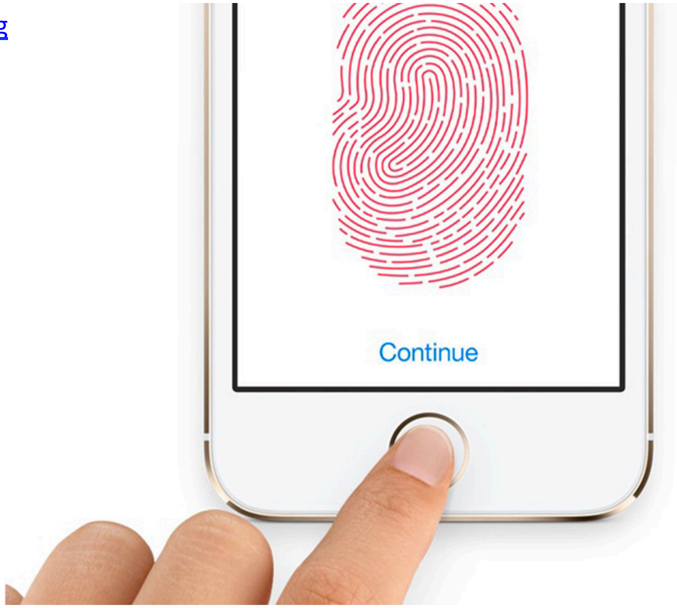
Biometric characteristics



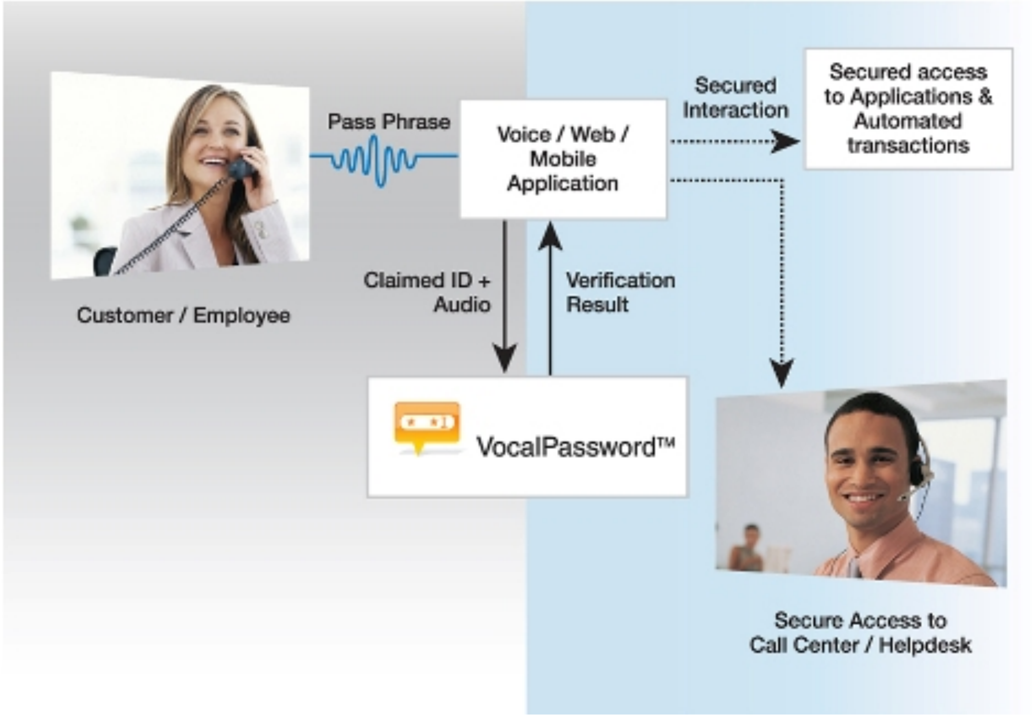
http://www.cl.cam.ac.uk/~jgd1000/UK_IRIS.png



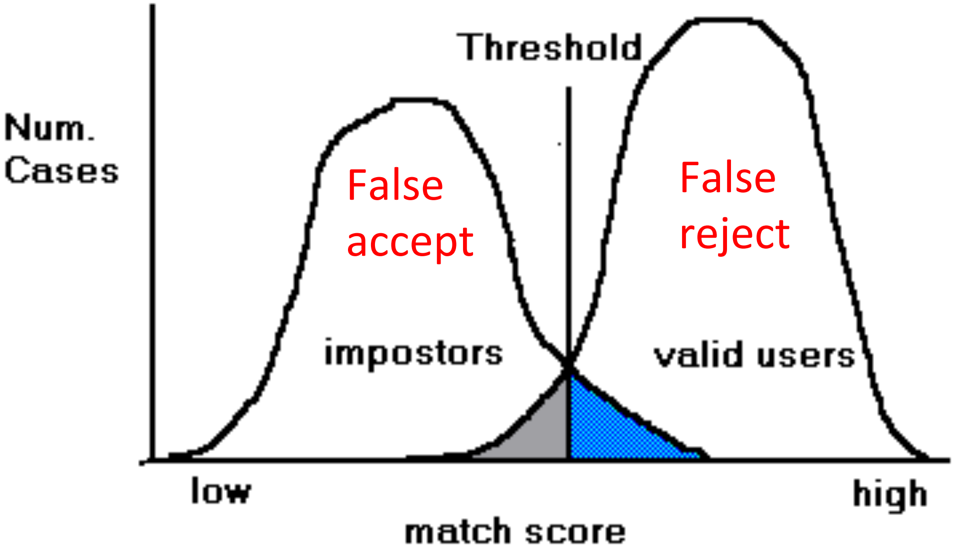
http://i.dailymail.co.uk/i/pix/2012/02/17/article-0-11C5E3C4000005DC-346_468x286.jpg



http://www.tomsguide.com/us/iphone-touchid-hack,news-17586.html


Customer / Employee
Pass Phrase
Voice / Web / Mobile Application
Secured Interaction
Secured access to Applications & Automated transactions
Claimed ID + Audio
Verification Result
VocalPassword™
Secure Access to Call Center / Helpdesk

http://www.nuance.com/for-business/by-solution/customer-service-solutions/solutions-services/inbound-solutions/voice-authentication-biometrics/vocal-password/index.htm



http://www.cs.cmu.edu/afs/cs/Web/People/jeongue/jeon/Speaker%20Verification.htm

Summary

- Cryptographic hash functions
  - Digital signing messages, password authentication, detecting changes in files
- Proving who you are
  - Passwords
    - Store only secure hash
    - Salting
    - Key stretching
  - Tokens, two-factor
  - Biometrics