CMSC389R

Cryptography I

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

homework questions

Exams handed back

patriotCTF @ GMU tomorrow

ShmooCon

BSidesNova

cryptography

cryptography

There’s more than just Caesar cipher?

cryptography

- Science behind securing digital information
  - Authentication
  - Data integrity
  - Message secrecy
  - Access control

history

- Secret codes to protect information
  - Substitution
    - Caesar, ROT13
  - Transposition
    - Rail Fence, Route Cipher
- Historically used for military purposes
- “Art form” - little scientific rigor

modern cryptography

- Mathematically rigorous
  - Formal definitions
  - Provability
  - Well-defined assumptions

uh, math?

- Hopefully not too much
- Take CMSC456, MATH456, or ENEE456 (formerly ENEE459E/CMSC489R)

ciphers

- Old ciphers are insecure
  - Caesar: keyspace of 26 -> bruteforceable
  - ROT13: keyspace of 1 -> ...bruteforceable?
  - Vigenere: unique scheme, but frequency analysis

  What makes a cipher secure?

ciphers

- Should be built off a *secure permutation*
  - Permutation ought to be key-dependent
  - Different keys -> different permutations
  - Permutation should *look* random
- Ciphers often referred to as permutations
  - Take in fixed-length input and produces fixed-length outputs

one-time pad

- Satisfies previous requirements for perfect secrecy
- Enc(K, M) = M ^ K = C (^ is binary XOR)
- Dec(K, C) = C ^ K = M
- len(M) = len(K) MUST be true for secrecy!
  - What's the probability of 1 ^ randbit() = 0?
  - Claude Shannon: Pr = ½, so rand K -> rand C!

on probability

- Probability is the likelihood of something happening or not -> Pr in range [0, 1]
- Pr = 0 means never happening
- Pr = 1 means always happening
- Cryptography uses probability to measure an attacker’s chance at success
- e.g. finding a correct n-bit key is 1 out of 2^n or $1/2^n$ -> best chances for a secure cipher

goals for security

- In practice, ciphers need to be secure against a given attack model
- Kerchoff’s principle: secrecy of a cipher should only depend on the key, not the scheme
- Black-box models
  - Given the algorithm, can query a box
- Grey-box models
  - Given the implementation, examine behavior

attacker models

- Ciphertext-only attacker (COA)
  - Have list of C’s, don’t know M’s
- Known-plaintext attacker (KPA)
  - Have list of (M, C) pairs
- Chosen-plaintext attacker (CPA)
  - Can pick M, query box, receive C
- Chosen-ciphertext attacker (CCA)
  - Can pick C or M, query box, receive M or C

security goals

- Indistinguishability (IND)
  - Attacker has M_1, M_2
  - Attacker sends (M_1, M_2) to Oracle
  - Oracle encrypts one of the messages and sends C
  - Attacker should not be able to guess better than a 50/50 chance which message was used

security goals

- Non-malleability (NM)
  - Attacker has (M, C)
  - Attacker should not be able to modify C -> C’ in a way that M’ relates to M
  - e.g. one-time pad fails non-malleability
    - recall: M ^ K = C
    - C’ = C ^ X -> M’ = M ^ X

security goals

- Goals need an attack model, often written as GOAL-ATTACKMODEL
- Semantic Security: IND-CPA
  - Ciphertexts should not leak info about msgs
  - i.e. encrypting M twice should yield $C_1 \neq C_2$
  - One way to achieve this is with randomness
    - pick random R: C=Enc(K,R,P), M=Dec(K,R,C)

hashing

hashing

- One-way cryptographic function
- Input -> hashing algorithm -> output
  - Small input change -> large output change
  - Unique* output for every unique input
- Used in git commits, intrusion detection, evidence integrity in forensic investigations

*collisions

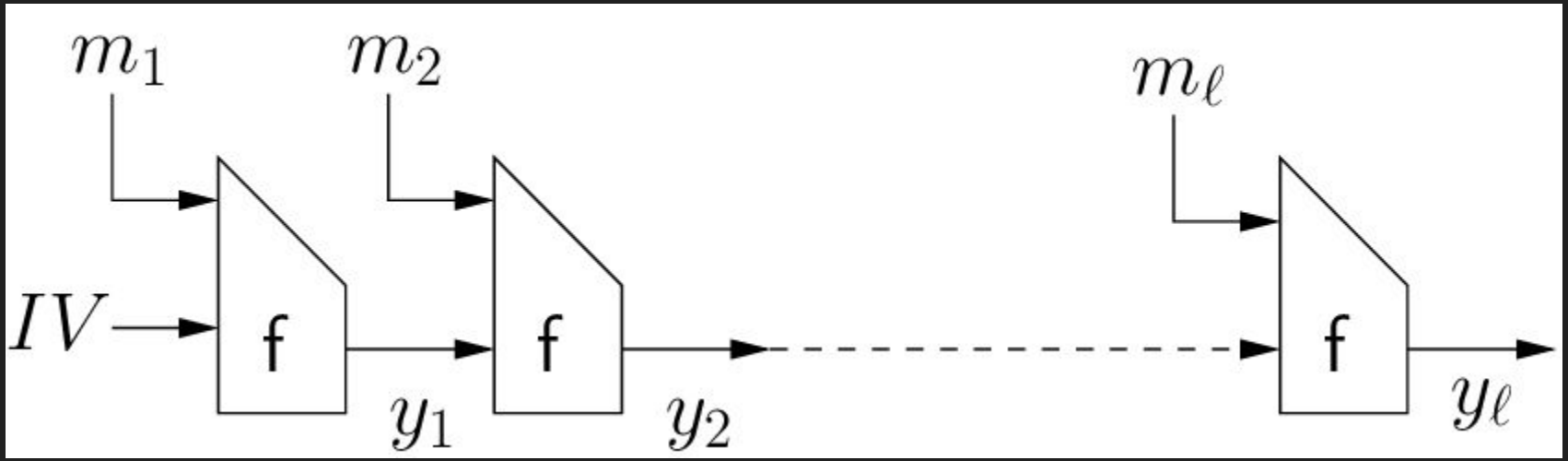
secure hashes

- Unpredictability
  - Small change in input -> large change in output
- Pre-image resistance
  - Given Hash(M), hard to find M
- Collision resistance
  - Given M, Hash(M), hard to find M’ such that Hash(M’) == Hash(M)
  - Collisions inevitable due to compression

building hash functions

- Usually compress large amounts of data -> fixed size outputs
- Use a form of compression mechanism
  - e.g. Davies-Meyer construct, sponge function
- Merkle-Damgard construct: chain together compression functions to accept infinitely large hash inputs

Merkle-Damgard



common hashing algorithms

- MD5 - very common, insecure
  - Internal security issues allow for ez collisions
- SHA1 - also common, recently declared insecure
  - Google & CWI *SHAttered* https://shattered.io/static/shattered.pdf
  - Used by git for commit hashes https://gist.github.com/masak/2415865
- SHA256
- SHA512

hashing on linux

- MD5: md5sum <text or file>
- SHA1: sha1sum <text or file>
- SHA256: sha256sum <text or file>
- SHA512: sha512sum <text or file>

Common syntax = easy to hash!

**NOTE: use echo -ne to NOT hash a trailing newline in your input, and to interpret escape sequences (e.g. \t as tab)

echo -ne “Hello World!” | md5sum

hashing in python

- Python’s hashlib
- Provides hashing capabilities for MD5, SHA1, SHA224, SHA256, SHA512, and more

```
>>> import hashlib

>>> h = hashlib.md5(“Hello CMSC389R!”)

>>> print h.hexdigest()

a0d708f132eda63ce4b0d11b60ec701c
```

where to find hashes?

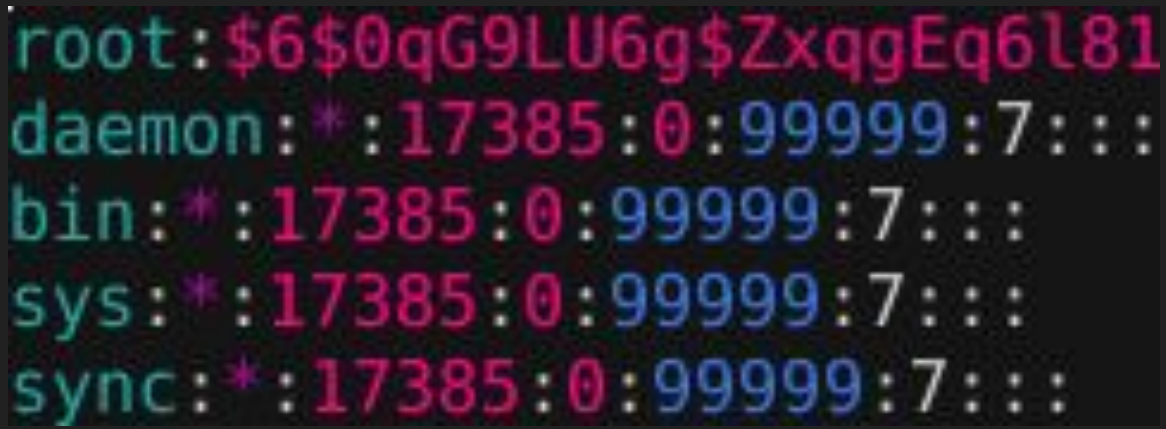
- OS passwords
  - Linux: /etc/shadow
  - Windows: C:/Windows/System32/config
    - Locked while OS is booted, loaded into RAM
    - Tools to recover these hashes
- Leaked databases
- From OSINT or Forensics discoveries

/etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
```

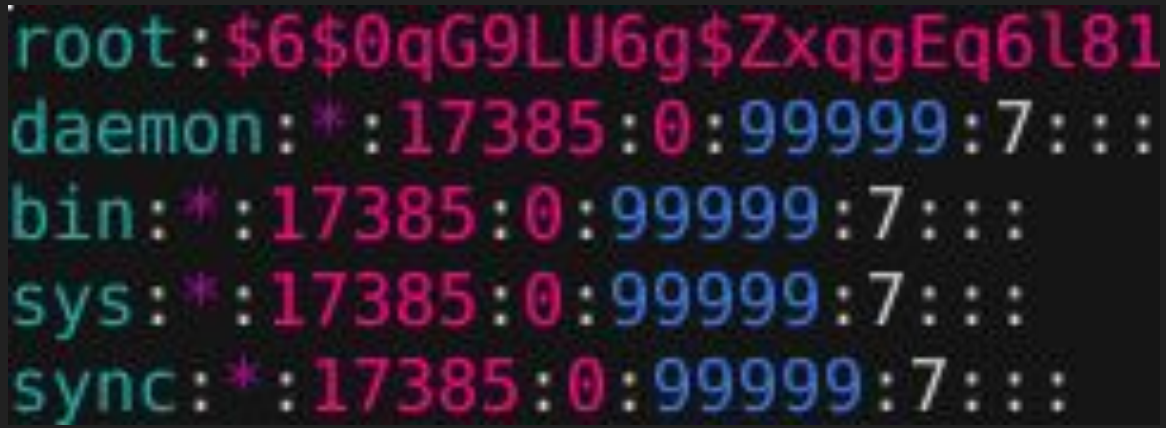
- login ID, password, user ID, group ID, username, home folder, command interpreter
- If password is ‘x’ then it is located in /etc/shadow

/etc/shadow



- 1. login name, 2. encrypted password (crypt(3)), 3. date of last pass change, 4. min pass age, 5. max pass age, 6. pass warning period, 7. pass inactivity period, 8. account expiration, 9. reserved

/etc/shadow



- ‘*’ or ‘!’ for password means account can’t be logged in normally w/ password
- password formatted as idsalt$encrypted

hash recovery

I thought it was one way?

hash recovery

brute forcing

- Define message space
  - All lowercase letters, letters + numbers, symbols, emojis?
- Enumerate each possible combination of items in your message space
  - aaaa, aaab, ..., zzzz
- Hash each enumeration and test against target hash

mask attacks

- Smarter brute force
- Say you know how the message was formatted...
  - 3 lowercase letters then 5 numbers
- Enumerate all combinations of your more-specific message space
  - aaa00000, aaa00001, ..., zzz99999
- Hash each enumeration and test against target hash

dictionary attacks

- Say you know the password was used from a leak...
  - i.e. SecLists
- Download password list
- Hash each password from list and test against target hash

hybrid attacks

- Brute force + dictionary
- Say the user’s password was found in a leak, and they only changed it slightly...
  - e.g. blink182 -> blink182!
- Download password list
- Enumerate potential modifications and append/prepend to password from list
- Hash each string and test against target hash

password recovery tools

- John the Ripper - cracks password hashes for UNIX and Windows
- Hashcat - cross-platform general hash cracker, better GPU support

john the ripper

- Syntax: john <flags> <password-files>
- Flags:
  - -wordlist:LIST_FILE
  - -incremental:MODE
- May need to “unshadow” /etc/passwd file
  - i.e. if passwords are ‘x’ and not hashes
  - unshadow /etc/passwd /etc/shadow
- Manpage kinda sucks, http://www.openwall.com/john/doc/

hashcat

- Syntax: hashcat <flags> <file-with-hashes>
- Flags:
  - -m <hash-type-code>
    - 0 = MD5, 100 = SHA1, 1400 = SHA256
  - -a <attack-mode-code>
    - 0 = dictionary, 3 = bruteforce, etc
  - -1 <custom-password-mask>
  - And many more... use the manpage!

aside

- Most people search for documentation online
  - “how 2 hashcat??”
  - “python socket how”
  - “arch linux?”
- Manpages are wildly powerful in the amount of information they provide

man

- type man <program> for basic help
- Manpages have various sections:
  - (1) executable programs (default when typing man)
  - (2) syscalls (like read, write)
  - (3) library calls (like printf, strcmp)
  - (5) file formats (like /etc/shadow!)
- Can discover more by typing man man

man

- When you see something in a manpage that says text(number), this means the manpage entry for text exists in section number.

```
SEE ALSO
       close(2), fcntl(2),
```

- Entry for close exists in section 2. What would that classify close as?

help()

- Python also has a help() function which provides a single-section manpage-like reference to the function or module
- To get help() on a module, import and call help() on it

```
>>> import math

>>> help(math)
```

reading manpages like a pro

- Blog post from last semester’s teacher https://blog.yossarian.net/2018/01/22/Reading-Manpages-Like-a-Pro
- Has more tips/tricks on using man

homework #7

will be posted soon.