Checksum and Cryptographic Hash Function

Xinwen Fu., Ph.D

Professor

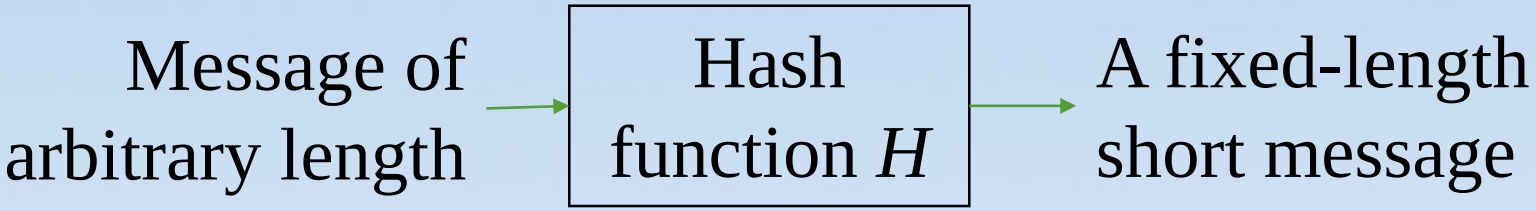
Department of Computer Science

University of Massachusetts Lowell

Outline

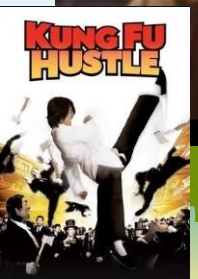
- Checksum and Hash Algorithm
- Hash application
- Hands-on lab

Hash Algorithm



- Also known as
  - Message digest
  - One-way transformation
  - One-way function

“Even his mom cannot recognize him!”



Hash Length

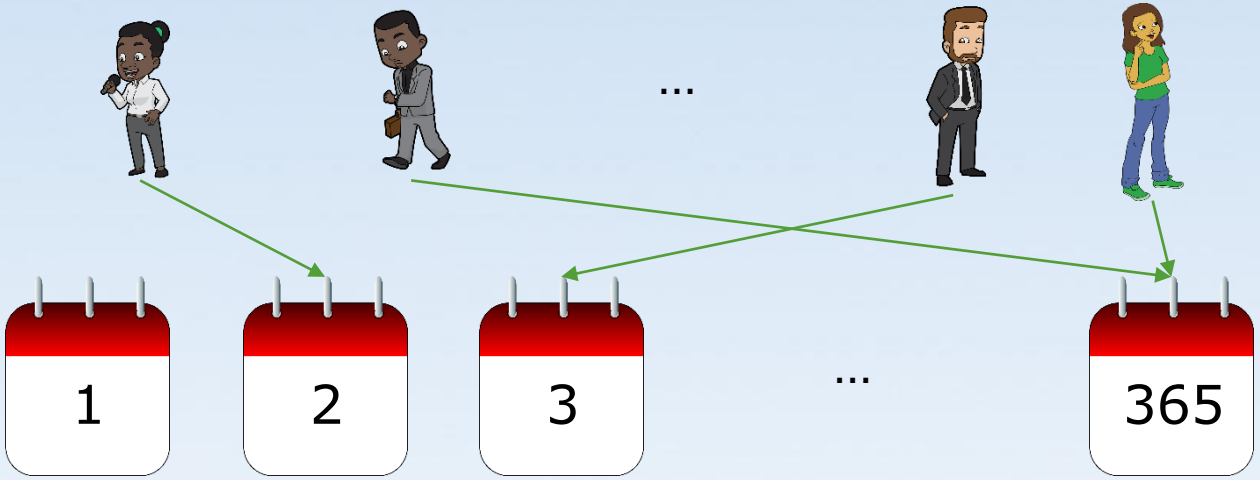
- Length of H(m) much shorter than length of message m
- Usually fixed lengths
  - MD5: 128 (16 bytes)
  - SHA1: 160 bits (20 bytes)

Application: Hash as Message/File Identity

- MD5 Hash algorithm
  - Generates 128 bit hash
- n files on a disk: one file one hash
- What is the chance that two files have the same hash?
  - Similar to the birthday problem/paradox

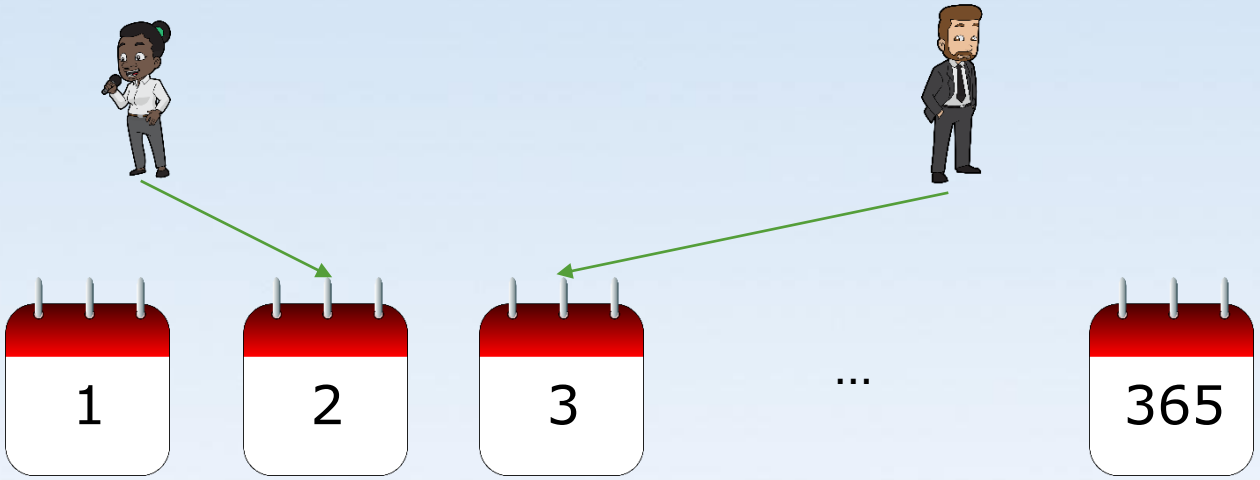
Birthday Problem

- What is the probability that in a set of *n* randomly chosen people, at least one pair of them will have the same birthday?



How to compute the probability?

- How many combinations of putting *n* people into 365 slots with no two people in the same slot?
  - $365 \times 364 \times \ldots \times (365\text{-}n+1)$. Why?
- How many combinations of putting n people into 365 slots?
  - 365^n. Why?

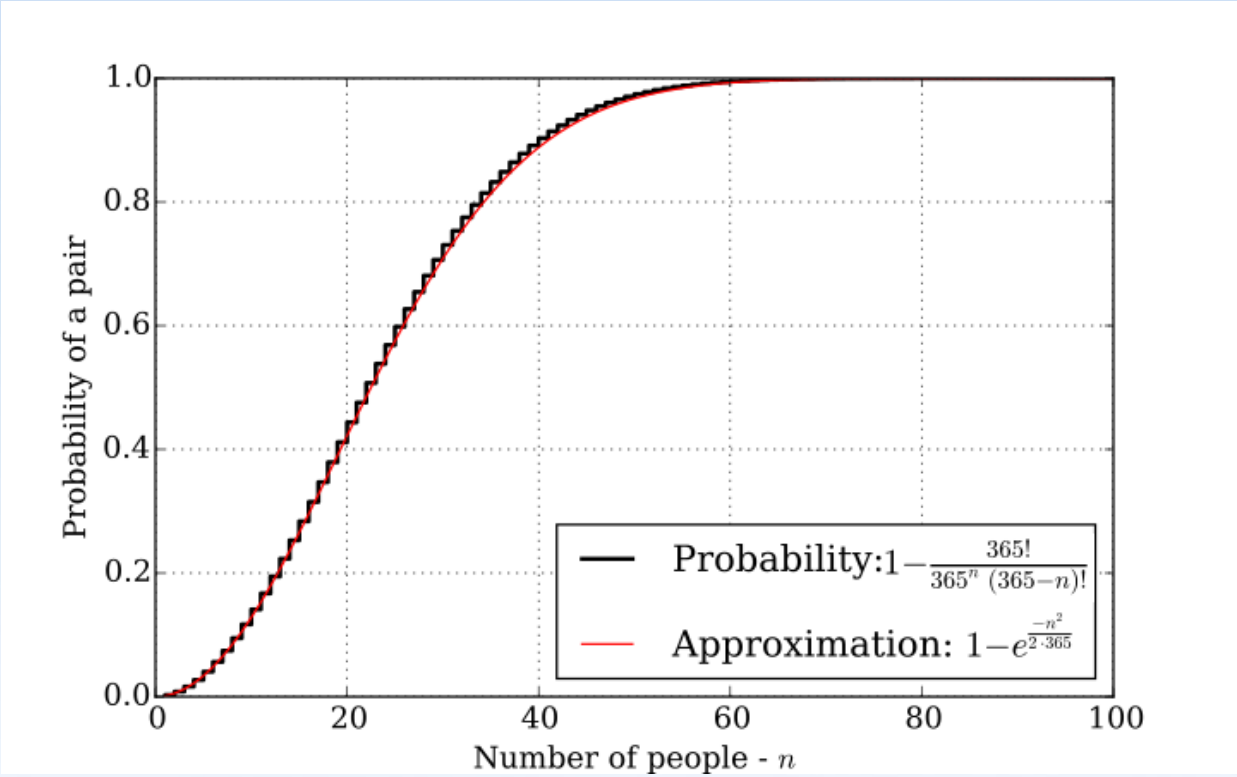


Probability?

- What is the chance that n people all have different birthdays?
  - $P = \frac{365 \times 364 \times \ldots \times (365-n+1)}{365^n}$
- What is the chance that at least two people out of n people have the same birthday
  - 1-P

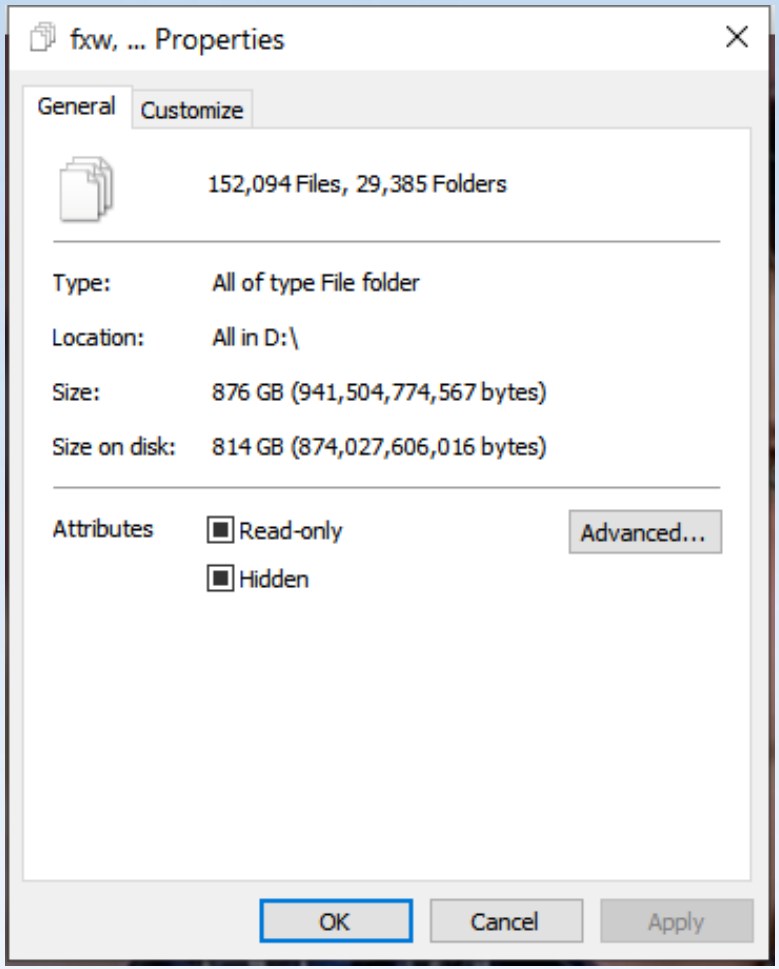
Birthday Problem Probability

- The computed probability of at least two people sharing a birthday versus the number of people



Hash as Identify Probability

- MD5 hash algorithm
- 2^{128} possible hashes
  - Roughly 3.4×10^{38}
  - Compared with 365 days
- n files on a disk
  - Compared with n randomly chosen people
  - E.g. 152,094 files on my hard disk D



Desirable Properties of Hash Functions

- Performance: easy to compute H(m)
- One-way property
  - Given H(m) but not m, it's difficult to find m
- Weak collision free
  - Given H(m), it's difficult to find m' such that H(m') = H(m).
- Strong collision free
  - Computationally infeasible to find m1, m2 such that $H(m_1) = H(m_2)$

Challenge

- Can you design a hash algorithm?
- Can you design a good hash algorithm?

Internet checksum: poor crypto hash function

Internet checksum has some properties of hash function:

- produces fixed length digest of message
- is many-to-one

But given message with given hash value, it is easy to find another message with same hash value:

message	ASCII format
I O U 1	49 4F 55 31
0 0 . 9	30 30 2E 39
9 B O B	39 42 4F 42
	B2 C1 D2 AC

different messages but identical checksums!

message	ASCII format
I O U 9	49 4F 55 39
0 0 . 1	30 30 2E 31
9 B O B	39 42 4F 42
	B2 C1 D2 AC

Outline

- Checksum and Hash Algorithm
- Hash application
- Hands-on lab

Application: Password hashing

- Doesn’t need to know password to verify it
- Store H(password|salt) and salt, and compare it with the user-entered password
- Salt makes dictionary attack more difficult

Application: Message Integrity

- Keyed hash
  - Agree on a secret key k
  - Compute H(m|k) and send with m
  - E.g. HMAC

Base64 Encoding

- The term Base64 is taken from the Multipurpose Internet Mail Extensions (MIME) standard
- Converts every 6 bits (64 possibilities) of a message into one character (8 bits) in a 64-character alphabet
  - Example 64-character alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
  - = may be used to indicate padding
- Example command---base64

```
kali@gencyber:~$ echo "hello" | base64
aGVsbG8K
kali@gencyber:~$
```

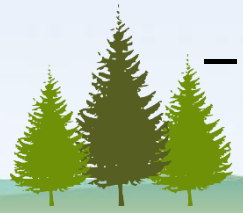
Base64 Encoding Example

- A public key encoded with base64

-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAwiC7M2YPYEI2R47Ozlu4
olZ2XtCGtt+i2UHUtjig2gKTRwxsjct3ZYRKawuYGGGaBQAcrqlZV3AJAwit9f+F
+zEWrGGyEEFypWdpOb3arNB0XJLxNsRNwSv09idSJ3o/rRfZMV0DR/dJcDznYnvt
/JSzckSktuwocspYa5QVEgMd6/SrW4ZyzK3OqoBW9ktzjTDP13uol/Lnv4Ts+hPE
lzkG9y+XZ3oa0vqd2oFjWU/13tif/1TrAqUE/Ph/4rHl0EwRCenrH1lceL9PnywF
L+GN2Iz1P1oG8n0d18BGn7XkejGPxjrarqpzEToKlLXAxp+i8iXypTSSRsYsj0Vo
PwIDAQAB
-----END PUBLIC KEY-----

Base16 Encoding

- Converts every 4 bits (16 possibilities) of a message into one character in a 16-character alphabet
  - 0 to 9, and "A"–"F“ (or "a"–"f“)
  - That is, hexadecimal
- Example: get base16 encoding a file called changelog
  - *base16 changelog*
    0a097375646f20616464757365722206b616c69207626f78
    73660a095375646f206170742069 6e7374616c6c2070797
    4686f6e332d7069700a097375646f206170742069 6e7374
    616c6c2074656c6e65740a097375646f2067656d20696e7
    374616c6c207265782d746578740a
  - *base16* part of basez package for Linux



Outline

- Checksum and Hash Algorithm
- Hash application
- Hands-on lab

MD5

MD5 Hash

- One student as Sender
  1. Compute hash of a message (md5sum)
     - echo -n ‘message' | md5sum
  2. Send message and its hash to chat server in the format of *message>>>hash*
- Anyone as Receiver
  1. Compute hash of received message locally (md5sum)
     - echo -n ‘received-message' | md5sum
  2. Compare newly computed hash with the hash sent over
- Discuss why hash only is not good for message integrity verification

HMAC

HMAC

- One student as Sender
  1. Share a secret key (a string) with Receiver
     - E.g. through Zoom's private chat
  2. Compute hmac of a message
     - *echo -n "message" | openssl sha1 -hmac "key"*
  3. Send the message and hmac to the chat server in the format of *message>>>hmac*
- The one with the key as Receiver
  1. Compute hmac of received message locally
     - *echo -n "received-message" | openssl sha1 -hmac "key"*
  2. Compare newly computed hmac with the received hmac

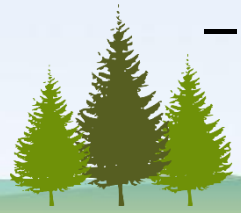
Password Cracking with John the Ripper

Password Cracking

- John the ripper---password cracking tool
  - Example use
- Run the following command, where mypasswd is the password hash file in the required format
  - *john mypasswd*
  - This command will try "single crack" mode first, then use a wordlist (i.e. a dictionary of password; default password list at /usr/share/john/password.lst), and finally go for "incremental" mode
  - Please refer to MODES for more information on these modes.

Password Cracking (Cont'd)

- If you've got some passwords cracked, use the following command to show cracked passwords
  - *john --show mypasswd*
  - Cracked passwords are stored in $JOHN/john.pot (~/.john/john.pot in kali) in a specific format
- Delete john.pot in order to crack again
  - *rm ~/.john/john.pot*)
- Test your own password
  - Create your password hash
    openssl passwd -1 -salt RnYtvEVV abc123
  - Replace a hash in mypasswd with the output above



Backup

AES

Encryption with AES

- *echo "OpenSSL" | openssl enc -iter 1000 -aes-256-cbc -a -k hello*
  - *-k hello*: The key will be generated from hello.
    - Without *-k hello*, the command will ask for a password, which will be translated into a key
  - *-iter 1000* is related to creating a strong key from the password
  - *-a*: means BASE64 output

Decryption with AES

- *echo "U2FsdGVkX1+lVCnMEVpKXisqA1IlycMvDFkv72ILasg=" | openssl enc -aes-256-cbc -iter 1000 -a -d -k hello*
  - *-d*: means decryption

Encrypting and Decrypting File with AES

- Encryption
  - openssl aes-256-cbc -a -salt -in secrets.txt -out secrets.txt.enc -iter 1000 -k hello
- Decryption
  - openssl aes-256-cbc -d -a -in secrets.txt.enc -out secrets.txt.new -iter 1000 -k hello

RSA

Generate Public and Private Key Pair

- Generate public and private key pair
  - *openssl genpkey -out privkey.pem -algorithm rsa*
  - Note: the created privkey.pem can be used as the private key although it contains the public key
- Extract the public key from privkey.pem and save it in pubkey.pem
  - *openssl rsa -in privkey.pem -outform PEM -pubout -out pubkey.pem*

Encryption with RSA

- Create a file
  - echo "Welcome to LinuxCareer.com" > encrypt.txt
- Encryption
  - openssl rsautl -encrypt -inkey pubkey.pem -pubin -in encrypt.txt -out encrypt.dat
- Encode the binary ciphertext with base64
  - *openssl enc -base64 -in encrypt.dat -out encrypt.dat.base64*
  - Encoding is not necessary, but needed for sending the ciphertext through our chat server

Decryption with RSA

- Decode *encrypt.dat.base64* and get the binary ciphertext
  - *openssl enc -base64 -d -in encrypt.dat.base64 -out encrypt.dat*
- Decryption
  - openssl rsautl -decrypt -inkey privkey.pem -in encrypt.dat -out new_encrypt.txt