2550 Intro to cybersecurity

L13: Authorization

abhi shelat

Thanks Christo for slides!

Authentication:

Authorization

After Authenticating a subject, what next?

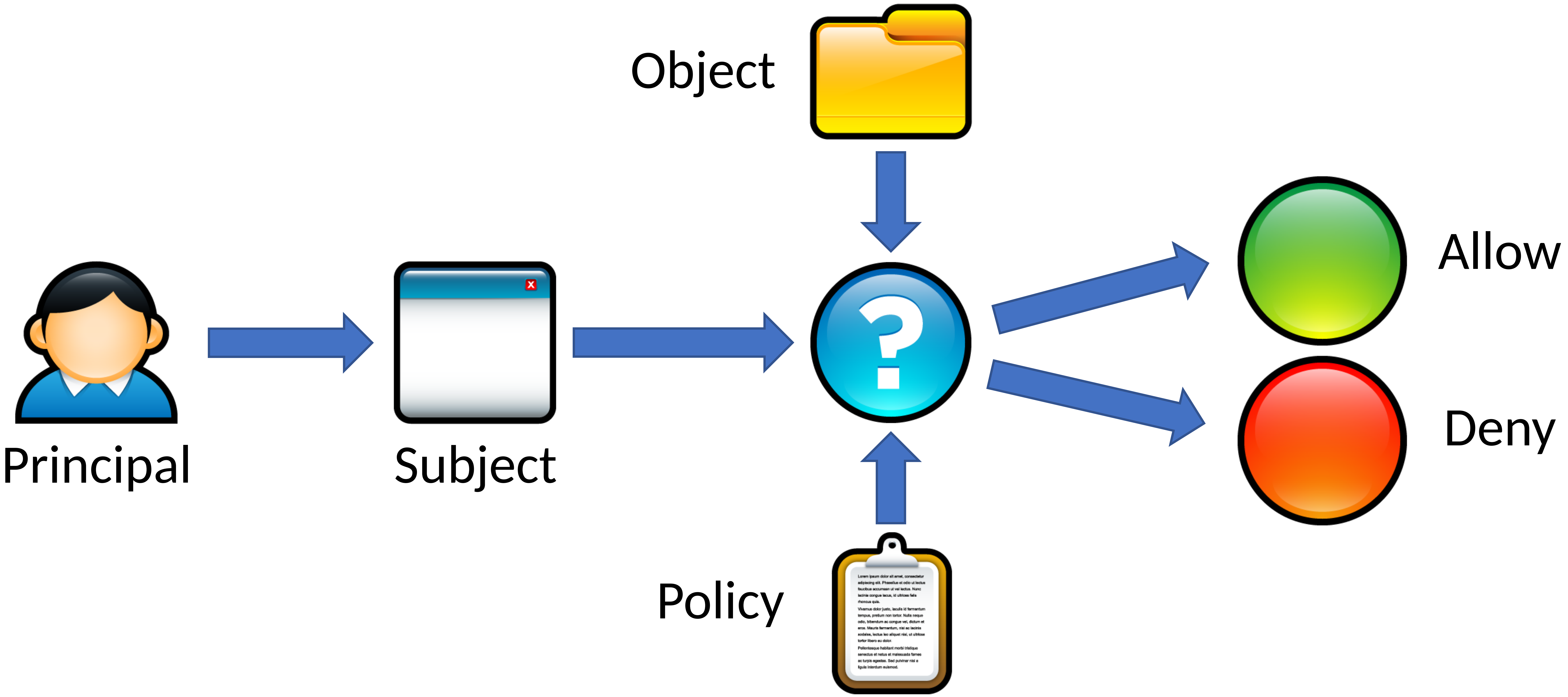
Access Control

- Policy specifying how entities can interact with resources
  - i.e., Who can access what?
  - Requires authentication and authorization
- Access control primitives

Principal	User of a system	
Subject	Entity that acts on behalf of principals	Software program
Object	Resource acted upon by subjects	Files Sockets Devices OS APIs

Access Control Check

- Given an access request from a subject, on behalf of a principal, for an object, return an access control decision based on the policy



Access Control Models

- Discretionary Access Control (DAC)
  - The kind of access control you are familiar with
  - Access rights propagate and may be changed at subject’s discretion

- Mandatory Access Control (MAC)
  - Access of subjects to objects is based on a system-wide policy
  - Denies users full control over resources they create

Discretionary Access Control

Access Control Matrices

Access Control Lists

Unix Access Control

Discretionary Access Control

- According to Trusted Computer System Evaluation Criteria (TCSEC)

"A means of restricting access to objects based on the identity and need-to-know of users and/or groups to which they belong.

Controls are discretionary in the sense that a subject with a certain access permission is capable of passing that permission (directly or indirectly) to any other subject."

Access Control Matrices

Given subjects $s_i \in S$, objects $o_j \in O$, rights {**R**ead, **W**rite, e**X**ecute},

	o_1	o_2	o_3
s_1	RW	RX	
s_2	R	RWX	RW
s_3		RWX	

- Introduced by Lampson in 1971
- Static description of protection state
- Abstract model of concrete systems

Access Control List (ACL)

- Each object has an associated list of subject→operation pairs
- Authorization verified for each request by checking list of tuples
- Used pervasively in filesystems and networks
  - "Users a, b, and c and read file x."
  - "Hosts a and b can listen on port x."

	o_1	o_2
s_1	RW	RX
s_2	R	RWX
s_3		RWX

Access Control List (ACL)

- Each object has an associated list of subject→operation pairs
- Authorization verified for each request by checking list of tuples
- Used pervasively in filesystems and networks
  - "Users a, b, and c and read file x."
  - "Hosts a and b can listen on port x."

ACL for o_2

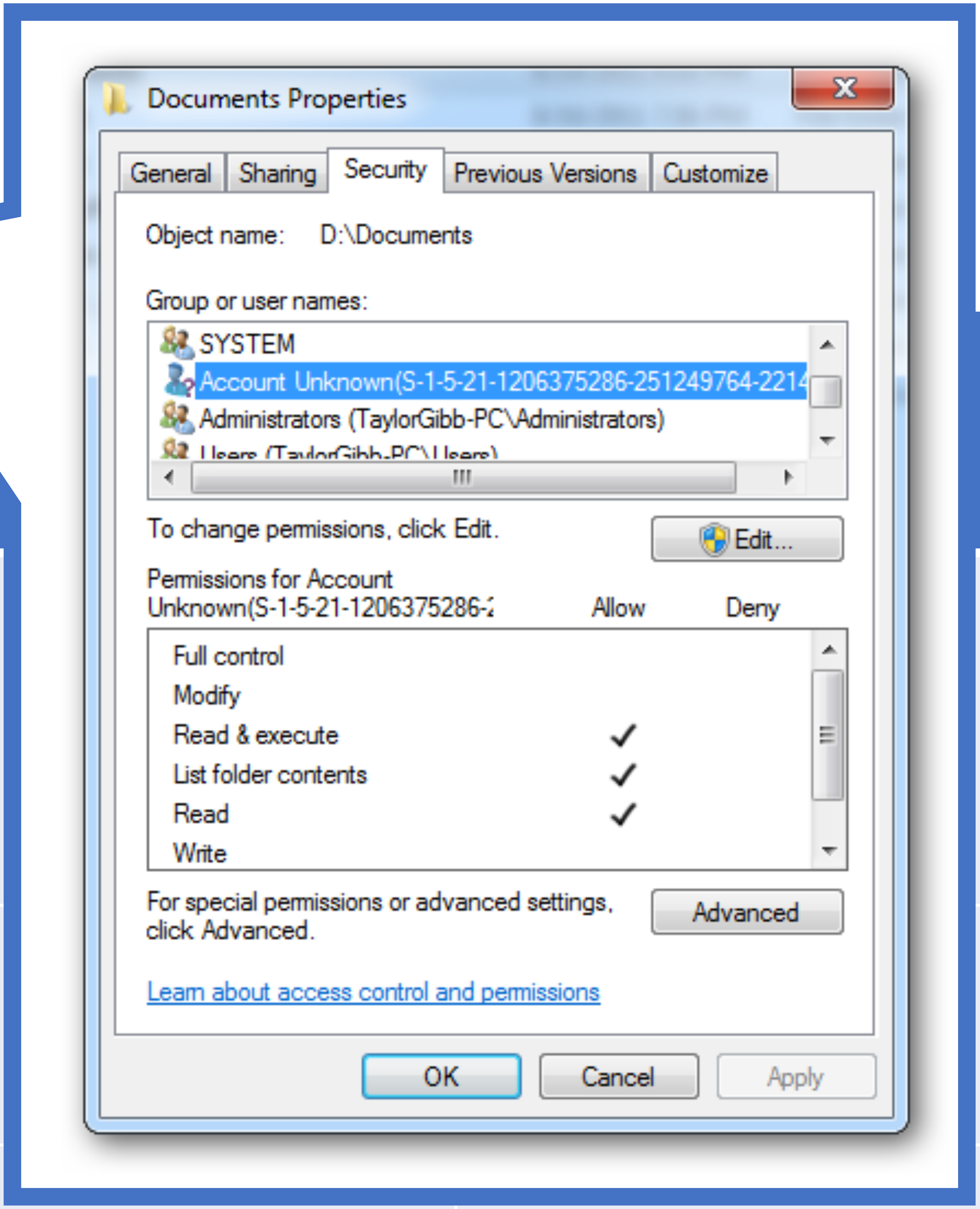
	o_1	o_2
s_1	RW	RX
s_2	R	RWX
s_3		RWX

Windows ACLs

	D:\Music	D:\Images
System	RWX	RWX
Administrators	RW	RW

Windows ACLs

	D:\Music
System	RWX
Administrators	RW



Object name: D:\Documents

Group or user names:

- SYSTEM
- Account Unknown(S-1-5-21-1206375286-251249764-2214
- Administrators (TaylorGibb-PC\Administrators)
- Users (TaylorGibb-PC\Users)

To change permissions, click Edit.

Permissions for Account Unknown(S-1-5-21-1206375286-2	Allow	Deny
Full control		
Modify		
Read & execute	✓	
List folder contents	✓	
Read	✓	
Write		

For special permissions or advanced settings, click Advanced.

Learn about access control and permissions

ACL Review

The Good

- Very flexible
  - Can express any possible access control matrix
  - Any principal can be configured to have any rights on any object

The Bad

ACL Review

The Good

- Very flexible
  - Can express any possible access control matrix
  - Any principal can be configured to have any rights on any object

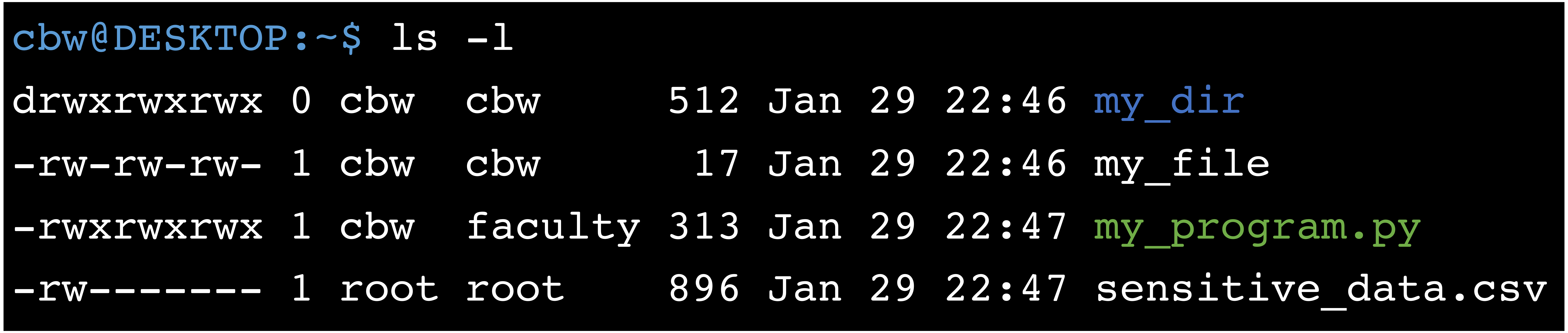
The Bad

- Complicated to manage
  - Every object can have wildly different policies
  - Infinite permutations of subjects, objects, and rights

Unix-style Permissions

- Based around the concept of owners and groups
  - All objects have an owner and a group
  - Permissions assigned to owner, group, and everyone else
- Authorization verified for each request by mapping the subject to owner, group, or other and checking the associated permissions

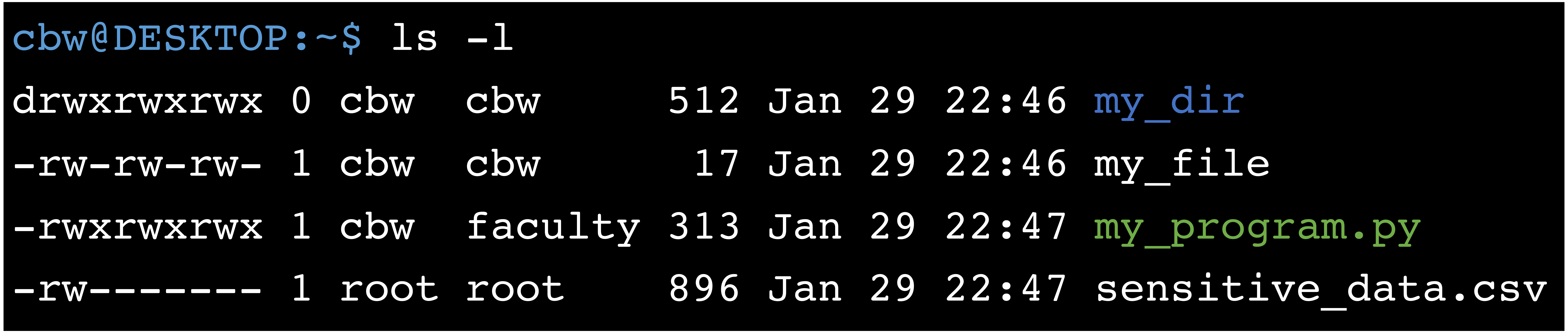
Unix Permissions



```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw     512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
```

d → Directory   **r → Read**   **w → Write**   **x → eXecute**

Unix Permissions

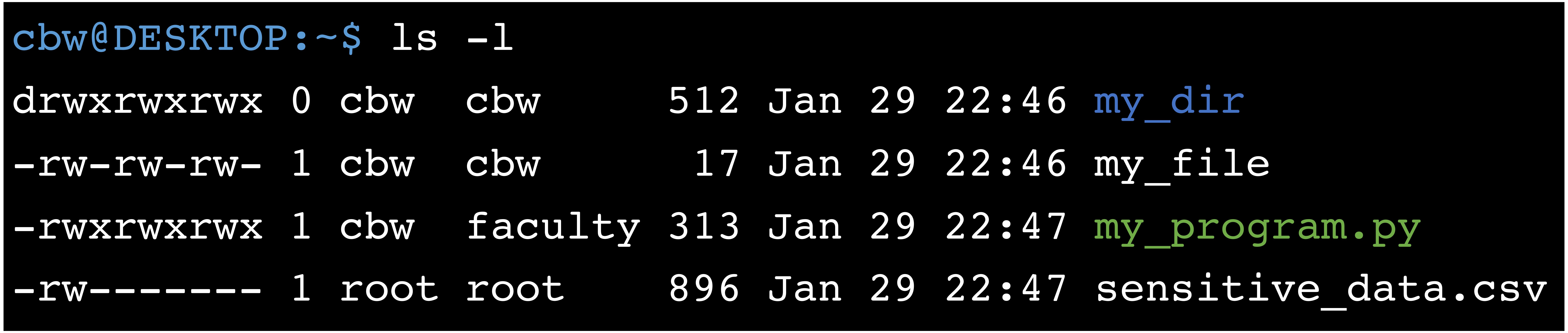


```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw      512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw       17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root     896 Jan 29 22:47 sensitive_data.csv
```

Owner

d →　Directory　　r →　Read　w →　Write　x →　eXecute

Unix Permissions



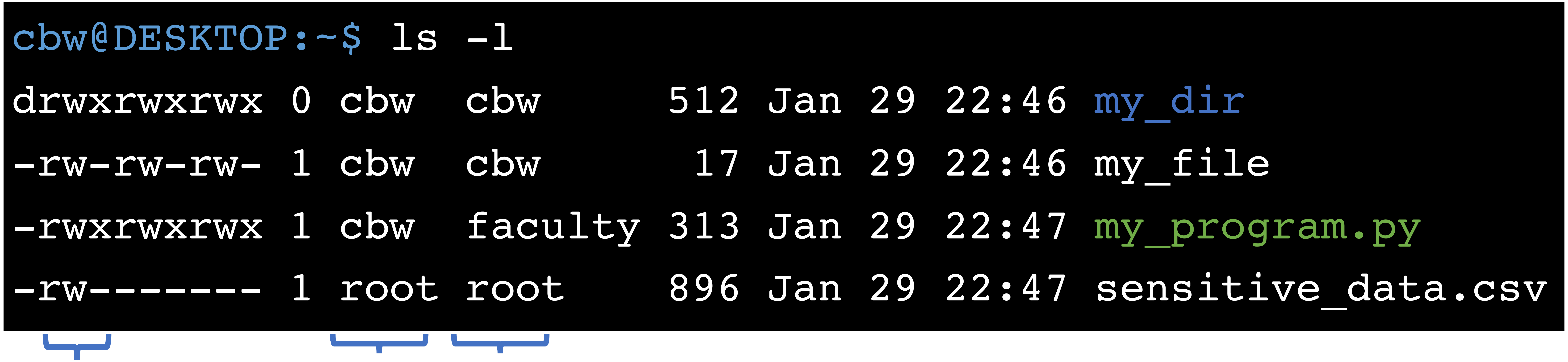
```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw     512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
```

Owner

Owner

d → **Directory** **r →** **Read** **w →** **Write** **x →** **eXecute**

Unix Permissions



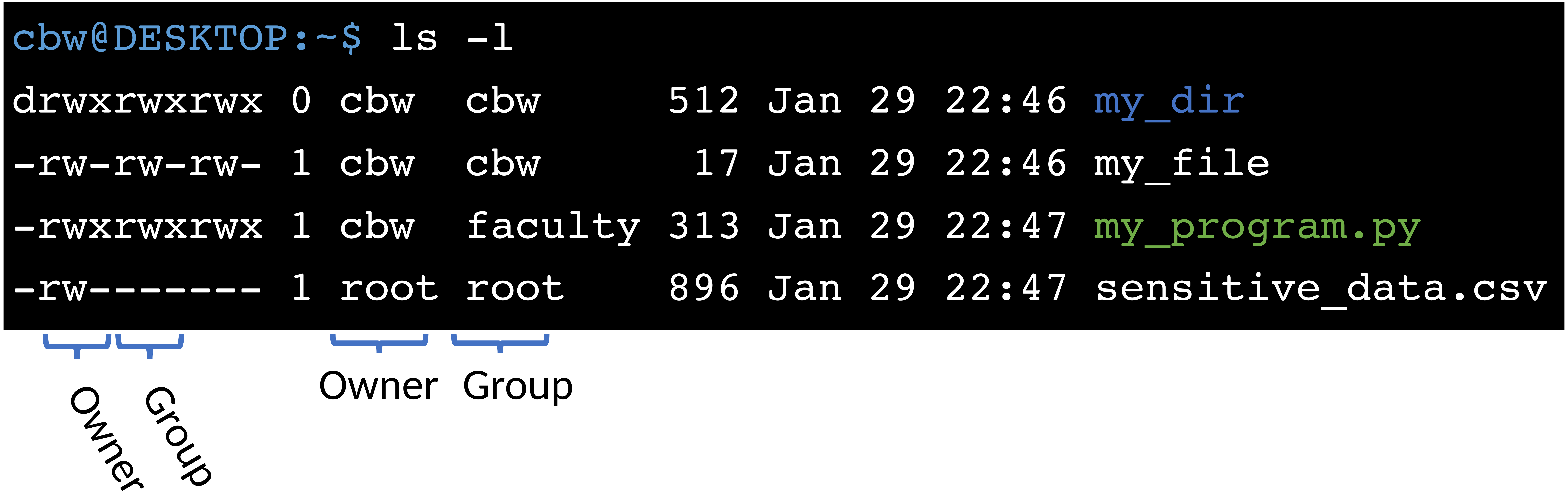
```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw      512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw       17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root     896 Jan 29 22:47 sensitive_data.csv
```

Owner

Owner Group

d → Directory   **r → Read**   **w → Write**   **x → eXecute**

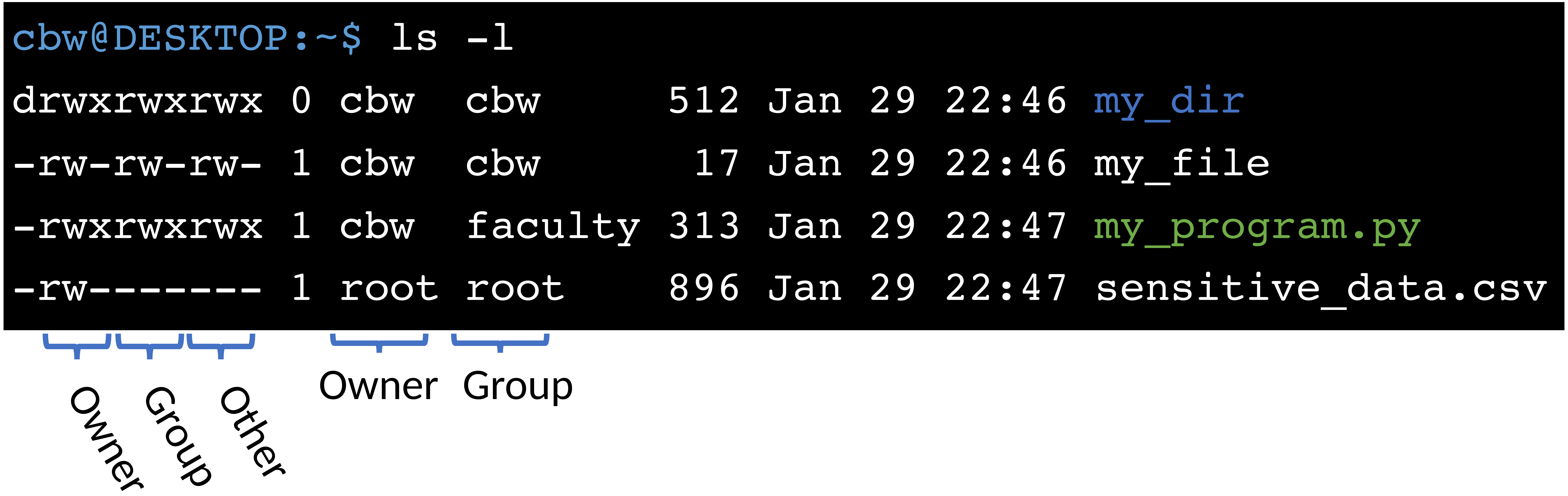
Unix Permissions



```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw     512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
```

d → Directory   r → Read   w → Write   x → eXecute

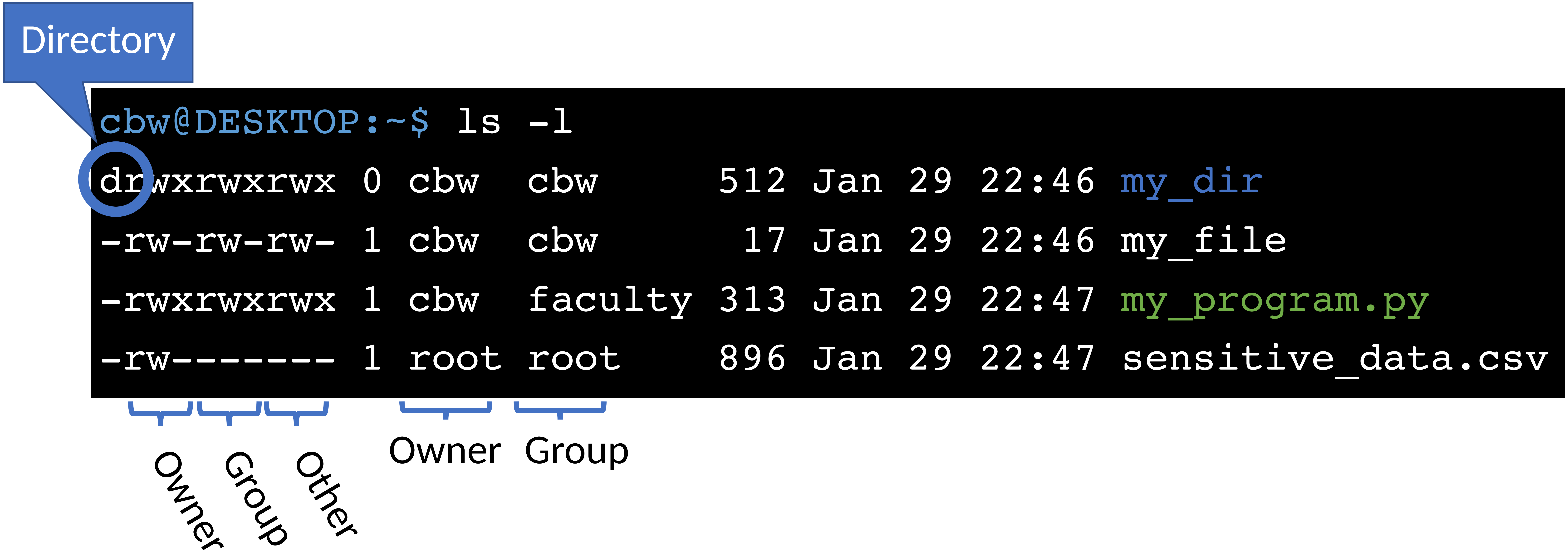
Unix Permissions



```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw     512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
```

d → Directory **r → Read** **w → Write** **x → eXecute**

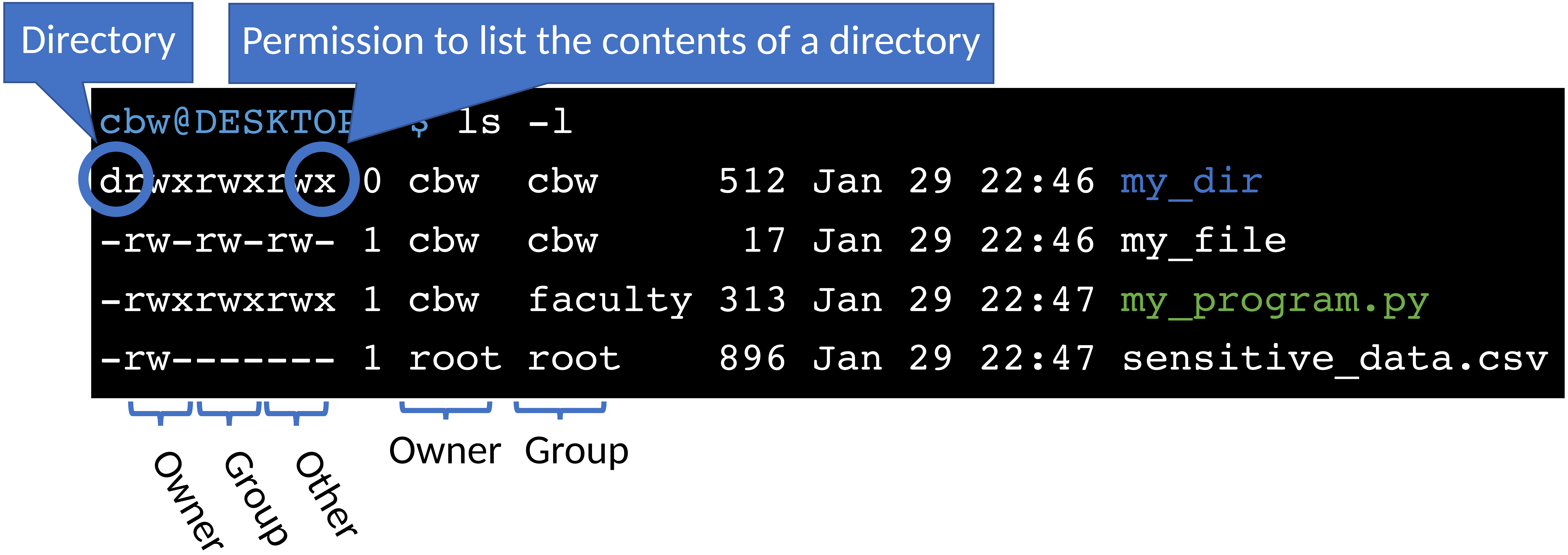
Unix Permissions



```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw      512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw       17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty  313 Jan 29 22:47 my_program.py
-rw------- 1 root root     896 Jan 29 22:47 sensitive_data.csv
```

d → **Directory** **r →** **Read** **w →** **Write** **x →** **eXecute**

Unix Permissions



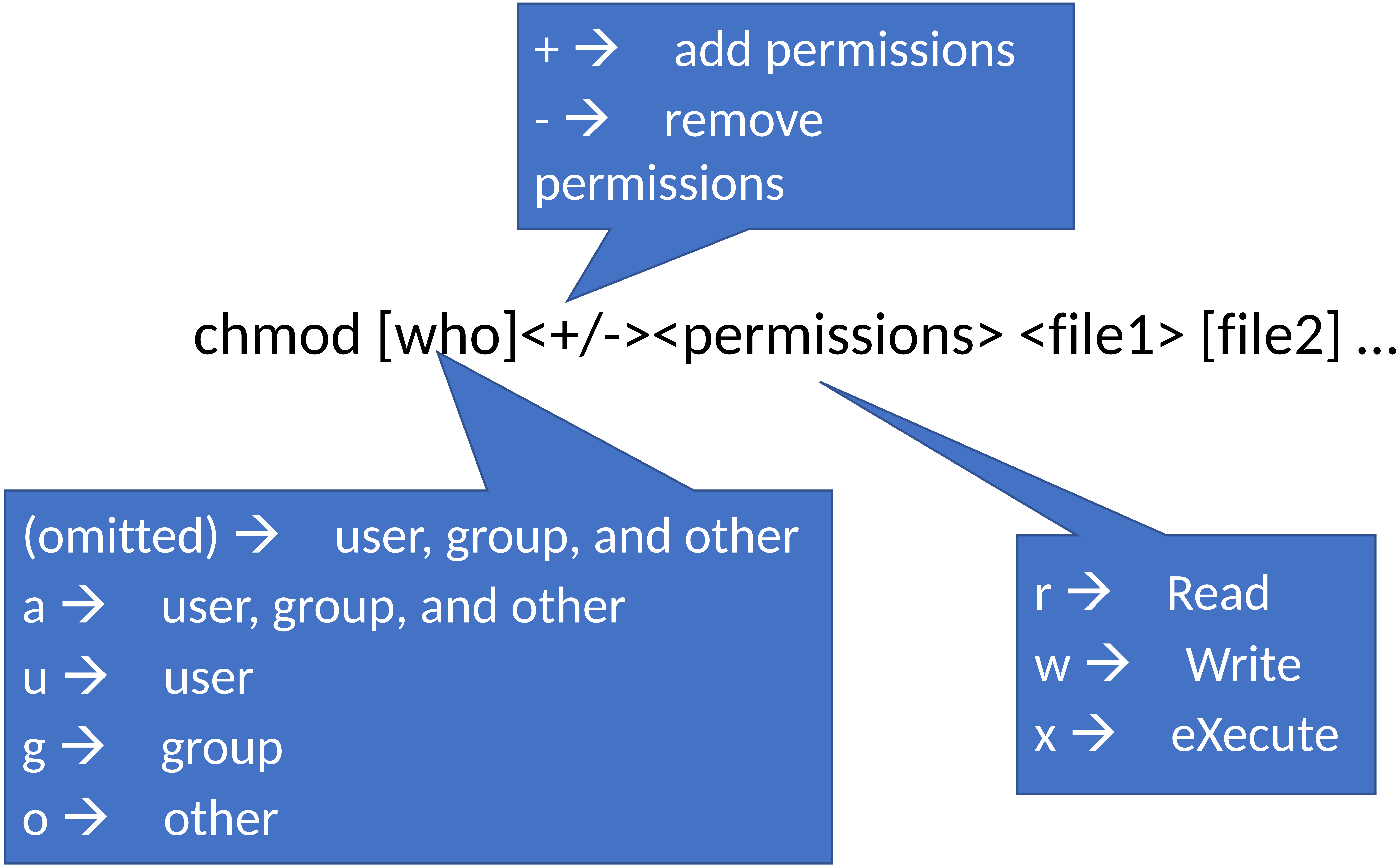
Directory

Permission to list the contents of a directory

```
cbw@DESKTOP   $ ls -l
drwxrwxrwx 0 cbw  cbw      512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw       17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
-rw------- 1 root root     896 Jan 29 22:47 sensitive_data.csv
```

d → Directory **r → Read** **w → Write** **x → eXecute**

Setting Permissions



+ → add permissions

- → remove permissions

```
chmod [who]<+/-><permissions> <file1> [file2] …
```

(omitted) → user, group, and other

a → user, group, and other

u → user

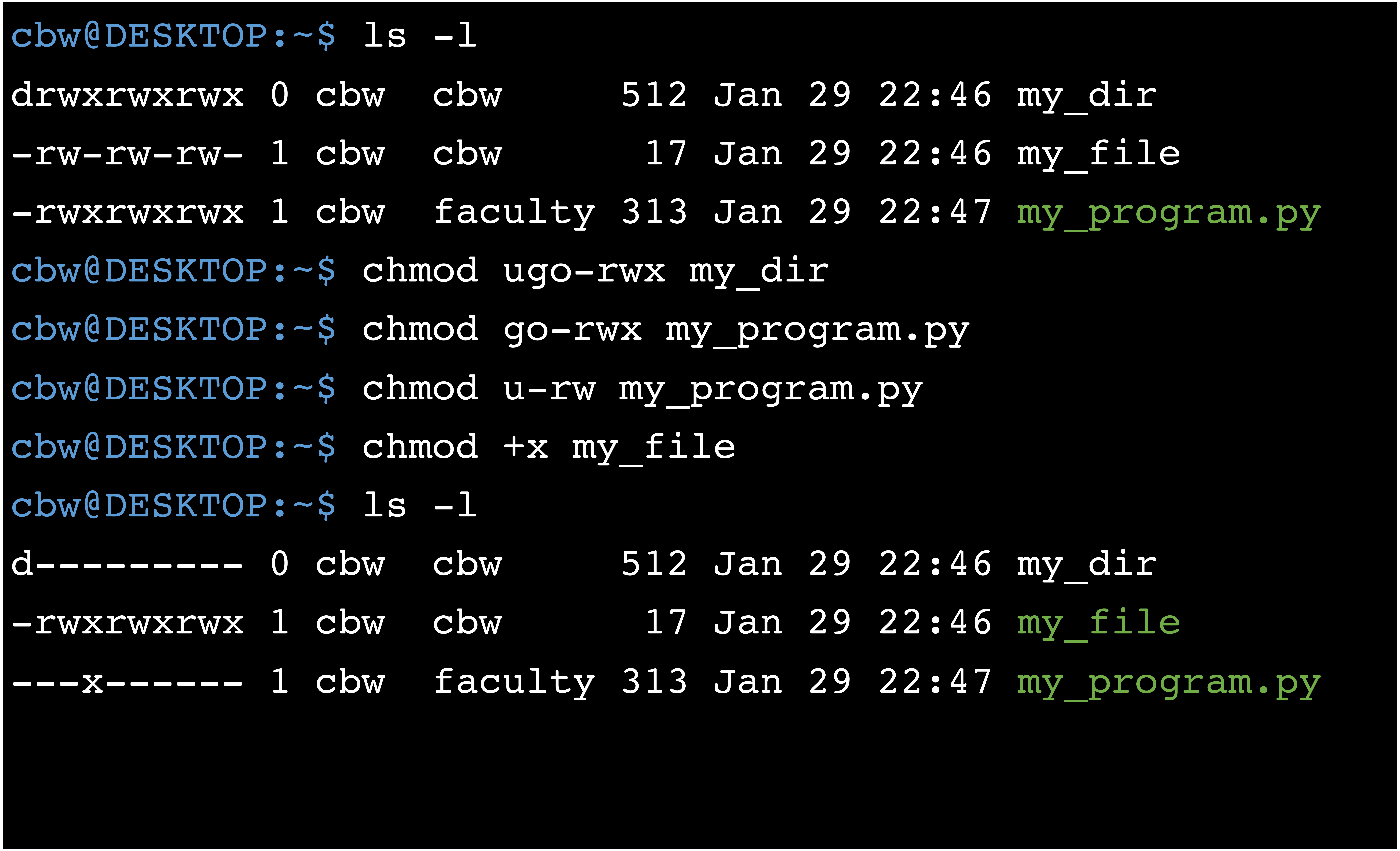
g → group

o → other

r → Read

w → Write

x → eXecute



```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw     512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
cbw@DESKTOP:~$ chmod ugo-rwx my_dir
cbw@DESKTOP:~$ chmod go-rwx my_program.py
cbw@DESKTOP:~$ chmod u-rw my_program.py
cbw@DESKTOP:~$ chmod +x my_file
cbw@DESKTOP:~$ ls -l
d--------- 0 cbw  cbw     512 Jan 29 22:46 my_dir
-rwxrwxrwx 1 cbw  cbw      17 Jan 29 22:46 my_file
---x------ 1 cbw  faculty 313 Jan 29 22:47 my_program.py
```

Alternate Form of Setting Permissions

chmod ### <file1> [file2] …

- #s correspond to owner, group, and other
- Each value encodes read, write, and execute permissions
  - 1 → execute
  - 2 → write
  - 4 → read

Alternate Form of Setting Permissions

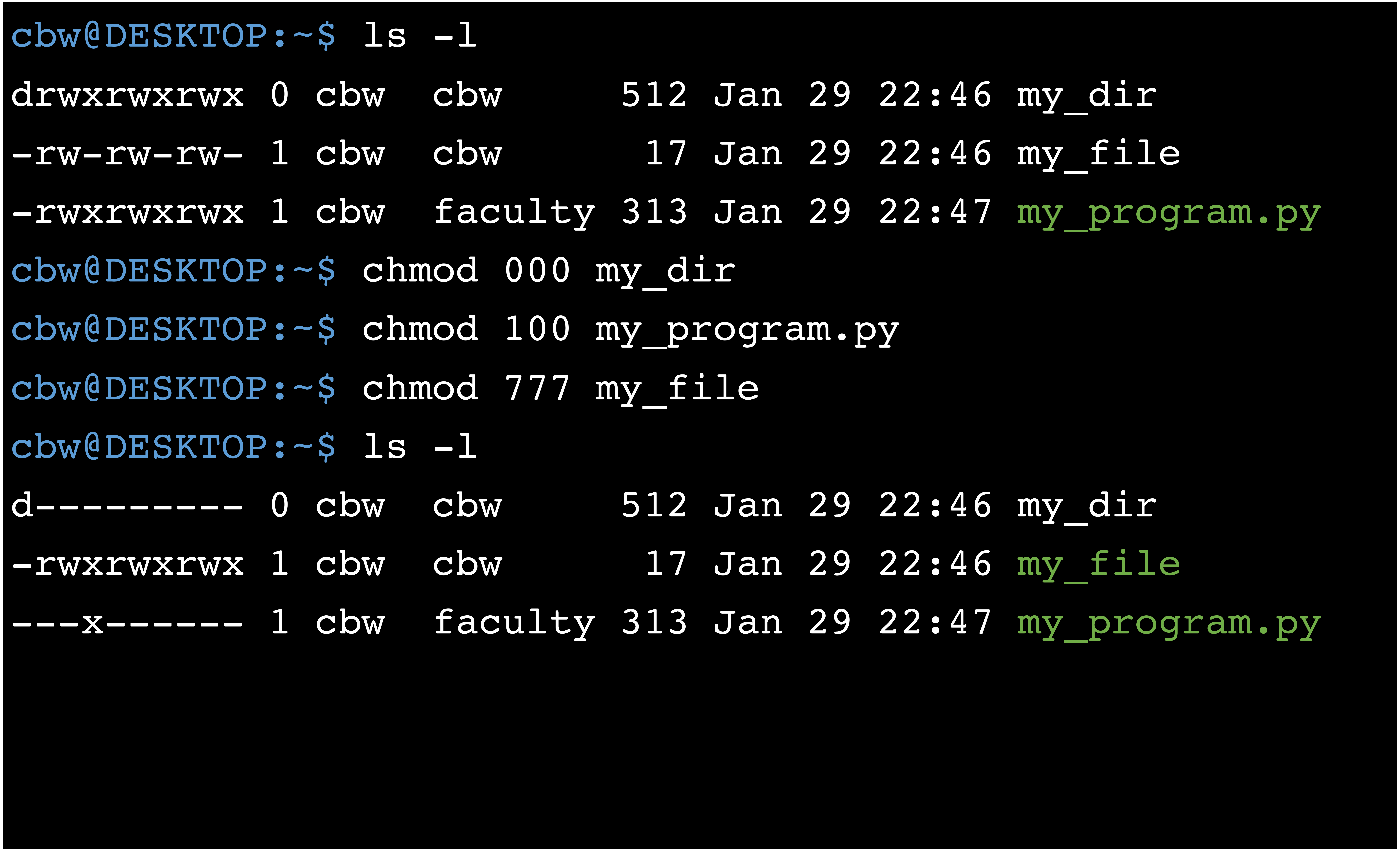
chmod ### <file1> [file2] …

- #s correspond to owner, group, and other
- Each value encodes read, write, and execute permissions
  - 1 → execute
  - 2 → write
  - 4 → read
- What if you want to set something as read, write, and execute?

Alternate Form of Setting Permissions

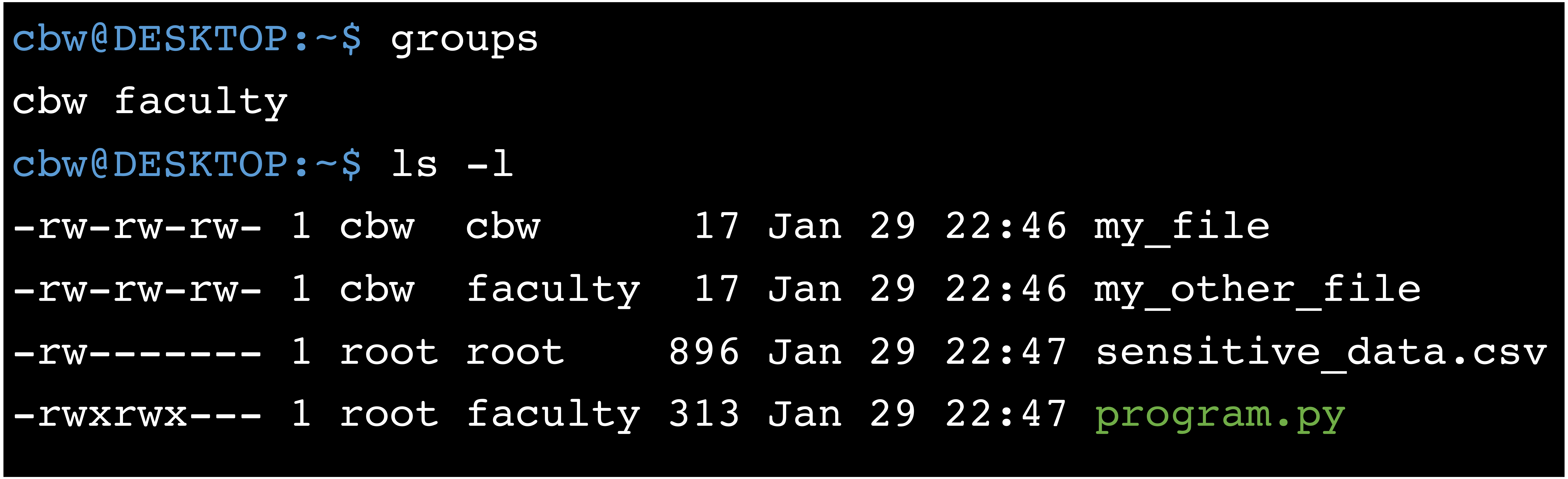
chmod ### <file1> [file2] …

- #s correspond to owner, group, and other
- Each value encodes read, write, and execute permissions
  - 1 → execute
  - 2 → write
  - 4 → read
- What if you want to set something as read, write, and execute?
  - 1 + 2 + 4 = 7



```
cbw@DESKTOP:~$ ls -l
drwxrwxrwx 0 cbw  cbw     512 Jan 29 22:46 my_dir
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rwxrwxrwx 1 cbw  faculty 313 Jan 29 22:47 my_program.py
cbw@DESKTOP:~$ chmod 000 my_dir
cbw@DESKTOP:~$ chmod 100 my_program.py
cbw@DESKTOP:~$ chmod 777 my_file
cbw@DESKTOP:~$ ls -l
d--------- 0 cbw  cbw     512 Jan 29 22:46 my_dir
-rwxrwxrwx 1 cbw  cbw      17 Jan 29 22:46 my_file
---x------ 1 cbw  faculty 313 Jan 29 22:47 my_program.py
```

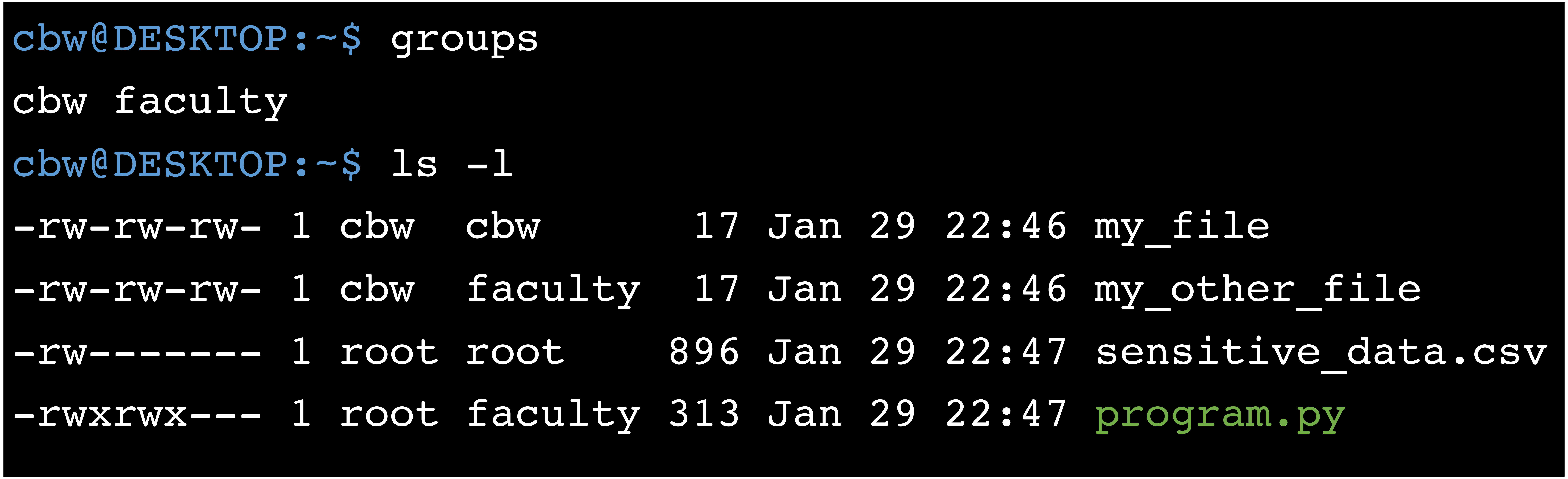
Who May Change Permissions?



```
cbw@DESKTOP:~$ groups
cbw faculty
cbw@DESKTOP:~$ ls -l
-rw-rw-rw- 1 cbw  cbw       17 Jan 29 22:46 my_file
-rw-rw-rw- 1 cbw  faculty   17 Jan 29 22:46 my_other_file
-rw------- 1 root root     896 Jan 29 22:47 sensitive_data.csv
-rwxrwx--- 1 root faculty 313 Jan 29 22:47 program.py
```

- Which files is user *cbw* permitted to *chmod*?

Who May Change Permissions?



```
cbw@DESKTOP:~$ groups
cbw faculty
cbw@DESKTOP:~$ ls -l
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rw-rw-rw- 1 cbw  faculty  17 Jan 29 22:46 my_other_file
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
-rwxrwx--- 1 root faculty 313 Jan 29 22:47 program.py
```

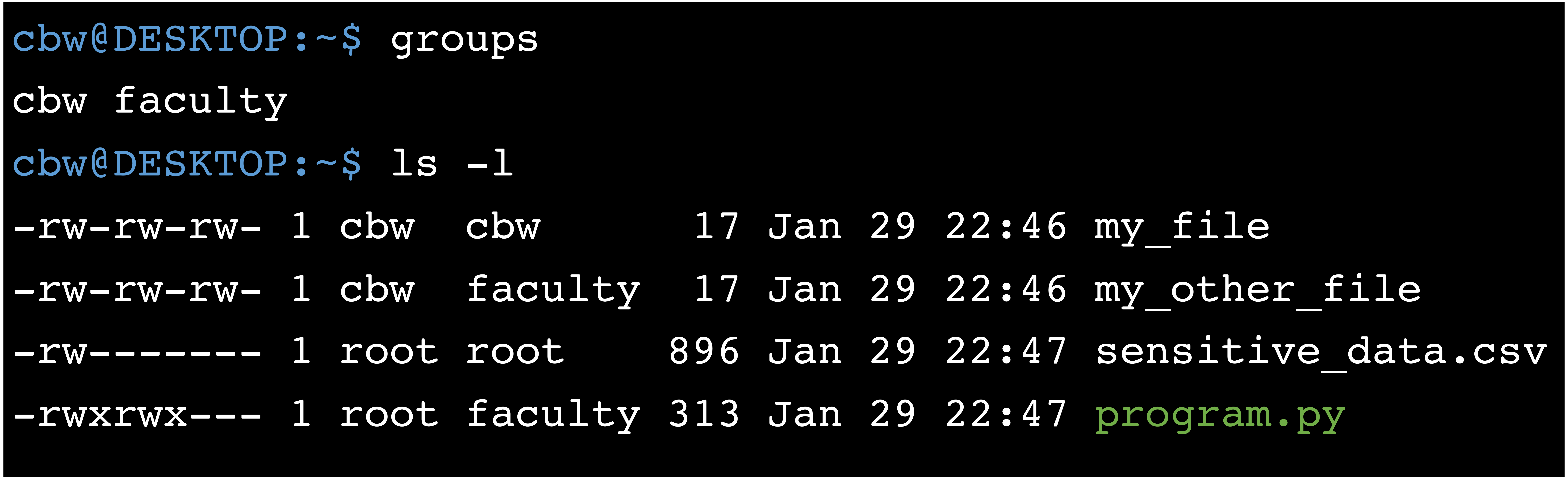
- Which files is user *cbw* permitted to *chmod*?
  - Only owners can *chmod* files
  - *cbw* can *chmod my_file* and *my_other_file*
  - Group membership doesn’t grant *chmod* ability (cannot *chmod program.py*)

Setting Ownership

- Unix uses discretionary access control
  - New objects are owned by the subject that created them
- How can you modify the owner or group of an object?

```
chown <owner>:<group> <file1> [file2] …
```

Who May Change Ownership?



```
cbw@DESKTOP:~$ groups
cbw faculty
cbw@DESKTOP:~$ ls -l
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rw-rw-rw- 1 cbw  faculty  17 Jan 29 22:46 my_other_file
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
-rwxrwx--- 1 root faculty 313 Jan 29 22:47 program.py
```

- Which operations are permitted?

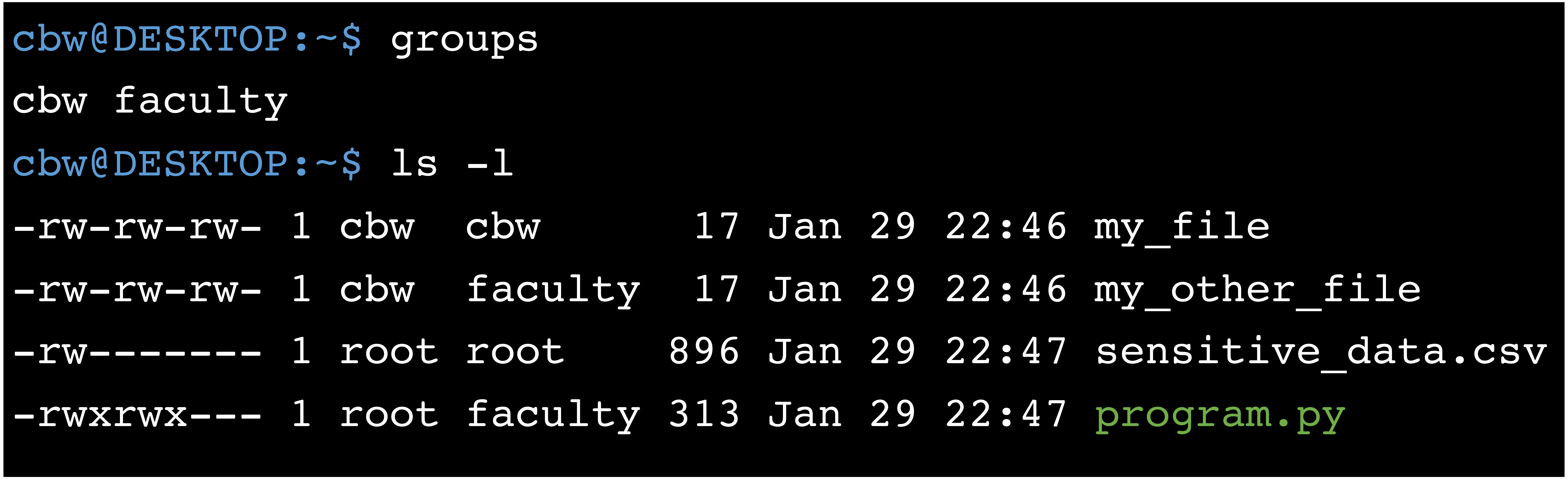
chown cbw:faculty my_file

chown root:root my_other_file

chown cbw:cbw sensitive_date.csv

chown cbw:faculty program.py

Who May Change Ownership?



```
cbw@DESKTOP:~$ groups
cbw faculty
cbw@DESKTOP:~$ ls -l
-rw-rw-rw- 1 cbw  cbw      17 Jan 29 22:46 my_file
-rw-rw-rw- 1 cbw  faculty  17 Jan 29 22:46 my_other_file
-rw------- 1 root root    896 Jan 29 22:47 sensitive_data.csv
-rwxrwx--- 1 root faculty 313 Jan 29 22:47 program.py
```

- Which operations are permitted?

chown cbw:faculty my_file	Yes, cbw belongs to the faculty group
chown root:root my_other_file	No, only root many change file owners!
chown cbw:cbw sensitive_date.csv	No, only root many change file owners!
chown cbw:faculty program.py	No, only root many change file owners!

Unix Access Control Exercise (1)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file1	file2
user1	r--	rwx
user2	r--	rw-
user3	r--	rw-
user4	rwx	rw-

Unix Access Control Exercise (1)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file1
user1	r--
user2	r--
user3	r--
user4	rwx

User	Groups
user1	user1
user2	user2
user3	user3
user4	user4

```
-rwxr--r-- 1 user4  user4  0 file1
-rwxrw-rw- 1 user1  user1  0 file2
```

Unix Access Control Exercise (2)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file1	file2
user1	r--	--x
user2	r-x	rwx
user3	r-x	r--
user4	rwx	r--

Unix Access Control Exercise (2)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file1
user1	r--
user2	r-x
user3	r-x
user4	rwx

User	Groups
user1	user1
user2	user2, group1
user3	user3, group1, group2
user4	user4, group2

```
-rwxr-xr-- 1 user4  group1  0 file1
-rwxr----x 1 user2  group2  0 file2
```

Unix Access Control Exercise (3)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file 1	file 2
user 1	---	rw-
user 2	r--	r--
user 3	rwx	rwx
user 4	rwx	---

Unix Access Control Exercise (3)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file 1	file 2
user 1	---	rw-
user 2	r--	r--
user 3	rwx	rwx
user 4	rwx	---

- Trick question! This matrix **cannot** be represented

Unix Access Control Exercise (3)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file 1	file 2
user 1	---	rw-
user 2	r--	r--
user 3	rwx	rwx
user 4	rwx	---

- Trick question! This matrix **cannot** be represented
- *file2*: four distinct privilege levels
  - Maximum of three levels (user, group, other)

Unix Access Control Exercise (3)

- What Unix group and permission assignments satisfy this access control matrix?

Desired Permissions

	file 1	file 2
user 1	---	rw-
user 2	r--	r--
user 3	rwx	rwx
user 4	rwx	---

- Trick question! This matrix **cannot** be represented

- *file2*: four distinct privilege levels
  - Maximum of three levels (user, group, other)

- *file1*: two users have high privileges
  - If *user3* and *user4* are in a group, how to give *user2* read and *user1* nothing?
  - If *user1* or *user2* are owner, they can grant themselve write and execute permissions :(

Unix Access Control Review

The Good

- Very simple model
  - Owners, groups, and other
  - Read, write, execute
- Relatively simple to manage and understand

The Bad

Unix Access Control Review

The Good

- Very simple model
  - Owners, groups, and other
  - Read, write, execute
- Relatively simple to manage and understand

The Bad

- Not all policies can be encoded!
  - Contrast to ACL

Unix Access Control Review

The Good

- Very simple model
  - Owners, groups, and other
  - Read, write, execute
- Relatively simple to manage and understand

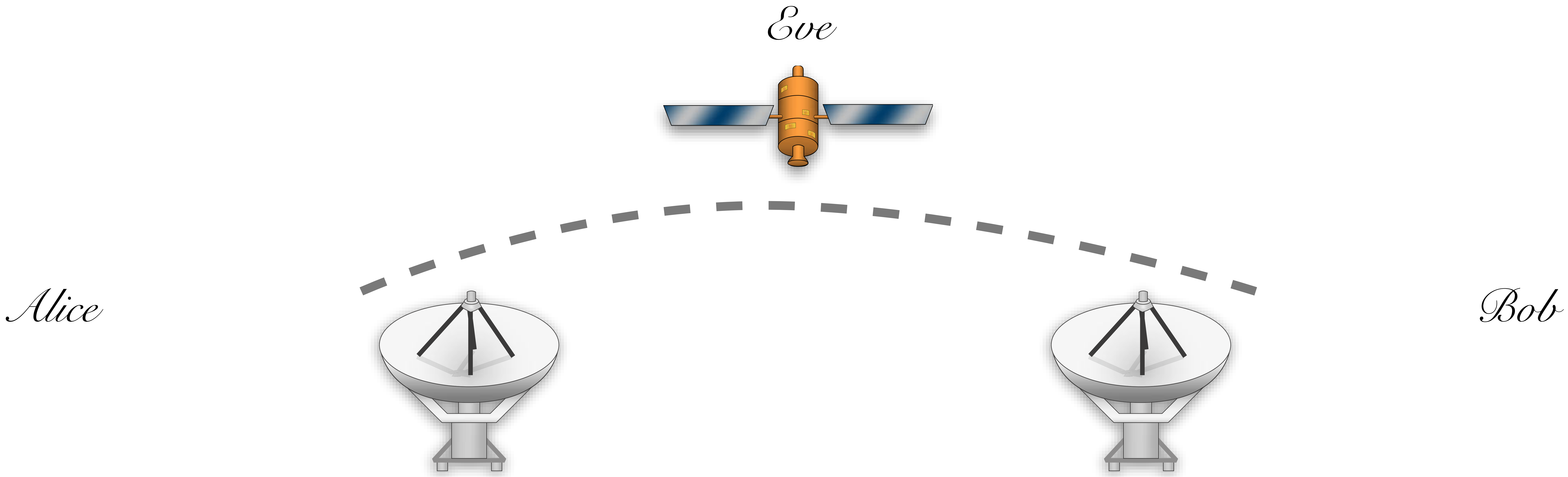
The Bad

- Not all policies can be encoded!
  - Contrast to ACL
- Not quite as simple as it seems
  - setuid

Midterm review

Security modeling

Symmetric Encryption

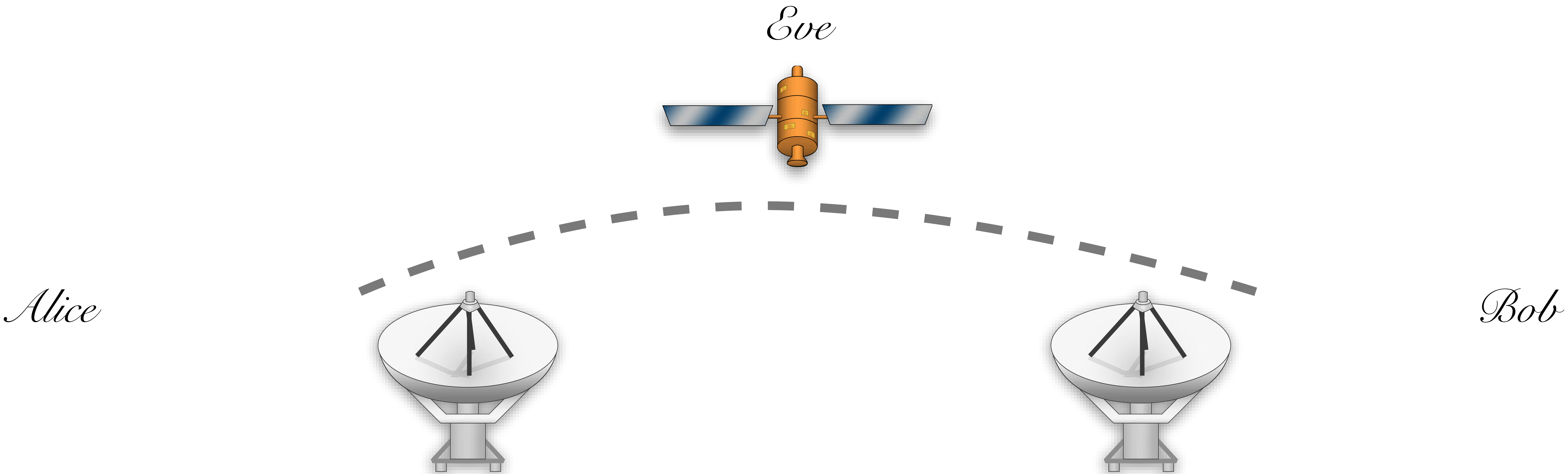


Eve

Alice

Bob

Public Key Encryption

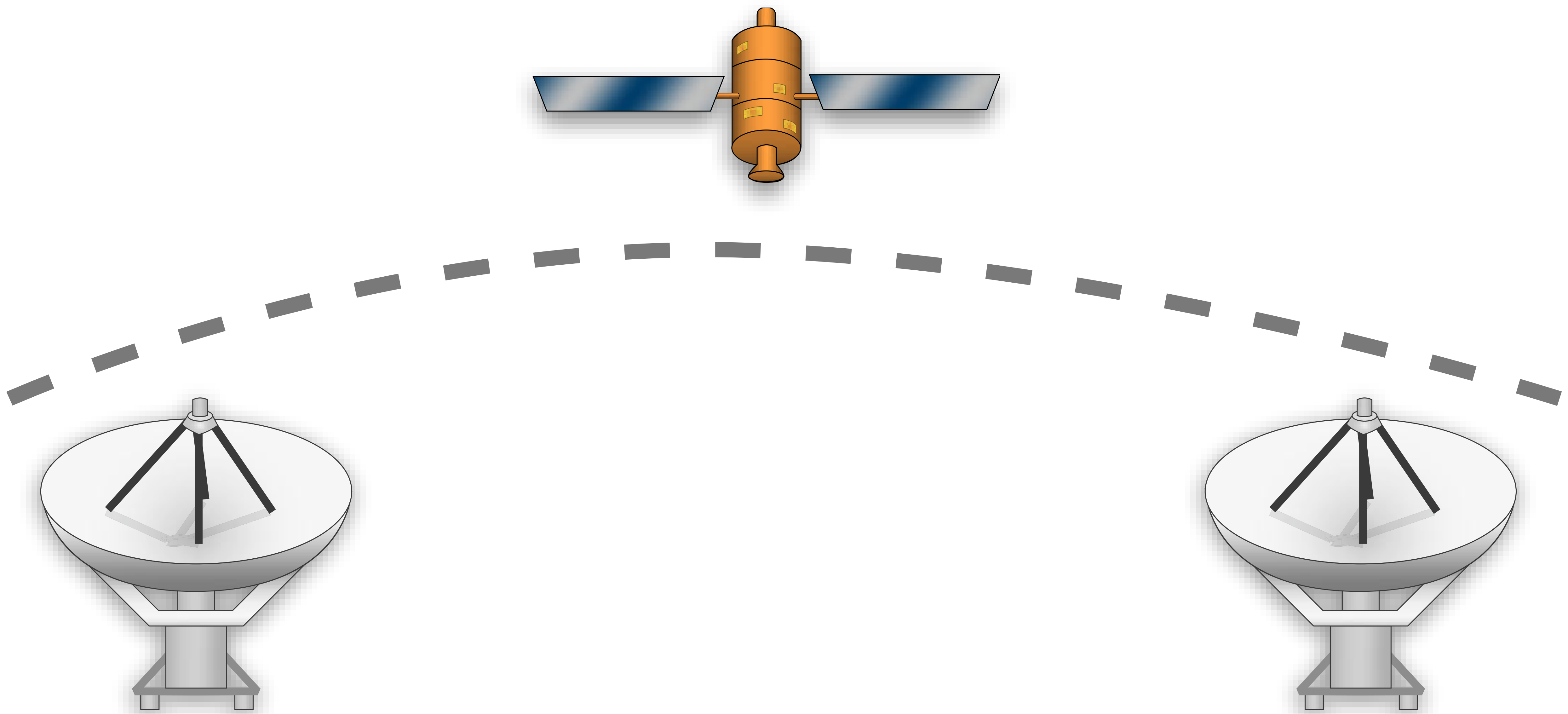


Eve

Alice

Bob

MAC

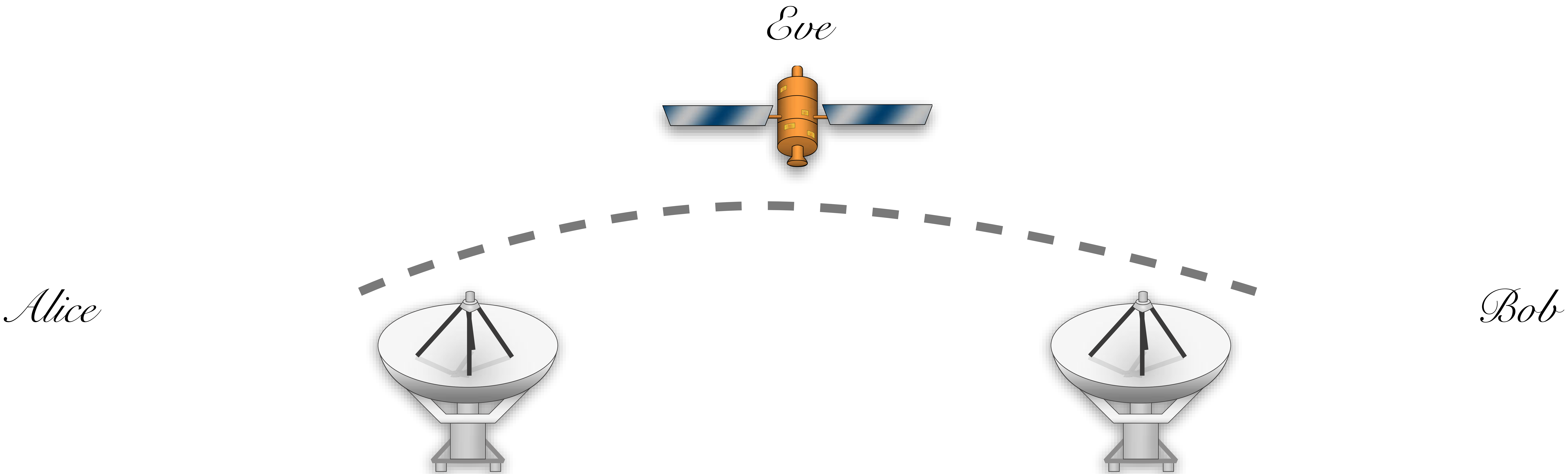


Eve

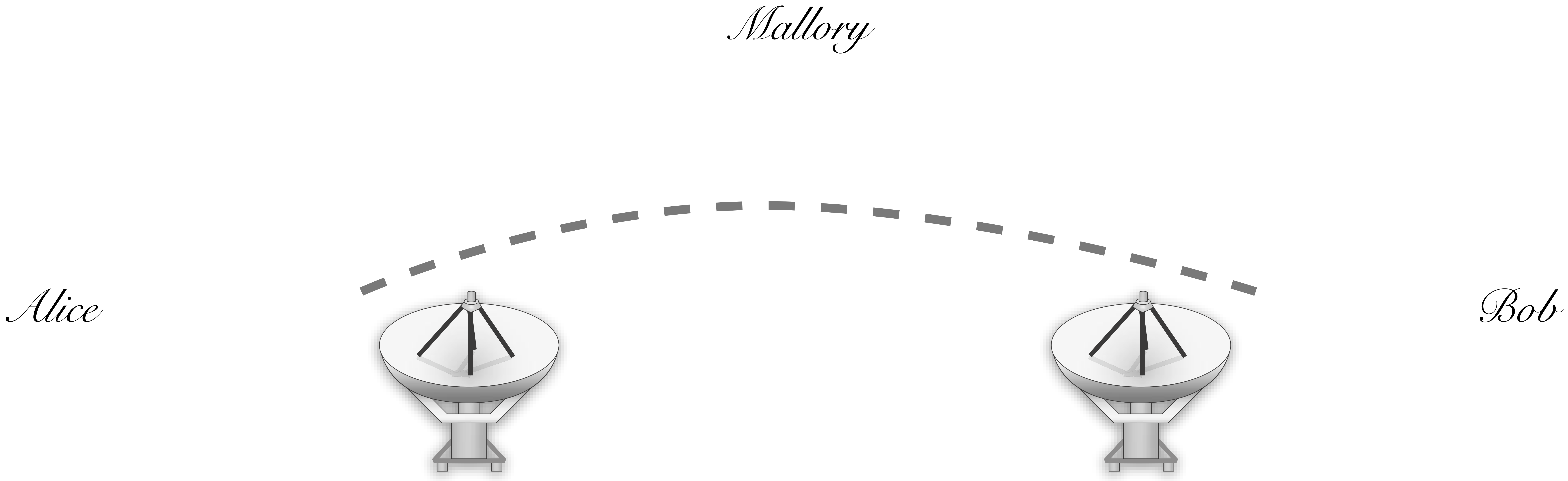
Alice

Bob

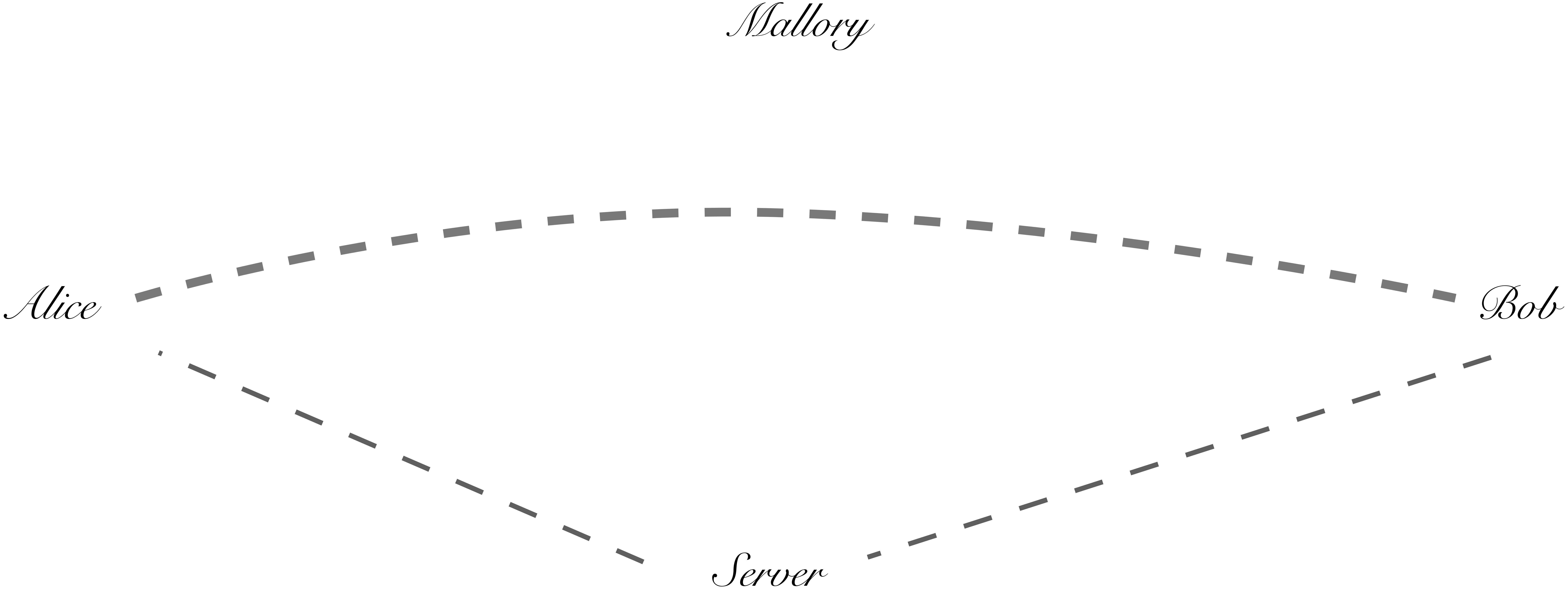
Digital Signatures



Password Authentication



Distributed Password Authentication



Mallory

Alice

Bob

Server

Security Model for Snell Library

Mallory

Alice

Snell Library

Topics

- Kerchoff’s principle
- Security experiments
- Given an example scenario, be prepared to develop a threat model and a security game to capture the threat
- Review our example cast of attackers, they may come in handy if you are asked to develop a threat model on the exam.
- Confidentiality, Authentication, Integray, Non-repudability
- Perfect and Shannon security
- One time pad
- Computational Indistinguishability
- Pseudo-random generators
- Symmetric key encryption
- Pseudo-random functions
- Message authentication codes
- Hash functions, definitions, security experiment, examples
- Public key encryption, IND-CPA security game, RSA cryptosystem example
- Digital Signature security game, why textbook RSA signing is insecure
- Password storage systems, salting and hashing, slow hash functions
- Pros and cons of biometrics
- two-factor authentication, U2F
- biometrics, their strengths, and their shortcomings
-