clicker.cs.illinois.edu
code 340

Security

Last time

- Hashing passwords, salt and pepper
- Good randomness
- Symmatric-key encryption
  - Diffie-Helman key exchange
- Assymmetric-key encryption (public and private)
- Digital signature
  - Certificate

Learning objectives

1. Continuing last class
    - More on ciphers and hashes
    - Understand what “cryptographically secure” means
2. Understand goals of security
    - CIA Triad (Confidentiality, Integrity, Availability)
    - Authentication, Authorization, and Least Privilege
3. How security shows up in various contexts
    - HTTPS
    - Unix-style permissions
    - Blockchain (next class)

Part 1: continuing last class

Cryptography primitives

Technique	Input	Output	Keys	Reversible?
Hash	Any size	Fixed size	0	No
`def hash(m:bytes) -> int: ...`				
Symmetric key encryption	Any size	Same size	1	Yes
`def symm_encrypt(m:bytes, key:int) -> bytes: ...` `def symm_decrypt(m:bytes, key:int) -> bytes: ...`				
Assymetric key encryption	Fixed size*	Fixed size	2	Yes
`def publ(m:int, public_key:int) -> int: ...` `def priv(m:int, public_key:int, private_key:int) -> int: ...`				

To *sign* a message, we

A. Hash the symmetric encryption of the message
B. Hash the public-key encryption of the message
C. Hash the private-key encryption of the message
D. Symmetrically encrypt the hash of the message
E. Public-key encrypt the hash of the message
F. Private-key encrypt the hash of the message

clicker.cs.illinois.edu
code 340 #2

Signature

- The signer has the message and both keys, and creates the signature as:

```
sig = priv_encrypt(hash(message), pub_key, priv_key)
```

- The recipient has the message, signature, and public key and checks the signature as:

```
assert hash(message) == pub_decrypt(sig, pub_key)
```

Cryptographically secure

- A procedure is **cryptographically secure** if *all* the following are true:
    - Computing its inverse requires prohibitively-expensive **guess-and-check**.
    - Computing it does not have **side channels** that leak information.
    - There are no **patterns** in its output that make other procedures easier to guess.
- Encryption schemes are weakened over time:
    - A pattern is found that reduces the space to guess from.
    - Compute power makes previously-prohibitive guess-and-check more doable.

Side channels are hard to avoid.

We will **not** create any cryptographically secure code in this class.

Suppose I find that your encryption code uses more computing cycles for 1 bits in the key than for 0 bits, so I put a power meter on your house and use the power drawn by your computer to detect how many 1 bits there are in your key, then use that information to narrow the search space when trying to guess your key.

I have used which insecurity?

clicker.cs.illinois.edu
code 340 #3

A. Failure to require expensive guess-and-check
B. Presence of a side-channel leaking information
C. Patterns in output that weakens results

Functions we'll use

- Cryptographically-secure hash SHA-256

```python
import hashlib

hash = hashlib.sha256(data).digest()
```

  - 256-bit (32-bye) results – big enough that collisions are incredibly rare
  - Uses a complex combination of bitwise operators
  - As of 2025, best attack reduces attack search space from 2^{256} to $2^{251.7}$

Functions we'll use (cont'd)

- Asymmetric cipher RSA

  ```
  encrypted = pow(data, private_key, public_key)
  decrypted = pow(encrypted, 0x10001, public_key)
  ```

  - `pow(a,b,c)` efficiently computes $a^b \mod c$
    - has many side channels (timing, power, cache)
  - Can use any size key; bigger keys are more secure
  - Many patterns known; 2048-bit keys have search space of around 2^{112}, not 2^{2048}
  - Quantum computers may render RSA obsolete

- We won't code with symmetric ciphers in this class

The importance of clocks

Problem: With enough time I can guess any key

Solution: Change keys periodically

Problem: Certificates signed by old keys are still signed

Solution: Put expiration date in each certificate

Problem: If my computer's clock is wrong, I might misunderstand expiration dates

Solution: Refuse to do anything secure if not all party's clocks agree

Part 2: goals of security

What does it mean to be “insecure”?

Definitions

Confidentiality
Only the intended recipients receive each message

Integrity
Messages are not modified during transit

Availability
The service is available and responsive when you need it

Authentication
Each party knows who they are communicating with

Authorization
Only parties that should be allowed to do something are

Least Privilege
A party granted rights to do x should not also get rights to do y

HTTPS uses symmetric-key encryption for both request and response. This provides

A. Confidentiality
B. Integrity
C. Availability
D. Authentication
E. Authorization
F. Least Privilege

clicker.cs.illinois.edu
code 340 #4

HTTPS uses digital certificates, which are documents which say “The legitimate owner of website x has public key y” and are signed by well-known certificate authority.

Digital certificates provide

A. Confidentiality
B. Integrity
C. Availability
D. Authentication
E. Authorization
F. Least Privilege

clicker.cs.illinois.edu
code 340 #5

Our course code submission page uses HTTP’s built-in login system to show you your submitted code and to get files you upload attached to your account.

Login provides

A. Confidentiality
B. Integrity
C. Availability
D. Authentication
E. Authorization
F. Least Privilege

clicker.cs.illinois.edu
code 340 #6

Part 3: security in practice

HTTPS

1. Open a socket (a TCP connection)
2. Client sends a set of supported cipher suites
3. Server picks one and tells client to use it
4. Server sends its certificate (and thus public key) to the client
5. Client uses server's public key to encrypt a Diffie-Helman Key Exchange
6. Communicate via HTTP using a symmetric cipher

Provides:

- Confidentiality – symmetric key encryption
- Integrity – symmetric key encryption, TCP packet order
- Authentication of Server – by certificate

HTTPS Connection Errors

- Clock misalignment
- Expired certificate
- No-longer-secure cipher suite
- Certificate for wrong website (such as certificate for `cs.illinois.edu` sent when visiting `siebelschool.illinois.edu`)
- Certificate signed by untrusted authority (such as the server's own key)

Unix-style permissions

> *Note* — UNIX™ was an operating system written in C by the inventor of C (among others) based on the earlier Multics operating system, with support from AT&T and Bell Labs.
>
> Linux, MacOS, Android, and iOS are all directly descended from UNIX, and Windows has many UNIX-inspired components.

- Every process, file, and directory has an owning User.
    - Every file and directory has an owning Group.
    - Each process belongs to all Groups that its owning User belongs to.
- One user is special: the “super user”, often named `root`, has all permissions
- The log-in screen is a process run by the super user
    - **Authenticates** a user
      then spawns the operating system’s user interface (a process) as that user.
- Files and Directories each have a 9-bit permission flag.
    - Provides **authorization**

User, Group, Other

- The 9-bit permission flag is divided into three 3-bit sub-flags:
  `uuugggooo`
- The three `u` (user) bits are used if:
  the process and the file/directory have the **same owning user**.
- Otherwise, the three `g` (group) bits are used if:
  the process's user is **a member of** the file/directory's **owning group**.
- Otherwise, the three `o` (other) bits are used.

File permissions

Each file+process pair selects a 3-bit flag: `rwx`

- If `r` (read) is 1, then the process is allowed to access the bytes inside the file.
- If `w` (write) is 1, then the process is allowed to change the bytes inside the file.
- If `x` (execute) is 1, then the process is allowed to run the program expressed by the bytes inside the file.

  `x` without `r` is meaningless: you're allowed to run the program
  but can't find out enough about the program to run it.

Commonly, these bits are represented with letters if the bit is 1 and hyphens if the bit is 0.

> *Example* — `rwxr-xr-x` means
>
> - The user has all three permissions (`r`, `w`, and `x`)
> - The group and others have two of the three (`r` and `x`, but not `w`)

Suppose I'm logged in as user `luthert`, which is a member of three groups: `luthert`, `csvm340-cls`, and `csvm340-stf`.

I use `ls -l` to show the following files. Which ones could I run by typing `./prog`[ABCDE]?

A. `rwxrwxrwx  luthert   luthert      progA`

B. `r-xrwxrwx  drschatz  drschatz     progB`

C. `rwxr-xr-x  drschatz  csvm340-stf  progC`

D. `r-x---r-x  drschatz  csvm340-stf  progD`

E. `rwx---r-x  drschatz  drschatz     progE`

clicker.cs.illinois.edu
code 340 #7

Directory permissions

In UNIX, the file system forms a tree*, with interior nodes called directories.

Files don't have names: edges in the tree do. That a file is named "README" is a property of the directory that contains the file, *not* of the file itself.

Each directory+process pair selects a 3-bit flag: `rwx`

- If `r` (read) is 1, then the process is allowed to find out what children this node has.
  - `ls /some/path` *reads* that node and shows the results.
- If `w` (write) is 1, then the process is allowed to change what children this node has.
  - Creating, renaming, and removing files are all *write* operations on those files' containing directory.
- If `x` (execute) is 1, then the process is allowed to visit a given child.
  - `x` without `r` means you can visit children only if you already know their names.
  - You can't do *anything* to a node you cannot visit.

Suppose I'm logged in as user `luthert`, which is a member of three groups: `luthert`, `csvm340-cls`, and `csvm340-stf`.

Suppose that `ls` shows the following permissions and owners:

```
drwxr-xr-x  root      root      /
drwxr-xr-x  root      root      /home
drwxr-xr-x  drschatz  drschatz  /home/drschatz
drwxr-xr--  drschatz  drschatz  /home/drschatz/code
drwx--x--x  drschatz  drschatz  /home/drschatz/code/mp8
-rwxr-xr-x  drschatz  drschatz  /home/drschatz/code/mp8/filter
```

clicker.cs.illinois.edu
code 340 #8

Why can't I run `/home/drschatz/code/mp8/filter`?

A. I'd need to use `./filter` not `/home/drschatz/code/mp8/filter`

B. I'm not `drschatz` so I can't use `drschatz`'s files

C. I don't have access to `/` or anything inside it

D. I don't have `x` permission on `/home/drschatz/code`

E. I don't have `r` permission on `/home/drschatz/code/mp8`

F. I don't have `x` permission on `/home/drschatz/code/mp8/filter`

File permissions and least privilege

Least Privilege
A party granted rights to do x should not also get rights to do y

Can Unix-style permissions be used to implement the principle of least privilege (w.r.t. files)?

- If **yes**, explain scenario with all needed, no unneeded rights
- If **no**, explain scenario where needed rights come with unneeded rights

Example scenarios:

- All CS faculty and student have accounts on shared computer
- Need to run an application but don't trust its creator to be non-malicious
- Shared file server for 100-employee tech company

`sudo`, root, and kernel mode

- Permissions are handled by
  1. User code executes a `syscall`, switching to kernel mode
  2. Kernel code compares the process's owner and the requested action to the permission bits
  3. If permitted, kernel does the action
  4. Kernel switches back to user mode and code
- For the super user `root`, step 2 always resolves to "permitted"
- `su` and `sudo` request changing the user of the current (`su`) or new (`sudo`) process
  - Some users have this permission, others do not

Learning objectives

1. Continuing last class
    - More on ciphers and hashes
    - Understand what “cryptographically secure” means
2. Understand goals of security
    - CIA Triad (Confidentiality, Integrity, Availability)
    - Authentication, Authorization, and Least Privilege
3. How security shows up in various contexts
    - HTTPS
    - Unix-style permissions
    - Blockchain (next class)