

Lecture 1

Introduction to ECE 422

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Introduction
- Course logistics
- Agile methodology
- TODOs for upcoming classes

An Ran's research

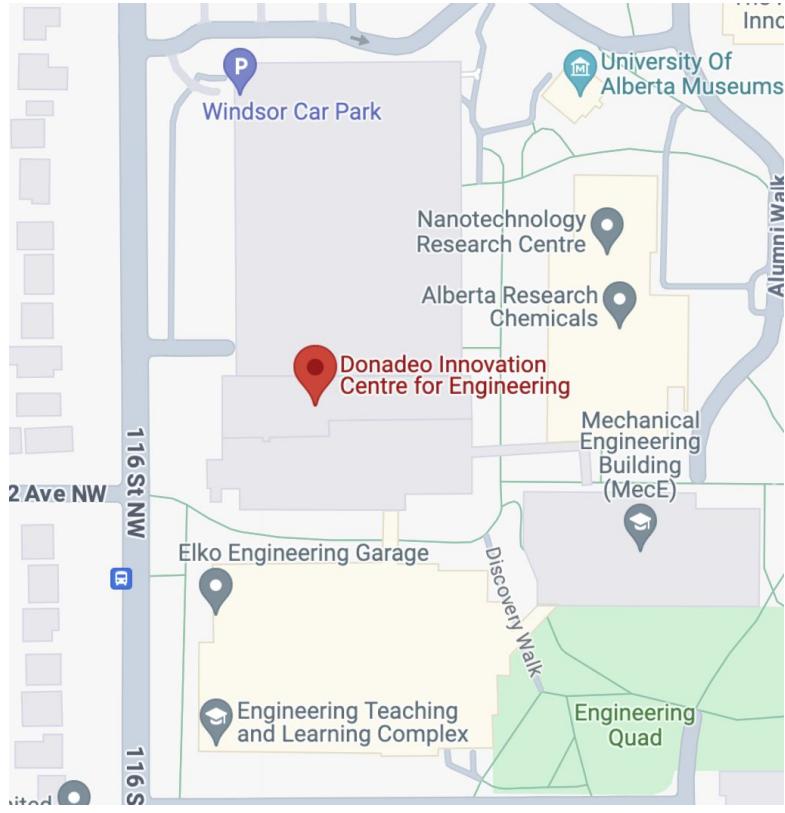


Software quality assurance



Mining software data

Office



Donadeo Innovation Centre For Engineering
11-366

Email: anran6@ualberta.ca

Office hour
MWF 13:00 - 13:30 (by appointment)

Course objectives

- Determine the reliability of a system
 - Lecture slides
- Design a fault-tolerant software component
 - Project deliverables, final report
- Analyze the security of a built system
 - Lecture slides
- Design a secure software system
 - Project deliverables, final report

Course format

There are 3 main components in this course:

- Course projects (45%)
- Midterm (25%)
- Final exam (30%)

Course projects

- Done in groups of 3 people
- Programming language of your choice (e.g., Python, Java, and C++)
- Following the agile methodology, more details to be announced.

Project deliverable (3-5 pages)

- Describes design, technologies, tools, user stories, and planning

Final report (6-10 pages)

- Expands of the deliverable, based on the finished product

Demonstration (10-15 minutes)

- After the final report submission, scheduled with the TAs.

Course projects

Project 1: Reliability Auto-Scaling

An auto-scaler that actively monitors the response time and scale in or out according to the workload.

Project deliverable (5%)

- Due Wednesday, January 24

Final report and demo (15%)

- Due Wednesday, February 7

Course projects

Project 2: Secure File System

A secure file system that allows its internal users to store data on an untrusted file server.

Project deliverable (10%)

- Due Friday, March 15

Final report and demo (15%)

- Due Monday, April 8

Slack Workspace

Slack is used to help your team with the projects. Make sure to join the Slack channel, the link is provided below.

Slack link: [ECE 422 Winter 2024](#)

A starter kit, information, and tutorials on Cybera and Docker will be posted on both eClass and Slack.

Team formation deadline: Friday, Jan 12, 2024

- Send your team information on the #team channel in Slack (Find a team name and include your names)
- Setup your Cybera account

Teaching and course assistant information

No lab or tutorial is scheduled.

Ronald Unrau

Email: rcunrau@ualberta.ca

Zhijie Wang

Email: zhijie.wang@ualberta.ca

Late Policy

- Deliverable and final reports must be submitted on-time.
 - 25% per day penalty
- Deadlines:
 - Wednesday, January 24: Project 1 Deliverable
 - Wednesday, February 7: Project 1 Final Report
 - Friday, March 15: Project 2 Deliverable
 - Monday, April 8, 2024: Project 2 Final Report
- All class deadlines are due at 11:59pm

Course format

There are 3 main components in this course:

- Course Projects (45%)
 - Project 1 (20%)
 - Project 2 (25%)
 - Done in groups of 3
 - Each project is graded on the deliverable, final report and demo
- Midterm (25%)
- Final Exam (30%)

Exams

Midterm (25%)

- Monday, February 26, 2024
- In-person exam
- Covers slides from Week 1 to 6 (up to February 16)

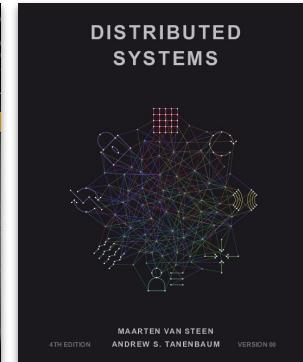
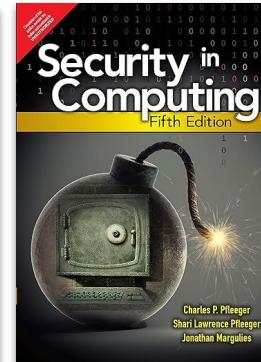
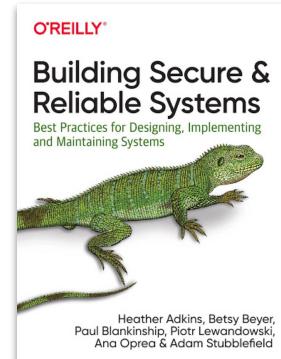
Final exam (30%)

- ~~Friday, April 19, 2024~~ **Wednesday, April 24, 2024**
- In-person exam
- Covers slides from Week 1 to 14

If you anticipate being unable to take the midterm for some *legitimate* reason, please inform me ahead of time. When this happens, the weight of the missed midterm will be added to the final exam (a total of 55%).

Textbooks

1. Marteen van Steen, Andrew S. Tanenbaum, "Distributed Systems", 4th edition version 01, distributed-systems.net, 2020.
2. C.P. Pfleeger, S.L. Pfleeger, J. Margulies, "Security in Computing, 5th Ed.", Pearson Education, 2015.
3. H. Adkins, B. Beyer, P. Blankinship, P. Lewandowski, A. Oprea, A. Stubblefield, "Building Secure and Reliable Systems", O'Reilly Media, 2020.



Deadlines

Activity	Due/Scheduled	Weight
Project 1 Deliverable	Wednesday, January 24, 2024	5%
Project 1 Final Report	Wednesday, February 7, 2024	15%
Project 2 Deliverable	Friday, March 15, 2024	10%
Project 2 Final Report	Monday, April 8, 2024	15%
Midterm Exam	Tentative - Monday, February 26, 2024	25%
Final Exam	Tentative - Friday, April 19, 2024 Wednesday, April 24, 2024	30%

Academic Honesty

Deliverable and final report: You are welcome to discuss the projects with other students, but write your deliverable and final report by yourself. Avoid plagiarize, both the deliverable and report should reflect the work of your team.

Code submission: You are allowed to consult other people's code for the structure of your project, but the main code should be written by you and your teammates alone.

Schedule for today

- Introduction
- Course logistics
- Agile methodology
- TODOs for upcoming classes

Agile methodology

Agile methodology proposes principles instead of an actual process

- Proposed by developers based on their experience
- The Agile Manifesto: <https://agilemanifesto.org/>

Agile methodology

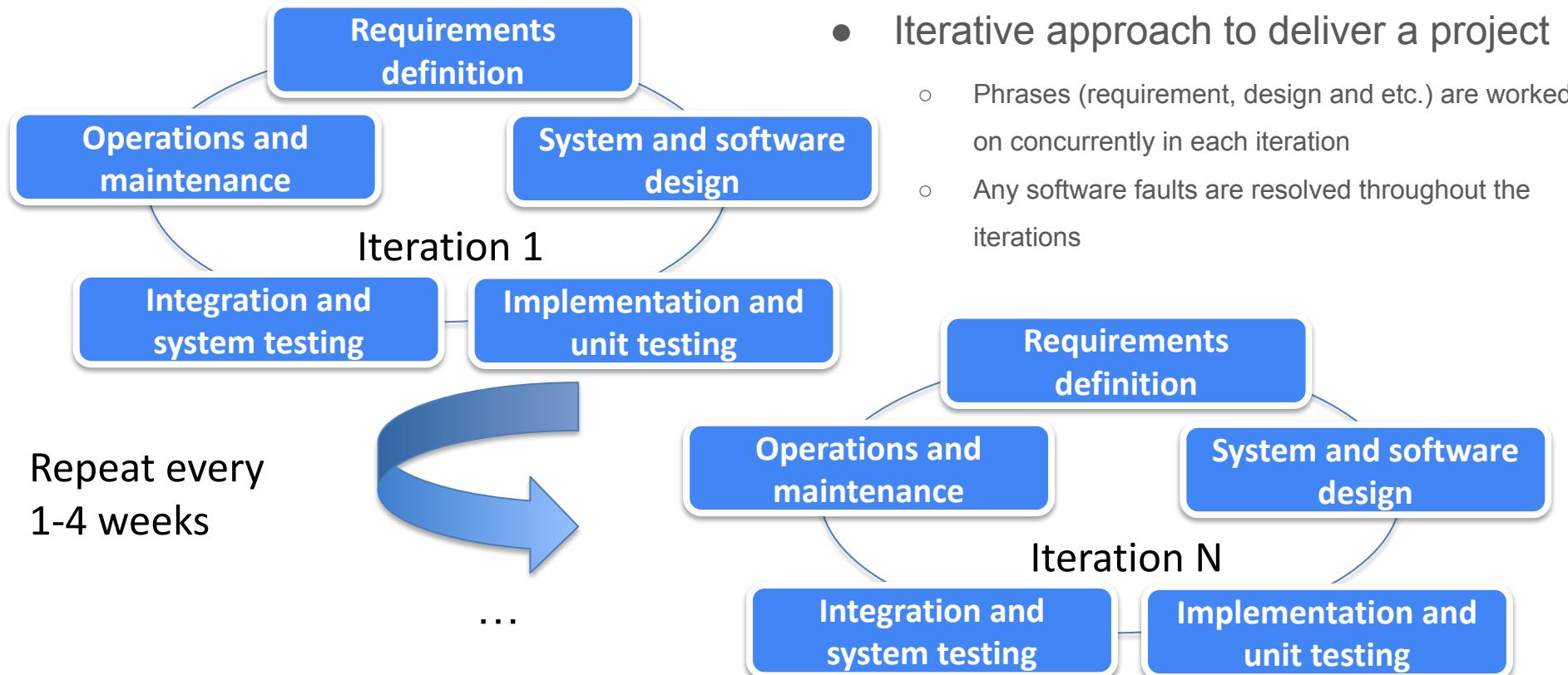
Agile methodology proposes principles instead of an actual process

- Proposed by developers based on their experience
- The Agile Manifesto: <https://agilemanifesto.org/>
- The Agile Manifesto contains 12 principles that focus on 4 values:
 - Working software over comprehensive documentation
 - Customer collaboration over contract negotiation
 - Responding to change over following a plan
 - Individuals and interactions over processes and tools

Agile methodology

- Suggests an iterative approach to deliver a project, different from traditional approaches (e.g., waterfall model)
 - Focuses on continuous releases
 - Flexibility to adjust and iterate during the development process
- Cornerstone of DevOps practices
 - By fostering collaboration and feedback (both with the customer and within the team)
 - Deliver earlier = detect faults earlier, less expensive to fix (developing software as building house, software faults as cracks in house's foundation)

Agile workflow



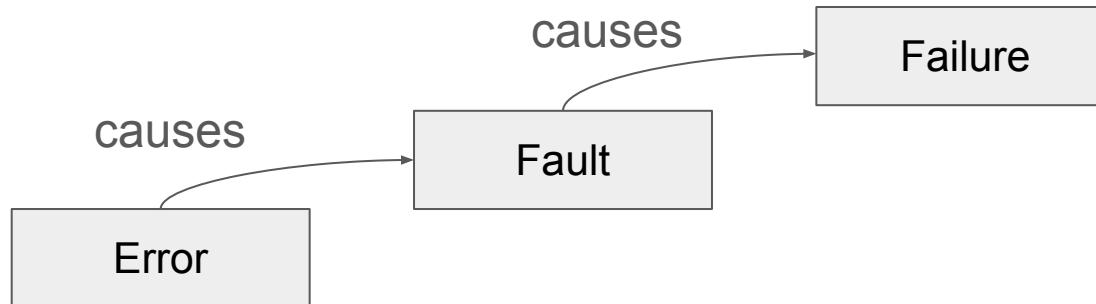
What is a software fault?



Error: a mistake in the code

Fault: a defect in the code that cause incorrect or unexpected results

Failure: occurs when the system fails to perform its required function



A developer makes an error that causes a fault in the software, which can cause a failure.

What is a software fault?



Error: a mistake in the code

Fault: a defect in the code that can cause incorrect or unexpected results

Failure: occurs when the system fails to perform its required function

```
public static int numZero (int[] x) {  
    int count = 0;  
    for (int i = 1; i < x.length; i++) {  
        if (x[i] == 0) {  
            count ++;  
        }  
    }  
    return count;  
}
```

Use case 1

Input: [0, 2, 3]

Output: 0

Error state

First iteration *for* where *i* = 1,
skipping index 0.

Is there a/an __ ?

Error	True / False
-------	--------------

Fault	True / False
-------	--------------

Failure	True / False
---------	--------------

What is a software fault?



Error: a mistake in the code

Fault: a defect in the code that can cause incorrect or unexpected results

Failure: occurs when the system fails to perform its required function

```
public static int numZero (int[] x) {  
    int count = 0;  
    for (int i = 1; i < x.length; i++) {  
        if (x[i] == 0) {  
            count ++;  
        }  
    }  
    return count;  
}
```

Use case 1

Input: [0, 2, 3]

Output: 0

Error state

First iteration *for* where *i* = 1,
skipping index 0.

Is there a/an __ ?

Error	True / False
-------	--------------

Fault	True / False
-------	--------------

Failure	True / False
---------	--------------

What is a software fault?



Error: a mistake in the code

Fault: a defect in the code that can cause incorrect or unexpected results

Failure: occurs when the system fails to perform its required function

```
public static int numZero (int[] x) {  
    int count = 0;  
    for (int i = 1; i < x.length; i++) {  
        if (x[i] == 0) {  
            count ++;  
        }  
    }  
    return count;  
}
```

Use case 2

Input: [2, 3, 0]

Output: 1

Error state

First iteration *for* where *i* = 1,
skipping index 0.

Is there a/an __ ?

Error	True / False
-------	--------------

Fault	True / False
-------	--------------

Failure	True / False
---------	--------------

What is a software fault?



Error: a mistake in the code

Fault: a defect in the code that can cause incorrect or unexpected results

Failure: occurs when the system fails to perform its required function

```
public static int numZero (int[] x) {  
    int count = 0;  
    for (int i = 1; i < x.length; i++) {  
        if (x[i] == 0) {  
            count ++;  
        }  
    }  
    return count;  
}
```

Use case 2

Input: [2, 3, 0]

Output: 1

Error state

First iteration *for* where *i* = 1,
skipping index 0.

Is there a/an __ ?

Error	True / False
-------	--------------

Fault	True / False
-------	--------------

Failure	True / False
---------	--------------

Scrum

- Scrum is an agile project management framework that helps teams structure and manage their work based on the Agile principles.
- Scrum is composed of three main aspects:
 - Roles
 - Artifacts
 - Workflow

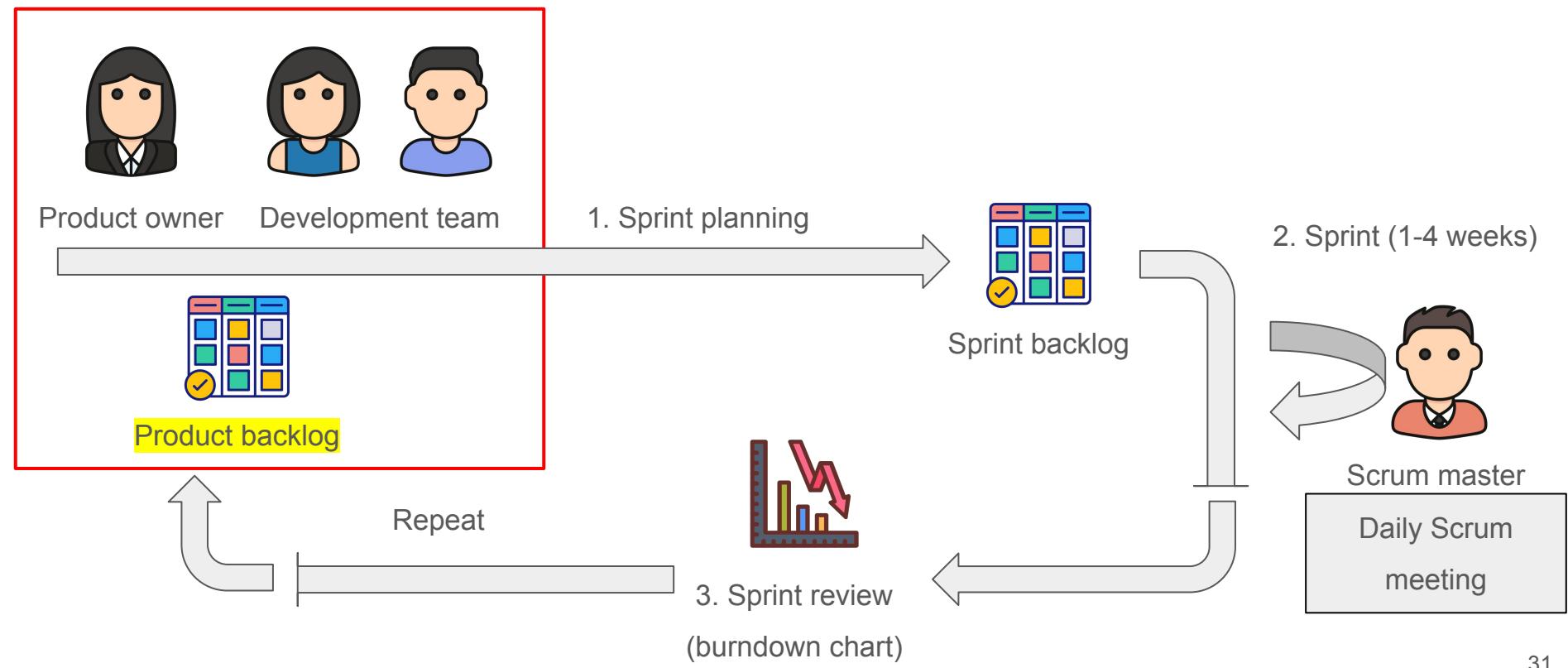
Roles

- Product owner
 - Makes sure the product is what the customer wants
 - Defines features to implement
- Scrum team
 - Individuals who are working on the software
 - Team members are cross-functional (consist of developers, testers, designers...)
- Scrum master
 - Makes sure scrum process is being followed
 - Hosts Scrum meetings
 - Removes impediments so that team members can finish their task

Artifacts

- Product backlog
 - List of work that the team needs to do to deliver a finished product (new user stories, bug fixing, etc)
- Iteration/sprint backlog
 - List of work that the team needs to do for the current iteration/sprint
- Burndown chart
 - Keeps track of the remaining tasks for the current iteration

Workflow



User stories

User stories: a software feature written from the end user's perspective.

- Written in non-technical language to provide context
- Why? To describe what value a software feature provides to the customer
- An user story should explain: persona + need + purpose

User stories

User stories: a software feature written from the end user's perspective.

- Written in non-technical language to provide context
- Why? To describe what value a software feature provides to the customer
- An user story should explain: persona + need + purpose

As a [persona], I want [need], so that [purpose].

- Persona: Who is the end user?
- Need: What is the goal of the end user?
- Purpose: What problem does the end user look to solve?

Example: As a sales director, I want a sales report, so that I can monitor the sales progress.

User stories

User stories: a software feature written from the end user's perspective.

- Written in non-technical language to provide context
- Why? To describe what value a software feature provides to the customer
- An user story should explain: persona + need + purpose

As a [persona], I want [need], so that [purpose].

- Persona: Who is the end user?

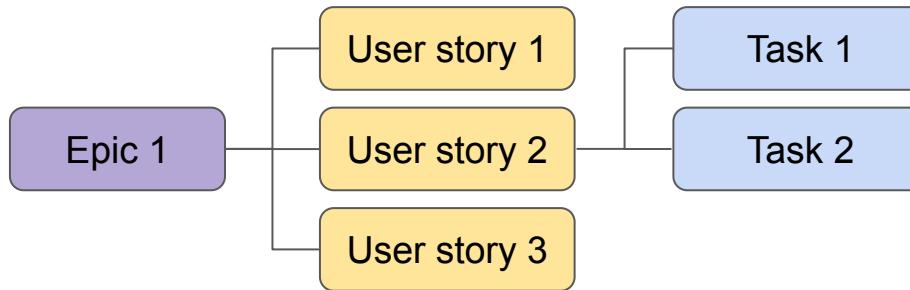
You will be asked to provide two user stories for your first project deliverable.

More details will be discussed in the next class.

Example: As a sales director, I want a sales report, so that I can monitor the sales progress.

Epics

Epics: large bodies of work that can contain a collection of user stories.



- Why? To organize the work and clarify on client priorities
- An epic cannot be completed in a single iteration

Examples of epics: “chatbot functionality” in a web app, “augmented reality features” in a mobile app, and etc.

Story points and estimation

Traditional software teams give estimates in a time format: days, weeks, months. Agile teams use story points to quantify the efforts required to carry out a task (user story).

Story points are relative to:

- Work complexity
- Amount of work and hours required to complete
- Risk and uncertainty

Planning poker

Planning poker: a consensus-based estimating technique for story points where everyone (e.g., developers, designers, testers ...) is involved

- The product owner reads a user story, describes it
- Everyone picks a card to represent their estimate
 - Everyone holds a deck of planning poker cards. Each card contains a value of the Fibonacci sequence (e.g., 0, 1, 2, 3, 5, 8, 13, 20, and 40)
- Reveal their card and discuss the differences between estimates
- Re-estimate until a consensus is reached within the team

Note: usually, you should never have user stories worth more than 16 hours of work (or 20 story points)

- larger the task, more difficult to have an accurate estimate

Planning poker

Common problem 1: Product uncertainty

Solution: put the story aside and research until uncertainty can be resolved

Common problem 2: Technical uncertainty

Solution: decide on the story points later, use a range as the estimate to begin with

Planning poker

Planning poker demo ▾

A An Ran ▾

Invite players

i

Ron

An Ran

Reveal cards

Zhijie

Choose your card 🎉

0 1 2 3 5 8 13 21 34 55 89 ? 🎲

Start new voting

2

3

Zhijie

Average:
2.3

Agreement:

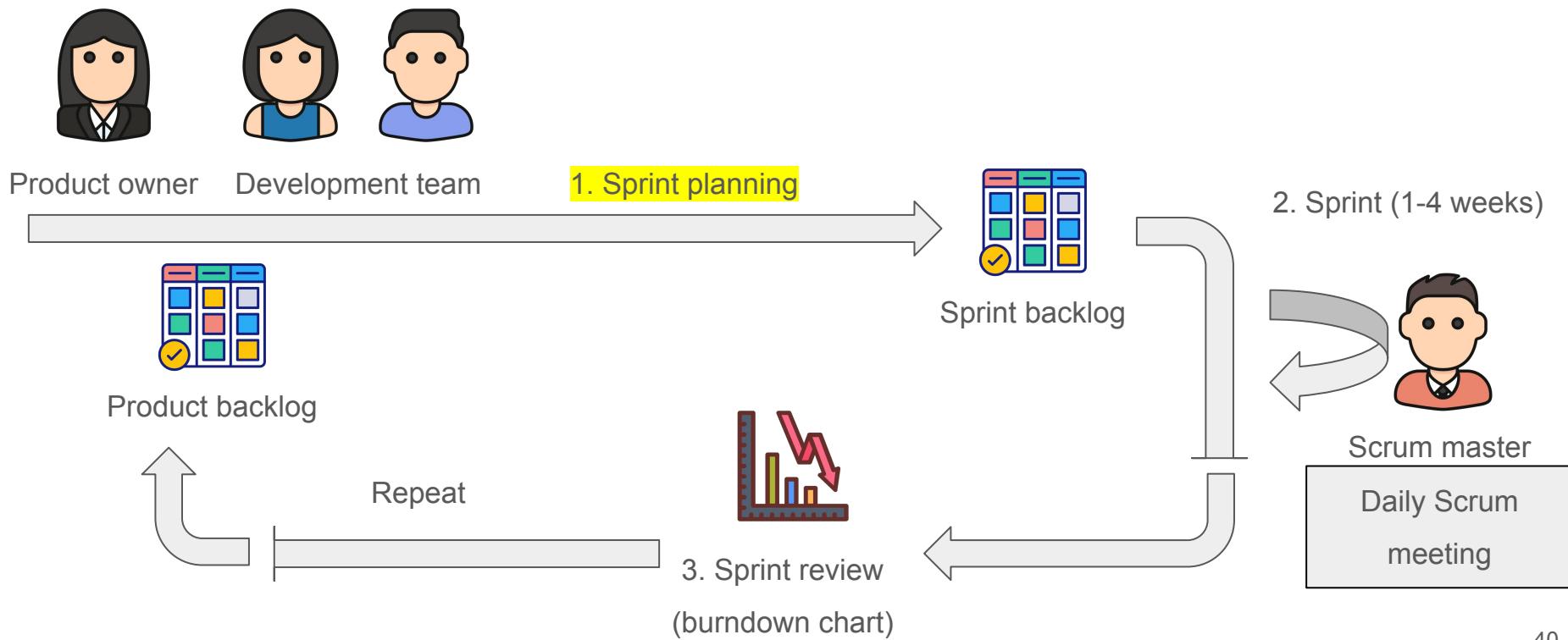
2 Votes 3 Votes

Try it yourself!

Planning poker
online:

<https://planningpokeronline.com/>

Workflow



An example of sprint backlog

Sprint Backlog			
Forecast	To-Do	In Progress	Done
<p>Fix My Profile</p> <p>5</p>		<p>aliquip</p>	<p>ipsum</p> <p>duis</p> <p>sit</p> <p>ipsum</p>
<p>Filter Service Tickets</p> <p>8</p>	<p>dolor</p> <p>ipsum</p> <p>culpa</p>	<p>vale</p> <p>culpa</p>	<p>aliquip</p>
<p>Quick Tips</p> <p>3</p>	<p>ipsum</p> <p>sit</p> <p>duis</p> <p>duis</p>		

Another example of sprint backlog

▼ Affogato 13 May – 27 May (14 issues)

As a user, I'm able to jump onto the Espresso app, order a brew and a snacky snack for now, and pre-order dessert for later.

0 14 23 Complete sprint ...

Issue	Category	Status	Assignee	
BREW-1 Content audit	QUICK WINS	4	IN PROGRESS	
BREW-17 Update project plan with key milestones	FY23 LAUNCH PLAN	5	IN PROGRESS	
BREW-2 Comp. analysis—Food delivery	FY23 LAUNCH PLAN	5	DONE	
BREW-3 Comp. analysis—Custom menus	FY23 LAUNCH PLAN	5	DONE	
BREW-4 Something's up with the load screen	FY23 LAUNCH PLAN	5	DONE	
BREW-11 Journey mapping workshop w/ Beanz	FY23 LAUNCH PLAN	5	IN PROGRESS	Jessie Spiteri
BREW-12 Taste test: Round 1 (vanilla extract)	QUICK WINS	5	TO DO	
BREW-5 Social content: Week 1	FY23 LAUNCH PLAN	10	TO DO	
BREW-14 FY23 Vision: Storyboarding	FY23 LAUNCH PLAN	10	DONE	
BREW-10 Explore personas: Geoff	FY23 LAUNCH PLAN	5	IN PROGRESS	
BREW-20 Review banner ads	QUICK WINS	5	DONE	
BREW-24 Onboarding tour refinements	FY23 LAUNCH PLAN	3	DONE	
BREW-22 Taste test: Flate white cups	QUICK WINS	3	IN PROGRESS	
BREW-23 Taste test: Latte glasses	FY23 LAUNCH PLAN	5	TO DO	

BREW-13 / BREW-11

1 like share ... X

Journey mapping workshop w/ Beanz

In Progress Actions

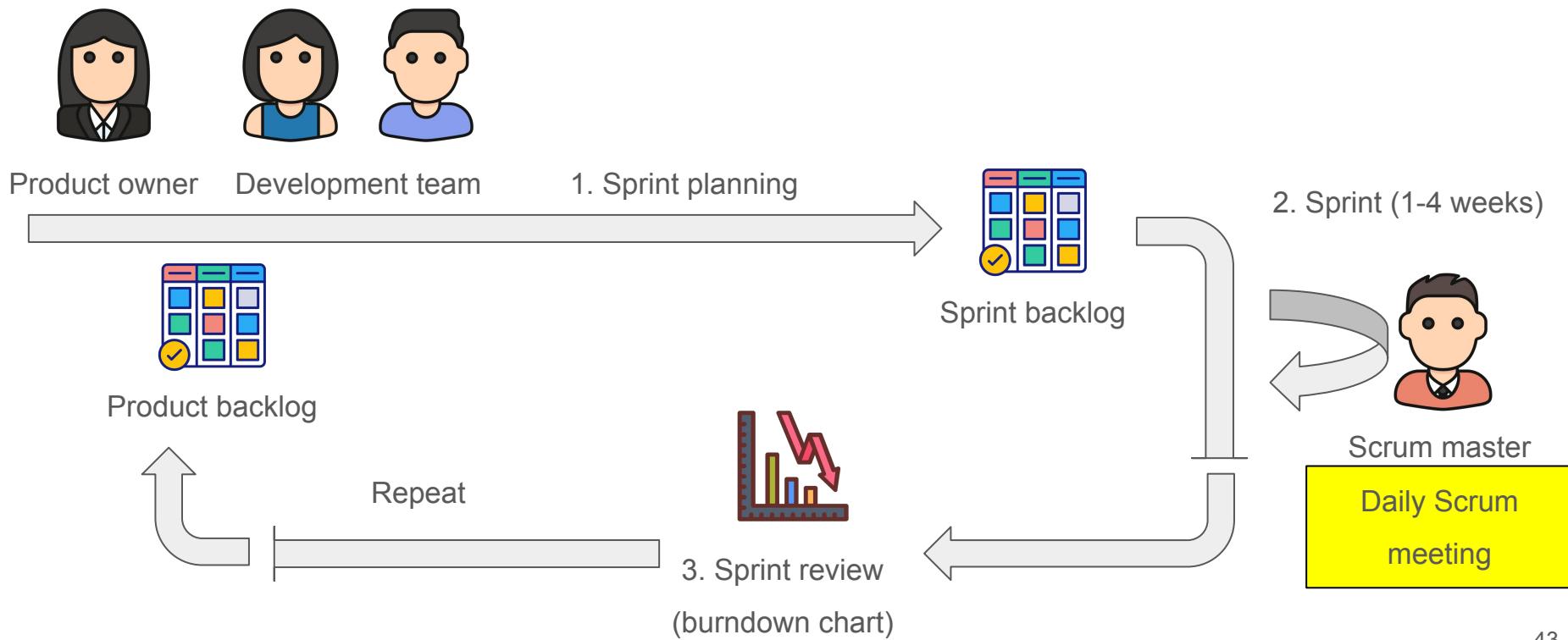
Description
Add a description...

Details

Assignee	Jessie Spiteri
Reporter	Jessie Spiteri
Labels	Android Workshop iOS
Sprint	Affogato
Story Points	5
Epic Link	FY23 Launch Plan
Priority	High

Add a comment...

Workflow



Daily Scrum meeting

15-minutes meeting held on each day of a sprint.

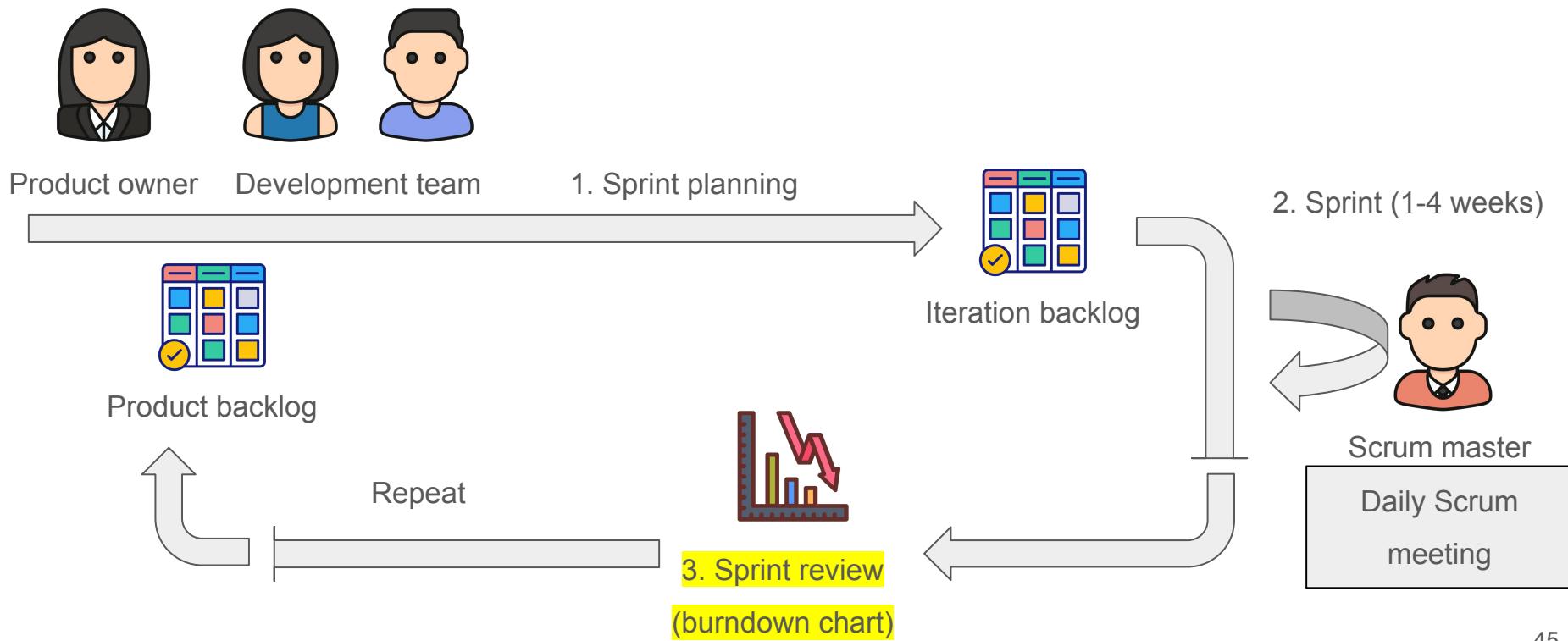
Three questions of the daily meeting:

- What did I do yesterday?
- What will I do today?
- Are there any impediments in my way?

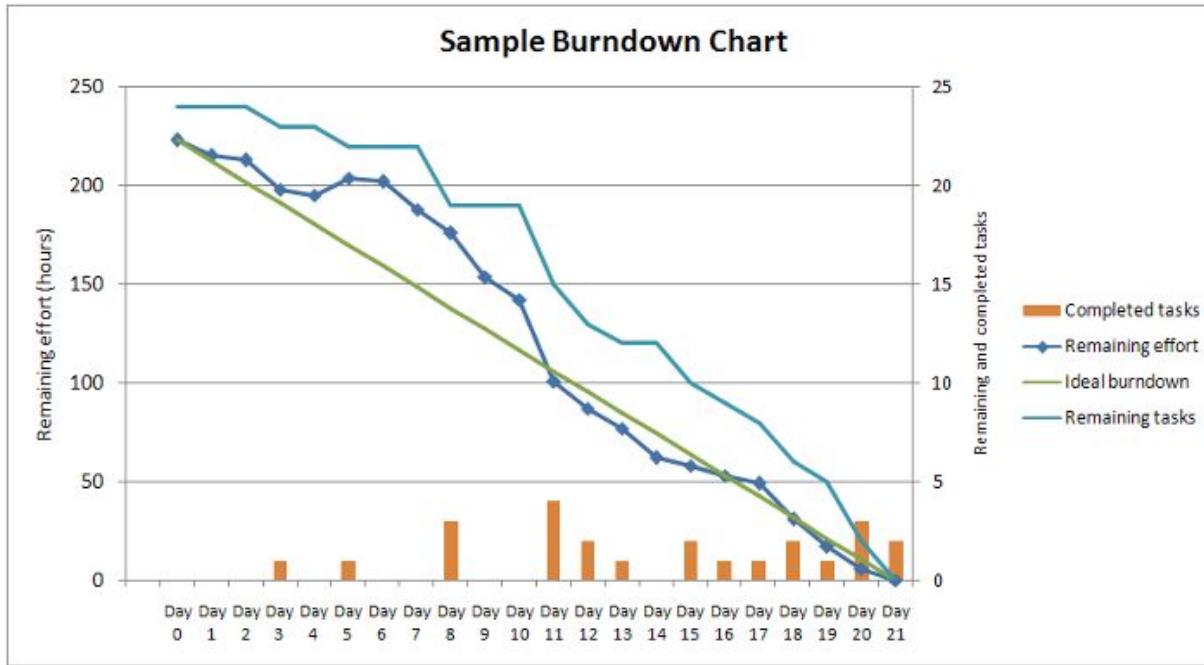
Scrum master is responsible for resolving the impediments.

- Examples of impediments: “I need help debugging a database problem.”
- Solution: “Assigning a team member for pair programming”

Workflow

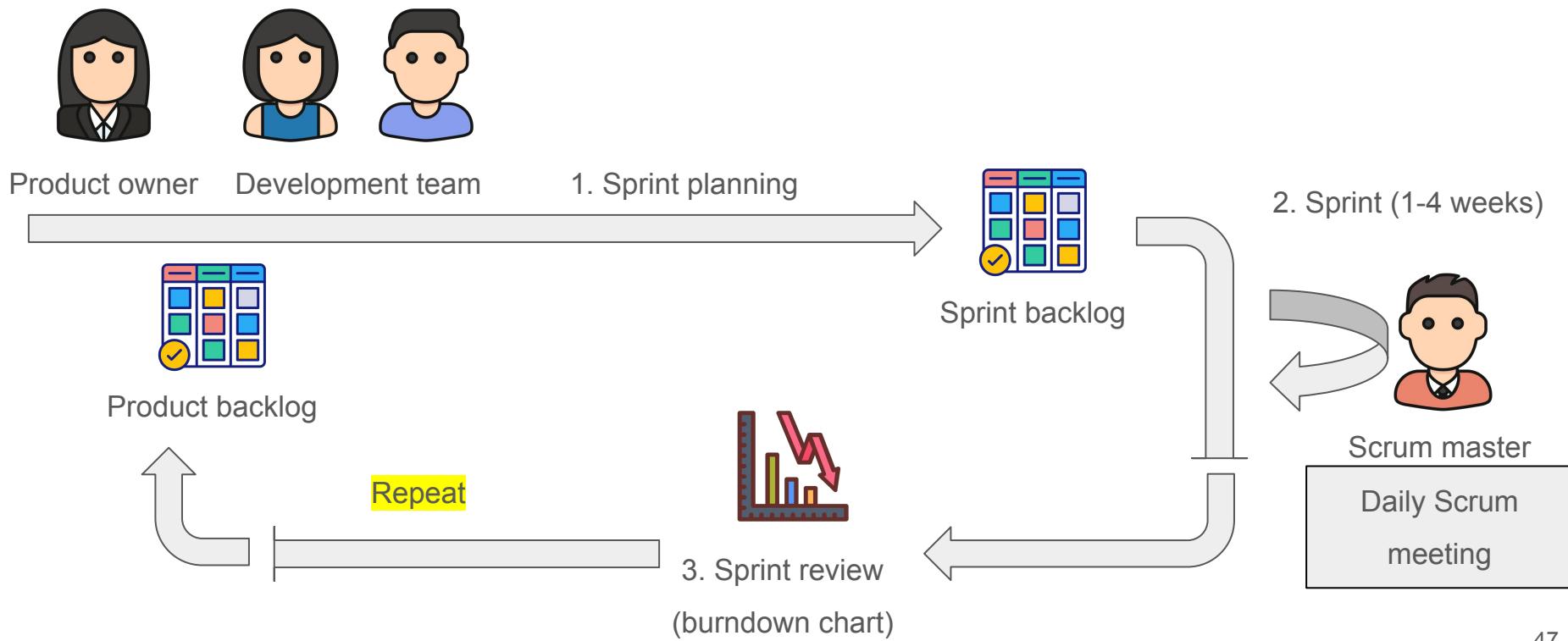


An example burndown chart



By Pablo Straub - Own work, Public Domain,
<https://commons.wikimedia.org/w/index.php?curid=7132232>

Workflow



Scrum



Silicon Valley Season 1 Episode 5

Agile methodology

- Pros
 - Strong flexibility
 - Able to change requirements efficiently
 - Close customer collaboration
 - Make sure the things we do are what the customers want
 - Promote teamwork
 - Help each other when one encounters roadblocks
- Cons
 - Lack of documentation
 - Require continuous planning

TODOs

- Join our Slack workspace
- In Slack #team channel: post your team information before Friday 23:59 pm
 - Team Name: Team ECE422
 - Name: An Ran Chen, Ron Unrau, Zhijie Wang
- Setup your Cybera account
 - In this course, you will be using Cybera, which is a tool to help deploy your projects.

Lecture 2

DevOps

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

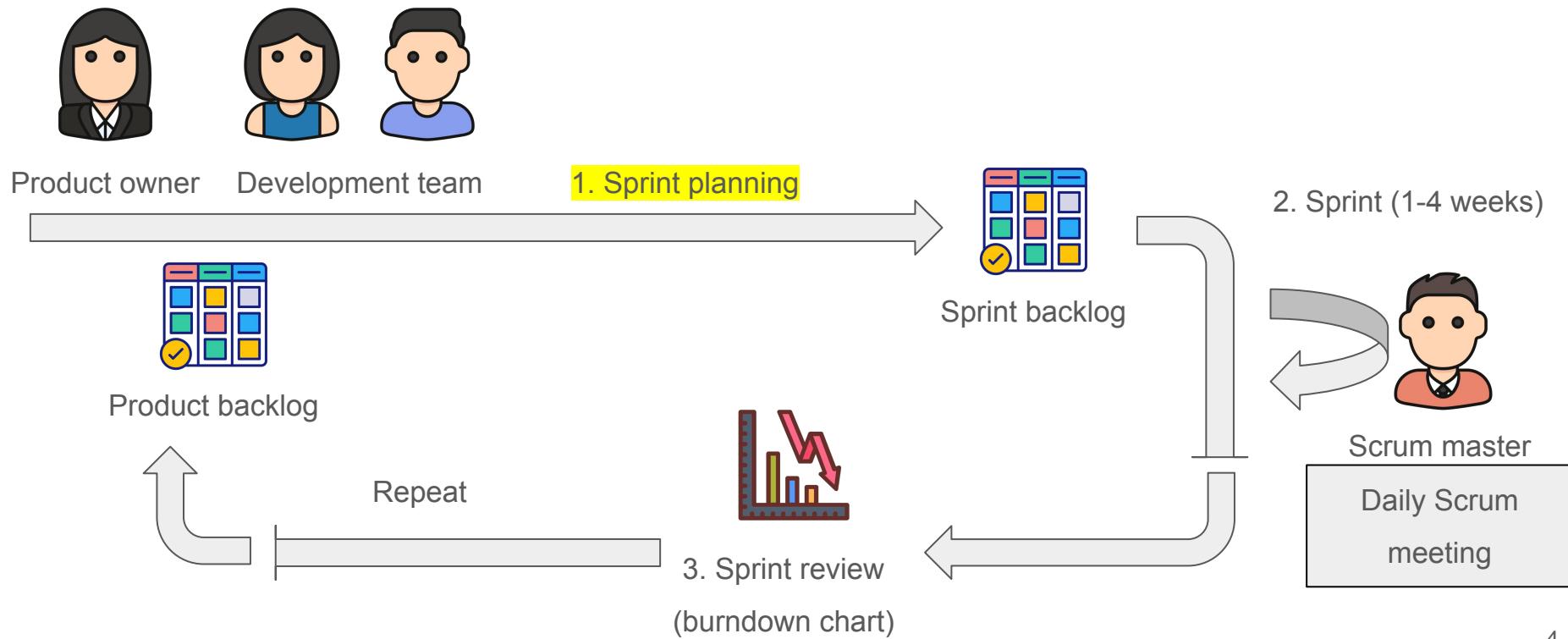
Schedule for today

- Key concepts from last class
- DevOps
- CI/CD
- Docker container
- TODOs for upcoming classes

Agile methodology

- Suggests an iterative approach to deliver a project, different from traditional approaches (e.g., waterfall model)
 - Focuses on continuous releases
 - Flexibility to adjust and iterate during the development process
- Cornerstone of DevOps practices
 - By fostering collaboration and feedback (both with the customer and within the team)
 - Deliver earlier = detect faults earlier, less expensive to fix (developing software as building house, software faults as cracks in house's foundation)

Scrum, an agile project management framework



User stories

User stories: a software feature written from the end user's perspective.

- Written in non-technical language to provide context
- Why? To describe what value a software feature provides to the customer
- An user story should explain: persona + need + purpose

As a [persona], I want [need], so that [purpose].

- Persona: Who is the end user?
- Need: What is the goal of the end user?
- Purpose: What problem does the end user look to solve?

Example: As a sales director, I want a sales report, so that I can monitor the sales progress.

Schedule for today

- Key concepts from last class
- DevOps
- CI/CD
- Docker container
- TODOs for upcoming classes

DevOps

DevOps is a partnership between software developers (Dev) and IT operators (Ops).

- Speeds up deployments
- Improves communication and collaboration
- Ensures quality and reliability of the system

Note: in the next class, we will see that DevOps is not a perfect solution if we are prioritizing on the reliability of the system.

Traditional software development process



Developers

Responsibilities

- Design
- Code
- Test
- Release engineering
- Improvements



Operators

Responsibilities

- Deploy
- Monitor
- Troubleshoot
- Anomaly detection
- Quality assurance
- Configuration tuning

Traditional software development process



Developers

Responsible for development of the system

1. looking to implement more features
2. want to release faster to the customers
3. work on the development and testing environments (no idea on how the system is deployed)



Operators

Responsible for the operations of the system

1. keeping the system stable and safe
2. ensure that the system is well-tested before the release
3. work on the production environment (no idea how the system is implemented and tested)

Challenges in traditional software development

- Poor communication
 - No collaboration between developer and operations teams
 - Developers implement the system
 - Operators deploy and operate the system
 - However, the operation teams are responsible for troubleshooting the problems in deployment while they lack the implementation information
 - Consequence: slow down the releases
- Isolated expertises
 - Because of the knowledge gap, both teams struggle in resolving problems
 - Developers have limited insights into operations
 - Operations staff have limited capacity in resolving anomalies

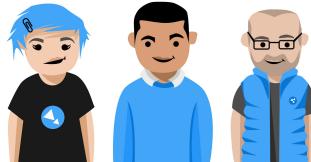
Challenges in traditional software development

- Conflicting goals
 - New features vs stability
 - Developers look to push new features quickly to the customer
 - Operators want to keep the system stable and safe
 - Operators want to ensure there are as little change to the infrastructure of the system as possible
- Different cultures
 - Common goal of being as productive as possible, but
 - Developers want to improve the system as much as possible
 - Operators want to limit the changes to reduce uncertainty

DevOps

DevOps promotes a collaborative culture, where development and operations teams work together to share responsibilities for software correctness and reliability.

- It focuses on **reducing time to market** for new features while ensuring **high quality**.



Development and operations teams work together under the **DevOps workflow** to shorten the development life cycle, provide continuous delivery while ensuring high quality.

DevOps

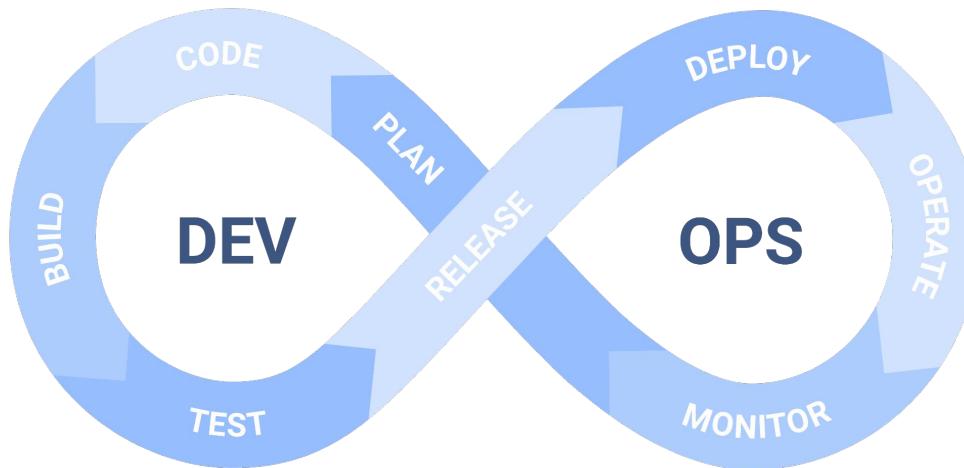
DevOps has three main components:

- Workflow
- Automation
- Tool supports

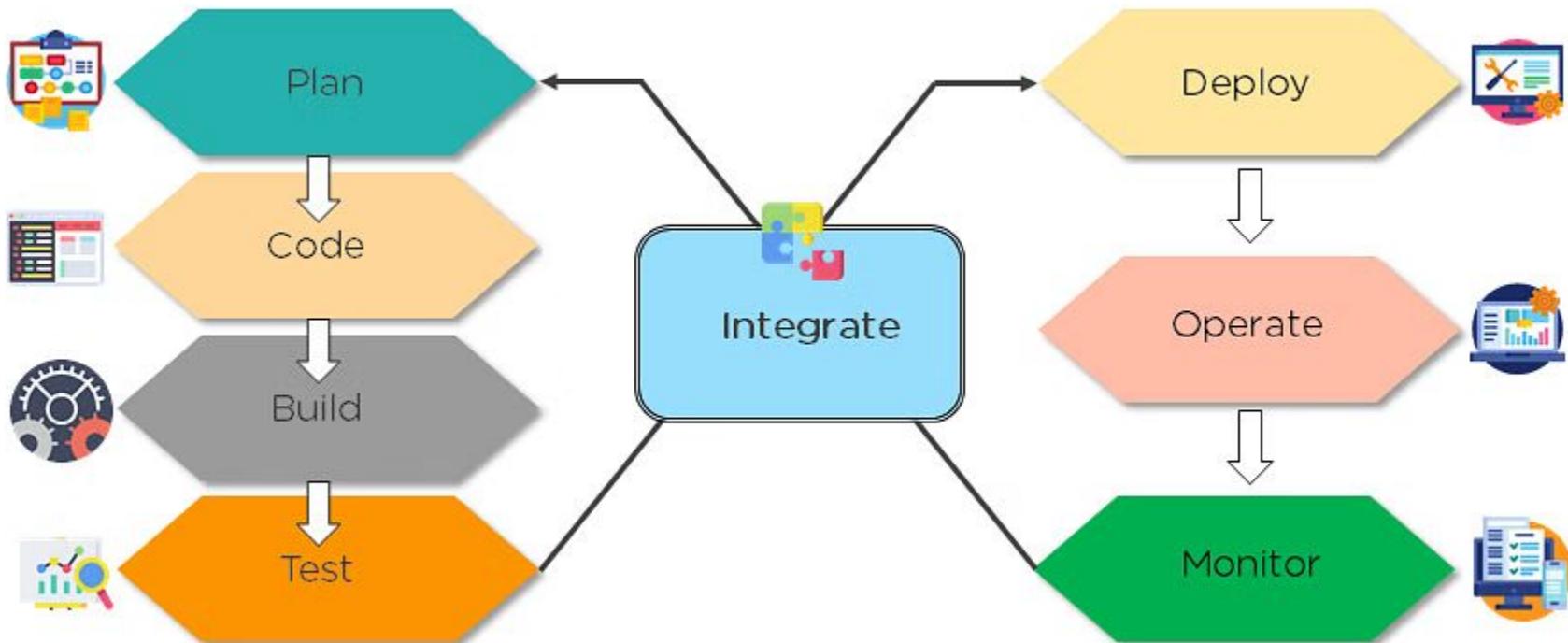
DevOps workflow

DevOps workflow is continuously looping through a set of activities between development and operations.

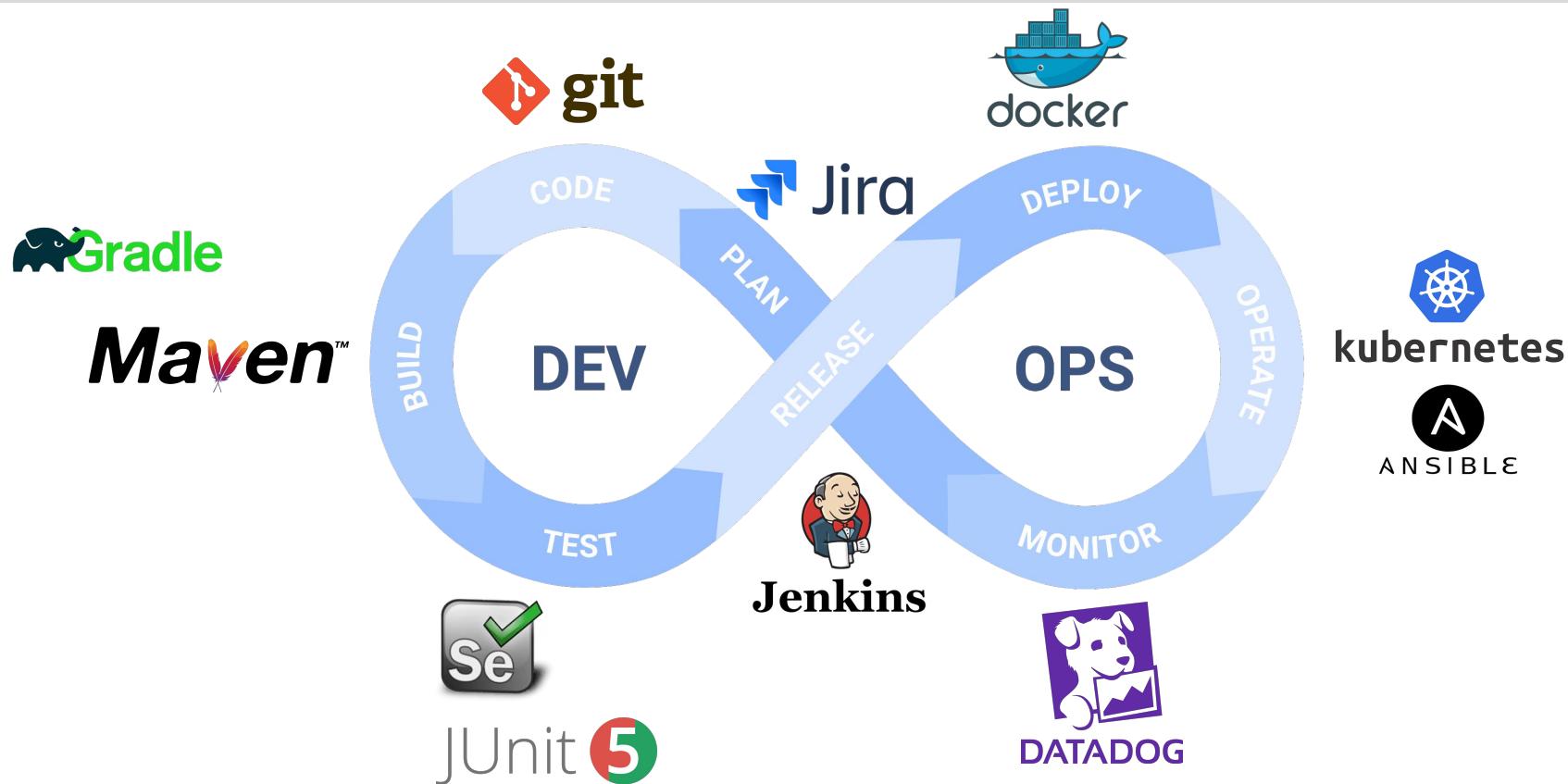
- involves continuous development, testing, monitoring, feedback, and deployment processes.



DevOps workflow



DevOps workflow



DevOps

DevOps has three main components:

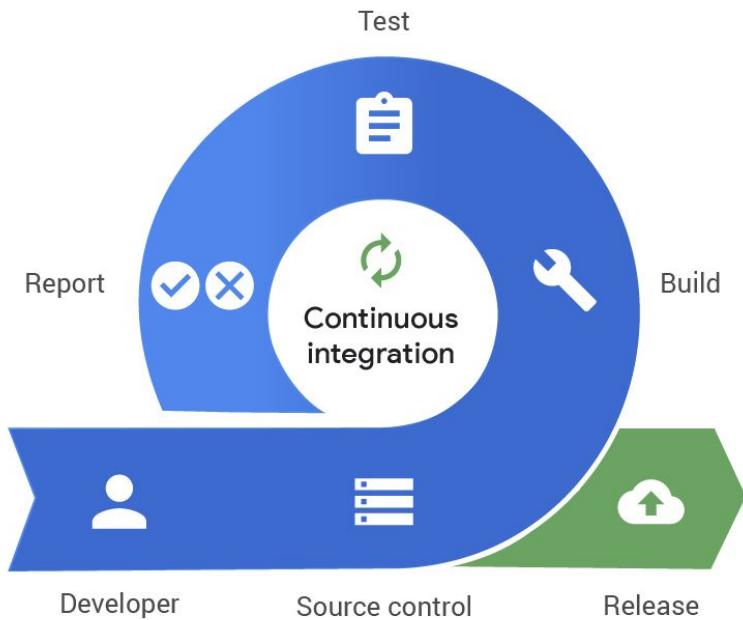
- Workflow
- Automation
 - DevOps uses automation to performs tasks which can reduce human assistance and facilitate feedback loops between operations and development teams.
 - E.g., CI/CD
- Tool supports

CI/CD

- CI stands for Continuous Integration
- CD can be Continuous Delivery and/or Continuous Deployment
- CI/CD aims to streamline and speed up the development and delivery processes
 - CI/CD automates the development process from an initial code change to its deployment to the production environment.



Continuous Integration (CI)



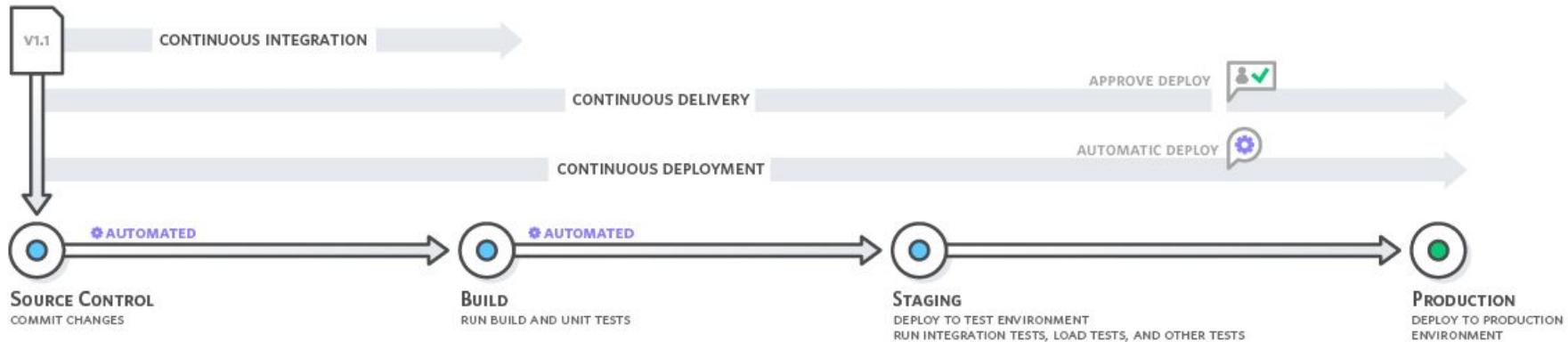
Continuous integration is the practice of automatically and frequently integrating code changes into a shared source code repository.

- Why? To detect integration issues early
- Developers frequently merge their code changes into a central repository
- Each commit triggers an automated workflow that builds and tests the system

Continuous Delivery (CD)

Continuous delivery is the practice of automatically releasing new code changes through automated testing process into the production environment **with manual approval**.

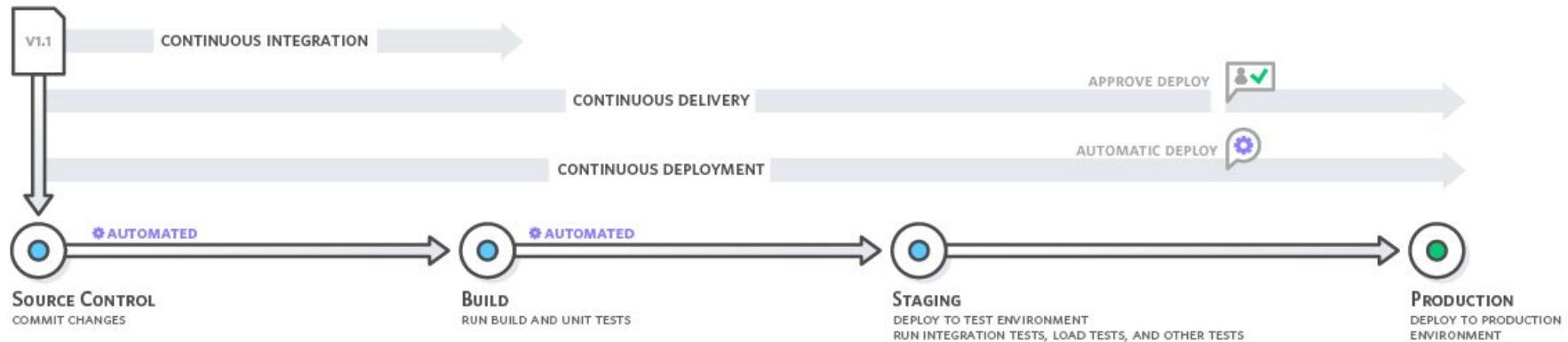
- Why? To deliver updates to customers faster and more frequently
- Every code change is built and tested, and then pushed to staging environment.
- Once the testing finishes on staging, the final decision to deploy to a live production environment is triggered manually.



Continuous Deployment (CD)

Continuous deployment is the practice of automatically releasing new code changes through automated testing process **directly** into the production environment.

- Why? To reduce manual processes
- Every code change is built and tested, and then pushed to staging environment.
- Once the testing finishes on staging, the changes are automatically deployed to the live production environment.



Continuous delivery vs continuous deployment



1. Ensure that code can always be safely deployed on the production servers
 - a) continuous delivery, b) continuous deployment, c) both
2. Make software releases faster and more robust
 - a) continuous delivery, b) continuous deployment, c) both
3. Ensure that the application works as expected
 - a) continuous delivery, b) continuous deployment, c) both

Continuous delivery automates the entire process up to the point of release readiness, but the actual release to production is triggered manually.

Continuous deployment automates the entire process, including the release to production, eliminating the need for manual intervention in deploying changes.

Continuous delivery vs continuous deployment



1. Ensure that code can always be safely deployed on the production servers
 - a) continuous delivery, b) continuous deployment, c) both
2. Make software releases faster and more robust
 - a) continuous delivery, b) continuous deployment, c) both
3. Ensure that the application works as expected
 - a) continuous delivery, b) continuous deployment, c) both

Continuous delivery automates the entire process up to the point of release readiness, but the actual release to production is triggered manually.

Continuous deployment automates the entire process, including the release to production, eliminating the need for manual intervention in deploying changes.

Continuous delivery vs continuous deployment



4. Deliver every change through rigorous automated testing
 - a) continuous delivery, b) continuous deployment, c) both
5. Every change that passes the automated tests is deployed to production automatically
 - a) continuous delivery, b) continuous deployment, c) both
6. Deployed with no explicit approval but require a culture of monitoring
 - a) continuous delivery, b) continuous deployment, c) both

Continuous delivery automates the entire process up to the point of release readiness, but the actual release to production is triggered manually.

Continuous deployment automates the entire process, including the release to production, eliminating the need for manual intervention in deploying changes.

Continuous delivery vs continuous deployment



4. Deliver every change through rigorous automated testing
 - a) **continuous delivery**, b) continuous deployment, c) both
5. Every change that passes the automated tests is deployed to production automatically
 - a) continuous delivery, b) **continuous deployment**, c) both
6. Deployed with no explicit approval but require a culture of monitoring
 - a) continuous delivery, b) **continuous deployment**, c) both

Continuous delivery automates the entire process up to the point of release readiness, but the actual release to production is triggered manually.

Continuous deployment automates the entire process, including the release to production, eliminating the need for manual intervention in deploying changes.

DevOps

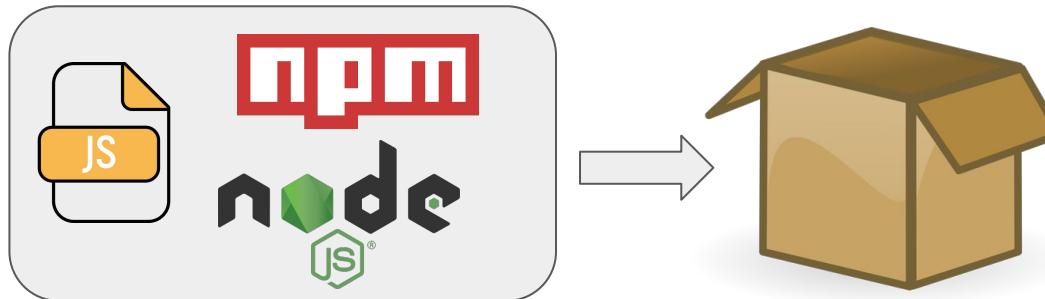
DevOps has three main components:

- Workflow
- Automation
- Tool supports
 - DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity (story points)
 - E.g., containerization

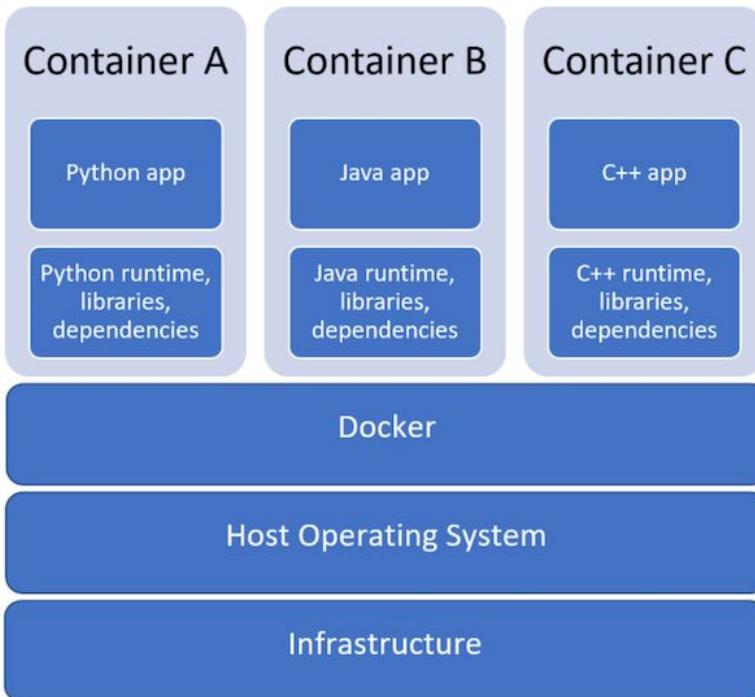
Containerization

A process that virtualizes resources for software application

- Packages an application with all the files and libraries it needs so it can run on any infrastructure
 - E.g., the dependencies, configuration, system tools and runtime become one standardized unit, *the container*
- Helps to build, test, and deploy applications easier and quicker



Container



Example

A Python container include:

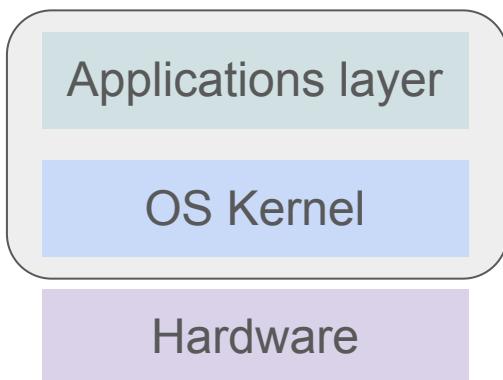
- Python application
- Python runtime
- Application dependencies

Docker container

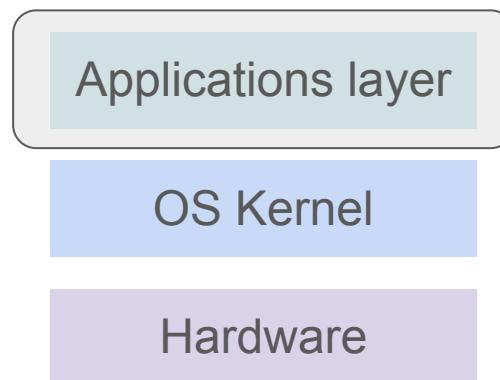
Docker packages software into standardized units (also known as containers) that include everything the software needs to run.

It works in a similar way as a virtual machine

VM: virtualization of the OS
Kernel + Application layer



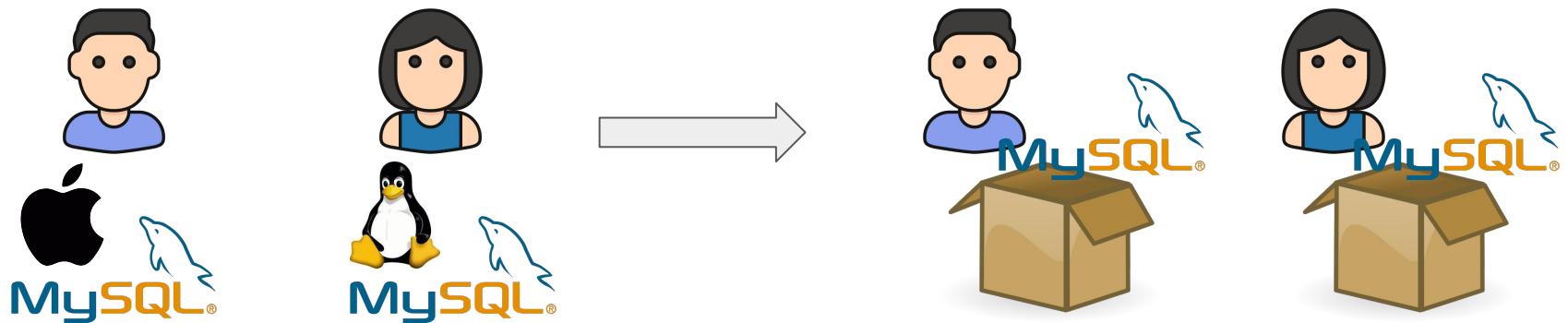
Docker: virtualization of the
Application layer only



Docker container

Containers help to

- Standardize services on all local development environments
- Contain all services, dependencies and configurations in one place
- Isolate dependencies from the local machine



Docker container on security

Docker is also effective against malicious third-parties dependencies as it has its own filesystem.

- Protect against typosquatting attacks
 - Typosquatting attacks create code packages that include a typo to the packages' names (e.g., atlas-client rather than atlas_client)
 - Hoping the victims install the incorrect version to deliver the malware
- Container only have accesses to its own files
- Avoid ransomware attack as we run Node.js inside the container

Docker container in DevOps

Development team can package everything into docker container, and send it to the operations team for deployment.

- Avoid installations and configurations on the server's operating system
 - Resolve dependency version conflicts on the server
- Mitigate the possibility of miscommunication
 - No manual configurations needed on the server

Dockerfile

```
FROM node:21-alpine

WORKDIR /app

COPY src /app/
COPY package.json /app/

RUN npm install \
    && npm run build

CMD [ "serve", "-s", "build" ]
```

- FROM instruction creates a base image
 - Starting point for the image
 - An empty first layer, e.g., node:21-alpine
 - List of images available on [Docker Hub](#)

Dockerfile

```
FROM node:21-alpine

WORKDIR /app

COPY src /app/
COPY package.json /app/

RUN npm install \
    && npm run build

CMD [ "serve", "-s", "build" ]
```

- WORKDIR instructions changes the working directory
 - Set /app as the main working directory
- COPY instructions copies local files
 - The src directory and package.json file are added into the container at path /app
- RUN instruction installs node packages and build the app
 - Install the dependencies inside the container

Dockerfile

```
FROM node:21-alpine

WORKDIR /app

COPY src /app/
COPY package.json /app/

RUN npm install \
    && npm run build

CMD [ "serve", "-s", "build" ]
```

- CMD instruction to run the app where the container is ran
 - Last instruction in Dockerfile
 - There can only be one CMD instruction in a Dockerfile

More information on the instructions: [Docker docs](#)

Benefits of Docker containers

- Modularity
 - Part of an application can be taken down for updates
- Layer and image version control
 - Each Docker image contains a series of layers
 - Full control over container images
- Rollback
 - Each Docker image contains a series of layers
 - Roll back the image to the previous version
- Rapid Deployment
 - Portable artifact
 - Easy to share and distribute

DevOps

DevOps is a partnership between software developers (Dev) and IT operators (Ops).

- Improves communication and collaboration
 - Developers and operators share responsibilities and combine work.
- Speeds up deployments
 - Reduce time between committing a change and shipping that change into production environment.
- Ensures quality and reliability of the system
 - Practices like continuous integration and continuous delivery ensure changes are functional and safe.

Project deliverable

Project 1: Reliability Auto-Scaling Deliverable (3-5 pages)

- Due Wednesday, January 24

Design

- Class diagram

Tools and technologies

- What technologies you plan to use? Why?

User stories

- Two user stories (persona + need + purpose)
- Each user story should be broken down into sub-tasks

Planning

- Timeline of the subtasks

TODOs

- In Slack #teams channel: post your team information before Friday 23:59 pm
 - Email me or send a private message on Slack if you need help finding a team. You are also welcome to send a *short* message in the #teams channel to find a team yourself.
- Check out the Cybera instructions shared by the TAs on Slack
- Check out the Docker documentations on eClass

Lecture 3

Site Reliability Engineering

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

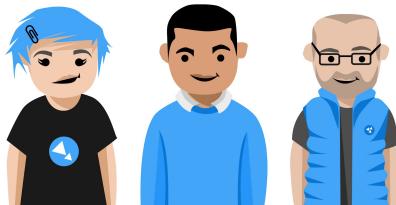
- Key concepts from last class
- Site Reliability Engineering
 - SLA, SLO, SLI
 - Error Budget
- Availability
- Reliability

DevOps

Traditional software development

- Developer for delivery and agility, Operators for stability and reliability

Let's respond to customer's requirements.



Developers

If it is not broken, let's not touch the code.



Operators

DevOps

Traditional software development

- Developer for delivery and agility, Operators for stability and reliability

DevOps as a collaborative culture to speed up the deployments

- CI/CD for incremental changes and continuous delivery
- Tools and automation to support DevOps principles
 - Example of tools: Jenkins and Docker
- Continuous delivery automates the entire process up to the point of release
- Continuous deployment automates the entire process, eliminating the need for manual approval

SRE (Site Reliability Engineering)

SRE is a practice of automating IT operations tasks.

- Focuses on response time and software reliability
- Operations-oriented automation

Examples of IT operations tasks that are automated:

- Incident response
- Production system management
- Performance monitoring
- Emergency response

SRE (Site Reliability Engineering)

Why? To encourage frequent and small releases while maintaining system reliability

Benefits of SRE:

- Metric reporting
 - Use relevant metrics to discover issues
 - Translate metrics into concrete objectives
 - metric says 99.9% success rate, set objectives to 99% to beat the competition.
- Clarify and evaluate customer expectations
- Better productivity
 - SRE team as a specialized team for reliability
- Detect issues before they reach end-users
 - SRE proactively monitor the system at a higher-level perspective than developers
 - Find and fix issues that only occur during production

SRE (Site Reliability Engineering)

SRE has three key components:

- Service Level Agreement (SLA)
 - Agreement that the business team make with the customer
- Service Level Objective (SLO)
 - Objectives that the SRE team agrees to meet
- Service Level Indicator (SLI)
 - Real metrics on the system performance

Site reliability engineers build **Service Level Indicators** to drive **Service Level Objective** which are then used to negotiate the **Service Level Agreement** with the customer.

Availability

Availability is the percentage of time that a system is operational.

- Expressed in terms of the number of “nines” in the availability percentage

Common levels of availability and its corresponding downtime per year

- 90%, “One Nines”: Around 36 days of downtime per year
- 99%, “Two Nines”: Less than 4 days
- 99.9%, “Three Nines”: Just over 8 hours
- 99.99%, “Four Nines”: Less than an hour of downtime
- 99.999%, “Five Nines”: Only about 5 minutes of downtime

Availability

MTBF (Mean Time Between Failure) is the average time between two consecutive failures.

$$\text{MTBF} = \frac{\text{Operational Time}}{\text{Number of Failures}}$$

MTTR (Mean Time to Repair) is the average time it takes to restore the system after a failure

$$\text{MTTR} = \frac{\text{Total Repair Time}}{\text{Number of Failures}}$$

Availability is the percentage of time that a system is operational

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

Availability

Given a system with MTBF of 100 hours and MTTR of 2 hours,
based on the availability formula:

$$\text{Availability} = \frac{100}{100 + 2} \times 100\% = \frac{100}{102} \times 100\% \approx 98.04\%$$

The system is estimated to be available 98.04% of the time.

Reliability

Reliability is the likelihood that a system will perform its function without failure.

- Calculated by MTBF (Mean Time Between Failure)

How to define reliability standards?

- Risks
 - Safety-critical systems vs personal websites
- Customer expectations
 - Translating “it should never fail” into “99.99% chance of successful operation”
 - Function, environment, and probability of failure
 - E.g., Sales transaction should work 99.99% of the time during Black Friday sales
- SLAs (Service level agreements)

Service Level Agreement (SLA)

An SLA is an agreement between a service provider and a customer that outlines the customer's expectations.

- Why? To provide a mutual understanding of service expectations between the business team and the customer
- Who is involved in SLA?
 - Tied to business and production decisions
 - Business team and customer, SREs are not involved
 - SLA is a legal binding agreement
 - But SRE can help the business team in understanding the likelihood and difficulty of meeting the SLOs contained in the SLA
 - SRE also help avoid the consequences of missed SLOs in the SLA

SLA

An SLA helps to:

- To improve trust with customers
- To improve communication with stakeholders
- To improve productivity

Examples of SLAs

- First Response Time (FRT)
- Time to Resolution (TTR)
- Availability

SLA vs KPI

A Key Performance Indicator (KPI) demonstrates how effectively a company achieves key business objectives.

Examples of KPI:

- Sales revenue
- Website traffic

A SLA is a contract that outlines the expectations and responsibilities of both parties, and includes performance metrics for services the business offers.

Examples of SLA:

- Availability
- Response time

- SLA is a legal binding agreement.
- SLA for service expectations, KPI for performance
- SLA may contain penalties, KPI does not

SLA challenges

- Hard to track
- Do not always reflect business priorities
- Little flexibility in reporting
 - Yes or no question, either SLA is met or it is not
 - Causing hostility between business and SRE teams

Solution?

- Have the SRE collaborate with the business team to consider real-world scenarios.
- In the SLA, specify the Service Level Objectives (SLOs) which are the individual promises to the customer.

Service Level Objective (SLO)

An SLO is an agreement within an SLA about a specific metric like uptime or response time.

- Why? To provide the SRE team an understanding of whether they are in compliance with the SLA.
- Who is involved in SLO?
 - SRE team
 - Product team (e.g., product marketing, design and management)
 - May also include the development team

SLO opens discussion between teams to create a realistic definition of objectives.

Service Level Objective (SLO)

Examples of SLO:

- **Uptime**
 - E.g., eClass should be up 99.9% of the time
- **Response time**
 - E.g., eClass should load within one second for each page request
- **Traffic**
 - E.g., eClass should support a concurrent student count of 10,000 during course registration period.
- **Success**
 - E.g., eClass should maintain a success rate of 99.9% for course registration requests.

Service Level Indicator (SLIs)

An SLI is a metric that measures the actual performance and availability of a service.

- Why? We can't have SLOs without SLIs, SLIs measure the compliance of an SLO.
- If the SLO is set to 99.95% uptime, then the SLI is the actual measurement of your uptime.

Examples of SLIs:

- Uptime
 - Time that a service is usable
- Request latency
 - Speed at which the team respond to a request
- Error rate
 - Ratio of bad events (e.g., system failures) to total event

Error Budget

Error budget is how much downtime a system can afford without upsetting customers.

- An error budget is 1 minus the SLO of the service.
 - To respect the SLO of the service
- A 99.9% SLO service has a 0.1% error budget.

Why? To avoid overspending the time on reliability

- How can we tell if we are overspending? By error budget

The team can “spend” error budget on making unreliable changes.

Error Budget

Example: Given an availability of 95% for SLO, what is your error budget per year?

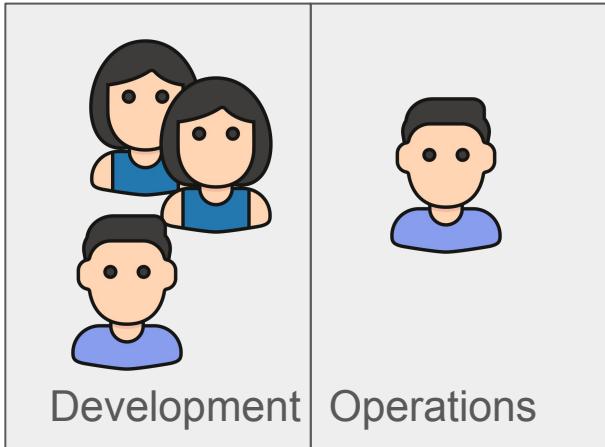
- 95% availability corresponds to 8.4 hours downtime per week
 - Downtime per week = $24 * 7 * 0.05 = 8.4$ hours
- 8.4 hours is the error budget you can spend in one week on making the system more reliable or pushing for new features.
- Whether you spend it on making the system more reliable or new features, depends on whether or not you exceed your current budget

Error Budget

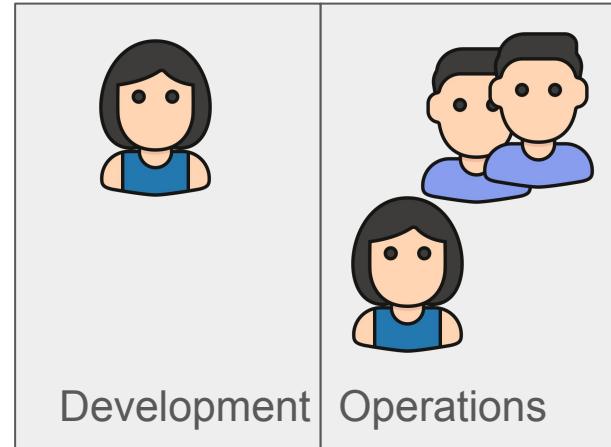
What happens if you overspend your error budget?

- Shift the priorities from development to improving the reliability of the system
- Once your error budget has recovered, you can work on functionalities again

When error budget is **underspent**



When error budget is **overspent**



SRE best practices

- Use non-technical language in SLAs
 - Keep SLA language simple to avoid misunderstandings with the customer
- Fewer SLOs as possible
 - Commit to fewer SLOs, and focus on the objectives that matter to the customer
- Build an error budget
 - Accept failures, leave room for agility (changes and new features)
- Don't shoot for the moon
 - Better to underpromise and overdeliver
- Do not include every metric as an SLI
 - Pick the SLIs based on your core SLOs

SRE vs DevOps

SRE focuses on reliability

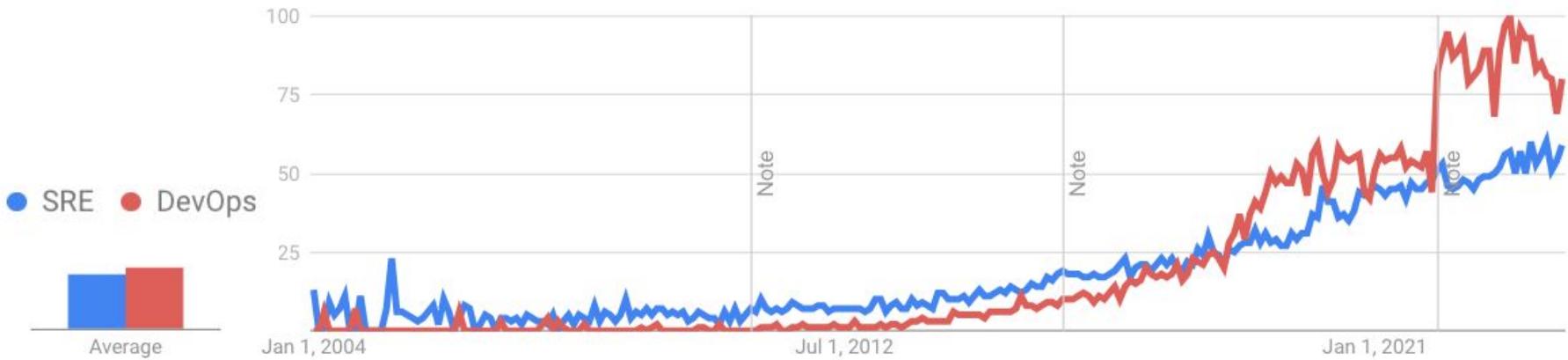
- Operating and monitoring
- Incident response
- Capacity planning

DevOps focuses on delivery

- CI/CD
- Release automation
- Configuration management

Note: SRE is to DevOps what Scrum is to Agile; one implementation of a philosophy and principles, but not a 100% subset.

Trends over time in Canada

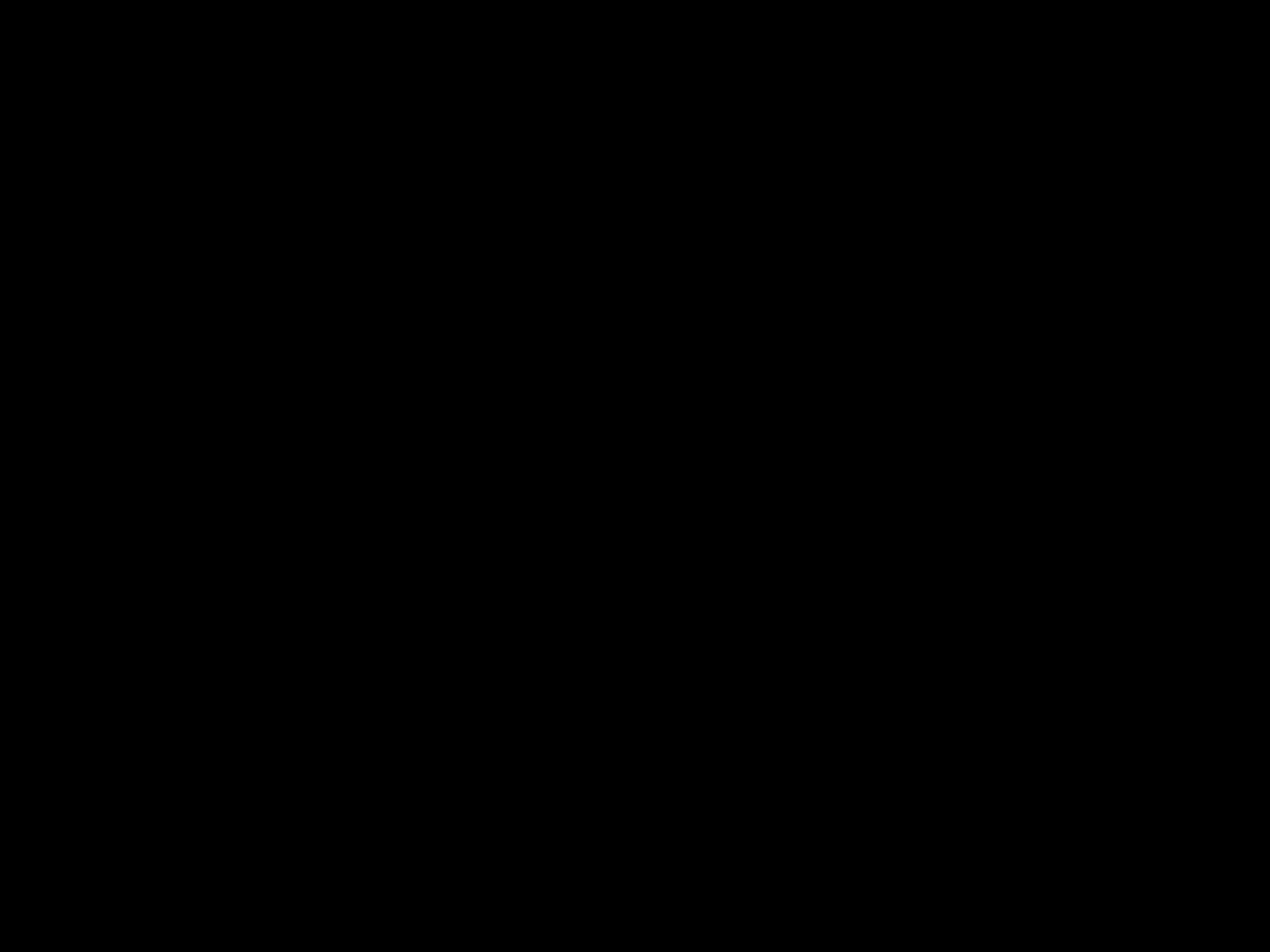


DevOps and SRE are not two competing methods, but rather similar approaches to design and deliver better software faster.

- SRE as one implementation of DevOps best practices

Google SRE team explains SRE

- [Article: Lessons Learned from Twenty Years of Site Reliability Engineering](#)
- [Video: Site reliability engineers at Google explaining SRE](#)



Google SRE team explains SRE

- SRE is a discipline that include software, hardware, networking, and human
- SRE as a new role, a team that is highly-skilled in monitoring and incident responses
- As a SRE team, the goal is to
 - Improve the system overtime
 - Mitigate the issues
 - Understand and learn from the issues

From a SRE at Shopify

Team structure

- Operations engineer only (including the manager is an operation engineer)
- Vs. DevOps that is a cross-functional team that includes developer, designer, data scientists, product manager

Daily tasks

- Maintain the infrastructure so that it fulfills SLO (service-level objective) under different loads

Lecture 4

Fault-Tolerant Design

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Fault tolerance
- Fault recovery techniques
 - Backward error recovery
 - Forward error recovery
- Multiple versions techniques
 - Recovery blocks
 - N-version programming
 - N self-checking programming

SRE (Site Reliability Engineering)

SRE is a practice of automating IT operations tasks.

- Focuses on response time and software reliability
- Operations-oriented automation

Examples of IT operations tasks that are automated:

- Incident response
- Production system management
- Performance monitoring
- Emergency response

SRE (Site Reliability Engineering)

SRE has three key components:

- Service Level Agreement (SLA)
 - Agreement that the business team make with the customer
- Service Level Objective (SLO)
 - Objectives that the SRE team agrees to meet
- Service Level Indicator (SLI)
 - Real metrics on the system performance

Site reliability engineers build **Service Level Indicators** to drive **Service Level Objectives** which are then used to negotiate the **Service Level Agreements** with the customer.

SRE (Site Reliability Engineering)

There are two important -ability in SRE:

- Reliability is the likelihood that a system will perform its function without failure.
 - Calculated by MTBF (Mean Time Between Failure)
 - The higher MTBF, the more reliable and available the system becomes
- Availability

Availability

Availability is the percentage of time that a system is operational.

- Expressed in terms of the number of “nines” in the availability percentage

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

A SLA is a contract that outlines the expectations and responsibilities of both parties, and includes performance metrics for services the business offers.

Examples of SLA:

- Availability

How to improve availability?

To increase availability, there are two solutions:

- Reduce the frequency of faults
 - By upgrading the hardware, expensive
 - By continuously improving the software, slow and expensive
 - Challenge: we can never reduce the probability of faults to zero
- Design systems that continue to work despite some faulty components; this solution is called **Fault Tolerance**

Software fault tolerance

Fault tolerance is the ability for systems to continue functioning as a whole despite the faults

- Example of fault tolerance: while a PDF reader may crash when editing a corrupted PDF file, the PDF reader will restart itself after saving the information
- Why? Building safety-critical systems where there is more requirements on building safe and reliable softwares
 - E.g., aircraft, electronic banking, automobiles
 - Failures in these systems can cause catastrophic consequences
- Fault tolerance is implemented and prioritized in these systems

State-of-the-practice on software faults

With the current state-of-the-practice, such as DevOps and SRE (People + Practices + Tools together):

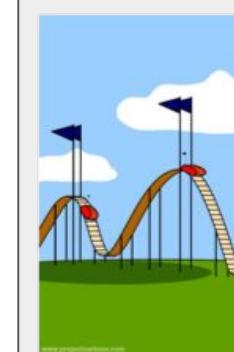
- Fewer faults introduced, but the software is never assumed fault-free
- Some faults can be allowed to remain in the system

Unique challenges to software engineering, faults can happen due to design and specifications:

- Building the product wrong
- Building the wrong product



Product
wrong



Wrong
product

Software fault vs hardware faults

- Physical vs logical faults
 - Software faults are logical faults written in programming language, which are difficult to visualize and detect.
 - Hardware faults are generally physical faults, which are limited by the number of physical components.
- Cause of faults
 - Software components fail due to bugs, hard to predict and monitor.
 - Hardware components generally fail due to wear and tear, which can be characterized and predicted over time.
- Resolution
 - Software faults are harder to fix, possibility of regression problems
 - Hardwares can simply be replaced when they break down

Fault tolerance techniques

Fault tolerance techniques provide mechanisms to the software system to prevent failures in the present of a fault.

It consists of:

- Error detection: identification of an error state
- Error diagnosis: assessment of the damage caused by the error
- Error confinement: prevention of further damages
- Error recovery: replacing the error state with an error-free state

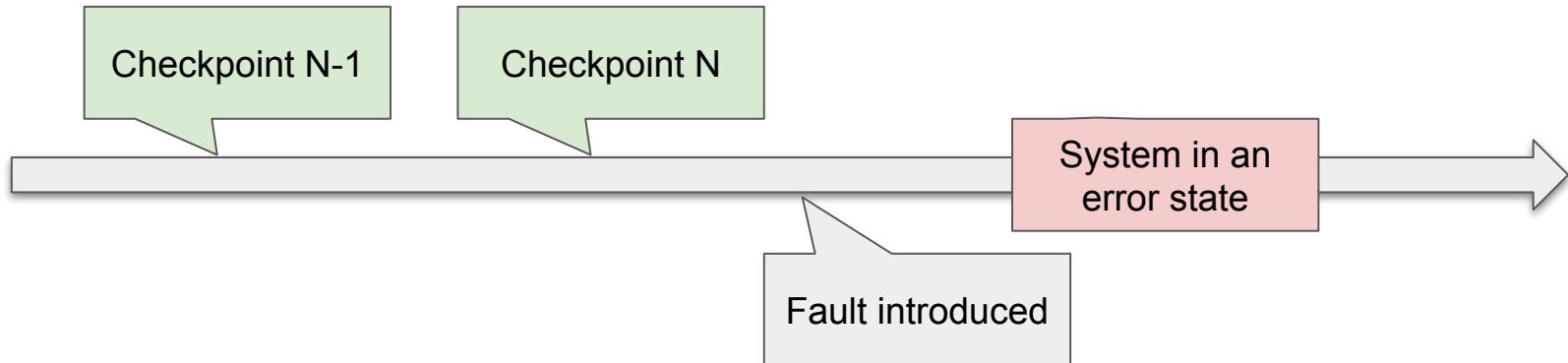
Schedule for today

- Key concepts from last class
- Fault tolerance
- Fault recovery techniques
 - Backward error recovery
 - Forward error recovery
- Multiple versions techniques
 - Recovery blocks
 - N-version programming
 - N self-checking programming

Backward error recovery

The system state is saved at predetermined recovery points (also known as checkpoints)

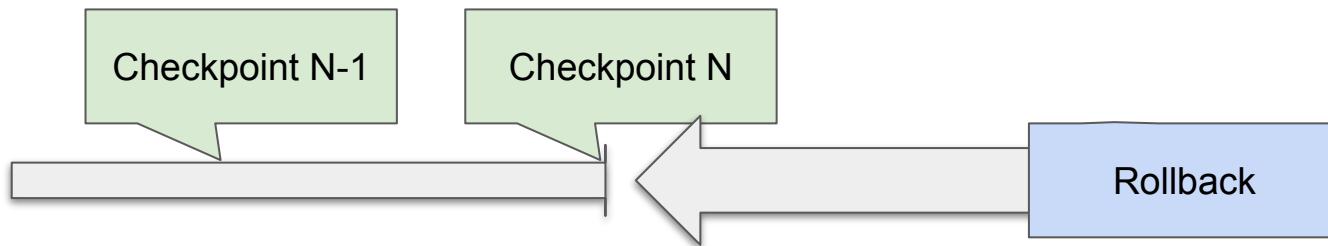
- Incremental checkpointing on error-free state
- When the system is in an error state, recover the system by rolling back to a previously saved state



Backward error recovery

Example of implementations

- Roll back the previous checkpoint
- Wait and test the checkpoint for the next request
- Repeat until the system is in an error-free state



Backward error recovery

Advantages

- No knowledge on the errors
- Generalizable solution
- Perfect for faults that are arbitrary or unpredictable

Disadvantages

- Requires significant resources
- Requires identification of consistent states (hard to define)
- No guarantee that the error will not persist when recovering (e.g., design errors)

Forward error recovery

The system builds a new error-free state from which it can continue to operate with error compensation for the corrupted and missed data.

- Assume that the location and cause of errors can be accurately assessed
- Build a new error-free state from it

The system continues to run in a “degraded” mode

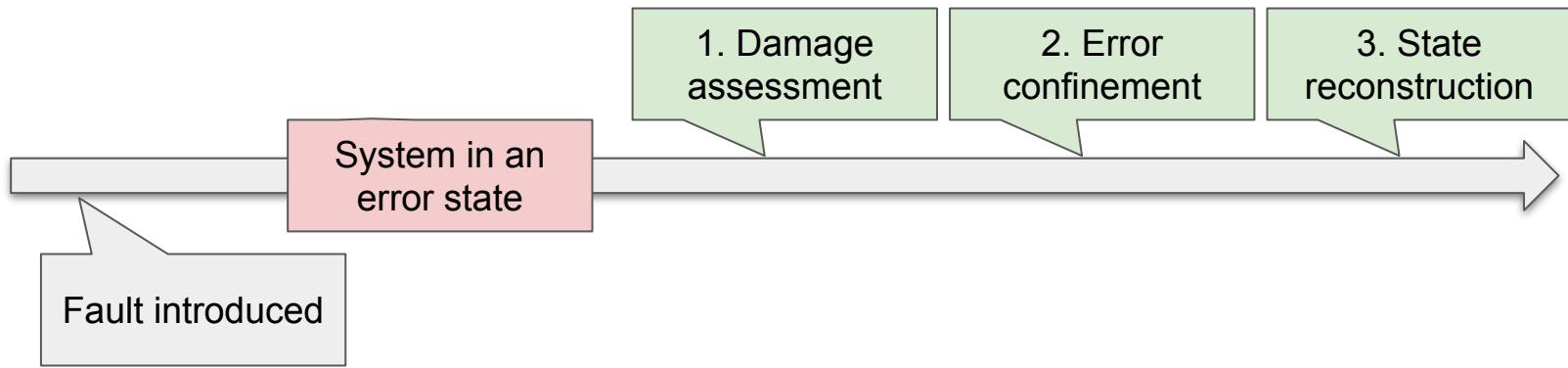
- Example: A printer operating in forward error recovery
 - Printing jobs can enter their critical sections
 - But not ordered by timestamp

Error compensation

Error compensation is based on the algorithm that uses redundancy to select the correct/acceptable version.

- For example, in recovery blocks, the state transformation can be induced by switching from a failed state to a non-failed one executing the same task
- Error compensation may be applied all the time, whether or not an error occurred
 - Fault masking (e.g., N-version programming)

Forward error recovery



Example of implementation

1. Determine to what extent the system has been corrupted
2. Structure the system to minimize the damage caused by the error
3. Make selective corrections to the system state

Forward error recovery

Advantages

- More efficient use of time/memory

Disadvantages

- Design specifically for a particular system
- Depends on the accuracy of damage assessment and prediction
- Anticipate the potential errors in advance
- Not applicable for unpredicted errors

Schedule for today

- Key concepts from last class
- Fault tolerance
- Fault recovery techniques
 - Backward error recovery
 - Forward error recovery
- **Multiple versions techniques**
 - Recovery blocks
 - N-version programming
 - N self-checking programming

Classification of fault tolerance techniques

- Single version techniques: use the redundancy applied to a single version of a software module to detect and recover from faults.
 - Exception handling
- Multiple version techniques
 - Recovery blocks
 - N-version programming
 - N self-checking programming
- Multiple data representation techniques (not covered in class)
 - Retry blocks
 - N-copy programming

Exception handling

Exception handling is the interruption of normal operation to handle abnormal responses.

Possible events triggering the exceptions:

- Interface exceptions
 - signaled by a module when it detects an invalid service request
- Local exceptions
 - signaled by a module when its fault detection mechanism detects a fault
- Failure exceptions
 - signaled by a module when it has detected that its fault recovery mechanism is enable to recover successfully

Classification of fault tolerance techniques

- Single version techniques
 - Exception handling
- Multiple version techniques: use two or more versions of the same software module to achieve fault tolerance through software redundancy.
 - Recovery blocks
 - N-version programming
 - N self-checking programming
- Multiple data representation techniques (not covered in class)
 - Retry blocks
 - N-copy programming

Multiple version techniques

Recovery blocks (by Randell, 1975, 2714 citations)

System structure for software fault tolerance

B Randell - ... of the international conference on Reliable software, 1975 - dl.acm.org

... In **software** the redundancy required is not simple replication ... **software fault tolerance** that we have developed can be ... Thus we start by considering the problems of **fault tolerance**, ie of ...

[☆ Save](#) [万分 Cite](#) [Cited by 2714](#) [Related articles](#) [All 23 versions](#)

N-version programming (by Chen and Avizie, 1978, 1001 citations)

N-version programming: A fault-tolerance approach to reliability of software operation

L Chen, [AAvizienis](#)

Proc. 8th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-8), 1978 • inf.pucrs.br

[☆ Save](#) [万分 Cite](#) [Cited by 1001](#) [Related articles](#) [All 7 versions](#) [»](#)

N self-checking programming (by Laprie, 1987, 117 citations and 1990, 549 citations)

Hardware and software fault tolerance: Definition and analysis of architectural solutions

JC Laprie, [J Arlat](#), C Béounes, [K Kanoun](#), C Hourtolle - Proc. 17th Int. Symposium on ..., 1987

[☆ Save](#) [万分 Cite](#) [Cited by 117](#) [Related articles](#)

Definition and analysis of hardware-and-software fault-tolerant architectures

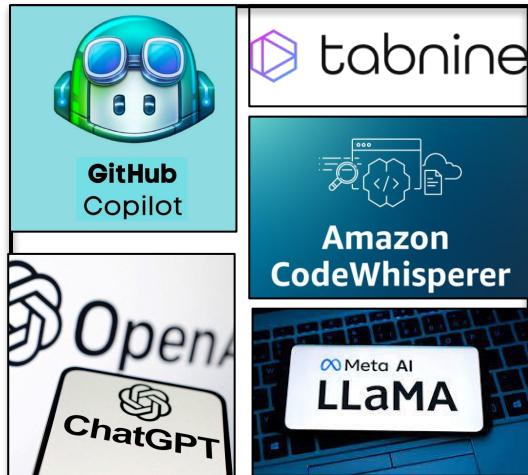
JC Laprie, [J Arlat](#), C Beounes, [K Kanoun](#)

Predictably Dependable Computing Systems, 1995 • Springer

[☆ Save](#) [万分 Cite](#) [Cited by 549](#) [Related articles](#) [All 15 versions](#)

Multiple version techniques

Can we use fault tolerant techniques to assist in building better AI technologies?



ChatGPT for Programmers: Build Python Apps in Seconds

Ardit Sulce

4.6 ★★★★★ (272)

1.5 total hours • 18 lectures

This screenshot shows a course page for a ChatGPT course. It features a central image of a person sitting at a desk with a computer monitor displaying code, accompanied by a small orange robot-like character. Above the image are icons for ChatGPT, Python, and TensorFlow. Below the image is the course title, author, rating, and duration.



<https://github.com/OpenBMB/ChatDev>

Classification of fault tolerance techniques

- Single version techniques
 - Exception handling
- Multiple version techniques: use two or more versions of the same software module to achieve fault tolerance through software redundancy.
 - Recovery blocks
 - N-version programming
 - N self-checking programming
- Multiple data representation techniques (not covered in class)
 - Retry blocks
 - N-copy programming

Recovery blocks

The recovery blocks technique

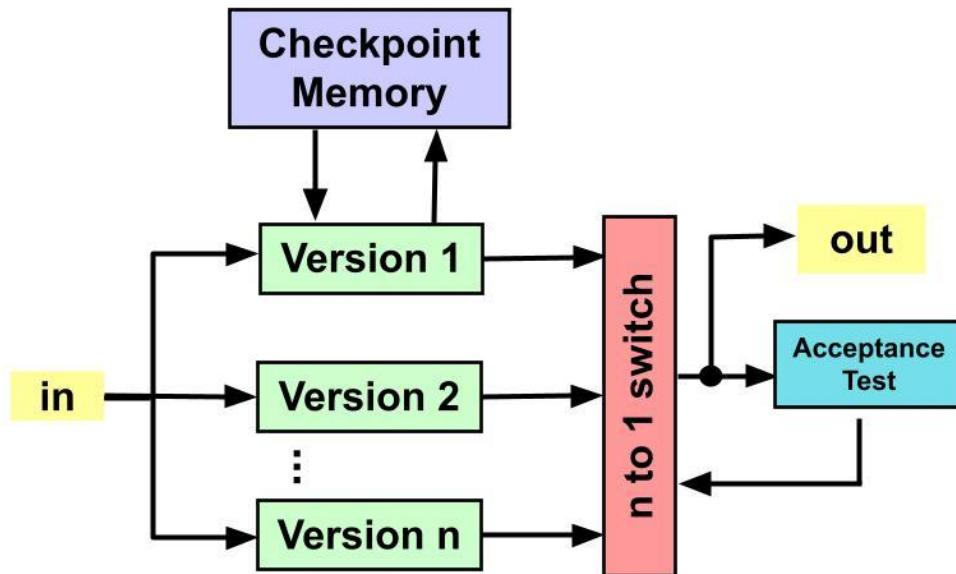
- Views the entire system as fault recoverable blocks
- Operates with an acceptance test, which confirms the results of different versions of the same function

There are three main actors in a recovery blocks technique:

- Primary version
 - The main implementation of a function
- Acceptance tests
 - Condition that the system is expected to meet
- Alternative versions
 - Each version is a different implementation of the same function

Recovery blocks

Example



1. Run the primary version for the acceptance test
2. If the primary version fails, roll back the state of the system before the execution
3. Run the next version for the same acceptance test until there is an acceptable output
4. If none produce acceptable outputs, the system fails

Recovery blocks

Cons:

- Reliability of the approach depends on the coverage and quality of the acceptance test (the decision mechanism for versions selection)
- State saving mechanisms can cause overheads
- Acceptance test needs to be small and fast to execute
 - Faster than the function itself
- Acceptance test used in an ad-hoc basis
 - If such tests exist, the faults would have already been fixed

N-version programming

The N-version programming technique

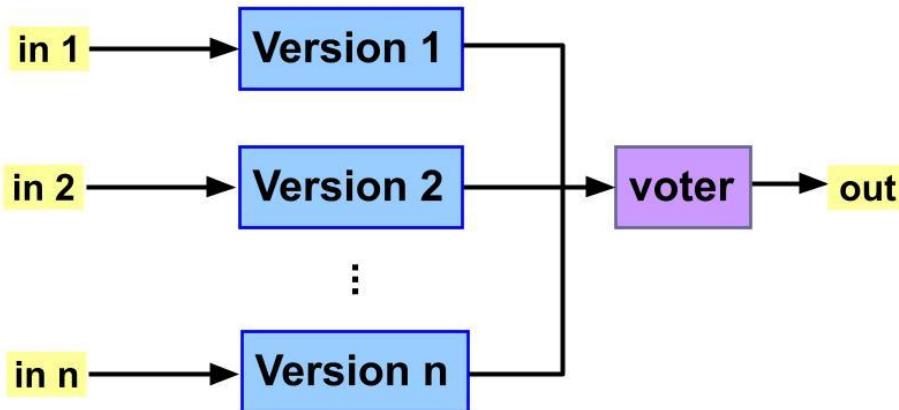
- Executes N versions of the function in parallel on identical input
- Returns the results based on a voting of the individual outputs

There are two main components:

- Versions
 - Different implementations of the same function
- Voter
 - A selection algorithm based on the individual outputs, or the consequences of the error
 - E.g., majority win, generalized median

N-version programming

Example



1. Run all versions concurrently
2. Run the decision mechanism based on the voter (e.g., majority win)
3. Return the most common output from the individual versions

N-version programming

Intuition behind N-version programming

- Have N teams of developers managing their own implementation of the software.
- If one of the versions fail, we can use the others as backups.
- The probability of two or more versions of a software system containing the same fault is very low.

Cons:

- The technique depends on design diversity, which may lead to other problems (e.g., specification/design errors)

Recovery Blocks vs N-version programming

N-version programming has the versions executed concurrently, while the recovery blocks execute the versions in sequence.

- In real-time systems, cost in time is always prioritized.
- N-version programming is generally more applicable

Different decision mechanisms on version selection

- N-version programming run a single decision mechanism (the voter)
- Recovery blocks require each version to run the acceptance test

Recovery blocks assume that a single acceptance test is enough to detect the fault

- Not always the case in real-world applications

N self-checking programming

The N self-checking programming technique combines the recovery blocks and N-version programming together.

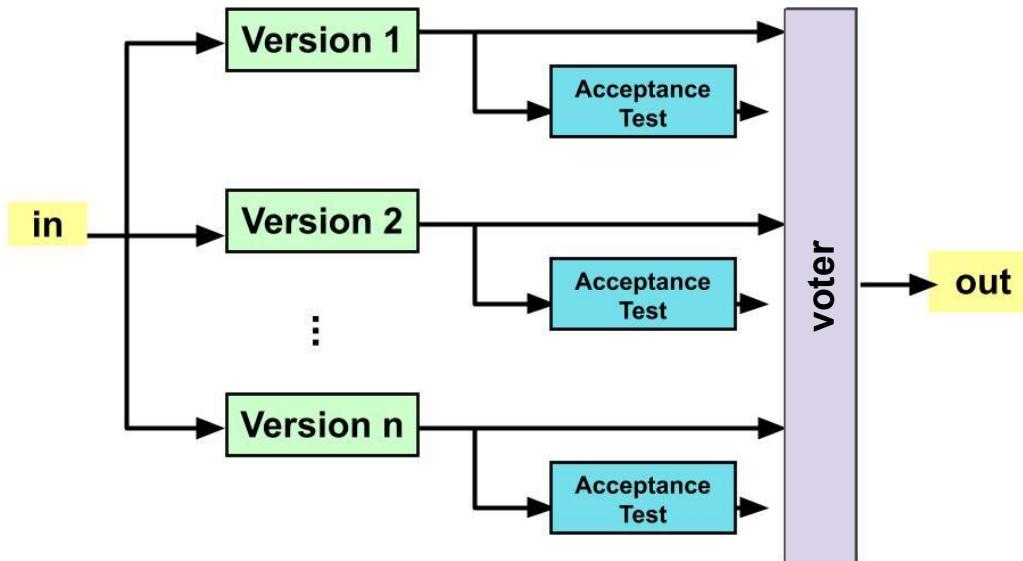
- Executes N versions of the function concurrently, or in sequence on identical input
- Return the results based on the acceptance tests

There are two main components:

- Versions
- Acceptance tests
 - Separate tests for each version

N self-checking programming

Example: N self-checking by acceptance tests



1. Run all versions concurrently or sequentially
2. Run the corresponding acceptance test for each version
3. Run the voter
4. Return the majority agreement as output (only applied to versions that have passed their acceptance test)

N self-checking programming

Cons:

- Include more overheads than N-version programming and “N” times the overhead of recovery blocks
- Vulnerable against design and specification errors
- Extra efforts in deriving individual acceptance tests

Example of applications:

- Airbus A-340 airplane (put in service 30 years ago)

Other challenges on multiple version techniques

- Decision mechanisms themselves must be free from faults
 - do not involve an alternative mechanism
- Difficult to maintain and document
 - “Versions” can range from a few lines of code to an entire program
- Difficult to generalize
 - Design choices are made based on the available resources and on the specific applications

Questions

- What is the goal of software fault tolerance?
- Name the two error recovery strategies, and briefly explain how they work.
- Name the three multiple version techniques, and briefly explain how they work.

Lecture 5

Fault Removal

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- An alternative solution, fault removal
 - Functional testing
 - Example of state-of-the-art
 - Structural testing
 - Effectiveness
- System dependability
- Dependability concept summary
- Questions

How to improve availability?

To increase availability, there are two solutions:

- Reduce the frequency of faults
 - By upgrading the hardware, expensive
 - By continuously improving the software
 - slow and expensive
 - after the fault occurs
 - Challenge: we can never reduce the probability of faults to zero
- Design systems that continue to work despite some faulty components; this solution is called **Fault Tolerance**

Fault tolerance techniques

Fault tolerance techniques provide mechanisms to the software system to prevent failures in the present of a fault.

It consists of:

- Error detection: identification of an error state
- Error diagnosis: assessment of the damage caused by the error
- Error confinement: prevention of further damages
- Error recovery: replacing the error state with an error-free state

Classification of fault tolerance techniques

- Single version techniques
 - Exception handling
- Multiple version techniques: use two or more versions of the same software module to achieve fault tolerance through software redundancy.
 - Recovery blocks
 - N-version programming
 - N self-checking programming
- Multiple data representation techniques (not covered in class)
 - Retry blocks
 - N-copy programming

What makes fault tolerance important?

It is practically impossible to build a perfect system, because each individual component in the system has its own reliability measure.

- Reliability is the likelihood that a system will perform its function without failure.
 - Calculated by MTBF (Mean Time Between Failure)
- System reliability is based on the reliability of all its components.
 - Calculated based on the following formula:

$$R = (1 - F1) * (1 - F2) * (1 - F3) * (1 - FX)....$$

where R refers to the overall system reliability, F1 refers to the failure rate of the first component and so on.

What makes fault tolerance important?

It is practically impossible to build a perfect system, because each individual component in the system has its own reliability measure.

- Reliability is the likelihood that a system will perform its function without failure.
 - Calculated by MTBF (Mean Time Between Failure)
- Since $1 - \text{failure rate} = \text{reliability}$, the reliability of all its components.
 - Calculated based on the following formula:

$$R = (1 - F1) * (1 - F2) * (1 - F3) * (1 - FX)....$$

where R refers to the overall system reliability, F1 refers to the failure rate of the first component and so on.

What makes fault tolerance important?

System reliability formula:

$$R = (1 - F1) * (1 - F2) * (1 - F3) * (1 - FX)....$$

Example:

- Suppose each component in a system has the reliability 99.99%
- A system consisting of 100 (non-redundant) components will have the reliability 99.01%
- A system consisting of 10,000 components will have the reliability 36.79%

As the complexity of a system grows, its reliability significantly decreases.

Schedule for today

- Key concepts from last class
- An alternative solution, fault removal
 - Functional testing
 - Example of state-of-the-art
 - Structural testing
 - Effectiveness
- System dependability
- Dependability concept summary
- Questions

How to improve availability?

To increase availability, there are two solutions:

- Reduce the frequency of faults
 - By upgrading the hardware, expensive
 - By continuously improving the software
 - slow and expensive
 - after the fault occurs
 - Challenge: we can never reduce the probability of faults to zero
- Design systems that continue to work despite some faulty components; this solution is called **Fault Tolerance**

How to improve availability?

To increase availability, there are two solutions:

- Reduce the frequency of faults; This solution is called **Fault Removal**
 - By upgrading the hardware, expensive
 - By continuously improving the software
 - slow and expensive
 - after the fault occurs
 - Challenge: we can never reduce the probability of faults to zero
- Design systems that continue to work despite some faulty components; this solution is called **Fault Tolerance**

Fault removal

- Why? Despite fault tolerance efforts, not all faults are tolerated, so we need fault removal.
 - To keep in mind: fault tolerance is a must-have property for safety-critical systems.
 - But for software that we use everyday, we want to remove faults to improve user experience.
- Improving **system dependability** by:
 - Detecting existing faults through software verification and validation
 - Eliminating the detected faults

Fault removal as two concepts:

1. as a solution to improve availability, and thus dependability
2. as a solution for faults that affect user's daily activities (not tolerated)

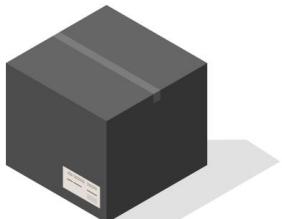
Software testing

Software testing is one of the most common fault removal techniques.

There are two types of software testing:

- Functional testing (black-box testing)
- Structural testing (white-box testing)

We do not know anything,
system as a black box



We know everything,
system as a white box

Schedule for today

- Key concepts from last class
- An alternative solution, fault removal
 - Functional testing
 - Example of state-of-the-art
 - Structural testing
 - Effectiveness
- System dependability
- Dependability concept summary

Functional testing

Functional testing checks the functionality of a program works as per the software requirements.

- In other words, functional testing checks for design errors.
- Also known as black-box testing as no knowledge of the internal code is required.

Example of functional testing:

- Suppose we test a program which adds two integers
- A functional test verifies the implemented operation is indeed addition, rather than multiplication.

Functional testing

- Functional testing: functionality + requirements

Why?

- Does the swing on the tree functions as a swing?
 - Yes
- Does the swing satisfies the user requirements?
 - Can the user use with swing?
 - No



Types of functional testing

Five types of functional testing (covered in class):

- Unit testing
- Integration testing
- Acceptance testing
- Regression testing
- ... and more
- Recovery testing (not covered in class)
- Sanity testing (not covered in class)
- Smoke testing (not covered in class)

Types of functional tests

- Unit testing allows developers to verify the quality and reliability of individual units of the code are working as expected.
- Integration testing verifies that the different units of the code as a combined entity is reliable.
- Acceptance testing verifies whether a component satisfies the user requirements / reliable from the end user's perspective.
 - In modern software development, it is part of agile methodology.
- Regression testing ensures the software reliability by identifying unintended side effects (e.g., new bugs) after code changes.

Example of state-of-the-art

For web/Java applications, Selenium can be used for functional testing to detect and remove errors/faults.

Acceptance Test 1:

- Open browser
- Enter username <username>
- Enter password <password>
- Press submit button
- Assert text <Course Equivalency System>

Schedule for today

- Key concepts from last class
- An alternative solution, fault removal
 - Functional testing
 - Example of state-of-the-art
 - Structural testing
 - Effectiveness
- System dependability
- Dependability concept summary

Structural testing

Structural testing checks the internal structure or internal implementation of a program for errors.

- Structural testing focuses on how the system is doing rather than the functionality
- Also known as white-box testing as the access of the code is required

Example of structural testing:

- Suppose we test a program which adds two integers
- A structural test checks the “steps that lead to the sum being calculated”, rather than whether the returned output is correct.

Types of structural testing

Examples of structural testing:

- Mutation testing: assess the quality of the test cases which should be robust enough to fail mutant code.
- Data flow testing: examines the data flow with respect to the variables used in the code.
- Control flow testing: examines the execution paths in the code.

Structural testing effectiveness

The effectiveness of structural testing is expressed in terms of test coverage metrics which measure the portion of code exercised by tests.

- Statement coverage = $(\text{Number of Statements Exercised} / \text{Total Number of Statements}) \times 100\%$
- Branch coverage = $(\text{Number of Decisions Outcomes tested} / \text{Total Number of Decision Outcomes}) \times 100\%$
- Path coverage = $(\text{Number Paths Exercised} / \text{Total Number of Paths in the Program}) \times 100\%$

Statement coverage

100% Statement coverage requires that each executable statement of a program is executed at least once during a test.

- Statement coverage only checks whether the loop body was executed or not
- It does not report whether loops reach their termination condition
- Statement coverage is insensitive to some control structures
 - E.g., && or || operators, switch statements

Example

```
x = 0;  
if (condition)  
    x = x + 1;  
y = 10/x;
```

Test 1 where condition = true

- 100% statement coverage
- No error found in the code

Example

```
x = 0;  
if (condition)  
    x = x + 1;  
y = 10/x;
```

Test 1 where condition = true

- 100% statement coverage
- No error found in the code

Test 2 where condition = false

- 75% statement coverage
- Error found in the code

Example

```
x = 0;  
if (condition)  
    x = x + 1;  
y = 10/x;
```

Test 1 where condition = true

- 100% statement coverage
- No error found in the code

Take-home 1: there
is an insensitivity of
statement coverage
to control structures.

Test 2 where condition = false

- 75% statement coverage
- Error found in the code

Take-home 2: 100%
statement coverage
does not mean there
is no bug in the
code.

Branch coverage

100% Branch coverage requires that each branch of a program is executed at least once during a test.

- Branch coverage checks boolean expressions as one predicate regardless of the presence of operators.
 - E.g., (condition A && condition B) is treated as one predicate
- Other statements such as switch statements and exception handlers are treated similarly.

Example

```
if (condition1)  
    x = 0;  
else  
    x = 2;  
if (condition2)  
    y = 10*x;  
else  
    y = 10/x;
```

Test 1 where condition1 = true,
and condition2 = true,

Test 2 where condition1 = false,
condition2 = false,

- 100% branch coverage
- No error found in the code

Test 3 where condition1 = true,
and condition2 = false,

- 50% branch coverage
- Error found in the code

Take-home 1: 100%
branch coverage
does not mean there
is not bug in the
code.

Branch coverage =
(Number of Decisions
Outcomes tested / Total
Number of Decision
Outcomes) x 100 %

Path coverage

Path coverage requires that each of the possible paths through the program is followed during a test.

- The most reliable metric, however, not applicable to large programs.
 - Number of paths is exponential to the number of branches
- Suppose a test that takes 1 microsecond (10^{-6} sec) to execute
- Testing all paths of a program that contain 30 if-statement will take 18 minutes
 - $(2^{30} * 10^{-6} \text{ sec}) / 60 = 18 \text{ min}$
- Testing all paths of a program that contain 60 if-statement will take 366 centuries
 - $(2^{60} * 10^{-6} \text{ sec}) / 3,600 = 3202,55974 \text{ hours} = 36,559 \text{ years} = 366 \text{ centuries}$

Example of state-of-the-art

For web/Java applications, Behat/JBehave can be used for structural testing to verify user stories.

1. Write story

Plain
text

Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0
When stock is traded at 5.0
Then the alert status should be OFF
When stock is traded at 16.0
Then the alert status should be ON

2. Map steps to Java

POJO

```
public class TraderSteps {  
    private TradingService service; // Injected  
    private Stock stock; // Created  
  
    @Given("a stock and a threshold of $threshold")  
    public void aStock(double threshold) {  
        stock = service.newStock("STK", threshold);  
    }  
    @When("the stock is traded at price $price")  
    public void theStockIsTraded(double price) {  
        stock.tradeAt(price);  
    }  
    @Then("the alert status is $status")  
    public void theAlertStatusIs(String status) {  
        assertThat(stock.getStatus().name(), equalTo(status));  
    }  
}
```

JBehave

- Is a behavior-driven development tool
- Lets you write tests in form of user stories.

Structural testing vs functional testing

Implementation vs specifications

- Structure testing evaluates code structure or internal implementation of the code
- Functional testing verifies whether the software functions in accordance with functional specifications

Code vs requirements

- Structure testing requires a understanding of the code
- Functional testing requires a understanding of the software's requirements

Logic error vs system error

- Structure testing find errors in the internal code logic
- Functional testing ensures that the system is error-free

Schedule for today

- Key concepts from last class
- An alternative solution, fault removal
 - Functional testing
 - Example of state-of-the-art
 - Structural testing
 - Effectiveness
- **System dependability**
- Dependability concept summary

System dependability

Dependability is the ability of a system to deliver its intended level of service to its users.

- Dependability reflects the user's degree of trust in the system
- In modern systems, dependability becomes important not only for traditional safety-, mission-, and business-critical applications, but also for everyday systems.

Why is it important?

- Undependable systems can cause information loss
 - Consequence: a high recovery cost.
- Systems that are not dependable may be rejected by their users / not trustworthy

Why trustworthiness matters

Fiscal Year From:	2019-2020
Fiscal Year To:	2022-2023
Keywords in:	trust
Research Subject:	Computer systems software Software and development Software engineering
By Institutions:	All
Add or Modify Criteria New Search	

NSERC's Fundings Database

- Research subject: software
- Keywords in (project title/summary): trust
- Year 2019 to 2023
- 96 records of successful application

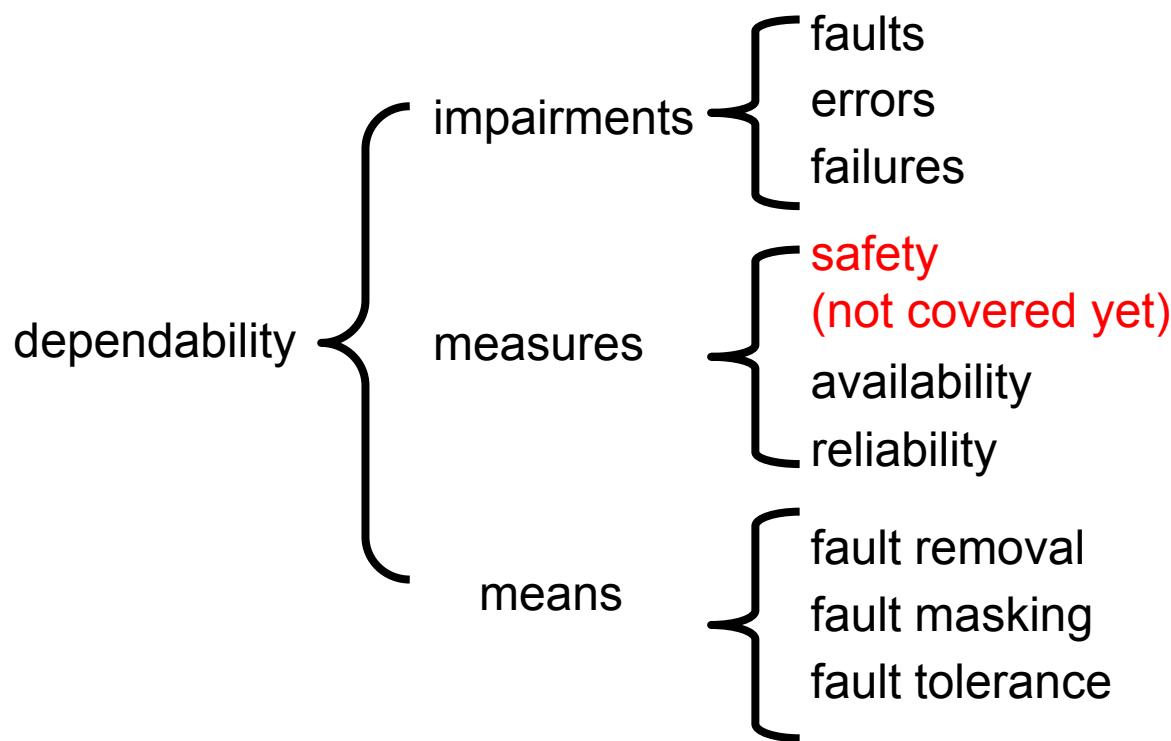
Showing 1 to 10 of 96 records				Show <input type="button" value="10"/> records
Name	Project Title	Amount(\$)	Fiscal Year	Program
		294,454	2021-2022	Collaborative Research and Training Experience
		294,454	2022-2023	Collaborative Research and Training Experience
		291,117	2020-2021	Collaborative Research and Training Experience
	Planning program			

Dependability

Dependability consists of three characteristics:

- Attributes: properties which are required of a system.
 - Depending on the application design and behavior
 - E.g., for ATMs, availability is important
 - E.g., for heart pacemaker, reliability is important
- Impairments: reasons for a system to cease to perform its function
 - Threats to dependability
- Means: methods and techniques enabling the development of a dependable system
 - E.g., fault tolerance

Dependability concept summary



Dependability means:

- Fault tolerance is one of the methods for achieving dependability
- Fault tolerance can be used in combination with other methods (e.g., fault removal)

Questions

- Give an example of a code in which a bug will not be detected in spite of 100% statement coverage.
- Does the example above implies you have 100% branch coverage?
- Is the opposite always true? In other words, does 100% branch coverage implies 100% statement coverage?
- Give an example of a code in which a bug will not be detected in spite of 100% branch coverage.

Lecture 6

Fault Localization

ECE 422: Reliable and Secure Systems Design



UNIVERSITY OF
ALBERTA

Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Fault localization
 - Traditional debugging
 - Spectrum-based technique
 - Information retrieval-based (IR-based) technique
- Deliverable

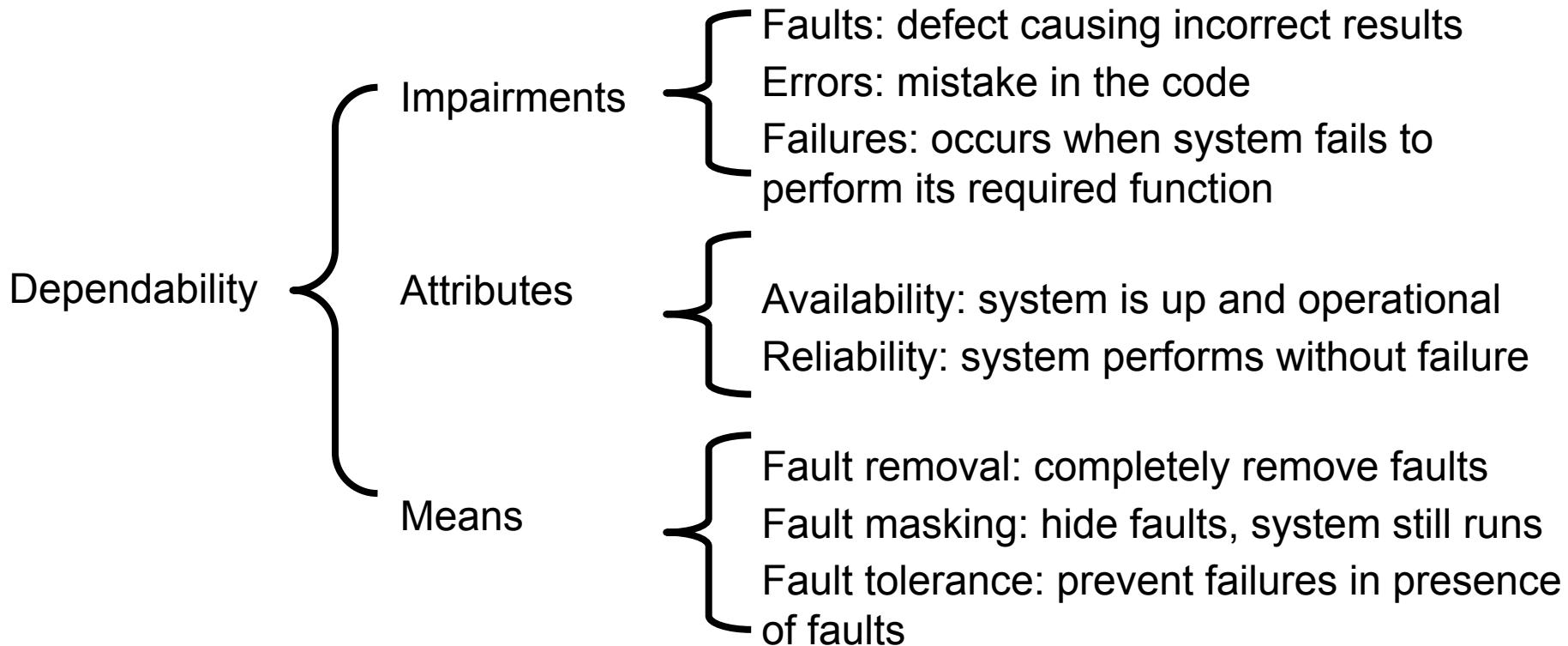
Fault removal

- Why? Despite fault tolerance efforts, not all faults are tolerated, so we need fault removal.
 - To keep in mind: fault tolerance is a must-have property for safety-critical systems.
 - But for software that we use everyday, we want to remove faults to improve user experience.
- Improving **system dependability** by:
 - Detecting existing faults through software verification and validation
 - Eliminating the detected faults

Fault removal as two concepts:

1. as a solution to improve availability, and thus dependability
2. as a solution for faults that affect user's daily activities (not tolerated)

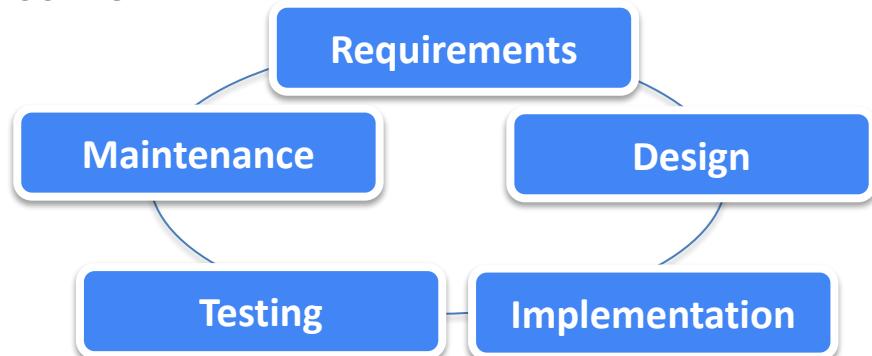
Dependability concept summary



Dependability in software development

Requirement phrase focus on the dependability attributes.

- Which dependability attributes are prioritized based on the user requirement?
 - Availability: to maximize operational time
 - E.g., the vending machine is always up and running
 - Reliability: to minimize system failures
 - E.g., the soda is never stuck in the vending machine



Dependability in software development

Design phrase focuses on the design properties.

- Do we need fault tolerance?
 - Fault tolerance as a design property
 - Fault tolerance is never integrated in the middle of the software development life cycle
 - Depend on the dependability attributes
- Which fault tolerance techniques to use?
 - Single vs multiple version fault tolerance techniques
 - Different consumption of resources
 - Multiple version techniques come with overheads for restoring the system state

Dependability in software development

Implementation phrase is about the actual implementation of the design.

- How to implement fault tolerance in the system?
 - Single vs multiple version fault tolerance techniques

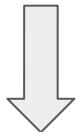
Testing phrase ensures faults are removed from the system.

- What type of testing should we prioritize on?
 - Structural and functional testing
- How do we measure the effectiveness of structural testing?
 - Coverage analysis (e.g., path, branch, and statement coverage)

Dependability in software development

Maintenance phase is about updating software to keep up with user requirements, including resolving faults in the system.

- Where is the fault?
- What are the root causes of the fault?



To answer these questions, developers use
fault localization techniques.

Fault localization

Fault localization techniques have been proposed to assist in locating and understanding the root causes of faults.

Why? Fault localization as a debugging technique to maintain the system:

- Pinpoint the location to fix in the code
- Recover fast from bugs, and reduce its impact on the users
- Reduce manual debugging efforts, more time for new feature



What makes fault localization important?

Fault tolerance, masking and removal as high level solutions / guidelines to deal with faults. Fault localization (FL) provides a solution from the coding perspective to questions like:

- Where is the fault?
 - FL provides the exact location of faults at different granularity levels.
- Which part of the system is affected?
 - FL tries to detect all the fix locations.
- How can I reproduce the faults?
 - FL provides hints such as relevant test cases revealing the fault.
- ... and more

Meta Research on software debugging

ICSE

2019

SapFix: Automated End-to-End Repair at Scale

Michael Pradel*
University of Stuttgart

Vijayaraghavan Murali
Facebook

Rebecca Qian
Facebook

TSE

2019

A Study of Bug Resolution Characteristics in Popular Programming Languages

Jie M. Zhang*, Feng Li, Dan Hao, Meng Wang, Hao Tang, Lu Zhang, Mark Harman

ICSE
2021

ISSTA

2020

Industry-scale IR-based Bug Localization: A Perspective from Facebook

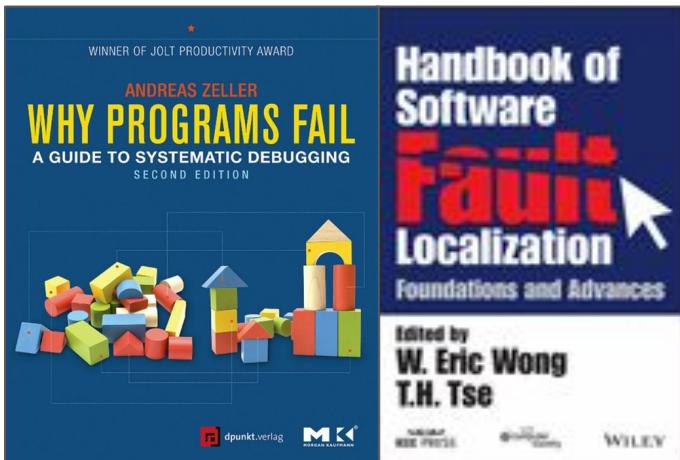
Vijayaraghavan Murali
Facebook, Inc.

Lee Gross
Facebook, Inc.

Rebecca Qian
Facebook, Inc.

Satish Chandra
Facebook, Inc.

Research community effort

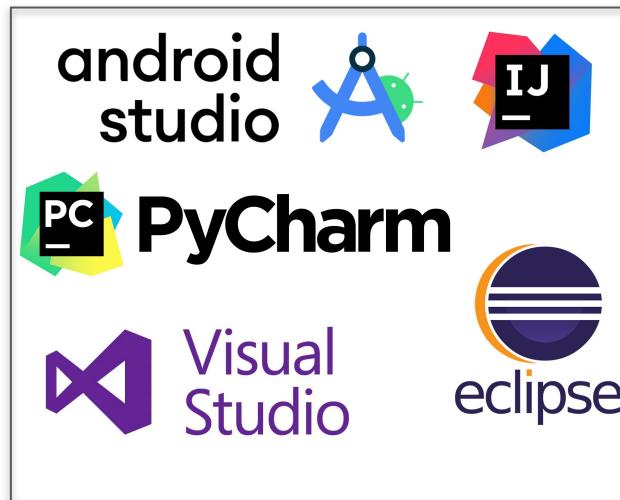


The Debugging Book

Tools and Techniques for Automated Software Debugging

by Andreas Zeller

<https://debuggingbook.org/>, 2021



This block shows a partial screenshot of a research paper search interface. It lists several papers with their titles, authors, and publication details. The visible titles include:

- 'SafeRevert: When Can Breaking Changes be Automatically Reverted?' by TAD Henderson, A Kondareddy, S Azad, E Nickell from hackthology.com - 3 days ago. Includes a 'PDF' link.
- 'Evolutionary Testing for Program Repair' by H Ruan, HL Nguyen, R Shariffdeen, Y Noller, A Roychoudhury from Seed - 6 days ago. Includes a 'PDF' link.
- 'Automated Test Case Repair Using Language Models' by AS Yaraghi, D Holden, N Kahani, L Briand from arXiv preprint arXiv:2401.06765 - 6 days ago. Includes a 'PDF' link.
- A section titled 'More articles from 6 days ago'.
- 'Two-Level Information-Retrieval-Based Model for Bug Localization Based on Bug Reports' by S Alsaeedi, AAA Gad-Elrab, A Noaman, F Eassa from Electronics - 7 days ago. Includes a 'PDF' link.
- A section titled 'More articles from 7 days ago'.
- 'DebugBench: Evaluating Debugging Capability of Large Language Models' by R Tian, Y Ye, Y Qin, X Cong, Y Lin, Z Liu, M Sun from arXiv preprint arXiv:2401.04621 - 9 days ago. Includes a 'PDF' link.

- Textbooks about software debugging alone
- Debuggers for IDEs
- New papers on FL or debugging everyday

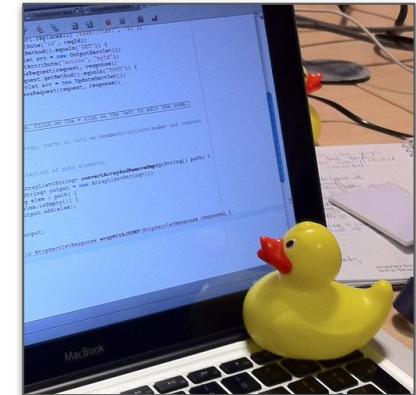
Rubber duck debugging

Introduced by Dave Thomas and Andy Hunt (credited Andrew Hunt) in “The Pragmatic Programmer” book.

From a [lists.ethernal.org post](https://lists.ethernal.org/post/111) by Andy:

- Beg, borrow, steal, buy, fabricate, or otherwise obtain a rubber duck.
- Place rubber duck on desk and inform it you are just going to go over some code with it, if that's all right.
- Explain to the duck what your code does line by line.
- At some point, you will realize what you are telling to the duck is not in fact what you are actually doing.

If you don't have a rubber duck, a co-worker works too.



Traditional debugging

Traditional debugging methods presents guides or techniques for findings faults manually.

However, there are many challenges:

- Searching process is time-consuming
- Challenging to understand the system as it evolves to be more complex
- Some faults can be time-sensitive, putting more pressures on developers

Fault localization

Fault localization present automated techniques for locating the faults in the source code.

There are many families of fault localization techniques:

- Spectrum-based techniques
- Information retrieval-based techniques
- Mutation-based techniques (not covered in class)
- Historical-based techniques (not covered in class)
- and more ...

Spectrum-based fault localization

Spectrum-based fault localization (SBFL), also known as statistical debugging, uses the results of test cases to identify the location of faults.

- Pinpoints the most suspicious program element (e.g., statement, method, file) based on the code coverage.
- Basic intuition: the location of code that is covered by more failing tests and less passing tests are more likely to contain faults.

Step 1 - Run all tests

Step 1: Run all tests

- Collect test results (passed or failed)
- Collect the code coverage (statement coverage)

	T ₁
S ₁	✓
S ₂	
S ₃	
S ₄	✓

For example

- T1 is a passing test.
- T1 covers statement 1 and 4.

Step 2 - Build test execution profiles

Step 2: Build test execution profiles

- For every executable statement in the code, collect the tests that executed that statement
- For example:
 - Statement S1 was executed by one passing test, T1
 - Statement S2 was executed by one failing test, T2
 - Statement S3 was executed by two failing tests, T2 and T3
 - Statement S4 was executed by one passing test, T1 and one failing test, T2

	T ₁	T ₂	T ₃
S ₁	✓		
S ₂		✓	
S ₃		✓	✓
S ₄	✓	✓	

Execution profile

Step 3: Calculate the suspiciousness score

Step 3: Calculate the suspiciousness score

- Use SBFL formulas to calculate a suspiciousness score for each program element
- Example of SBFL formula: Ochiai formula

$$Ochiai(element) = \frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}}$$

e_f Number of failed tests that execute the program element.

e_p Number of passed tests that execute the program element.

n_f Number of failed tests that do not execute the program element.

n_p Number of passed tests that do not execute the program element.

Step 3: Calculate the suspiciousness score

- For example

	T ₁	T ₂	T ₃
S ₃		✓	✓

$$\frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}} \rightarrow \boxed{e_f = 2 \\ n_f = 0 \\ e_p = 0} \rightarrow \frac{2}{\sqrt{(2 + 0) * (2 + 0)}} \downarrow$$

Suspiciousness
score = 1

e_f Number of failed tests that execute the program element.

e_p Number of passed tests that execute the program element.

n_f Number of failed tests that do not execute the program element.

n_p Number of passed tests that do not execute the program element.

Step 3: Calculate the suspiciousness score

	T ₁	T ₂	T ₃
S ₁	✓		
S ₂		✓	
S ₃		✓	✓
S ₄	✓	✓	

Execution profiles

$$\bullet \quad S_1 = \frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}} = 0$$

$$\bullet \quad S_2 = \frac{1}{\sqrt{(1+1) * (1+0)}} = 0.71$$

...



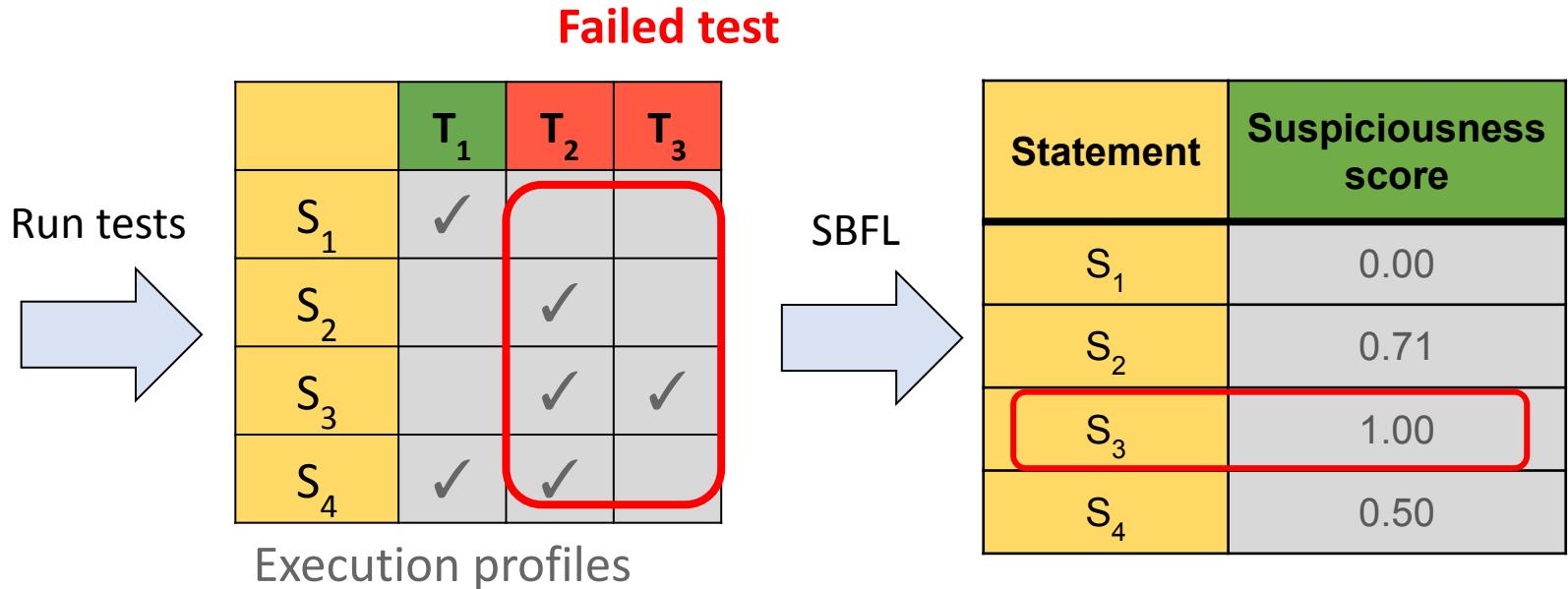
Statement	Suspiciousness score
S ₁	0.00
S ₂	0.71
S ₃	1.00
S ₄	0.50

Step 4: Rank elements by suspiciousness

Step 4: Rank the program elements by their suspiciousness score

- Output: a ranked list of suspicious elements is provided to the developer for manual bug fix.
- The element with the higher suspiciousness score is more likely to contain the fault.

Spectrum-based fault localization



Hint: program elements that are covered by more failing tests but less passing tests are more suspicious.

Limitations of SBFL

- Require the code coverage information, which may not always be available.
- Possible tie issue: same score is given to the elements covered by the equal number of passing and failing tests.
- Assume that test failures are related to the fault. Not the case for flaky tests.

Information retrieval-based fault localization

Information retrieval-based fault localization (IR-based FL) uses the textual description in the bug reports to locate the fault.

- Pinpoints the most suspicious program element (e.g., statement, method, file) based on the textual similarity between the bug description and the source code.
- Basic intuition: the description in the bug report and the faulty program element are likely to share the same tokens (words)

Step 1 - Preprocessing

Step 1.1 - Preprocess the bug report

- Text normalization: transforming text into a single canonical form
 - E.g., convert “stopwords”, “stop words”, “stop-words” to just “stopwords”
- Stopword removal: removing common, non-meaningful words
 - E.g., remove “is”, “a”
- Stemming: reducing text to their base form
 - E.g., convert “singing”, “sing”, “sung” to “sing”

Step 1 - Preprocessing

Step 1.2 - Preprocess the source code

- Keyword removal: removing programming language specific keywords
 - E.g., remove “for”, “if” for Java
- Concatenated words splitting
 - E.g., convert “getAverage” to “get” and “average”

Step 2 - Build vector space model

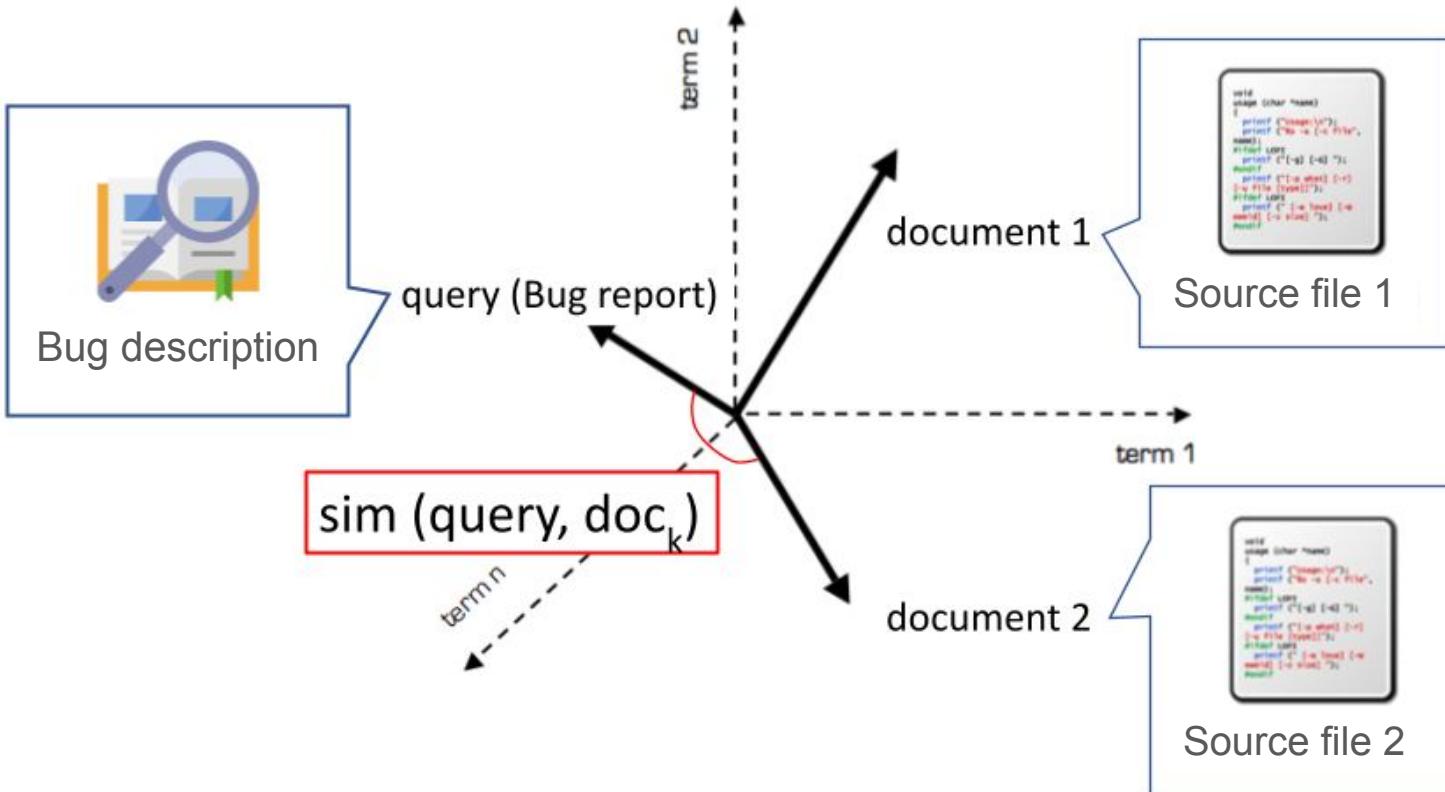
Step 2 - Build vector space model

- Vector space model: representing text documents as vectors so that we can calculate the similarity between vectors
- Intuition: transforms fault localization to a search problem

Example of search problem:

- Search query: bug description
- Documents: source code files
- Find the document with the highest similarity score to our search query
- Documents with the highest similarity are more likely to contain faults.

Vector representations



Example of vector representation

Suppose that:

- Bug report/query = “a problem with the classNotFound exception.”
- Source file/document = “ get classNotFound exception return exception”

	a	problem	with	the	classNotFound	exception	get	return
A	1	1	1	1	1	1	0	0
B	0	0	0	0	1	2	1	1

Vector representation:

- A = [1, 1, 1, 1, 1, 1, 0, 0]
- B = [0, 0, 0, 0, 1, 2, 1, 1]

Step 3 - Calculate the similarity metrics

Step 3 - Calculate the similarity metrics

- The similarity metric (e.g., cosine similarity) is based on the angle between the two vectors:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \cdot \sqrt{\sum_{i=1}^n B_i^2}}$$

Limitations of IR-based FL

- Assume high quality of bug reports
- In reality, there is always back-and-forth communication between the developer and the users.
- Only leverages the “visible” information in bug reports
- Useful debugging hints are often attached as error logs, screenshot, or even test cases as part of the bug report.

Deliverable

Due next Wednesday midnight, the grading scheme (total of 5 points):

- A class diagram (1 point)
- A section describing tools and technologies (1 point)
- Two user stories (2 points)
- A timeline showing your planning of the sub-tasks (1 point)

Cosine similarity

Cosine similarity is calculated by the equation:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \cdot \sqrt{\sum_{i=1}^n B_i^2}}$$

Suppose that our goal is to calculate the similarity of a bug report below:

- Bug report/query = “a problem with the classNotFound exception.”
- Source file/document = “ get classNotFound exception return exception”

Cosine similarity

Step 1: create a vector representation of the query and document.

- Bug report/query = “a problem with the classNotFound exception.”
- Source file/document = “ get classNotFound exception return exception”

	a	problem	with	the	classNotFound	exception	get	return
A	1	1	1	1	1	1	0	0
B	0	0	0	0	1	2	1	1

Vector representation:

- A = [1, 1, 1, 1, 1, 1, 0, 0]
- B = [0, 0, 0, 0, 1, 2, 1, 1]

Cosine similarity

Step 2: calculate the dot product and magnitude of these vectors

Vector representation:

- $A = [1, 1, 1, 1, 1, 1, 0, 0]$
- $B = [0, 0, 0, 0, 1, 2, 1, 1]$

Dot product of the vectors:

$$A * B = 1 \times 0 + 1 \times 0 + 1 \times 0 + 1 \times 0 + 1 \times 1 + 1 \times 2 + 0 \times 1 + 0 \times 1 = 3$$

Magnitude of the vectors:

$$\| A \| = \sqrt{(1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0 + 0)} = \sqrt{6}$$

$$\| B \| = \sqrt{(0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 2^2 + 1^2 + 1^2)} = \sqrt{5}$$

Cosine similarity

Step 3: calculate the cosine similarity

$$\text{similarity}(A, B) = \frac{A * B}{\|A\| \|B\|} = \frac{3}{\sqrt{6} * \sqrt{5}} = 0.5477$$

The bug report and source code file could be said to be 55% similar.

Lecture 7

Information Redundancy

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Code, codespace, codeword, word
- Hamming distance
- Code distance in error detection/correction
- Repetition codes
- Parity codes
- Next class: Hamming codes

Fault localization

Fault localization techniques have been proposed to assist in locating and understanding the root causes of faults.

Why? Fault localization as a debugging technique to maintain the system:

- Pinpoint the location to fix in the code
- Recover fast from bugs, and reduce its impact on the users
- Reduce manual debugging efforts, more time for new feature



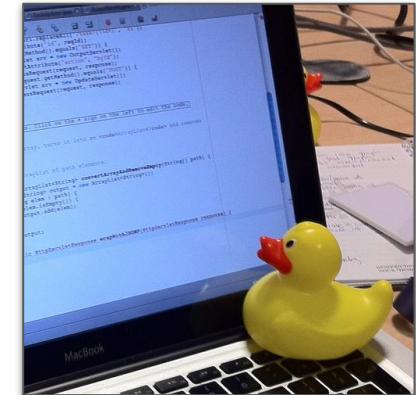
Rubber duck debugging

Introduced by Dave Thomas and Andy Hunt (credited Andrew Hunt) in “The Pragmatic Programmer” book.

From a [lists.ethernal.org post](https://lists.ethernal.org/post/1333) by Andy:

- Beg, borrow, steal, buy, fabricate, or otherwise obtain a rubber duck.
- Place rubber duck on desk and inform it you are just going to go over some code with it, if that's all right.
- Explain to the duck what your code does line by line.
- At some point, you will realize what you are telling to the duck is not in fact what you are actually doing.

If you don't have a rubber duck, a co-worker works too.



Fault localization

Fault localization present automated techniques for locating the faults in the source code.

There are many families of fault localization techniques:

- Spectrum-based techniques
- Information retrieval-based techniques
- Mutation-based techniques (not covered in class)
- Historical-based techniques (not covered in class)
- and more ...

Redundancy and fault tolerance

Main goal of designing reliable and secure systems:

- Maintain availability
 - Uptime
 - The system is up and running
- Maintain reliability
 - Mean Time Between Failure
 - The system continues to function without failure
- No system/software/hardware failure
 - Fault tolerance

Redundancy

To achieve fault tolerance, redundancy is usually used.

Types of redundancy:

- Software redundancy (e.g., N-version programming)
- Information redundancy (e.g., parity bit)
- Hardware redundancy (e.g., replication of the processor)
- Time redundancy (e.g., extra time)

For fault tolerant systems, there is often a combination of these redundancies.

Information redundancy

Information redundancy tolerate faults by adding information to the original data.

- Tolerate faults by means of coding
- Avoid unwanted information changes
- E.g., information loss during data storage or transmission

Used in data applications to ensure data integrity:

- Communication systems
- Storage devices
- Computer networks

The term “coding” is used in the context
of communication and data storage

Information redundancy

There are two common forms of information redundancy:

- Error detection code
- Error correction code
 - to ensure the accuracy, integrity and reliability of transmitted or stored data.

Code selection depends on the nature of the faults:

- Rate
- Consequences
- Overhead in encoding and decoding

Code

- **Code of length n** is a set of n-tuples satisfying some well-defined set of rules.

0000	0
0001	1
0010	2
...	
1001	9

Example: binary-coded decimal (BCD)

- A BCD with code length 4 is a set of 4-tuples for each decimal digit, from 0 to 9.
- Since there are 10 unique decimal digits, the **codespace** only includes the first 10 binary 4-tuples, from 0000 to 1001.
- Code: {0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001}
- 1010 is not part of the code, the decimal digit 10 is represented by 0001 0000.

Code

- A **codeword** is an element of the code satisfying the rules of the code.
- A **word** is an n-tuple not satisfying the rules of the code.
- For example:
 - In binary-coded decimal, 1010 is invalid, thus a word and not a codeword.
 - In binary code, 2024 is invalid, thus a word and not a codeword.
- To make code error detection/correction possible, codewords should be a subset of all possible 2^n binary tuples in the code.
- The number of codewords in a code C is called the **size** of C.

Encoding and decoding

- Encoding: transforms data into code word



- Decoding: transforms code word back to data



- There is a difference in length between the codeword and data.
- This difference gives us the number of check bits which are required to make the encoding (for separable codes).

Encoding and decoding

There are 2 possible scenario when an error happens during encoding/decoding:

1. Correct codeword → another codeword
2. Correct codeword → word

Error

Error happens when the data is corrupted in transmission or storage.

- E.g., bad spots on disks, scratches on DVD or CD, electrical interference

The most common error is a bit flip during a stream of data

- Bit flip: a 1 becomes a 0, or a 0 becomes a 1
- E.g., the sender passes 1010, the receiver gets 1000 instead.

It is also possible for a bit to get deleted or for an extra bit to be inserted.

Error

Common error patterns:

- Single-bit error = one bit has been corrupted
- Burst error = more than one bits have been changed

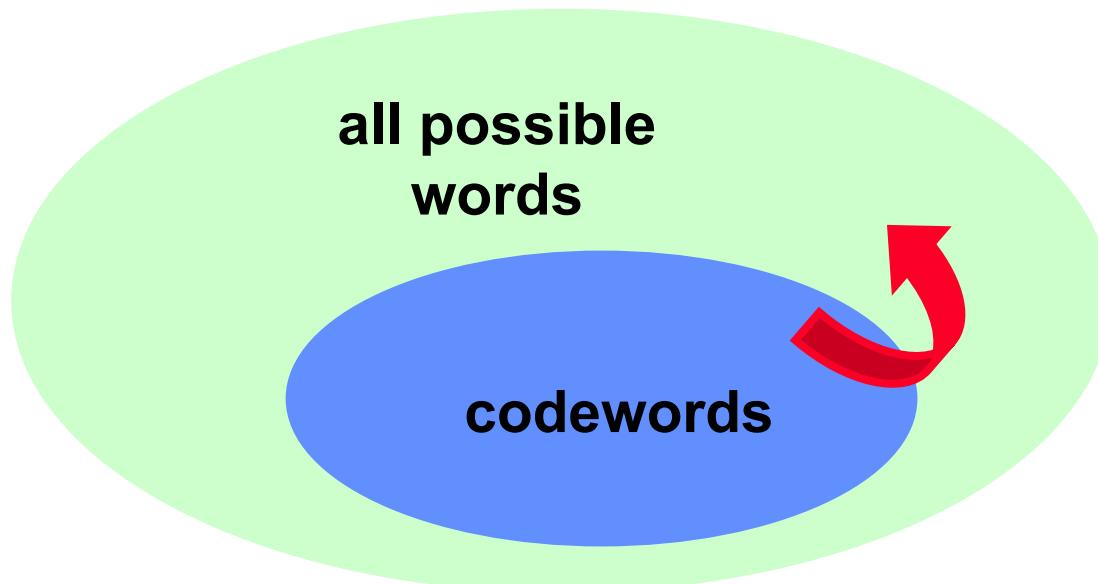
Error detecting/correcting code

Error detecting/correcting code is used to detect and correct corrupted bits during transmission.

- Characterized by the number of bits that can be detected/corrected
 - E.g., double-bit detecting code can detect two single-bit errors
 - E.g., single-bit correcting code can correct one single-bit error

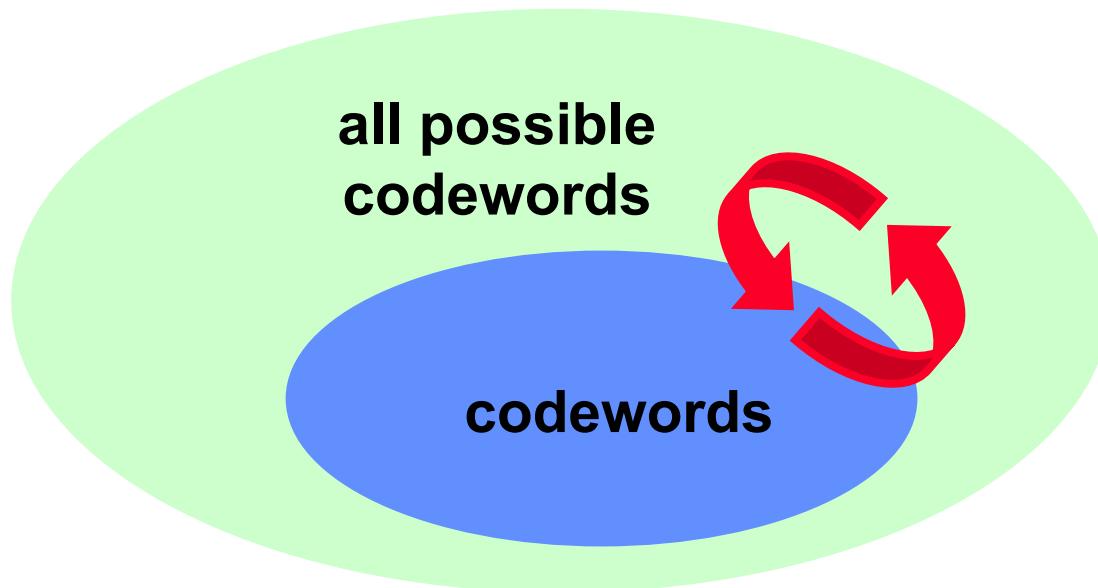
Error detecting code

Basic intuition: we define a code so that errors introduced in a codeword force it to lie outside the range of codewords.



Error correcting code

Basic intuition: we define a code so that it is possible to determine the correct codeword.



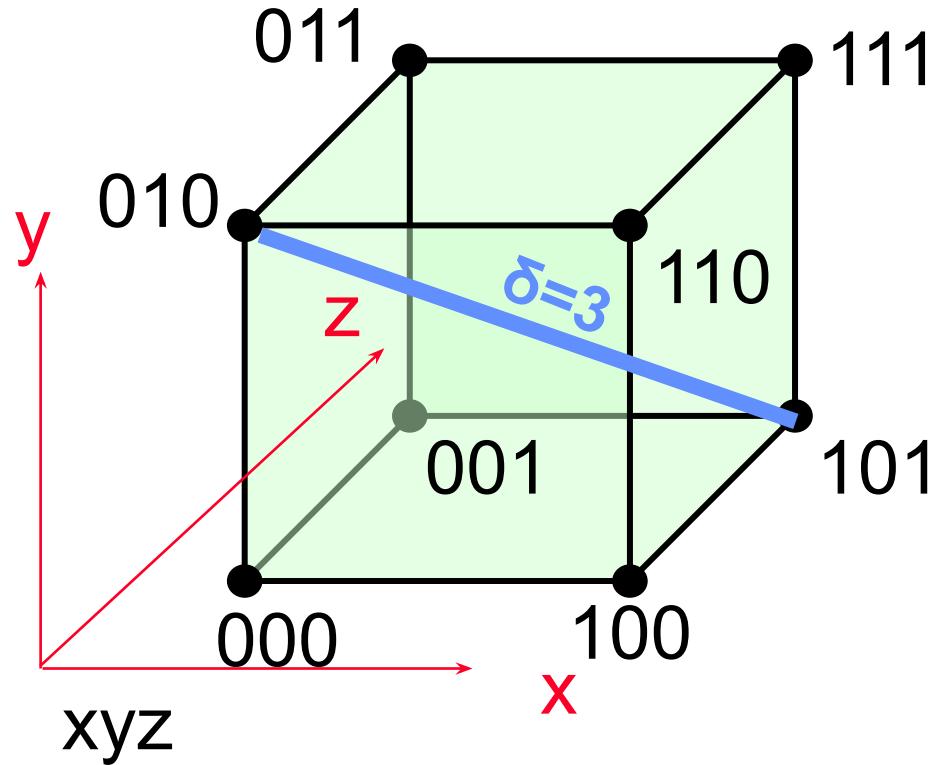
Hamming distance

Hamming distance measures the number of bit positions in which two n-tuples differ.

x	0000
y	0101

$$\delta(x,y) = 2$$

Code in 3-dimensional space



Hamming distance has 3 properties:

- Reflexivity:

$$H_d(x, y) = 0 \text{ if and only if } x = y$$

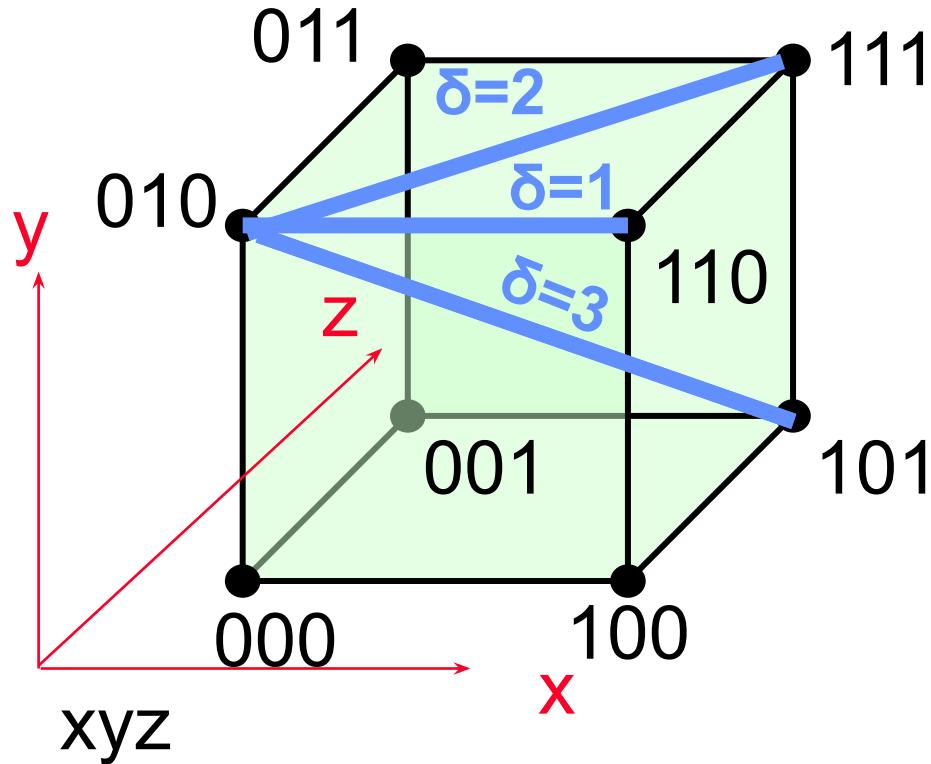
- Symmetry

$$H_d(x, y) = H_d(y, x)$$

- Triangle inequality

$$H_d(x, y) + H_d(y, z) \geq H_d(x, z)$$

Code in 3-dimensional space



Hamming distance has 3 properties:

- **Reflexivity:**

$$H_d(x, y) = 0 \text{ if and only if } x = y$$

$$H_d(010, 010) = 0$$

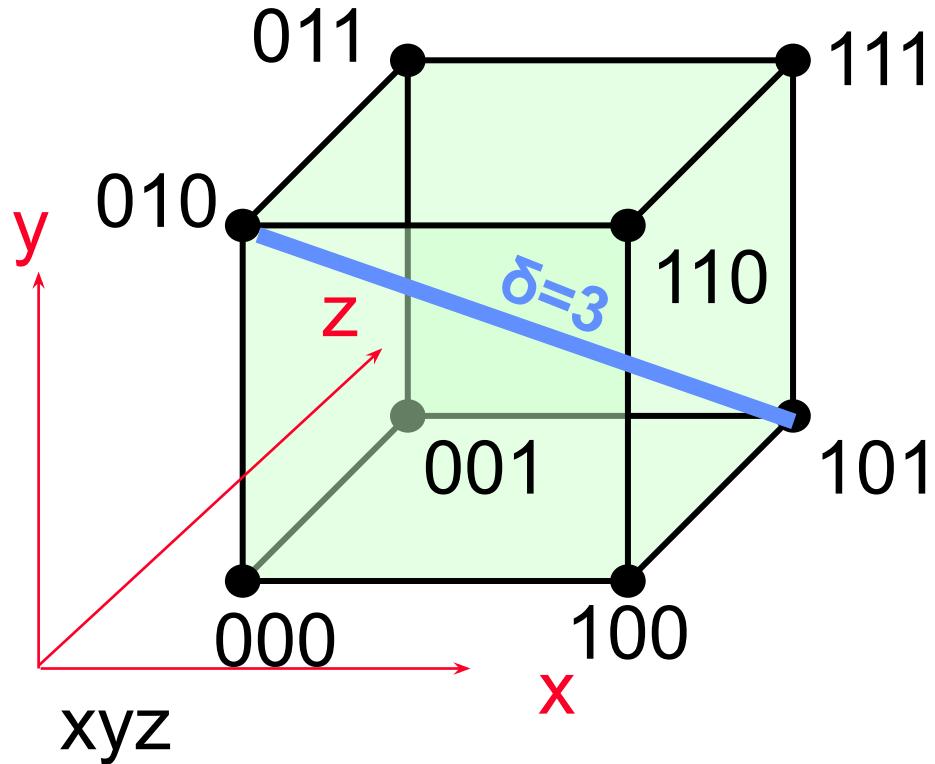
$$H_d(010, 110) = 1$$

$$H_d(010, 111) = 2$$

$$H_d(010, 101) = 3$$

- Symmetry
- Triangle inequality

Code in 3-dimensional space



Hamming distance has 3 properties:

- Reflexivity:

$$H_d(x, y) = 0 \text{ if and only if } x = y$$

- Symmetry

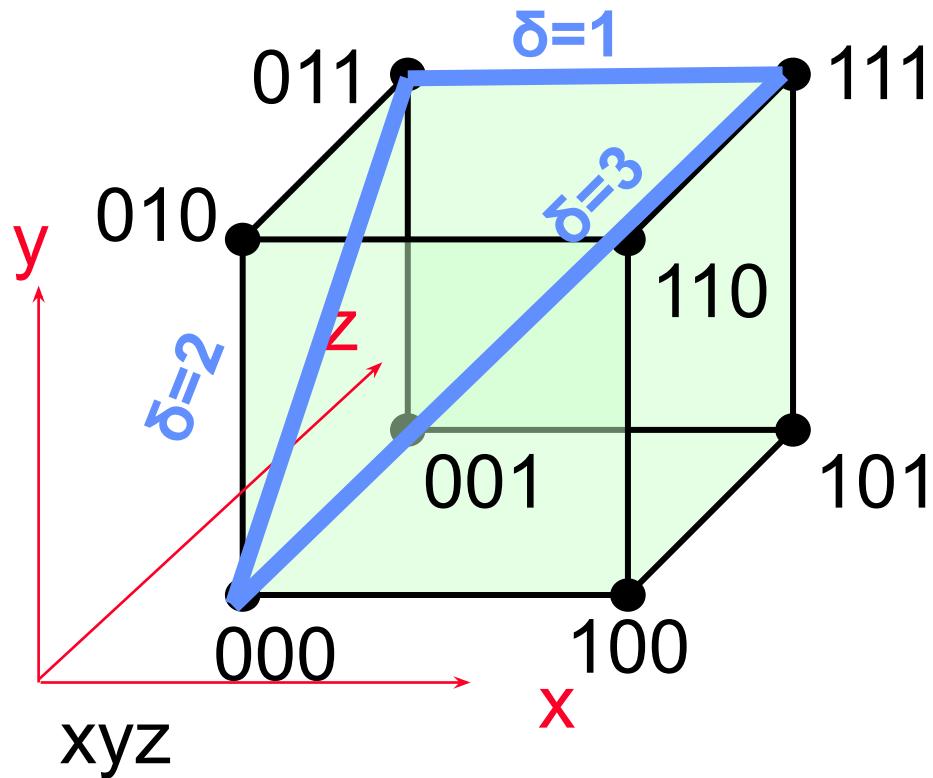
$$H_d(x, y) = H_d(y, x)$$

$$H_d(101, 010) = 3 = H_d(010, 101)$$

- Triangle inequality

$$H_d(x, y) + H_d(y, z) \geq H_d(x, z)$$

Code in 3-dimensional space



Hamming distance has 3 properties:

- Reflexivity:

$$H_d(x, y) = 0 \text{ if and only if } x = y$$

- Symmetry

$$H_d(x, y) = H_d(y, x)$$

- Triangle inequality

$$H_d(x, y) + H_d(y, z) \geq H_d(x, z)$$

$$H_d(000, 011) + H_d(011, 111)$$

$$= 2 + 1 = 3 \geq H_d(000, 111)$$

Code distance

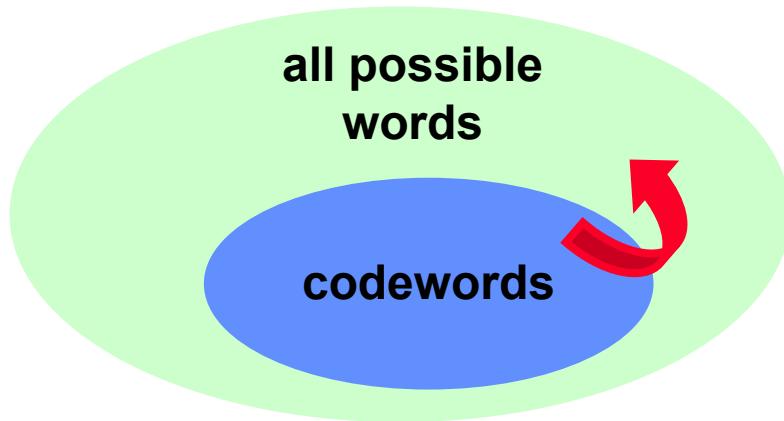
Code distance, denoted as C_d , is the minimum Hamming distance between any two distinct pairs of codewords of the code.

- Determines the error-detecting and error-correcting capabilities of a code.

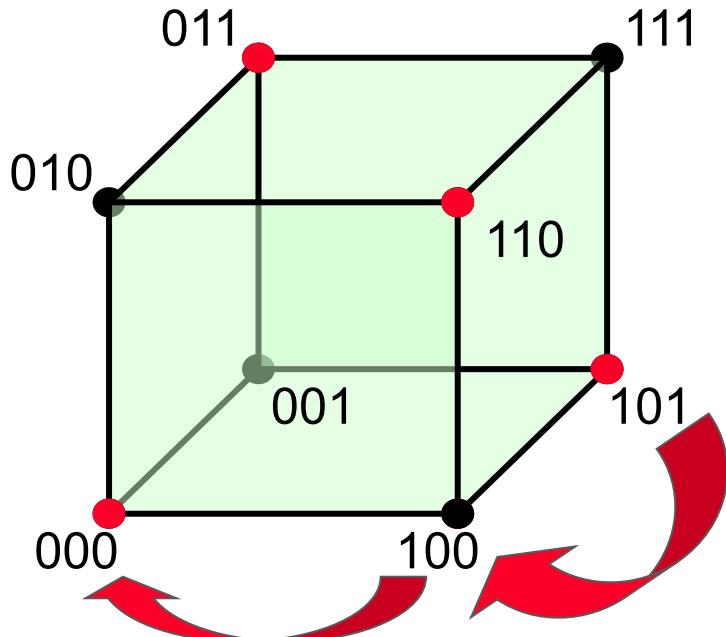
Code distance in detection

To design a code that can detect d bit errors, the code distance for the codewords must be larger or equal to $d+1$.

- This ensures that no errors in d or fewer bits can change one (valid) codeword into another (valid) codeword.



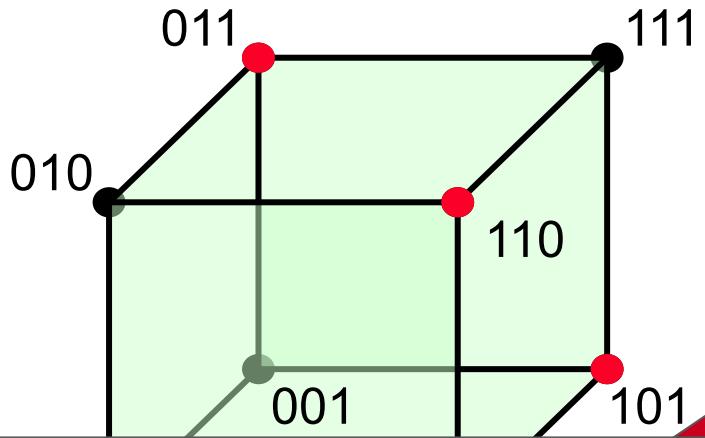
Code distance in detection



Example:

- Consider the code $\{000, 011, 101, 110\}$.
- The code distance is 2.
- Suppose an error has occurred in the first bit of the codeword 000.
- The resulting word 100 has a distance of 1 from the affected codeword 000.
- Since all codeword have a distance of 2 from each other, we detect 100 to be a single-bit error.

Code distance in detection



To detect d bit errors, the code distance for the codewords must be larger or equal to $d+1$.

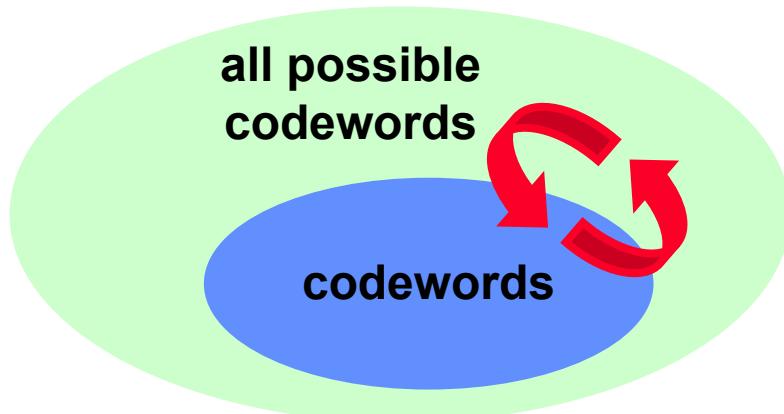
Example:

- Consider the code $\{000, 011, 101, 110\}$.
- The code distance is 2.
- Suppose an error has occurred in the first bit of the codeword 000.
- The resulting word 100 has a distance of 1 from the affected codeword 000.
- Since all codeword have a distance of 2 from each other, we detect 100 to be a single-bit error.

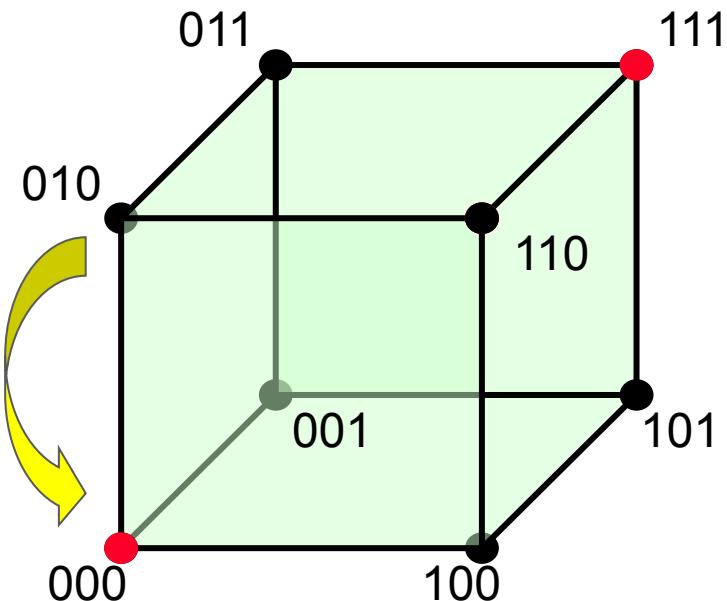
Code distance in correction

To design a code that can correct d bit errors, the code distance for the codewords must be larger or equal to $2d+1$.

- This puts the valid codewords so far apart that even after d bit errors, it is still less than half the distance to another valid codeword.



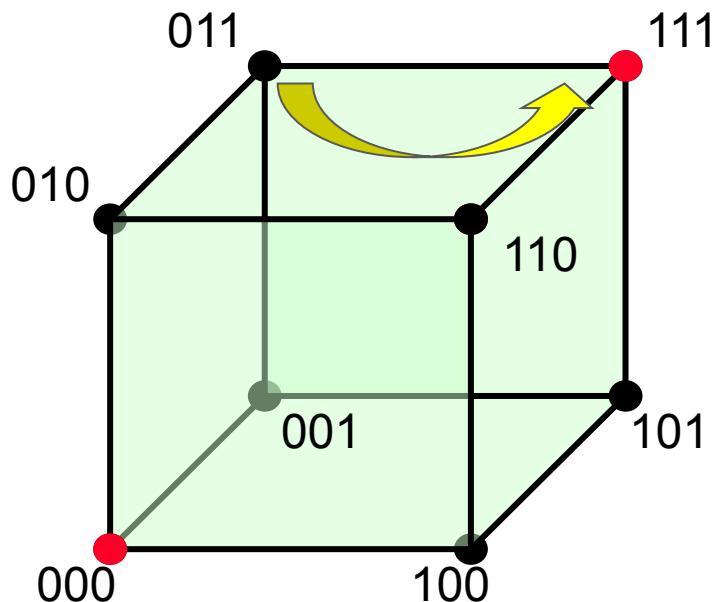
Code distance in correction



Example:

- Consider the code $\{000, 111\}$.
- The code distance is 3.
- Suppose a single-bit error has occurred in the second bit of the codeword 000.
- The resulting word 010 has a distance of 1 from the affected codeword 000.
- We correct 010 to the codeword 000, since it is the closest codeword.

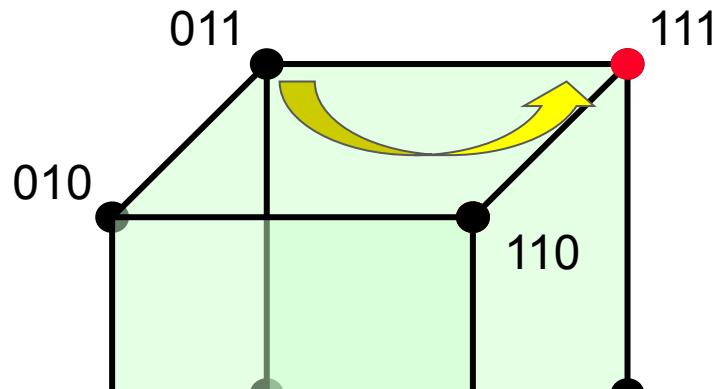
Code distance in correction



Example:

- Consider the code $\{000, 111\}$.
- The code distance is 3.
- Suppose a **double-bit error** has occurred in the second and third bit of the codeword 000.
- The resulting word **011** has a distance of 2 from the affected codeword 000.
- We **cannot correct the error** because the word 011 is closer to the codeword 111.

Code distance in correction



To correct d bit errors, the code distance for the codewords must be larger or equal to $2d+1$.

Example:

- Consider the code $\{000, 111\}$.
- The code distance is 3.
- Suppose a **double-bit error** has occurred in the second and third bit of the codeword 000.
- The resulting word **011** has a distance of 2 from the affected codeword 000.
- We **cannot correct the error** because the word 011 is closer to the codeword 111.

Code distance

Code distance is the minimum Hamming distance between any two distinct pairs of codewords of the code.

- Determines the error-detecting and error-correcting capabilities of a code.

Error detecting code

Suppose $C_d = 2$, code detects all single-bit errors.

Code: {00, 11}

Invalid words: 01 and 10

Error correcting code

Suppose $C_d = 3$, code corrects all single-bit errors.

Code: {000, 111}

Invalid words: 001, 010, 100,

101, 011, 110

Separable/non-separable code

- Separable code
 - Codeword = data + check bits
 - E.g., Parity: $11011 = 1101 + 1$
- Non-separable code
 - Codeword = data mixed with check bits
 - E.g., Cyclic: $1010001 \rightarrow 1101$
- Decoding process is much easier for separable codes (remove check bits)

Information rate

- The information rate of the code is the ratio k/n , where
 - k is the number of data bits
 - n is the number of data + check bits
- Example: a code obtained by repeating data three times has the information rate $\frac{1}{3}$.
 - In other words, 3 bits of message for each bit of data
 - Bigger code rates, stronger codes

Types of codes

- Repetition codes
- Parity codes
- Hamming codes
- Reed-Solomon codes
- Berger codes
- ... and more

Repetition codes

Repetition codes is used to detect errors by repeating the data several times.

- Raises the probability of the correct bit being uncorrupted

For example:

- Code {1 1 0} can be sent as {1 1 1 1 0 0}
- If the bits are not matching in each group of two, then an error is detected

However, two repetitions does not allow us to determine what the original values was.

Repetition codes

We can repeat each bit three times:

- Code {1 1 0} can be sent as {1 1 1 1 1 1 0 0 0}
- If the bits are not matching in each group of three, then an error is detected
- We can assume the majority results for each group of three is likely to be the original code
- However, this solution comes with high overhead
 - E.g., if we transfer 8 bits, it will require 24 bits, 200% increase in data size

Parity codes

Parity codes detect errors using parity bits, decreasing the number of bits sent.

Parity/check bits are used to determine if the total 1s in the sent code is even or odd.

- Single-bit parity code: can determine an odd number of errors.
- Multiple parity bits code: can determine errors regardless of its number, and locate where they occur.

Single-bit parity codes

Single-bit parity codes add an extra bit (a parity bit) to data so that resulting codeword has either even or odd number of 1s.

- Even parity: even number of 1s
- Odd parity: odd number of 1s

If a single bit is flipped, the received data will have the wrong parity, then we know there is an error in the transmission or storage.

The parity bit can either be added to the front or the back, as long as the order is consistent.

Single-bit parity codes

Example 1: odd parity code

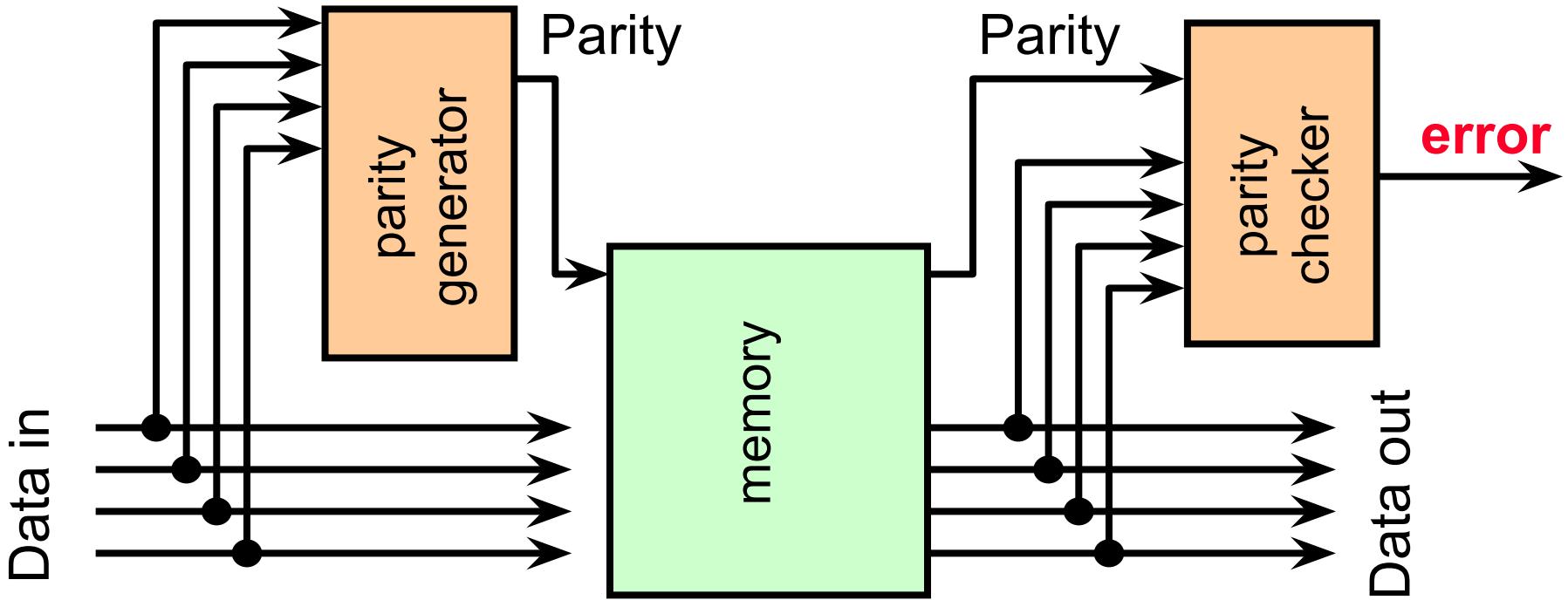
- Suppose the data “0011” is encoded using an odd parity code
- Because the data contain an even number of 1s
- We add a parity bit of “1” to make the number of 1s odd
- The resulting codeword is “0011 1”

Single-bit parity codes

Example 2: even parity code

- Suppose the data “0011” is encoded using an even parity code
- Because the data already contain an even number of 1s
- The resulting codeword is “0011 0”

Memory with built-in parity bits



Challenges with single-bit parity codes

Challenge 1: Multiple-bit errors which affect even number of bits cannot be detected

Scenario 1: with odd number of bits affected

- Consider the codeword “00111” with **of a 5-bit odd parity code**
- If an error affects the **first** bit, we receive the word “10111”
- “10111” has an even number of 1s, we detect the error.

Challenges with single-bit parity codes

Challenge 1: Multiple-bit errors which affect even number of bits cannot be detected

Scenario 2: with even number of bits affected

- Consider the codeword “00111” with **of a 5-bit odd parity code**
- If an error affects the **first two** bits, we receive the word “11111”
- “11111” has an odd number of 1s, the error is **not detected**.

Challenges with single-bit parity codes

Challenge 2: Limited error detection

- Can sometimes result in false positive (e.g., two bit flips)

Challenge 3: Unclear detection

- Can not tell which bit is corrupted

Schedule for today

- Key concepts from last class
- Code, codespace, codeword, word
- Hamming distance
- Code distance in error detection/correction
- Repetition codes
- Parity codes
- Next class: Hamming codes

Lecture 8

Information Redundancy - Part II

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
 - An analogy on error detection and correction codes
- Hamming codes
 - Basic intuition
 - Error detection
 - Error correction
 - Information rate
- Extended Hamming codes
- TODOs

Encoding and decoding

- Encoding: transforms data into code word



- Decoding: transforms code word back to data



- There is a difference in length between the codeword and data.
- This difference gives us the number of check bits which are required to make the encoding (for separable codes).

Code distance in error detection

To detect d bit errors, the code distance for the codewords must be larger or equal to $d+1$.

Example

Code: {000, 001}

$$C_d = 1$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

Analogy

Dictionary: {accept, accent}

$$\text{Distance} = 1$$

Typed word: accept

A typo happens, accept becomes accent

Code distance in error detection

To detect d bit errors, the code distance for the codewords must be larger or equal to $d+1$.

Example

Code: {000, 001}

$$C_d = 1$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

Analogy

Dictionary: {accept, accent}

$$\text{Distance} = 1$$

Typed word: accept

A typo happens, accept becomes accent

Code distance in error detection

To detect d bit errors, the code distance for the codewords must be larger or equal to $d+1$.

Example

Code: {000, 001}

$$C_d = 1$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We cannot tell there is an error in 001.

Analogy

Dictionary: {accept, accent}

$$\text{Distance} = 1$$

Typed word: accept

A typo happens, accept becomes accent

We cannot tell it is a typo.

Code distance in error detection

To detect d bit errors, the code distance for the codewords must be larger or equal to $d+1$.

Example

Code: {000, 101}

$$C_d = 2$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We can tell that 001 is an error (not a codeword in the code space)

Analogy

Dictionary: {accept, except}

Distance = 2

Typed word: accept

A typo happens, accept becomes except

We can tell it is a typo.

Code distance in error correction

To correct d bit errors, the code distance for the codewords must be larger or equal to $2d+1$.

Example

Code: {000, 101}

$$C_d = 2$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We cannot tell how to correct 001 (000 or 101).

Analogy

Dictionary: {accept, except}

Distance = 2

Typed word: accept

A typo happens, accept becomes except

We cannot tell which word is mistyped.

Code distance in error correction

To correct d bit errors, the code distance for the codewords must be larger or equal to $2d+1$.

Example

Code: {000, 101}

$$C_d = 2$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We cannot tell how to correct 001 (000 or 101).

Analogy

Dictionary: {accept, except}

Distance = 2

Typed word: accept

A typo happens, accept becomes except

We cannot tell which word is mistyped.

Code distance in error correction

To correct d bit errors, the code distance for the codewords must be larger or equal to $2d+1$.

Example

Code: {000, 111}

$$C_d = 3$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We can correct 001 to 000 (closest).

Analogy

Dictionary: {except, exception}

Distance = 3

Spelling word: except

A typo happens, except becomes eccept

We can correct the typo.

Schedule for today

- Key concepts from last class
 - An analogy on error detection and correction codes
- Hamming codes
 - Basic intuition
 - Error detection
 - Error correction
 - Information rate
- Extended hamming codes

Hamming codes

Hamming code is a linear code that can detect and correct single-bit errors.

- Why? To detect and correct errors while sending less redundant data (than repetition codes).

Basic intuition: if we apply multiple parity checks to some carefully selected subsets of the data, we can pin down the location of any single-bit error.

A **linear code** is an error-correcting code for which any linear combination of codewords is also a codeword.

Logic behind Hamming codes

The overall idea is have the parity checks as a series of questions that help us detect and locate the single-bit error.

- Each question narrows down the search space
- Analogous to Sudoku
 - Rule for each row: comprise the numbers 1-9
 - Rule for each column: comprise the numbers 1-9
 - ...
- Hamming codes
 - Parity check for some specific groups of bits: odd parity
 - ...

(15, 11) Hamming code

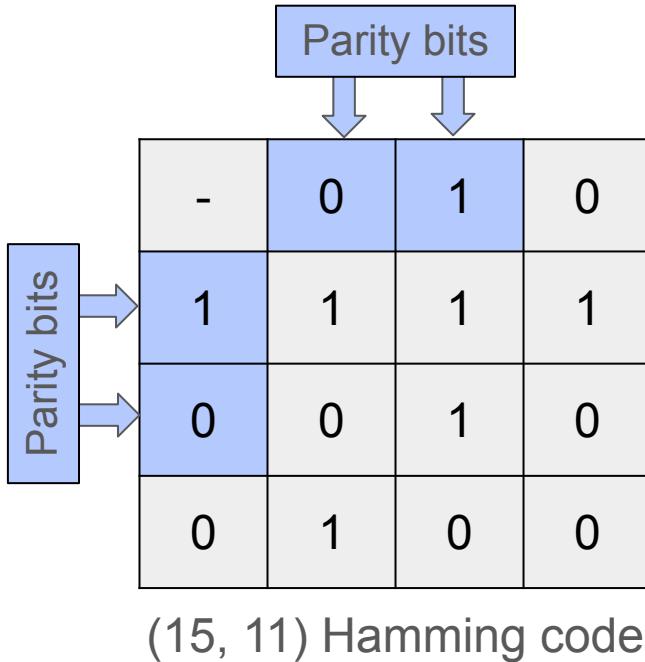
Not used			
-	0	1	0
1	1	1	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Definition of (15, 11)

- 15 bits to encode 11 bits of data
- The remaining 4 bits are parity bits
- The first position is not used

(15, 11) Hamming code



Definition of (15, 11)

- 15 bits to encode 11 bits of data
- The remaining 4 bits are parity bits
- The first position is not used

Position of the parity bits

- Set at position $\{2^n\}$, starting from $n = 0$
- Example
 - Parity bit #1: position 1
 - Parity bit #2: position 2
 - Parity bit #3: position 4
 - Parity bit #4: position 8

0	1	2	3
4	5	6	7
8	9	10	11
11	12	13	14

Positions of the bits

Error detection



-			0
	1	1	1
	0	1	0
0	1	0	0

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Step 1: calculate parity bit #1 at position 1

- We check the parity of the bits in the second and last columns.
- The bits {0 1 1 0 0 1 0} contain three “1”s.
- That is an odd number of “1”s.
- Therefore, we set the **first parity bit to 1** to make up for an even parity.

Error detection

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

-	1		0
	1	1	1
	0	1	0
0	1	0	0

Step 2: calculate parity bit #2 at position 2

- We check the parity of the bits in the third and last columns.
- The bits {0 1 1 1 0 0 0} contain three “1”s.
- That is an odd number of “1”s.
- Therefore, we set the **second parity bit** to **1** to make up for an even parity.

Error detection

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

-	1	1	0
	1	1	1
	0	1	0
0	1	0	0

Step 3: calculate parity bit #3 at position 4

- We check the parity of the bits in the second and last rows.
- The bits {1 1 1 0 1 0 0} contain four “1”s.
- That is an even number of “1”s.
- Therefore, we set the **third parity bit** to 0 as it is already an even parity

Error detection

Example: using even parity

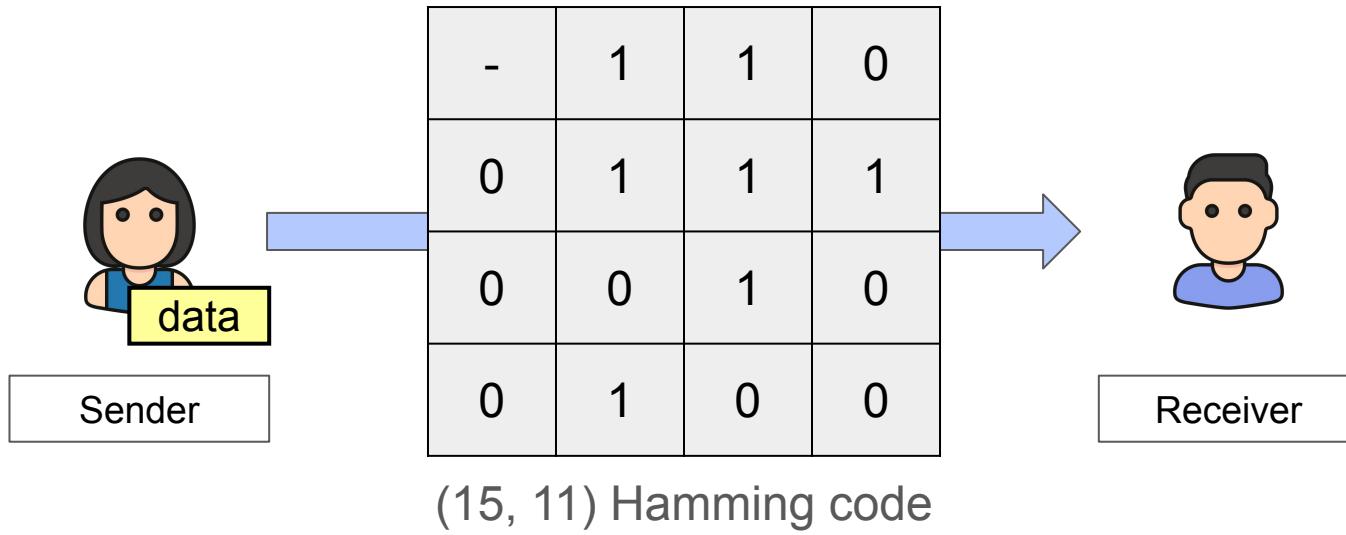
Data: {0 1 1 1 0 1 0 0 1 0 0}

-	1	1	0
0	1	1	1
	0	1	0
0	1	0	0

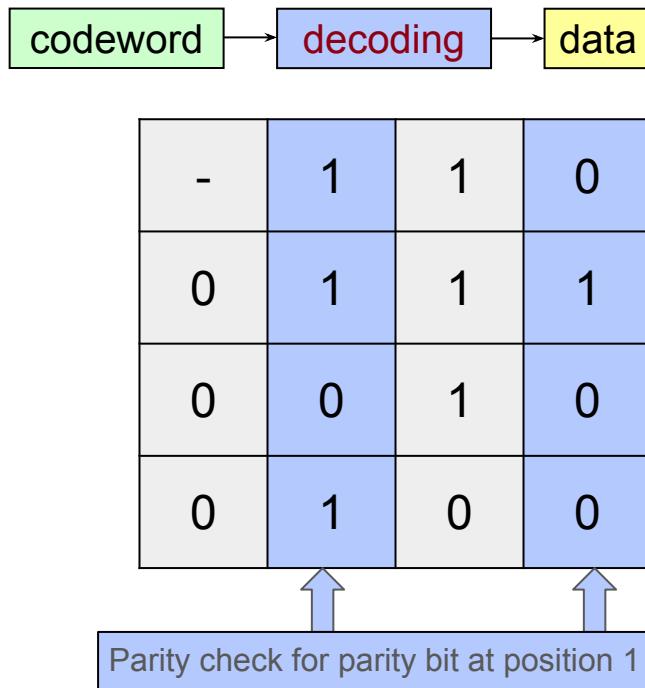
Step 3: calculate parity bit #4 at position 8

- We check the parity of the bits in the third and last rows.
- The bits {0 1 0 0 1 0 0} contain two “1”s.
- That is an even number of “1”s.
- Therefore, we set the **last parity bit to 0** as it is already an even parity

Error detection



Error detection



Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: No error

The receiver repeat the same process.

Parity bit at position 1: {1 0 1 1 0 0 1 0}, even parity, no error

Parity bit at position 2: {1 0 1 1 1 0 0 0}, even parity, no error

Parity bit at position 4: {0 1 1 1 0 1 0 0}, even parity, no error

Parity bit at position 8: {0 0 1 0 0 1 0 0}, even parity, no error

Error detection

Bit flip

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: Error at position 6

The receiver repeats the same process.

Error detection

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: Error at position 6

The receiver repeats the same process.

Parity bit at position 1: {1 0 1 1 0 0 1 0}, even parity, no error

Error detection

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: Error at position 6

The receiver repeats the same process.

Parity bit at position 1: {1 0 1 1 0 0 1 0}, even parity, no error

Parity bit at position 2: {1 0 0 1 1 0 0 0}, odd parity, **error**

Error detection

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: Error at position 6

The receiver repeats the same process.

Parity bit at position 1: {1 0 1 1 0 0 1 0}, even parity, no error

Parity bit at position 2: {1 0 0 1 1 0 0 0}, odd parity, **error**

Parity bit at position 3: {0 1 0 1 0 1 0 0}, odd parity, **error**

Error detection

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code

Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: Error at position 6

The receiver repeats the same process.

Parity bit at position 1: {1 0 1 1 0 0 1 0}, even parity, no error

Parity bit at position 2: {1 0 0 1 1 0 0 0}, odd parity, **error**

Parity bit at position 3: {0 1 0 1 0 1 0 0}, odd parity, **error**

Parity bit at position 8: {0 0 1 0 0 1 0 0}, even parity, no error

Error detection and correction

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error



-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Detection and correction by elimination

- First parity check: No error in the second and last columns.
- Second parity check: Error is somewhere in the last two columns.
- Intuition: the error must be in the third column.

Error detection and correction

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Detection and correction by elimination

- Third parity check: Error is somewhere in the second and last rows.
- Fourth parity check: No error in the third and last row.
- Intuition: Error must be at position 6.

Parity checks

The number of parity checks is based on the number of data bits.

- Every parity check reduces the possibility of an error occurring at a certain location by half.
- k bits = 2^n bits, requires n parity checks
- Examples:
 - E.g., 4 bits = 2^2 bits, 2 parity checks
 - E.g., 16 bits = 2^4 bits, 4 parity checks
 - E.g., 256 bits = 2^8 bits, 8 parity checks

Information rate

The information rate improves as we deal with larger block size.

- Bigger information rates produce stronger codes.
- Examples:
 - E.g., 16 bits, 11 data bits, 4 parity bits, information rate = $11/15 \approx 73\%$
 - E.g., 128 bits, 121 data bits, 6 parity bits, information rate = $121/128 \approx 95\%$
 - E.g., 256 bits, 247 data bits, 8 parity bits, information rate = $247/255 \approx 97\%$
 - E.g., 516 bits, 505 data bits, 10 parity bits, information rate = $505/515 \approx 98\%$
 - ...

The information is the ratio k/n , where

- k is the number of data bits
- n is the number of data bits + parity bits

Information rate

Data bits	Parity checks	Hamming codes
16	4	73%
128	6	95%
256	8	97%
516	10	98%
...
1,048,576	20	Nearly 100%

One million bits of data only requires 20 parity checks to locate the error.

Error detection and correction

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error (0)

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Parity bit at
position 1

Binary

0

Error detection and correction

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error (0)

Parity bit at
position 1

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error (1)

Parity bit at
position 2

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Binary

1	0
---	---

Error detection and correction

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error (0)

Parity bit at
position 1

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error (1)

Parity bit at
position 2

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error (1)

Parity bit at
position 4

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error (0)

Parity bit at
position 8

Binary

0	1	1	0
---	---	---	---



6

Decimal

Decimal value

8	4	2	1
---	---	---	---

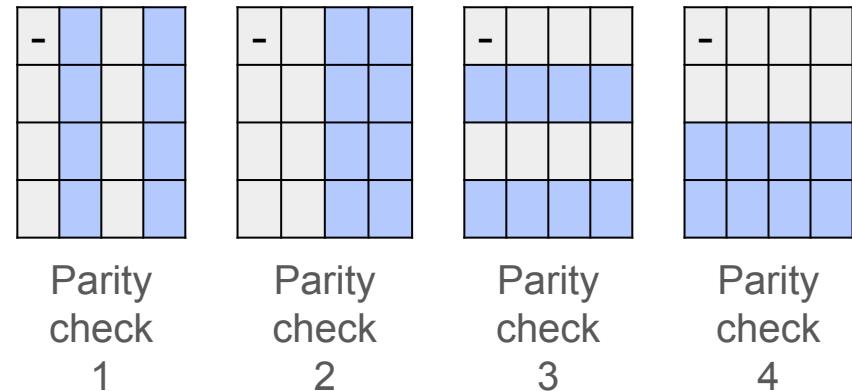
What happens if there is an error on a parity bit?



Bit flip

-	1	0	0
0	1	1	1
0	0	1	0
0	1	0	0

(15, 11) Hamming code
using even parity



Schedule for today

- Key concepts from last class
 - An analogy on error detection and correction codes
- Hamming codes
 - Basic intuition
 - Error detection
 - Error correction
 - Information rate
- Extended Hamming codes
- TODOs

Extended Hamming codes

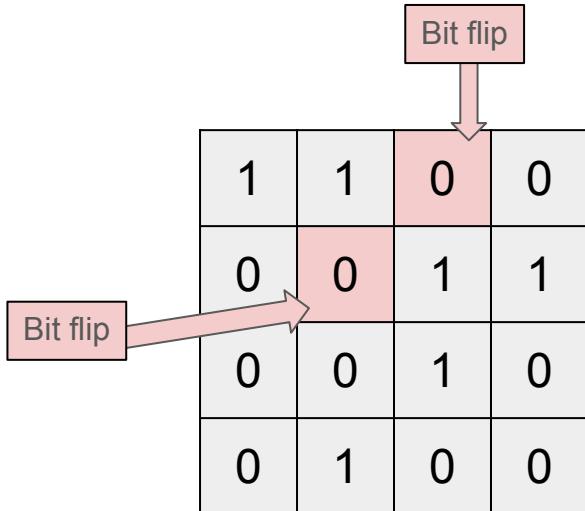
Extended Hamming code is a linear code that can detect and correct single-bit errors, and also detect double-bit errors.

- Uses an extra parity check for the whole block of bits
- For example, parity check on {1100 111 0010 0100}

1	1	1	0
0	1	1	1
0	0	1	0
0	1	0	0

Extended Hamming code (even parity)

Extended Hamming codes



Extended Hamming code
(even parity)

- There are six 1s in the whole block.
- That is an even number of 1s, the parity check at position 0 passes.
- However, the other parity checks (at position 1, 2 and 4) detect an error.
- Therefore, there are at least two errors.

TODOs

- For the deliverable, please don't forget to add your team and team member names (a title page).

Lecture 9

Information Redundancy - Part III

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Cyclic codes
 - Two properties: linear and cyclic
 - Generator polynomial
 - Encoding
 - Polynomial multiplication
- Next class: decoding and error detection in cyclic codes

Code distance in error detection

To detect d bit errors, the code distance for the codewords must be larger or equal to $d+1$.

Example

Code: {000, 001}

$$C_d = 1$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We cannot tell there is an error in 001.

Analogy

Dictionary: {accept, accent}

$$\text{Distance} = 1$$

Typed word: accept

A typo happens, accept becomes accent

We cannot tell it is a typo.

Code distance in error correction

To correct d bit errors, the code distance for the codewords must be larger or equal to $2d+1$.

Example

Code: {000, 111}

$$C_d = 3$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We can correct 001 to 000 (closest).

Analogy

Dictionary: {except, exception}

Distance = 3

Spelling word: except

A typo happens, except becomes eccept

We can correct the typo.

Error detection and correction

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error (0)

Parity bit at
position 1

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error (1)

Parity bit at
position 2

-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

Error (1)

Parity bit at
position 4

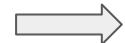
-	1	1	0
0	1	0	1
0	0	1	0
0	1	0	0

No error (0)

Parity bit at
position 8

Binary

0	1	1	0
---	---	---	---



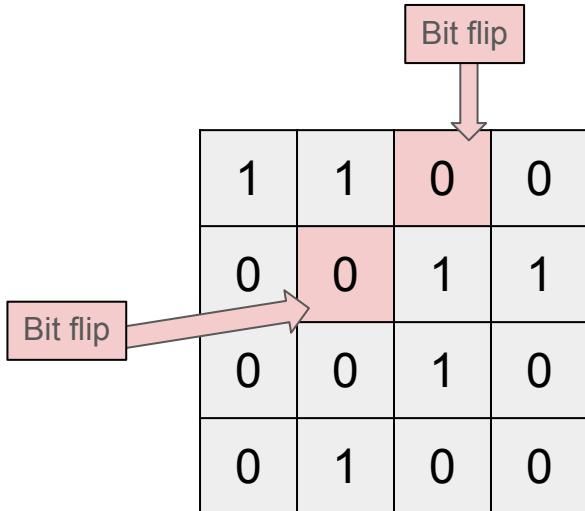
6

Decimal

Decimal value

8	4	2	1
---	---	---	---

Extended Hamming codes



Extended Hamming code
(even parity)

- There are six 1s in the whole block.
- That is an even number of 1s, the parity check at position 0 passes.
- However, the other parity checks (at position 1, 2 and 4) detect an error.
- Therefore, there are at least two errors.

Cyclic codes

Cyclic code is a special class of codes used in systems where burst errors can happen.

- Burst errors can happen in digital communication and storage devices (e.g., Disks, CDs)

Examples of cyclic codes:

- Cyclic Redundancy Check (CRC)
- Reed-Solomon codes (RS codes)

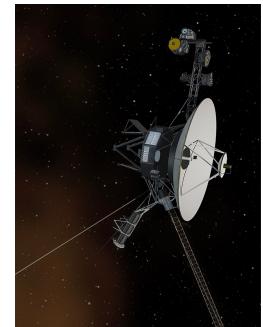
Burst error = more than one bits have been changed

Application of Reed-Solomon codes

RS codes have been widely applied in modern systems thanks to its efficiency in error correction.

Examples of modern systems:

- Data storage
 - E.g., DVD and CD
- Satellite communication
 - E.g., Voyager II
- Hi-speed modems



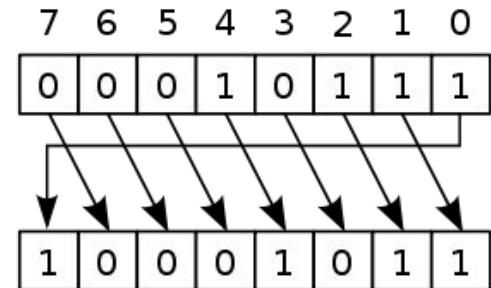
Cyclic codes

Cyclic: any circular shift of a codeword produces another codeword

- Move the rightmost bit to the leftmost position
- Shift all other bits by one position to the right

For example:

- Consider the codeword 10110
- Circular shift by one position to the right: 01011
- Circular shift by two positions to the right: 10101
- ...



Cyclic codes are not necessary linear

Cyclic codes are not necessary linear.

- a linear code is code for which any linear combination of codewords is also a codeword.
- any linear combination of codewords is also a codeword

For cyclic codes, the addition of two codewords does not necessarily lead to another codeword.

A **linear code** is an error-correcting code for which any linear combination of codewords is also a codeword.

Example of linear/cyclic codes

Linear code

- Suppose the code {0000, 0100, 0011, 1100, 0111, 1000, 1011, 1111}
- The sum of any codewords must produce another codeword

$$\begin{array}{r} 0100 \\ + 0011 \\ \hline 0111 \end{array}$$

Cyclic code

- Suppose the code {10110, 01011, 10101, 11010, 01101}
- Their sum does not produce a codeword

$$\begin{array}{r} 10110 \\ + 01011 \\ \hline 11101 \end{array}$$

Addition for codewords

The notion of “addition” here is different from the ordinary addition of numbers.

- Done in a mod 2 arithmetic system
- The terms “addition” and “xor” are used interchangeably

It differs in two respects:

- No carry over, each bit is independent of each other bit
- 2 is the same as 0, so 1 and 1 is none.
 - E.g., $1 + 0 \equiv 1 \pmod{2}$
 - E.g., $1 + 1 \equiv 0 \pmod{2}$

+	0	1
0	0	1
1	1	0

Example of addition of codewords

For example:

- Codeword 1: 10111
- Codeword 2: 00011
- Calculate the sum of these two codewords:

$$\begin{array}{r} 10111 \\ + 00011 \\ \hline 10100 \end{array}$$

or

$$\begin{array}{r} 10111 \\ \oplus 00011 \\ \hline 10100 \end{array}$$

Cyclic codes are not necessary linear

Linear code

- Suppose the code {0000, 0100, 0011, 1100, 0111, 1000, 1011, 1111}
- The sum of any codewords must produce another codeword

A **linear code** is an error-correcting code for which any linear combination of codewords is also a codeword.

$$\begin{array}{r} 0100 \\ + 0011 \\ \hline 0111 \end{array}$$

Cyclic code

- Suppose the code {10110, 01011, 10101, 11010, 01101}
- Their sum does not produce a codeword

For **cyclic codes**, the addition of two codewords does not necessarily lead to another codeword.

$$\begin{array}{r} 10110 \\ + 01011 \\ \hline 11101 \end{array}$$

Properties of linear cyclic codes

In practice, cyclic codes designed for error detection and correction should have two main properties:

Property 1: Linear

- The sum of any two or more codewords in C is again a codeword in C .

Property 2: Cyclic

- For a codeword in C , all its cyclic shifts are also codewords.

Example of linear cyclic codes



Code $\{000000, 100100, 110110, 010010, 011011, 001001, 101101, 111111\}$ is both linear and cyclic.

Question: Prove that the above code contains both cyclic and linear properties.

Property 1: Linear

- The sum of any two or more codewords in C is again a codeword in C .

Property 2: Cyclic

- For a codeword in C , all its cyclic shifts are also codewords.

Example of linear cyclic codes



Question: Is the code $\{000, 100, 010, 001\}$ a linear cyclic code?

Polynomials

Cyclic codes represent codewords as polynomials.

- E.g., a codeword $[a_0 a_1 \dots a_{n-1}]$ is represented as a polynomial

$$a(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + \dots + a_{n-1} \cdot x^{n-1}$$

Polynomials

Cyclic codes represent codewords as polynomials.

- E.g., a codeword $[a_0 a_1 \dots a_{n-1}]$ is represented as a polynomial

$$a(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + \dots + a_{n-1} \cdot x^{n-1}$$

Note that:

- Since the code is binary, the coefficients are 0 and 1
- For example, $d(x) = 1 \cdot x^0 + 0 \cdot x^1 + 1 \cdot x^2 + 1 \cdot x^3$ represents the data (1011)
- Polynomial: $x^3 + x^2 + 1$

$$\begin{array}{cccc} 1 & 0 & 1 & 1 \\ \swarrow & \downarrow & \searrow & \downarrow \\ 1x^0 + 0x^1 + 1x^2 + 1x^3 \end{array}$$

Polynomials

The degree of a polynomial equals to its highest exponent:

- E.g., the degree of $1 + x^1 + x^3$ is 3

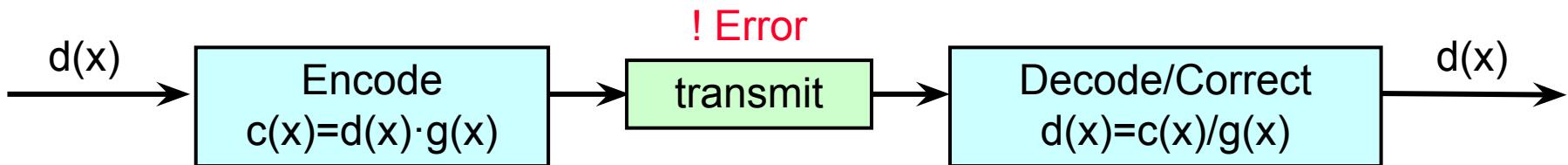
A cyclic code with the **generator polynomial** of degree $(n-k)$ detects all burst errors affecting $(n-k)$ bits or less.

- n is the number of bits in codeword
- k is the number of bits in data

Generator polynomial

Generator polynomial, denoted as $g(x)$, is used to:

- encode the **data polynomial** into **codeword polynomial**.
- decode the **codeword polynomial** back to the **data polynomial**.



Encoding

Multiply data polynomial by generator polynomial:
 $c(x) = d(x).g(x)$

- $g(x)$ is the generator polynomial for a linear cyclic code of length n if and only if $g(x)$ divides $1 + x^n$ without a remainder.

$$1 + x^n \bmod g(x) = 0$$

- Multiplication in modulo 2 arithmetic = AND operation

Polynomial multiplication

To multiply two polynomials:

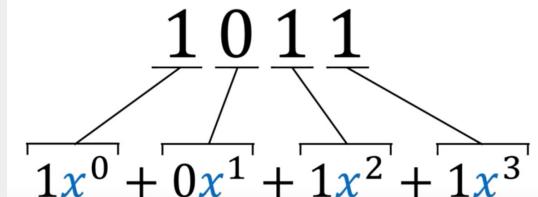
- multiply each term in one polynomial by each term in the other polynomial
- add those answers together, and simplify if needed

Example of polynomial multiplication

$$d(x) = (1011) = x^3 + x^2 + 1$$

$$g(x) = x^3 + x + 1$$

Data bits $k = 4$



$$c(x) = d(x).g(x)$$

$$= (x^3 + x^2 + 1).(x^3 + x + 1)$$

$$= x^6 + x^4 + x^3 + x^5 + x^3 + x^2 + x^3 + x + 1$$

$$= x^6 + x^5 + x^4 + x^3 + x^2 + x^1 + 1$$

Length $n = 7$

Cyclic property on multiplication

Suppose a codeword $\{b_0 b_1 b_2 b_3\}$, and $g(x) = (x^4 - 1) \equiv 0$, or $x^4 \equiv 1$

$$c(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3$$

Let's multiply the codeword by x :

$$x.c(x) = x(b_0 + b_1 x + b_2 x^2 + b_3 x^3)$$

$$x.c(x) = b_0 x + b_1 x^2 + b_2 x^3 + b_3 x^4$$

Cyclic property on multiplication

Suppose a codeword $\{b_0 b_1 b_2 b_3\}$, and $g(x) = (x^4 - 1) \equiv 0$, or $x^4 \equiv 1$

$$c(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3$$

Let's multiply the codeword by x :

$$x.c(x) = x(b_0 + b_1 x + b_2 x^2 + b_3 x^3)$$

$$x.c(x) = b_0 x + b_1 x^2 + b_2 x^3 + b_3 x^4$$

$$x.c(x) = b_3 + b_0 x + b_1 x^2 + b_2 x^3$$

Cyclic property on multiplication

Suppose a codeword $\{b_0 b_1 b_2 b_3\}$, and $g(x) = (x^4 - 1) \equiv 0$, or $x^4 \equiv 1$

$$c(x) = b_0 + b_1 x + b_2 x^2 + b_3 x^3$$

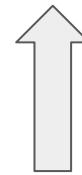
Let's multiply the codeword by x :

$$x.c(x) = x(b_0 + b_1 x + b_2 x^2 + b_3 x^3)$$

$$x.c(x) = b_0 x + b_1 x^2 + b_2 x^3 + b_3 x^4$$

$$x.c(x) = b_3 + b_0 x + b_1 x^2 + b_2 x^3$$

Take-home: $x.c(x)$ is basically shifting $c(x)$ by one position



codeword $\{b_3 b_0 b_1 b_2\}$ is a circular shift of $\{b_0 b_1 b_2 b_3\}$

Example of polynomial multiplication

$$d(x) = (1011000) = x^3 + x^2 + 1$$

$$c(x) = d(x).g(x)$$

$$g(x) = x^3 + x + 1$$

$$= (x^3 + x^2 + 1).(x^3 + x + 1)$$

$$(x^3 + x^2 + 1).(x^3 + x + 1)$$

$$= x^3.(x^3 + x^2 + 1) + x.(x^3 + x^2 + 1) + (x^3 + x^2 + 1)$$

Example of polynomial multiplication

$$d(x) = (1011000) = x^3 + x^2 + 1$$

$$c(x) = d(x).g(x)$$

$$g(x) = x^3 + x + 1$$

$$= (x^3 + x^2 + 1).(x^3 + x + 1)$$

$$(x^3 + x^2 + 1).(x^3 + x + 1)$$

$$= x^3.(x^3 + x^2 + 1) + x.(x^3 + x^2 + 1) + (x^3 + x^2 + 1)$$

$$1011000 \rightarrow 1x^0 + 0x^1 + 1x^2 + 1x^3 + 0x^4 + 0x^5 + 0x^6 + 0x^7$$

Example of polynomial multiplication

$$d(x) = (1011000) = x^3 + x^2 + 1$$

$$c(x) = d(x).g(x)$$

$$g(x) = x^3 + x + 1$$

$$= (x^3 + x^2 + 1).(x^3 + x + 1)$$

$$(x^3 + x^2 + 1).(x^3 + x + 1)$$

$$= x^3.(x^3 + x^2 + 1) + x.(x^3 + x^2 + 1) + (x^3 + x^2 + 1)$$



Shift 1011000 by three positions to the right

$$= 00010111$$

Example of polynomial multiplication

$$d(x) = (1011000) = x^3 + x^2 + 1$$

$$c(x) = d(x).g(x)$$

$$g(x) = x^3 + x + 1$$

$$= (x^3 + x^2 + 1).(x^3 + x + 1)$$

$$(x^3 + x^2 + 1).(x^3 + x + 1)$$

$$= x^3.(x^3 + x^2 + 1) + x.(x^3 + x^2 + 1) + (x^3 + x^2 + 1)$$



Shift 1011000 by one position to the right

$$= 00010111 + 0101100$$

Example of polynomial multiplication

$$d(x) = (1011000) = x^3 + x^2 + 1$$

$$c(x) = d(x).g(x)$$

$$g(x) = x^3 + x + 1$$

$$= (x^3 + x^2 + 1).(x^3 + x + 1)$$

$$(x^3 + x^2 + 1).(x^3 + x + 1)$$

$$= x^3.(x^3 + x^2 + 1) + x.(x^3 + x^2 + 1) + (x^3 + x^2 + 1)$$



No shifting, add 1011000

$$= 00010111 + 0101100 + 1011000$$

$$= 1111111 = x^6 + x^5 + x^4 + x^3 + x^2 + x^1 + 1$$

Example: (7, 4) cyclic code

Question: Find a generator polynomial for (7, 4) cyclic code.

Answer:

(7, 4) cyclic code means 7 bits to encode 4 bits of data ($n=7$, $k=4$).

$g(x)$ must contain the two following properties:

- $g(x)$ should be of a degree $(n - k) = 7 - 4 = 3$
- $g(x)$ should divide $1+x^7$ without a remainder

$1+x^7$ can be factored as:

$$1+x^7 = (1+x+x^3)(1+x^2+x^3)(1+x)$$

Two important properties to remember:

- $g(x)$ has a degree $(n-k)$
- $g(x)$ divides $1 + x^n$ without a remainder

- so, we can choose for $g(x)$ either $1+x+x^3$ or $1+x^2+x^3$

Next class: decoding and error detection in cyclic codes

Lecture 10

Information Redundancy - Part IV

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Cyclic codes
 - One more example on encoding
 - Decoding
 - Error detection
- TODOs

Cyclic codes

Cyclic: any circular shift of a codeword produces another codeword

- Move the bit at the rightmost position to the leftmost position
- Shift all other bits by one position to the right

For example:

- Consider the codeword 10110
- Circular shift by one position to the right: 01011
- Circular shift by two positions to the right: 10101
- ...

Properties of linear cyclic codes

In practice, cyclic codes designed for error detection should have two main properties:

Property 1: Linear

- The sum of any two or more codewords in C is again a codeword in C .

Property 2: Cyclic

- For a codeword in C , all its cyclic shifts are also codewords.

Example: (7, 4) cyclic code

Question: Find a generator polynomial for (7, 4) cyclic code.

Answer:

(7, 4) cyclic code means 7 bits to encode 4 bits of data ($n=7$, $k=4$).

$g(x)$ must contain the two following properties:

- $g(x)$ should be of a degree $(n - k) = 7 - 4 = 3$
- $g(x)$ should divide $1+x^7$ without a remainder

$1+x^7$ can be factored as:

$$1+x^7 = (1+x+x^3)(1+x^2+x^3)(1+x)$$

Two important properties to remember:

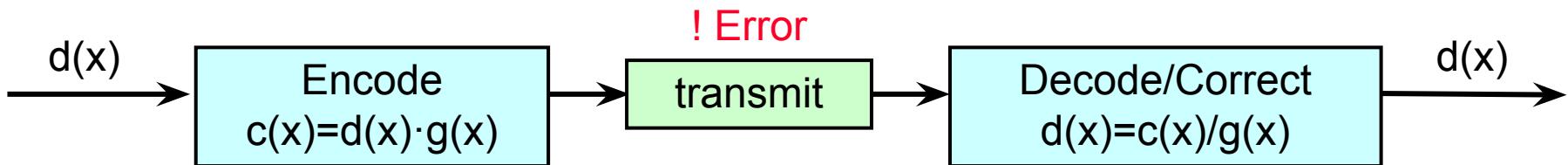
- $g(x)$ has a degree $(n-k)$
- $g(x)$ divides $1 + x^n$ without a remainder

- so, we can choose for $g(x)$ either $1+x+x^3$ or $1+x^2+x^3$

Generator polynomial

Generator polynomial, denoted as $g(x)$, is used to:

- encode the **data polynomial** into **codeword polynomial**.
- decode the **codeword polynomial** back to the **data polynomial**.



Example on encoding



Suppose $g(x) = (1+x+x^3)$ for a (7,4) cyclic code

Question: Find the codewords for the following data: 0001, 1001, 0110, 1000

Multiply data polynomial by generator polynomial:

$$c(x) = d(x).g(x)$$

Example on encoding



Suppose $g(x) = (1+x+x^3)$ for a (7,4) cyclic code

Question: Find the codewords for the following data: 0001, 1001, 0110, 1000

Solution: For each entry in the data (e.g., 0001, 1001):

- Step 1: convert it into a data polynomial
- Step 2: solve the codeword polynomial multiplication

Hint: use the circular shifting property to calculate the codeword polynomial instead. It is much faster.

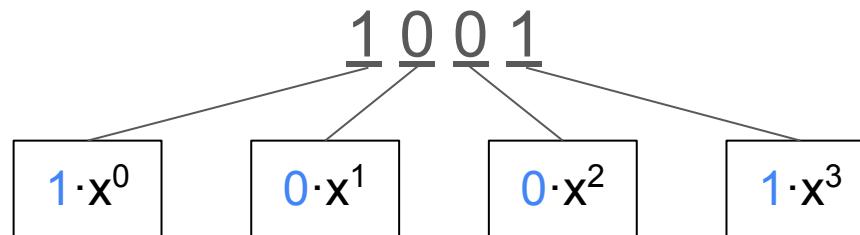
Step 1: data polynomial

Data = {1001}

The data can be represented as a polynomial:

$$a(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + \dots + a_{n-1} \cdot x^{n-1}$$

The data can also be visualized as:



So we represent 1001 as:

$$d(x) = 1 \cdot x^0 + 0 \cdot x^1 + 0 \cdot x^2 + 1 \cdot x^3 = 1 + x^3$$

Step 2: solve polynomial multiplication

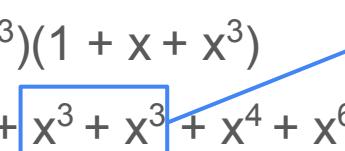
Data polynomial: $d(x) = 1 + x^3$

Generator polynomial: $g(x) = 1 + x + x^3$

Solve $c(x) = d(x).g(x)$

$$\begin{aligned}c(x) &= d(x).g(x) = (1 + x^3)(1 + x + x^3) \\&= 1 + x + \boxed{x^3 + x^3} + x^4 + x^6 \\&= 1 + x + x^4 + x^6\end{aligned}$$

$a_3 = 2$, same as 0



Step 2: solve polynomial multiplication

Data polynomial: $d(x) = 1 + x^3$

Generator polynomial: $g(x) = 1 + x + x^3$

Solve $c(x) = d(x).g(x)$

$$\begin{aligned}c(x) &= d(x).g(x) = (1 + x^3)(1 + x + x^3) \\&= 1 + x + x^3 + x^3 + x^4 + x^6 \\&= 1 + x + x^4 + x^6\end{aligned}$$

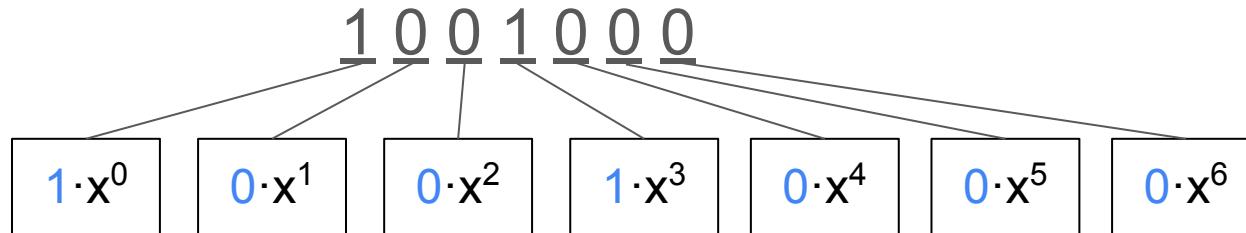
$$c(x) = \{1100101\}$$

Time-consuming if you need to calculate more than one codeword!

Alternative solution: with cyclic property

Given (7,4) cyclic code, $n = 7$

$$d(x) = 1 + x^3 = 1001000$$



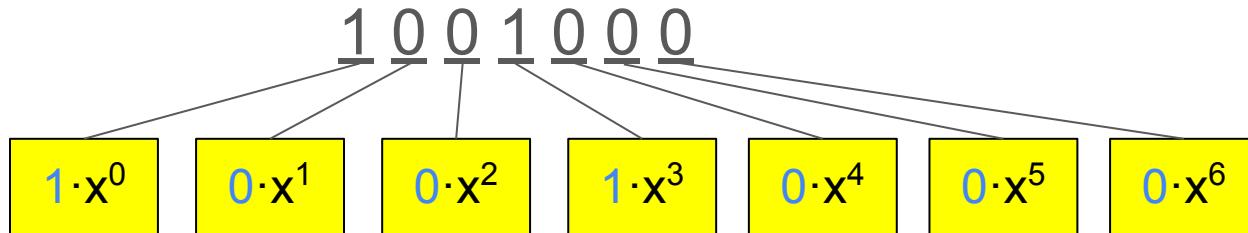
- In codeword polynomial, we always write least significant digit on the left
- $1001000 = 1 + x^3 = 1001$

Alternative solution: with cyclic property

Given (7,4) cyclic code, $n = 7$

$$d(x) = 1 + x^3 = 1001000$$

Updated Lecture 9 - slide 29



- In codeword polynomial, we write least significant digit on the left
- $1001000 = 1 + x^3 = 1001$

Alternative solution: with cyclic property

$$d(x) = 1 + x^3 = 1001000$$

$$g(x) = 1 + x + x^3$$

We multiply $d(x)$ by $g(x)$ to get $c(x)$:

Cyclic property on multiplication: $x \cdot c(x)$ is the same as shifting $c(x)$ by one position

We can use the cyclic property to compute the multiplication for each term in $g(x)$:

- Multiplication ($1 \cdot (1 + x^3)$): No shifting, 1001000

Alternative solution: with cyclic property

$$d(x) = 1 + x^3 = 1001000$$

$$g(x) = 1 + x + x^3$$

We multiply $d(x)$ by $g(x)$ to get $c(x)$:

Cyclic property on multiplication: $x \cdot c(x)$ is the same as shifting $c(x)$ by one position

We can use the cyclic property to compute the multiplication for each term in $g(x)$:

- Multiplication $(1 \cdot (1 + x^3))$: No shifting, 1001000
- Multiplication $(x \cdot (1 + x^3))$: Shifting by one position, 1001000 → 0100100

Alternative solution: with cyclic property

$$d(x) = 1 + x^3 = 1001000$$

$$g(x) = 1 + x + x^3$$

We multiply $d(x)$ by $g(x)$ to get $c(x)$:

Cyclic property on multiplication: $x \cdot c(x)$ is the same as shifting $c(x)$ by one position

We can use the cyclic property to compute the multiplication for each term in $g(x)$:

- Multiplication $(1 \cdot (1 + x^3))$: No shifting, 1001000
- Multiplication $(x \cdot (1 + x^3))$: Shifting by one position, 1001000 → 0100100
- Multiplication $(x^3 \cdot (1 + x^3))$: Shifting by three positions, 1001000 → 0001001

Alternative solution: with cyclic property

We get the codeword by adding the three terms together (distributive law)

$$\begin{array}{r} 1001000 \\ 0100100 \\ + 0001001 \\ \hline 1100101 \end{array}$$

By cyclic property, $d(x) \cdot g(x) = 1100101$

Example on encoding



Suppose $g(x) = (1+x+x^3)$ for a (7,4) cyclic code

No shifting, 1 shifting, 3 shifting

Question: Find the codewords for the following data: 0001, 1001, 0110, 1000

Solution: Using the circular shifting property, compute for $d(x) \cdot g(x)$

For data = {0001000}, $d(x) \cdot g(x) = 0001000 + 0000100 + 0000001 = 0001101$

Example on encoding



Suppose $g(x) = (1+x+x^3)$ for a (7,4) cyclic code

Question: Find the codewords for the following data: 0001, 1001, 0110, 1000

Solution: Using the circular shifting property, compute for $d(x) \cdot g(x)$

For data = {0001000}, $d(x) \cdot g(x) = 0001000 + 0000100 + 0000001 = 0001101$

For data = {1001000}, $d(x) \cdot g(x) = 1001000 + 0100100 + 0001001 = 1100101$

Example on encoding



Suppose $g(x) = (1+x+x^3)$ for a (7,4) cyclic code

Question: Find the codewords for the following data: 0001, 1001, 0110, 1000

Solution: Using the circular shifting property, compute for $d(x) \cdot g(x)$

For data = {0001000}, $d(x) \cdot g(x) = 0001000 + 0000100 + 0000001 = 0001101$

For data = {1001000}, $d(x) \cdot g(x) = 1001000 + 0100100 + 0001001 = 1100101$

For data = {0110000}, $d(x) \cdot g(x) = 0110000 + 0011000 + 0000110 = 0101110$

For data = {1000000}, $d(x) \cdot g(x) = 1000000 + 0100000 + 0001000 = 1101000$

Schedule for today

- Key concepts from last class
- Cyclic codes
 - One more example on encoding
 - Decoding
 - Error detection
- TODOs

Decoding

Divide codeword polynomial $c(x)$ by
the generator polynomial $g(x)$:
$$d(x) = c(x)/g(x)$$

If no error occurred:

- The received codeword is the correct codeword $c(x)$
- Therefore, $d(x) = c(x)/g(x)$, the remainder from the division is zero

Polynomial division

To divide two polynomials:

- We do the division of polynomials the same way as we do long division.
- Except the subtraction is modulo 2.

Polynomial division (1)

$$\begin{array}{r} x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\ \hline x^3 + x + 1 \end{array}$$

Polynomial division (1)

$$\begin{array}{r} x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\ \underline{-} \quad x^6 + x^4 + x^3 \\ \hline x^5 + x^2 + x + 1 \end{array} \qquad \left| \begin{array}{r} x^3 + x + 1 \\ \hline x^3 \end{array} \right.$$

We define addition and subtraction as modulo 2 with no carries.

This means addition = subtraction = XOR.

Subtraction for codewords

The “subtraction” here is different from the ordinary subtraction of numbers in two respects:

- No borrowing, each bit is independent of each other bit
- If $1 + 1 = 0$, then $0 - 1 = 1$, hence:
 - $0 - 0 = 0$, or $0 \oplus 0 = 0$
 - $0 - 1 = 1$, or $0 \oplus 1 = 1$
 - $1 - 0 = 1$, or $1 \oplus 0 = 1$
 - $1 - 1 = 0$, or $1 \oplus 1 = 0$

-	0	1
0	0	1
1	1	0

Polynomial division (1)

$$\begin{array}{r} x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\ - (x^6 + x^4 + x^3) \\ \hline x^5 + x^2 + x + 1 \end{array} \quad \left| \begin{array}{r} x^3 + x + 1 \\ \hline x^3 \end{array} \right.$$

Think of the xor
operation

Polynomial division (1)

$$\begin{array}{r} x^6 + \boxed{x^5} + x^4 + x^3 + x^2 + x + 1 \\ - (x^6 + x^4 + x^3) \\ \hline x^5 \end{array} \quad \left| \begin{array}{r} x^3 + x + 1 \\ \hline x^3 \end{array} \right.$$

$$1 \oplus 0 = 1$$

Polynomial division (1)

$$\begin{array}{r} x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\ \underline{- (x^6 + x^4 + x^3)} \\ x^5 \\ \hline \end{array} \quad \left| \begin{array}{r} x^3 + x + 1 \\ \hline x^3 \end{array} \right.$$

Diagram illustrating polynomial division:

- The dividend is $x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$. The term x^4 is highlighted with a blue box.
- The divisor is $x^3 + x + 1$.
- The quotient is x^3 .
- The remainder is x^5 .
- A blue arrow points from the highlighted x^4 in the dividend to the highlighted x^4 in the divisor, indicating the subtraction step: $x^4 - x^4 = 0$.
- A blue box at the bottom contains the equation $1 \oplus 1 = 0$, representing the binary addition of coefficients (1 and 1) resulting in 0.

Polynomial division (1)

$$\begin{array}{r} x^6 + x^5 + x^4 + \boxed{x^3} + x^2 + x + 1 \\ \underline{- (x^6 + x^4 + \boxed{x^3})} \\ x^5 \end{array} \quad \left| \begin{array}{r} x^3 + x + 1 \\ \hline x^3 \end{array} \right.$$

Diagram illustrating polynomial division. The dividend is $x^6 + x^5 + x^4 + \boxed{x^3} + x^2 + x + 1$. The divisor is $x^3 + x + 1$. The quotient is x^3 . The remainder is x^5 . Blue boxes highlight terms being subtracted: x^3 from the dividend and x^3 from the divisor. An arrow points from the term x^3 in the dividend box to the term x^3 in the divisor box. Another arrow points from the term x^3 in the divisor box to the result $1 \oplus 1 = 0$ in the remainder box.

Polynomial division (1)

$$\begin{array}{r} x^6 + x^5 + x^4 + x^3 + \boxed{x^2} + x + 1 \\ \underline{- (x^6 + x^4 + x^3)} \\ \hline x^5 + x^2 \\ \end{array} \quad \left| \begin{array}{r} x^3 + x + 1 \\ \hline x^3 \end{array} \right.$$

A blue arrow points from the term x^2 in the dividend to a blue box containing the equation $1 \oplus 0 = 1$.

Polynomial division (1)

$$\begin{array}{r} x^6 + x^5 + x^4 + x^3 + x^2 + \boxed{x + 1} \\ \underline{- (x^6 + x^4 + x^3)} \\ x^5 + x^2 + x + 1 \end{array} \quad \begin{array}{c} x^3 + x + 1 \\ \hline x^3 \end{array}$$

A blue arrow points from the term x in the dividend to the term x^3 in the quotient. Another blue arrow points from the term 1 in the remainder to the equation $1 \oplus 0 = 1$.

$$1 \oplus 0 = 1$$

Polynomial division (1)

$$\begin{array}{r} x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\ \underline{-} x^6 + x^4 + x^3 \\ \hline x^5 + x^2 + x + 1 \\ \underline{-} x^5 + \boxed{x^3} + x^2 \\ \hline x^3 + x + 1 \end{array} \quad \left| \begin{array}{r} x^3 + x + 1 \\ \hline x^3 + x^2 \end{array} \right.$$

0 \oplus 1 = 1

Polynomial division (1)

$$\begin{array}{r} x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\ \underline{-} x^6 + x^4 + x^3 \\ \hline x^5 + x^2 + x + 1 \\ \underline{-} x^5 + x^3 + x^2 \\ \hline x^3 + x + 1 \\ \underline{-} x^3 + x + 1 \\ \hline 0 \end{array} \quad \left| \begin{array}{r} x^3 + x + 1 \\ \hline x^3 + x^2 + 1 \end{array} \right.$$

No error

Polynomial division (2)

$$x^8 + x^5 + x^4 + x^2 + 1$$

$$x^8 + x^6 + x^5$$

$$x^6 + x^4 + x^2 + 1$$

$$x^6 + x^4 + x^3$$

$$x^3 + x^2 + 1$$

$$x^3 + x + 1$$

$$x^2 + x$$

$$x^3 + x + 1$$

$$x^5 + x^3 + 1$$

Error

Decoding (no error)

- If no error occurred
 - The received codeword is the correct codeword $c(x)$
 - Therefore, $d(x) = c(x)/g(x)$, the remainder from the division is zero

Decoding (in presence of error)

- Suppose an error has occurred, then

$$c^{\text{received}}(x) = c(x) + e(x), \quad e(x) = \text{error polynomial}$$

$$d^{\text{received}}(x) = (c(x) + e(x))/g(x)$$

Unless $e(x)$ is a multiple of $g(x)$, the received codeword will not be evenly divisible by $g(x)$.

- If $e(x)$ is a multiple of $g(x)$, the remainder of $e(x)/g(x)$ is 0 and the error will not be detected.

Schedule for today

- Key concepts from last class
- Cyclic codes
 - One more example on encoding
 - Decoding
 - Error detection
- TODOs

Example of error detection

Suppose that there is an error during transmission with the following polynomials:

$$d(x) = (1011) = x^3 + x^2 + 1 \quad g(x) = x^3 + x + 1 \quad e(x) = x^3 + 1$$

Question: Prove that this cyclic code can detect the error, then find the remainder.

- Calculate the codeword polynomial:

$$c(x) = d(x).g(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$$

- Calculate the received codeword:

$$c^{\text{received}} = c(x) + e(x) = x^6 + x^5 + x^4 + x^2 + x$$

- Calculate the data polynomial through division:

$$d(x) = c^{\text{received}}/g(x) = (x^6 + x^5 + x^4 + x^2 + x) / (x^3 + x + 1)$$

- Remainder is x, so the error is detected.

Summary of cyclic code

- Any circular shift of a codeword produces another codeword.
- Code is characterized by its generator polynomial $g(x)$, with a degree $(n-k)$, where n = bits in codeword, k = bits in data.
- All calculations are done in mod 2 arithmetic.
 - Multiplication of polynomial for encoding
 - Division of polynomial for decoding
- Cyclic code detects all single errors and all multiple adjacent error affecting $(n-k)$ bits or less.
 - It does not correct the error.

TODOs

- Demo guide is available on eClass (posted last Friday), it will help you to implement the system.
- The deadline for the final report is extended to Friday, 9 February 2024, 11:59 PM.

Lecture 11

Byzantine Fault Tolerance

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Failure classification
- Halting failures
- Byzantine failure
- The Two Generals Problem
- Next class: Byzantine General problem

Example on encoding



Suppose $g(x) = (1+x+x^3)$ for a (7,4) cyclic code

Question: Find the codewords for the following data: 0001, 1001, 0110, 1000

Solution: For each entry in the data (e.g., 0001, 1001):

- Step 1: convert it into a data polynomial
- Step 2: solve the codeword polynomial multiplication

Hint: use the circular shifting property to calculate the codeword polynomial instead. It is much faster.

Example on encoding



Suppose $g(x) = (1+x+x^3)$ for a (7,4) cyclic code

No shifting, 1 shifting, 3 shifting

Question: Find the codewords for the following data: 0001, 1001, 0110, 1000

Solution: Using the circular shifting property, compute for $d(x) \cdot g(x)$

For data = {0001000}, $d(x) \cdot g(x) = 0001000 + 0000100 + 0000001 = 0001101$

Summary of cyclic code

- Any circular shift of a codeword produces another codeword.
- Code is characterized by its generator polynomial $g(x)$, with a degree $(n-k)$, where n = bits in codeword, k = bits in data.
- All calculations are done in mod 2 arithmetic.
 - Multiplication of polynomial for encoding
 - Division of polynomial for decoding
- Cyclic code detects all single errors and all multiple adjacent error affecting $(n-k)$ bits or less.
 - It does not correct the error.

Information redundancy

Information redundancy tolerate faults by adding information to the original data.

- Tolerate faults by means of coding
- Avoid unwanted information changes
- E.g., information loss during data storage or transmission

The term “coding” is used in the context
of communication and data storage

Software fault tolerance

Fault tolerance is the ability for systems to continue functioning as a whole despite the faults

- Example of fault tolerance: while a PDF reader may crash when editing a corrupted PDF file, the PDF reader will restart itself after saving the information

The system contains a **fail-stop failure**.

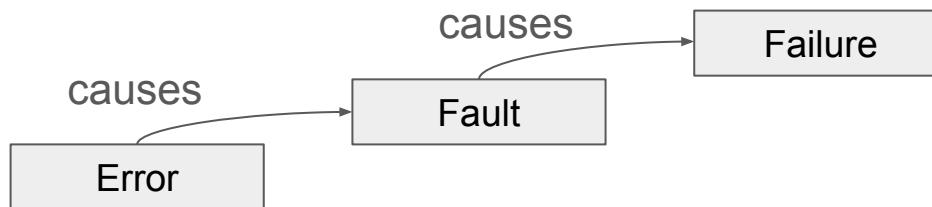
- Upon a fault, the system stops.
- To tolerate this type of behavior:
 - Solution 1: Restart the system (e.g., resetting the states)
 - Solution 2: Failover to redundancy

Schedule for today

- Key concepts from last class
- Failure classification
- Halting failures
- Byzantine failure
- The Two Generals Problem
- Next class: Byzantine General problem

Failure

Failure occurs when the system fails to perform its required function.



A failure is always related to a required function

- Based on the system specifications and functional requirements
- E.g., according to the user requirement, the maximum response time of a server shall be no longer than 15 seconds.
- A failure occurs when the response time exceeds 15 seconds.

Failure

A failure is an event that occurs at a specific point in time.

- It is not always possible to observe when a failure happens
- E.g., server performance degradation
- E.g., dormant faults in codes

The failure may:

- Come in different forms
- Originate from different kinds of faults
- Have different consequences on the system

Failure classification

Failure classification describes the way how a system can fail that is perceived by the rest of the system.

- It aims to answer one question: “**What kind of failure are we dealing with when considering the whole system into account.**”
- It helps understand how to build and design for fault-tolerant systems.

There are four types of failure:

- Timing failure, also known as performance failure
- Omission failure
- Crash failure
- Arbitrary failure, also known as Byzantine failure

Timing failure

Timing failure happens when the system delivers correct results, but outside the expected time interval.

- Failure lies in the amount of time the system took to execute a task
- Not in the results of the task itself

Example: Delayed response from a server

- The server must responds between 5 milliseconds and 15 seconds
- A response exceeds our expected time interval
- This produces a timing failure

Omission failure

Omission failure happens when the system never appears to respond.

- Can also be understood as “infinitely late” timing failure

There are two forms of omission failure:

- Send omission failure: fail to send a response
- Receive omission failure: fail to receive a response

Example: No response to incoming requests

- The client sends a request
- The server fails to respond to the request
 - Failed to send messages
 - Failed to receive messages

Crash failure

Crash failure happens when the system crashes, but is working correctly until it crashes.

- The system experiences omission once and becomes completely unresponsive (i.e., crashes)

Example: Server crashes

- The client sends a request
- The server experiences an omission failure, then fails and stops responding,

Byzantine failure

Byzantine failure happens when the system sends arbitrary responses at arbitrary times.

- Derived from Byzantine faults
- The system responds with different (erroneous) responses each time, and is inconsistent in what kind of response it delivers

Example: Arbitrary response to the same request

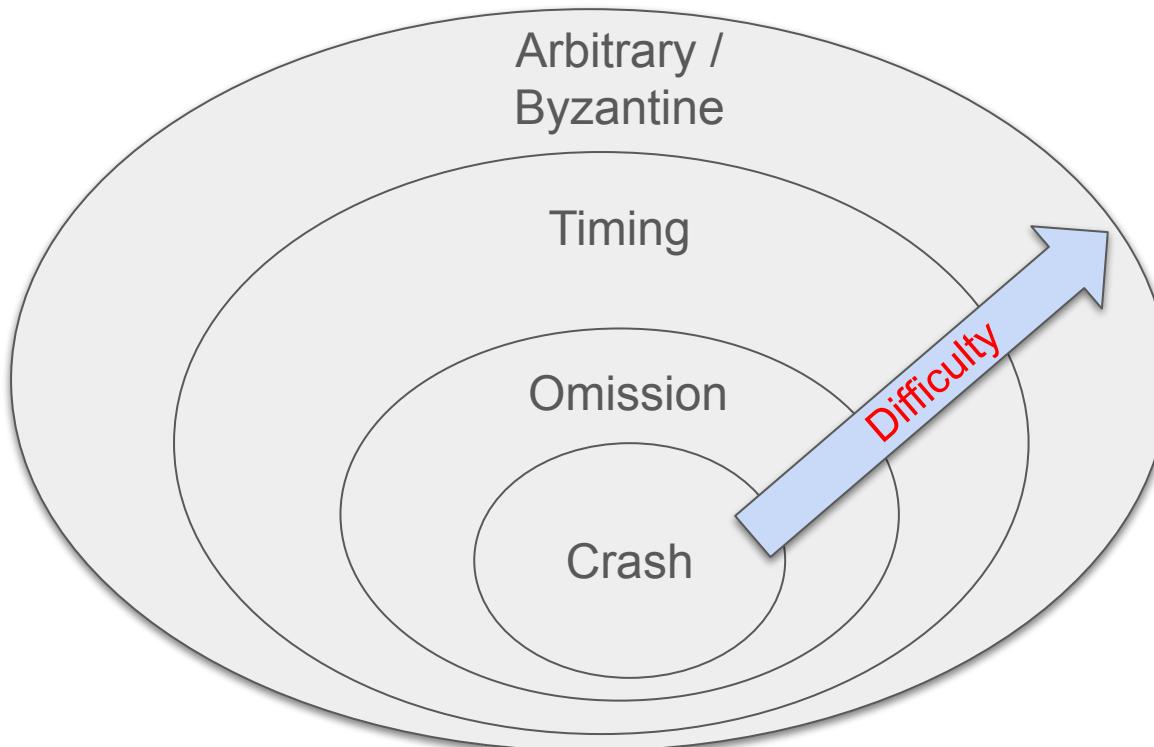
- The client A sends a request
- The server responds with a message A
- The client B sends the same request as client A
- The server arbitrarily responds with a message B

Examples of failure



Type of failures	Example of server's response
Crash failure	Valid responses until it crashes
Omission failure	No response to incoming requests
Timing failure	Delayed response, outside of a specified time interval
Arbitrary failure	Arbitrary response at arbitrary times

Difficulties of failures



Halting failures

Failure classification deal with the situation that we no longer perceive any actions from the system.

- Can we conclude that a system has come to a halt?
- How do we distinguish between crash/omission/timing failure?

There are five categories of halting failures on the reliability of failure detection (from the least to the most severe):

- Fail-stop
- Fail-noisy
- Fail-silent
- Fail-safe
- Fail-arbitrary

Types of halting failures

Let a *client* attempt to detect that the *server* has failed.

Fail-stop: failures that can be reliably detected.

- Crash failure, but reliably detectable
- Assuming that:
 - Non-faulty communication exists between the client and the server
 - There is a worst-case delay on the response from the server.
- E.g., Server stops and the client can detect this failure.

Fail-noisy: failures that are eventually reliably detected

- Crash failure, but eventually reliably detected
- The client will only eventually come to the correct conclusion that the server has crashed.
- E.g., Server's response time slows down, and eventually fails.

Types of halting failures

Fail-silent: failures that cannot be distinguished between crash and omission

- Omission or crash failures
- Assuming that:
 - Non-faulty communication exists between the client and the server
 - The client cannot distinguish crash failures from omission failures.
- E.g., No response from the server.

Fail-safe: failures that cannot do any harm.

- Arbitrary, yet benign failures
- The server fails without doing any harm
- E.g., Server slows down, but remains available.

Types of halting failures

Fail-arbitrary: arbitrary, with malicious failures

- Server may fail in any possible way; failures may be unobservable.
- Arbitrary where a server is producing output that it should never have produced, but which cannot be detected as being incorrect.
- E.g., The server may be working with other servers to produce intentionally wrong answers; it still responds to the client.

Schedule for today

- Key concepts from last class
- Failure classification
- Halting failures
- **Byzantine failure**
- The Two Generals Problem
- Next class: Byzantine General problem

Byzantine failure

Byzantine failure: a node/component may fail arbitrary due to:

- Exhausted resources
 - E.g., null response from the server due to resource exhaustion
- Conflicting information from different parts of the system
 - E.g., Malicious attack providing conflicting information
- ... and more

Byzantine failure

Byzantine failure: a node/component may fail arbitrary due to:

- Exhausted resources
- Conflicting information from different parts of the system

Why would nodes/components fail arbitrarily?

- Software bugs in the code
- Hardware failures
- Malicious attack on the system

What makes Byzantine failure important?

Byzantine failure provides an abstraction of issues in real-world applications

Example 1: Bitcoin

- Byzantine failure as the results of malicious nodes in the network
- A malicious node can cause failure in the network

Example 2: Aircraft flight control system (e.g., Boeing 777, 787)

- Byzantine failure as the results of potential hardware failures
- Hardware components can fail arbitrarily based on the temperature at which they operate

Solution? Designing Byzantine fault-tolerant systems for reliability

The Two Generals Problem

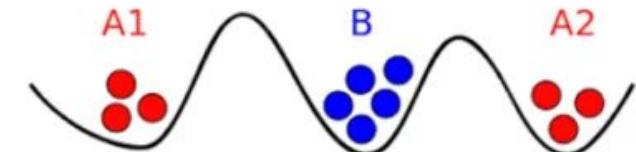
The two generals problem is a consensus problem where two Generals (General A1 and General A2) need to coordinate attack on the army B.

- If both Generals attack at the same time, they win the battle.
- If neither Generals attack, they will try to coordinate again the next day.
- But if only one of the two Generals attacks, they lose the battle.

There is only one way to communicate between the two Generals:

- By sending messengers on a high-risk path through enemy lines.

What messages can the generals send to reach a consensus on whether or not to attack?



The Two Generals Problem

Let General A1 be the one who decides to attack the next morning.

- Send the message to General A2 about the attack.
- Make sure that General A2 got the message.

What kind of message can General A1 send to General A2 to make sure that a consensus is reached?

Solution: TCP 3-way handshake

A1

A1 wants to
attack



If you respond, I'll attack!

A2

A1 will attack;
B wants to attack

Scenario 1: If the message was delivered to A2, everything is fine.

Solution: TCP 3-way handshake

A1

A1 wants to
attack



If you respond, I'll attack!

A2

A1 will attack;
A2 wants to attack

Scenario 2: If the messenger was captured by B, A1 will not receive any response from A2, they will not attack.

Solution: TCP 3-way handshake

A1

A1 wants to
attack

A1 will attack;
A2 will attack



If you respond, I'll attack!

A2

A1 will attack;
A2 wants to attack

Scenario 1: If the message
was delivered back to A1,
everything is fine.

Solution: TCP 3-way handshake

A1

A1 wants to
attack

A1 will attack;
A2 will attack



A2

A1 will attack;
A2 wants to attack

If you respond, I'll attack!

If you respond, I'll attack!



Scenario 2: If the messenger
was captured by B, A2 will not
receive any response from A1,
they will not attack.

Solution: TCP 3-way handshake

A1

A1 wants to
attack

A1 will attack;
A2 will attack



A2

A1 will attack;
A2 wants to attack

A1 will attack;
A2 will attack

If you respond, I'll attack!

If you respond, I'll attack!

We'll attack!

Scenario 1: If the message was delivered back to A2, everything is fine. They will both attack.

Solution: TCP 3-way handshake

A1

A1 wants to attack

A1 will attack;
A2 will attack



A2

Scenario 2: If the message was not delivered back to A2, A1 is committed to attack but A2 isn't.

If you respond, I'll attack!

If you respond, I'll attack!

We'll attack!
X

A1 will attack;
A2 wants to attack

A1 will attack;
A2 will attack

Solution: TCP 3-way handshake

A1

A1 wants to attack

A1 will attack;
A2 will attack



A2

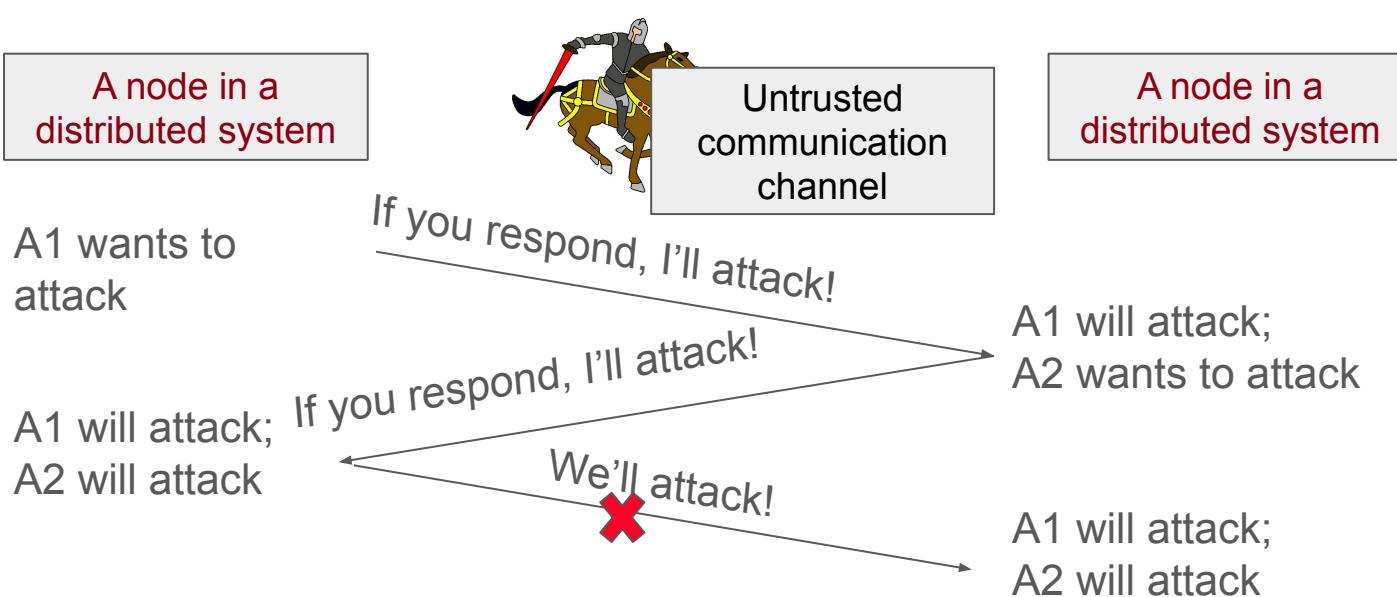
A1 will attack;
A2 wants to attack

A1 will attack;
A2 will attack



There is no solution to the two general problem.

Solution: TCP 3-way handshake in practice



Assumption: Failure is not 100% Byzantine.

The Two Generals Problem

The two generals problem is a classic computer science problem that remains unsolvable.

- It remains unsolvable as there is a need to be a last acknowledgement from General A1 to A2.
- And this starts a never-ending cycle of acknowledgement as the message may get lost.
- This is why the problem is unsolvable.

Next class: The Byzantine Generals Problem

Lecture 12

Byzantine Generals Problem

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Byzantine fault tolerance
- The Byzantine Generals Problem
 - Definition
 - Analogy to system design
 - The oral message solution
 - What happens when $m = 2$?

Failure classification

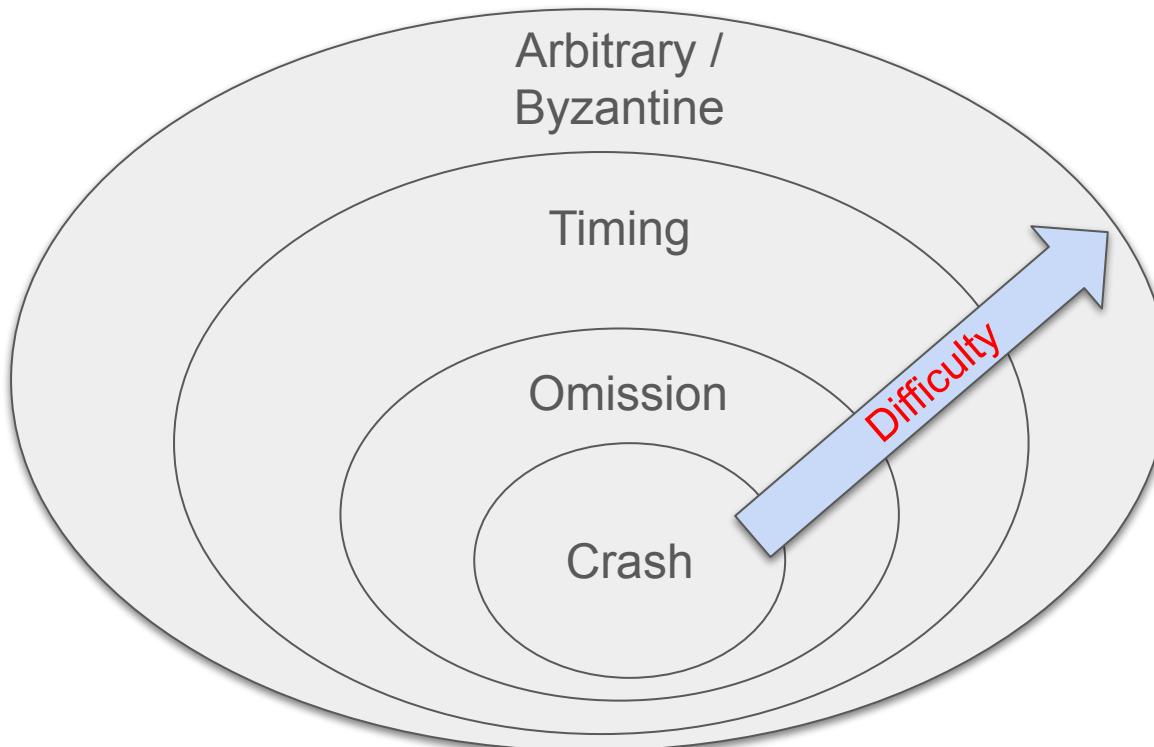
Failure classification describes the way how a system can fail that is perceived by the rest of the system.

- It aims to answer one question: “**What kind of failures are we dealing with when taking the whole system into account.**”
- It helps understand how to build and design for fault-tolerant systems.

There are four types of failure:

- Timing failure, also known as performance failure
- Omission failure
- Crash failure
- Arbitrary failure, also known as Byzantine failure

Difficulties of failures



Solution: TCP 3-way handshake

A1

A1 wants to attack

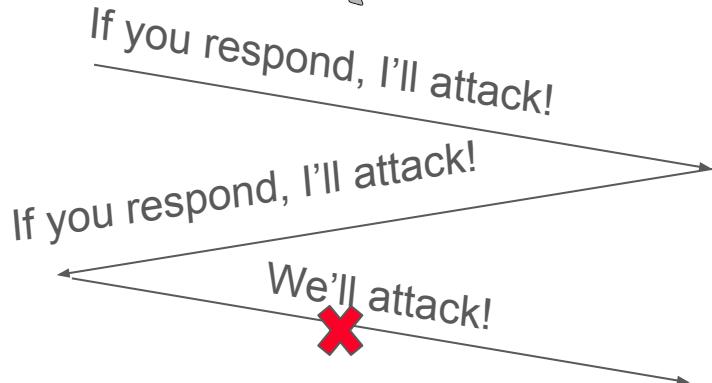
A1 will attack;
A2 will attack



A2

A1 will attack;
A2 wants to attack

A1 will attack;
A2 will attack



There is no solution to the two general problem.

Byzantine failure

Byzantine failure: a node/component may fail arbitrary due to:

- Exhausted resources
- Conflicting information from different parts of the system

Why would nodes/components fail arbitrarily?

- Software bugs in the code
- Hardware failures
- Malicious attack on the system

There is an **inconsistency** in the functionality of the component. It presents different symptoms to different observers.

Byzantine fault tolerance

Byzantine fault tolerance: the ability of a system to continue functioning even if some of its nodes/components fail randomly or act maliciously.

- The system can keep functioning even if certain components stop working
- For example, safety-critical systems need to be able to work when if some of its components fail
 - E.g., train, airplane, spacecraft
 - E.g., heart-lung machine, robotic surgery

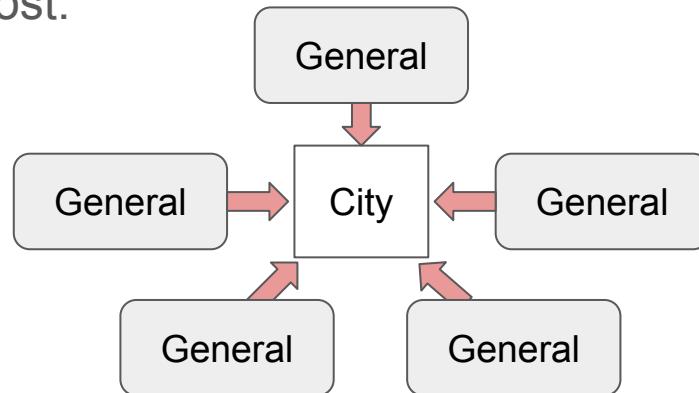
Schedule for today

- Key concepts from last class
- Byzantine fault tolerance
- The Byzantine Generals Problem
 - Definition
 - Analogy to system design
 - The oral message solution
 - What happens when $m = 2$?

The Byzantine Generals Problem

Byzantine Generals Problem: a group of Byzantine generals who must coordinate a attack by the means of messengers to reach consensus.

- The generals must decide as a group to attack or retreat.
- If they all make the same decision, they can eventually win the battle.
- But after the decision making phrase, if some of the generals attack while the other retreat, then the battle is lost.



The Byzantine Generals Problem

New assumptions:

- If a message is sent, it is always delivered correctly.
 - Sending lots of messengers increases the probability that one getting through
- One or more generals may be “traitors”
 - Analogous to a malicious node/component in the system
 - Traitors’ mission is to break the consensus among the “royal” generals (non-traitors) so that they lose the battle.

Consensus in the presence of a traitor

Suppose we have five generals, one of them is a traitor.

Each royal general shares their decision to others:

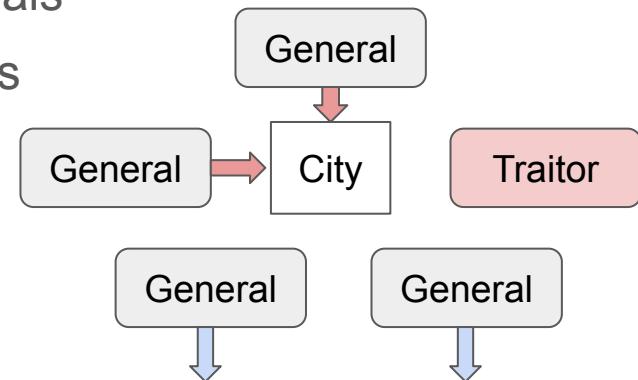
- Two generals send an “attack” message.
- Two other generals send a “retreat” message.

The traitor tries to break the consensus:

- Sends an “attack” message to the first two generals
- Send a “retreat” message to the last two generals

Next day: consensus is broken

- Two royal generals attack
- Two royal generals retreat



Analogy to system design

In a distributed system, some nodes can be:

- Flaky nodes
- Malicious nodes (e.g., Hackers)

The Byzantine problem attempts to propose a solution to these failures from two two main questions:

- How can we ensure the **consistency**?
 - All “royal” nodes produce the same results.
- How we can we ensure the **validity**?
 - All “royal” nodes produce a valid response.
- (Despite the presence of flaky or malicious nodes)

Byzantine Generals Problem

Lamport et al. proposed to solve the Byzantine Generals Problem with two solutions:

- Solution 1: Oral message
- Solution 2: Signed message

[PDF] The Byzantine generals problem

L Lamport, R Shostak, M Pease - Concurrency: the works of leslie ..., 1982 - dl.acm.org
... three **generals** that coped with one traitor, then we could construct a three-**general** solution to the **Byzantine Generals Problem** that also worked in the presence of one traitor. Suppose ...
☆ Save ⏪ Cite Cited by 9590 Related articles All 179 versions

The oral message solution simplifies the problem:

- A commanding general sends an order to his lieutenants.
 - A commanding general is the first person to send a decision
 - The rest become the lieutenants
 - Lieutenant can decide to execute or not execute the order from the command general
- Either of them can be traitors

Byzantine Generals Problem

When a commanding general sends an order to his lieutenants, there are two properties:

- **Consistency:** Loyal lieutenants must execute the same order.
- **Validity:** If the commanding general is loyal, then every loyal lieutenant must obey his order.

Solution to Byzantine Generals Problem

Let $m = \#$ traitors (malicious nodes), $n = \text{total } \#$ generals (nodes)

Theorem: We need $n = (3m + 1)$ generals to tolerate (m) traitors.

- If $m < \frac{1}{3} n$, then it is solvable
- We can somehow achieve both consistency and validity

Example: $m = 1$, $n = 3$

Scenario 1: Why is it unsolvable?

General A = commanding general, royal

General B = lieutenant, royal

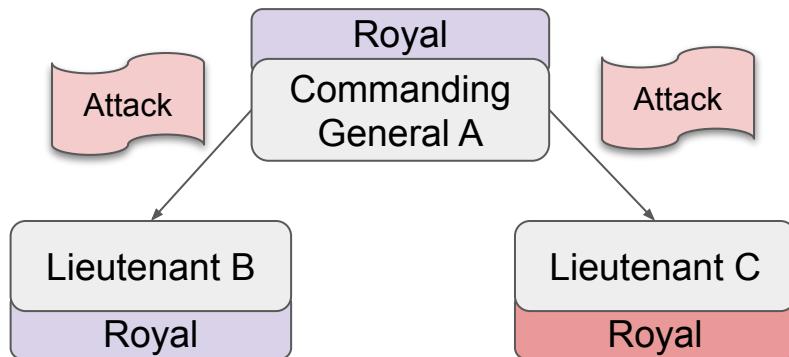
General C = lieutenant, traitor



Example: $m = 1, n = 3$

Scenario 1: Why is it unsolvable?

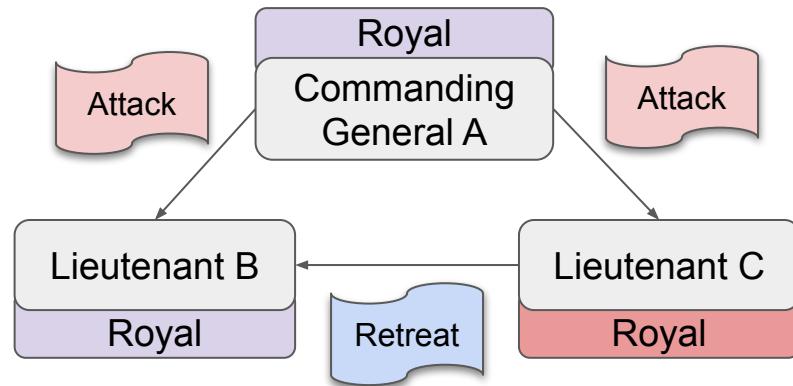
- A sends an “attack” message to the two others.
- B receives the message, and asks C to share the message that he/she received from A.



Example: $m = 1, n = 3$

Scenario 1: Why is it unsolvable?

- A sends an “attack” message to the two others.
- B receives the message, and asks C to share the message that he/she received from A.
- C is a traitor, thus changes the message to “retreat”.
- B cannot tell who is the traitor.
- **Unsolvable**



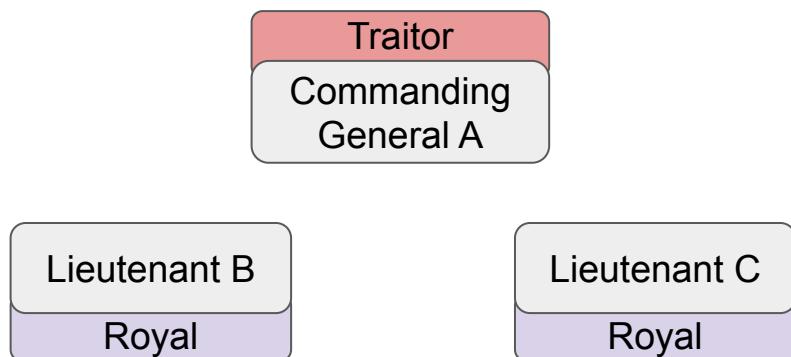
Example: m = 1, n = 3

Scenario 2: Why is it unsolvable?

General A = commanding general, **traitor**

General B = lieutenant, **royal**

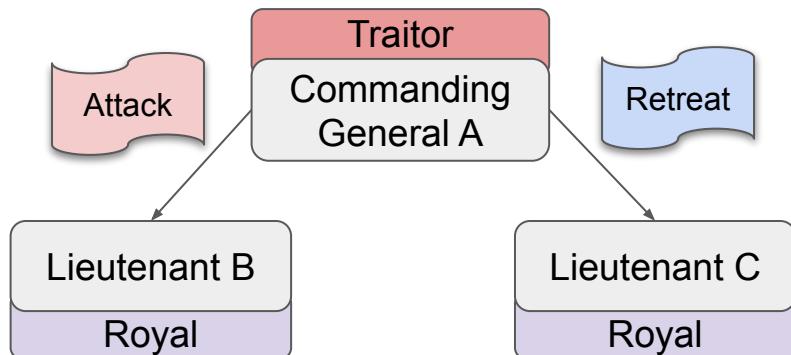
General C = lieutenant, **royal**



Example: $m = 1$, $n = 3$

Scenario 2: Why is it unsolvable?

- A sends different messages to others; “attack” to B, and “retreat” to C.
- B receives the message, and asks C to share the message that he/she received from A.

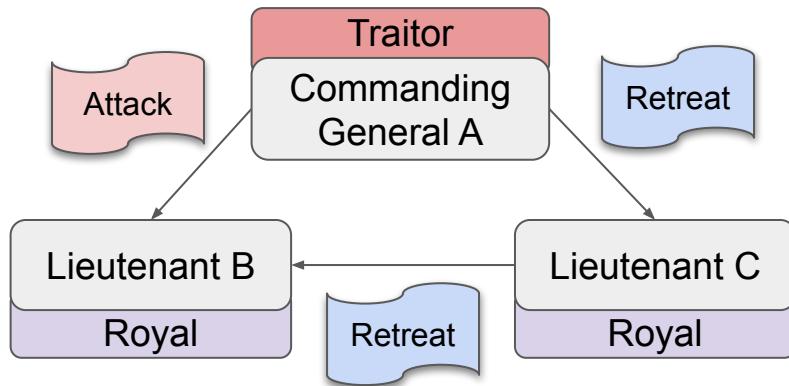


Example: $m = 1, n = 3$

Scenario 2: Why is it unsolvable?

- A sends different messages to others; “attack” to B, and “retreat” to C.
- B receives the message, and asks C to share the message that he/she received from A.
- C shares the message “retreat” with B.
- B still cannot tell who is the traitor.
- **Unsolvable**

If $m < \frac{1}{3} n$, then it is solvable,
Otherwise it is unsolvable.



Example: $m = 1$, $n = 4$

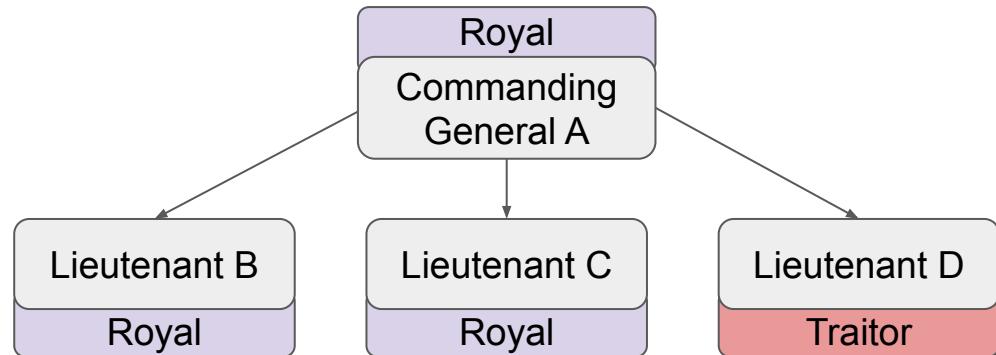
Scenario 1: Why is it solvable?

General A = commanding general, royal

General B = lieutenant, royal

General C = lieutenant, royal

General D = lieutenant, traitor

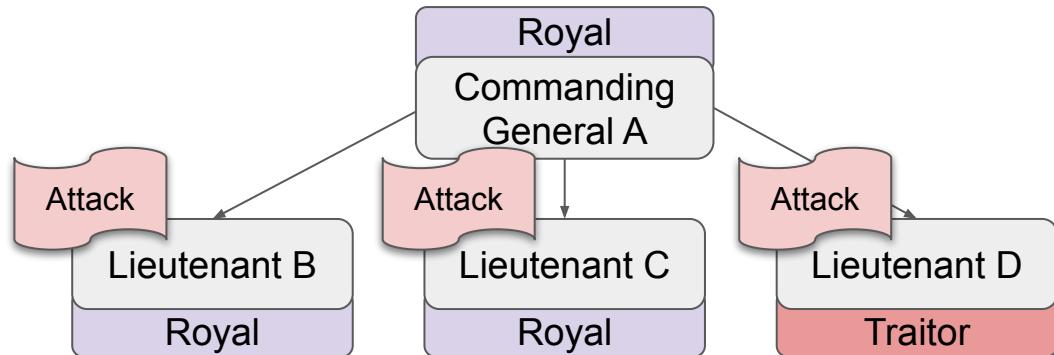


Example: $m = 1, n = 4$

Scenario 1: Why is it solvable?

- A sends an “attack” message to others.
- B asks C and D for the message they received.
 - $V(B) = (A, A, R)$
- B receives: “attack” from A, “attack” from C, and “retreat” from D
 - $V(B) = (A, A, R)$
- B decides to attack

- B decides to attack,
 $V(B) = (A, A, R)$

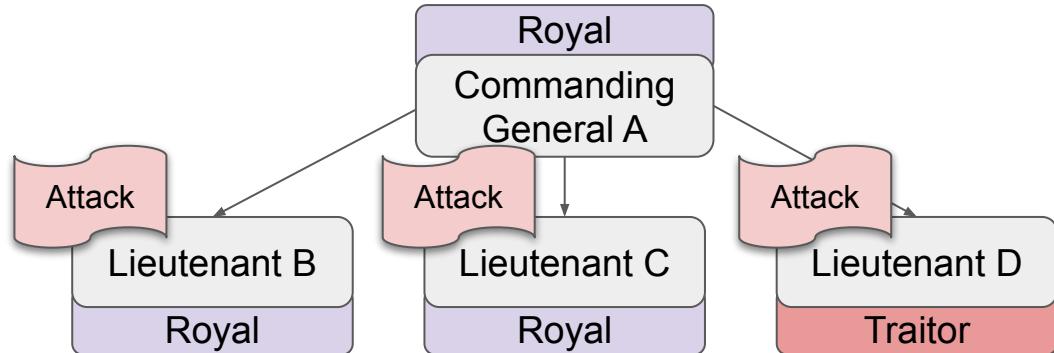


Example: $m = 1, n = 4$

Scenario 1: Why is it solvable?

- A sends an “attack” message to others.
- C asks B and D for the message they received.
- C receives: “attack” from A, “attack” from B, and “retreat” from D
 - $V(C) = (AA R)$
- C decides to attack

- B decides to attack,
 $V(B) = (AA R)$
- C decides to attack
 $V(C) = (AA R)$



Example: m = 1, n = 4

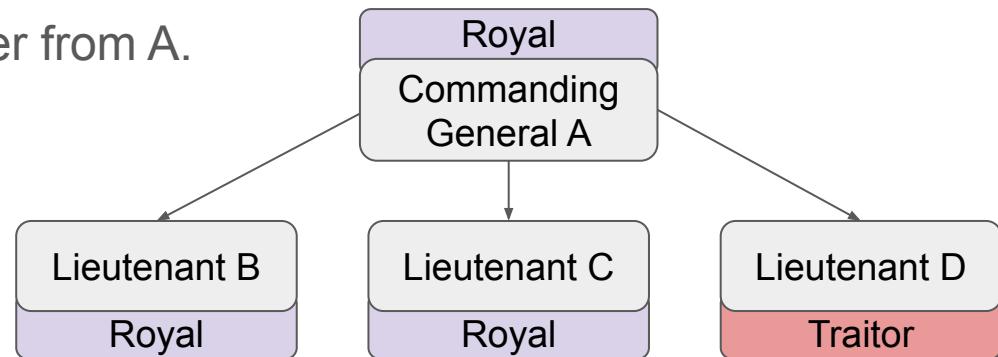
Scenario 1: Why is it solvable?

- A sends an “attack message” to other.

- B decides to attack,
 $V(B) = (A \ A \ R)$
- C decides to attack
 $V(C) = (A \ A \ R)$

Problem solved!

- **Consistency:** B and C executed the same order.
- **Validity:** B and C obeyed the order from A.



Example: $m = 1, n = 4$



Scenario 2: Is this solvable? Can we still achieve both properties?

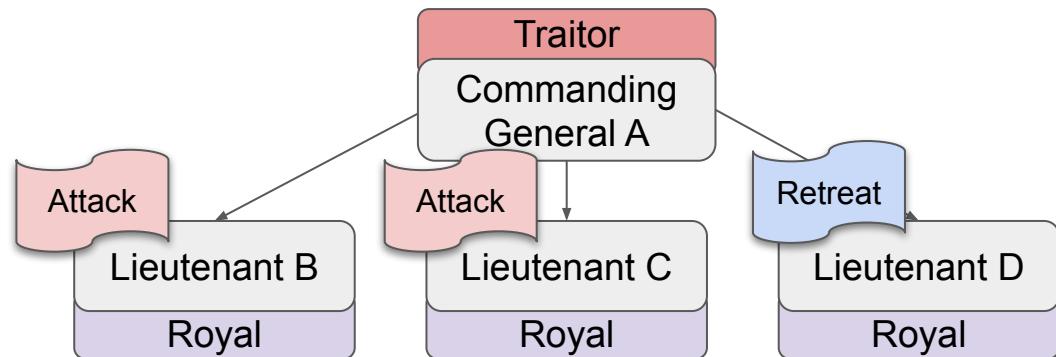
General A = commanding general, **traitor**

General B = lieutenant, **royal**

General C = lieutenant, **royal**

General D = lieutenant, **royal**

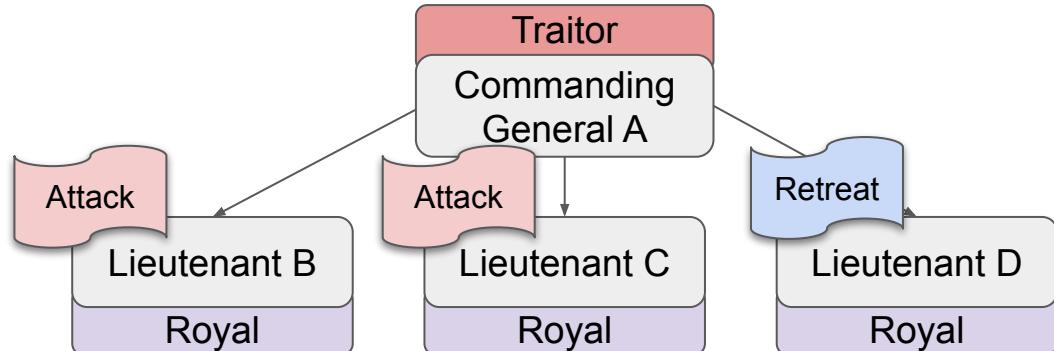
Validity: if and only if the commanding general is loyal, then every loyal lieutenant must obey his order.



Example: $m = 1$, $n = 4$

Scenario 2: Why is it solvable?

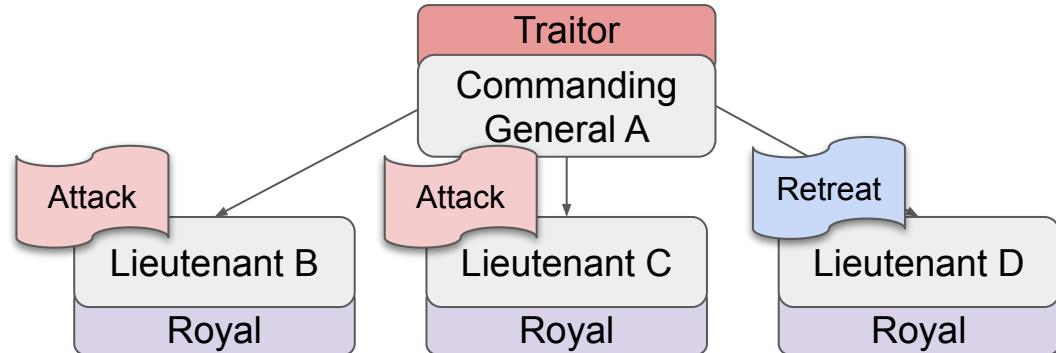
- A sends “attack” to the first two lieutenants, and “retreat” to the last.
- B receives: “attack” from A, “attack” from C, and “retreat” from D
 - $V(B) = (A \text{ A } R)$
- B decides to attack



Example: $m = 1$, $n = 4$

Scenario 2: Why is it solvable?

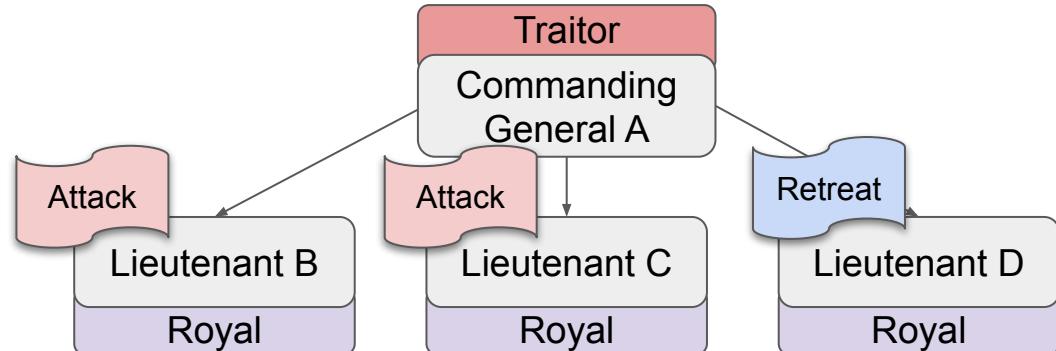
- A sends “attack” to the first two lieutenants, and “retreat” to the last.
- C receives: “attack” from A, “attack” from B, and “retreat” from D
 - $V(C) = (A \text{ A} \text{ R})$
- C decides to attack



Example: m = 1, n = 4

Scenario 2: Why is it solvable?

- A sends “attack” to the first two lieutenants, and “retreat” to the last.
- D receives: “retreat” from A, “attack” from B, and “attack” from C
 - $V(D) = (R \ A\ A)$
- D decides to attack



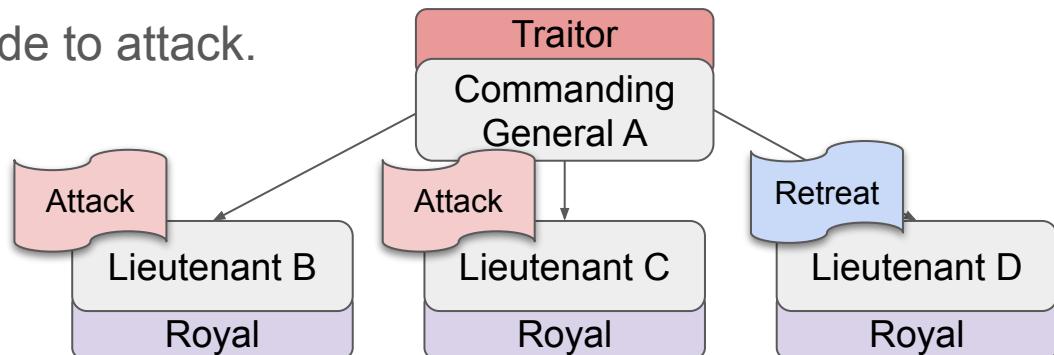
Example: $m = 1$, $n = 4$

Scenario 2: Why is it solvable?

- A sends “attack” to the first two lieutenants, and “retreat” to the last.
- $V(B) = (A\ A\ R)$
- $V(C) = (R\ A\ A)$
- $V(D) = (R\ A\ A)$
- All three royal lieutenants decide to attack.

Problem solved!

- **Consistency** fulfilled.



Schedule for today

- Key concepts from last class
- Byzantine fault tolerance
- The Byzantine Generals Problem
 - Definition
 - Analogy to system design
 - The oral message solution
 - What happens when $m = 2$?

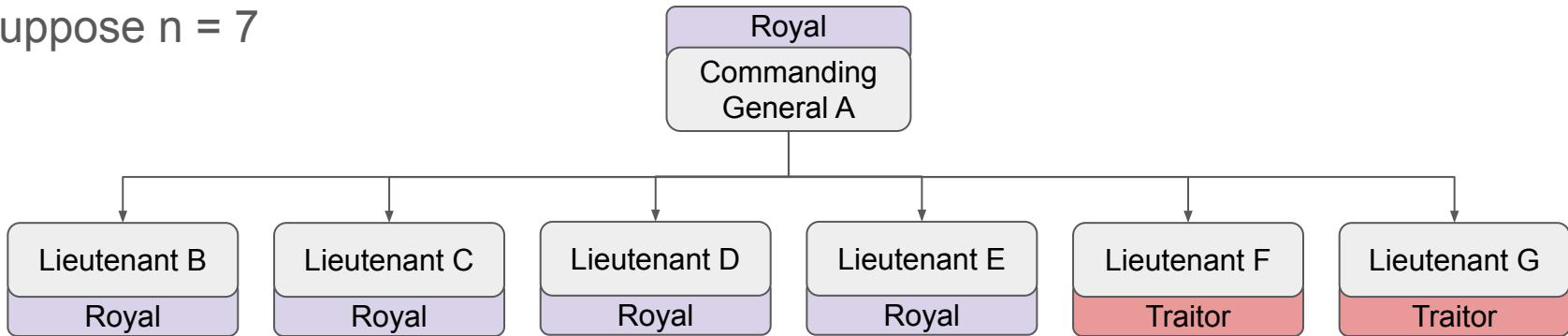
What happens when $m = 2$?

A recursive solution must be used to solve $m > 1$:

- The commanding general sends the message to the other lieutenants.
- Each lieutenant becomes a commanding general and sends the message they received from the original commanding general to the other lieutenants.
- ...

What happens when $m = 2$?

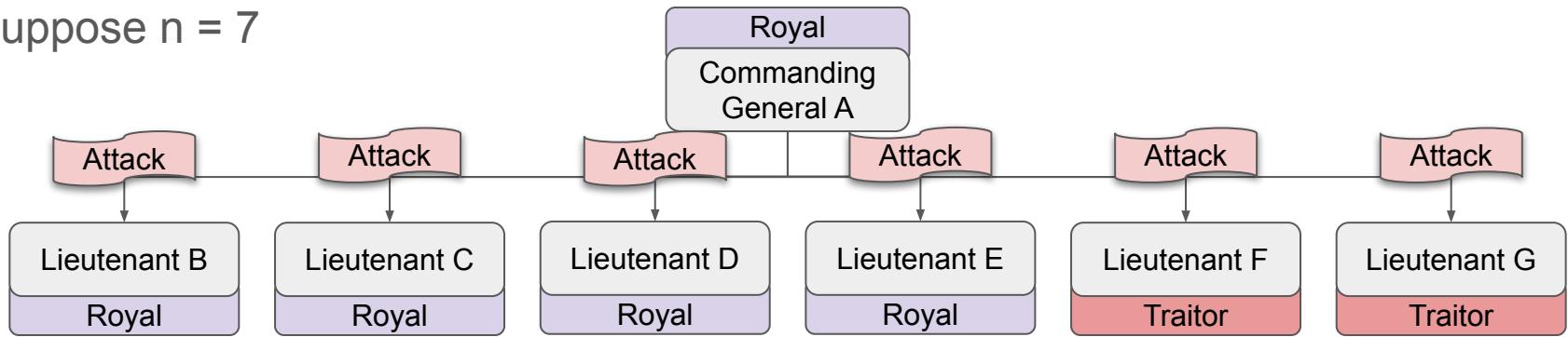
Suppose $n = 7$



- General A = commanding general, royal
- General B = lieutenant, royal
- General C = lieutenant, royal
- General D = lieutenant, royal
- General E = lieutenant, royal
- General F = lieutenant, traitor
- General G = lieutenant, traitor

What happens when $m = 2$?

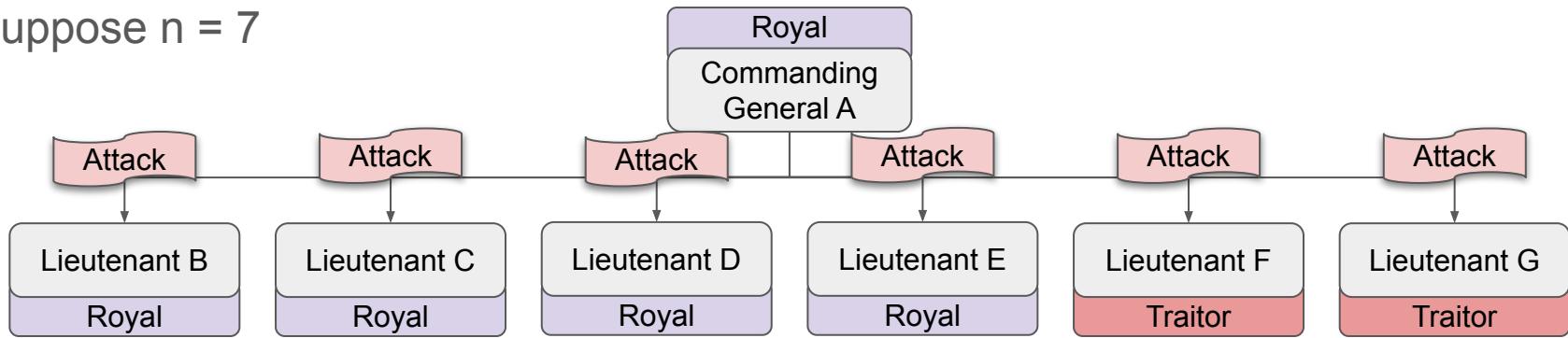
Suppose $n = 7$



- A sends “attack”
- B asks C the message he/she received
 - C replies “attack”
- B asks D, E, F, G: “what C received from A?”
 - D replies “attack”, E replies “attack”, F replies “retreat”, G replies “retreat”
 - $V_C = (A, A, A, R, R)$, B confirms that C received “attack”

What happens when $m = 2$?

Suppose $n = 7$



- A sends “attack”
- B asks C, D, E, F, G: “what D received from A?”
 - C replies “attack”, D replies “attack”, E replies “attack”, F replies “retreat”, G replies “retreat”
 - $V_D = (A, A, A, R, R)$, B confirms that D received “attack”

What happens when $m = 2$?

Suppose $n = 7$

For Lieutenant B

- Lieutenant B pick the most popular action from the other generals:
 - $V_B = A$, B receives “attack” from A
 - $V_C = (A, A, A, R, R)$, B confirms that C received “attack”
 - $V_D = (A, A, A, R, R)$, B confirms that D received “attack”
 - $V_E = (A, A, A, R, R)$, B confirms that E received “attack”
 - $V_F = (R, R, R, R, R)$, B confirms that F received “retreat”
 - $V_G = (R, R, R, R, R)$, B confirms that G received “retreat”
- Lieutenant B decides to attack, because there are 4 attacks and 2 retreats

For Lieutenant C ...

What happens when $m = 2$?

Suppose $n = 7$

- Commanding general A will attack.
- Lieutenant B decides to attack, because there are 4 attacks and 2 retreats
- Lieutenant C decides to attack, because there are 4 attacks and 2 retreats
- Lieutenant D decides to attack, because there are 4 attacks and 2 retreats
- Lieutenant E decides to attack, because there are 4 attacks and 2 retreats

All four royal lieutenant decide to attack, obeying A.

Problem solved!

- Consistency fulfilled
- Validity fulfilled

Cost of solving Byzantine Generals Problem

Solving the Byzantine Generals Problem is expensive

- $m = 0$, sending n messages to every lieutenant
- $m = 1$, sending n^2 messages to every lieutenant
- $m = 2$, sending n^3 messages to every lieutenant
- ...

m	Messages Sent
0	$O(n)$
1	$O(n^2)$
2	$O(n^3)$
3	$O(n^4)$

Timeline

- In 1978, Robert Shostak worked on a NASA-sponsored project about fault-tolerant for aircraft control.

SIFT: Design and analysis of a fault-tolerant computer for aircraft control
JH Wensley, L Lamport, J Goldberg... - Proceedings of the ..., 1978 - ieeexplore.ieee.org

SIFT (Software Implemented Fault Tolerance) is an ultrareliable computer for critical aircraft control applications that achieves fault tolerance by the replication of tasks among

[☆ Save](#) [芻 Cite](#) [Cited by 857](#) [Related articles](#) [All 17 versions](#)
- In 1980, Leslie Lamport, Robert Shostak, and Marshall Pease together worked on an algorithm for processors to reach consensus in the presence of faults.

Reaching agreement in the presence of faults
M Pease, R Shostak, L Lamport - Journal of the ACM (JACM), 1980 - dl.acm.org

... In the absence of **faults reaching** a satisfactory mutual **agreement** is usually an easy matter.
In ... include those of **reaching** approximate **agreement** and **reaching agreement** under various ...

[☆ Save](#) [芻 Cite](#) [Cited by 3167](#) [Related articles](#) [All 5 versions](#)
- In 1982, they later come up with the analogy of the Byzantine generals problem.

[PDF] The Byzantine generals problem
L Lamport, R Shostak, M Pease - Concurrency: the works of leslie ..., 1982 - dl.acm.org

... three **generals** that coped with one traitor, then we could construct a three-**general** solution to the **Byzantine Generals Problem** that also worked in **the** presence of one traitor. Suppose ...

[☆ Save](#) [芻 Cite](#) [Cited by 9590](#) [Related articles](#) [All 179 versions](#)

Leslie Lamport is also the initial developer of LaTeX.



Timeline

- In 1998, “Practical Byzantine Fault Tolerance” (PBFT) was published.
 - PBFT was proposed as a practical solution for Byzantine fault tolerance that takes into consideration there can be delays in the network.
- In 2008, “Bitcoin: A peer-to-peer electronic cash system” was published.
 - Key idea: A ledger via some complex math records the transactions.
 - Proof of work (PoW) to validate transactions on a blockchain.
 - Individuals around the world can come to a consensus without relying on any third-party as an intermediate for trust.



Lecture 13

The CIA Triad

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last classes (Reliability)
- The CIA triad
 - Confidentiality
 - Integrity
 - Availability
- Digital Signature
 - Birthday problem
 - Hash function

SRE vs DevOps

SRE focuses on reliability

- Operating and monitoring
- Incident response
- Capacity planning

DevOps focuses on delivery

- CI/CD
- Release automation
- Configuration management

Note: SRE is to DevOps is what Scrum is to Agile; one implementation of a philosophy and principles, but not a 100% subset.

Fault tolerance techniques

- Single version techniques
 - Exception handling
- Multiple version techniques: use two or more versions of the same software module to achieve fault tolerance through software redundancy.
 - Recovery blocks
 - N-version programming
 - N self-checking programming
- Multiple data representation techniques (not covered in class)
 - Retry blocks
 - N-copy programming

Fault removal

- Why? Despite fault tolerance efforts, not all faults are tolerated, so we need fault removal.
 - To keep in mind: fault tolerance is a must-have property for safety-critical systems.
 - But for software that we use everyday, we want to remove faults to improve user experience.
- Improving **system dependability** by:
 - Detecting existing faults through software verification and validation
 - Eliminating the detected faults

Fault removal as two concepts:

1. as a solution to improve availability, and thus dependability
2. as a solution for faults that affect user's daily activities (not tolerated)

Fault localization

Fault localization techniques have been proposed to assist in locating and understanding the root causes of faults.

Why? Fault localization as a debugging technique to maintain the system:

- Pinpoint the location to fix in the code
- Recover fast from bugs, and reduce its impact on the users
- Reduce manual debugging efforts, more time for new feature



Information redundancy

Information redundancy tolerate faults by adding information to the original data.

- Tolerate faults by means of coding
- Avoid unwanted information changes
- E.g., information loss during data storage or transmission

The term “coding” is used in the context
of communication and data storage

Byzantine fault tolerance

Byzantine fault tolerance: the ability of a system to continue functioning even if some of its nodes/components fail randomly or act maliciously.

- The system can keep functioning even if certain components stop working
- For example, safety-critical systems need to be able to work when if some of its components fail
 - E.g., train, airplane, spacecraft
 - E.g., heart-lung machine, robotic surgery

Schedule for today

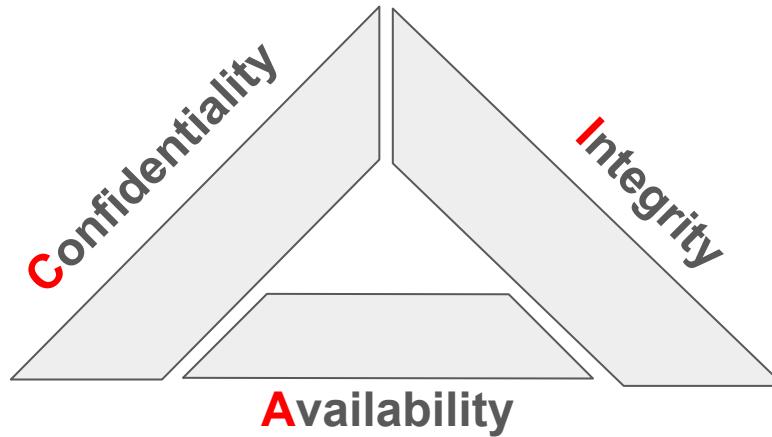
- Key concepts from last classes (Reliability)
- The CIA triad
 - Confidentiality
 - Integrity
 - Availability
- Digital Signature
 - Birthday problem
 - Hash function

The CIA triad

CIA: Confidentiality, integrity and availability

- Guiding model in information security
- Also known as the AIC Triad

Despite the acronym, this is not related to the Central Intelligence Agency in any way.



The CIA triad

The CIA triad:

- Confidentiality: only the authorized user can **access** particular resources
- Integrity: ensure data are **trustworthy, complete, and have not been modified** by unauthorized parties
- Availability: ensure data are **accessible** when needed

Both security and reliability are concerned with these three concepts

- Difference: the presence or lack of a malicious adversary

Confidentiality

Confidentiality

- Only the authorized user can access particular resources

Methods to achieve confidentiality:

- Encryption: encoding/decoding of the plaintext
 - E.g., Symmetric/asymmetric encryption
- Access controls: restricted access
 - E.g., Our library website
- Two-factor authentication: additional security checks
 - E.g., Mobile authentication for faculty and staff



Integrity

Integrity

- Ensure data are trustworthy, complete, and have not been modified by unauthorized parties

Methods to achieve integrity:

- Hashing: transforms any given data into fixed-size values
 - E.g., Comparing stored data
- Digital signature: verifies the authenticity of data
 - E.g., Emails, software application codes

Availability

Availability

- Ensure data are accessible when needed

Methods to achieve integrity:

- Fault tolerance
 - E.g., Airplane, heart-lung machine
- Redundancy
 - E.g., Hardware duplicates
- Testing and maintenance
 - E.g., Function or structural tests

The CIA triad

Confidentiality

Integrity

Availability



A hacker intercepts wireless traffic from users

A push-to-talk microphone stuck in the transmit position

A hacker modifies the messages from users

Server overload

Trade-off: Redundancy

Reliability assumption: some things will go wrong at some point.

- The primary risks are non malicious in nature
- In the absence of an adversary, systems often **fail safely (open)**

Therefore, a reliable design should consider redundancy.

- E.g., an electronic lock is designed to remain open in case of power failure, to allow safe exit through the door.

However, redundancy increases the attack surface.

- An adversary only needs find one vulnerability in one path to be successful.

Trade-off: Incident management

Security assumption: an adversary can make things go wrong at any point.

- The risks are malicious in nature
- The presence of an adversary necessitates the systems to **fail securely (remain closed)**

Therefore, a secure design should consider incident management.

- E.g., a door fails securely and remains closed when not powered.

However, incident management shares information on a need-to-know basis.

- A larger volume of logs may be useful to reduce the time to recovery, and thus keeping the system reliable.
- But, these logs can also be a valuable target for an attacker.

Schedule for today

- Key concepts from last classes (Reliability)
- The CIA triad
 - Confidentiality
 - Integrity
 - Availability
- Digital Signature
 - Birthday problem
 - Hash function

Birthday problem

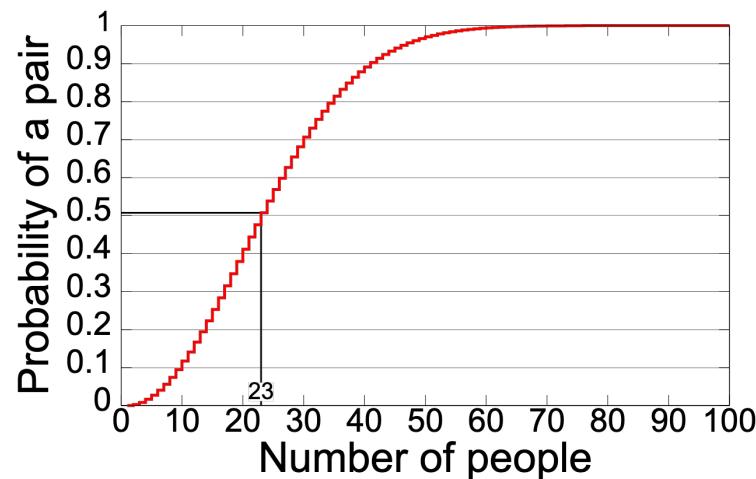
Let P be the probability that two people not having their birthday on the same day.

- $P_1 = \frac{365}{365}$
- $P_2 = \frac{365}{365} \times \frac{364}{365}$
- $P_3 = \frac{365}{365} \times \frac{364}{365} \times \frac{363}{365}$
- $P_n = \frac{365 \times 364 \times \dots \times (365 - n + 1)}{365^n}$

Birthday problem

Let $Q = (1 - P)$, the probability that two people have the same birthday, then:

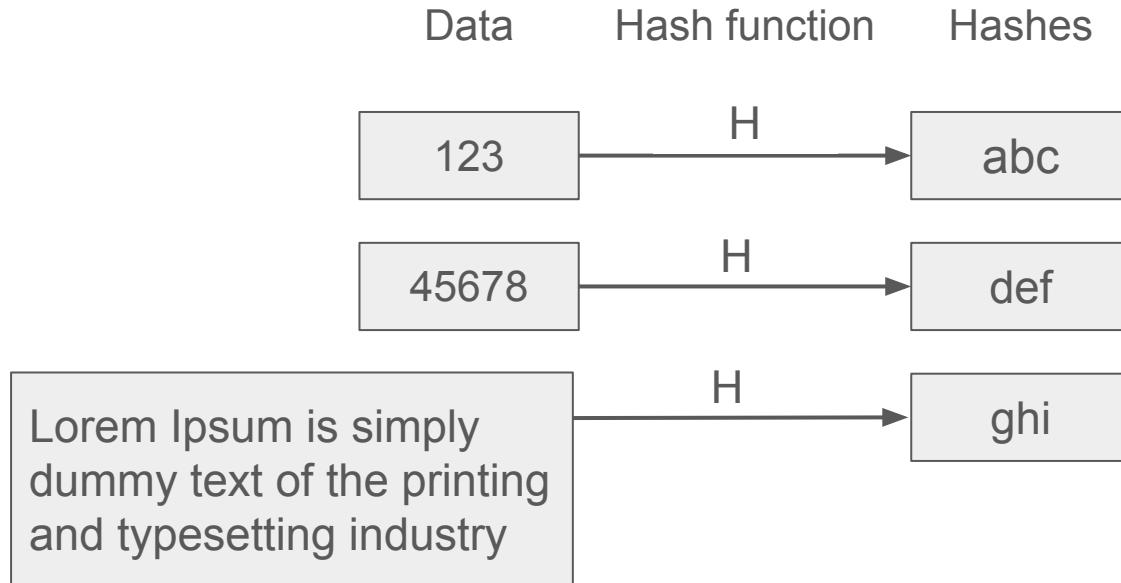
- $P_{10} = 88.31\%$ $Q = 1 - P = 11.69\%$
- $P_{25} = 43.13\%$ $Q = 1 - P = 56.87\%$
- $P_{50} = 2.96\%$ $Q = 1 - P = 97.04\%$
- $P_{78} = 00.01\%$ $Q = 1 - P = 99.99\%$



Hash function

Hash function: transforms any given data into fixed-size values

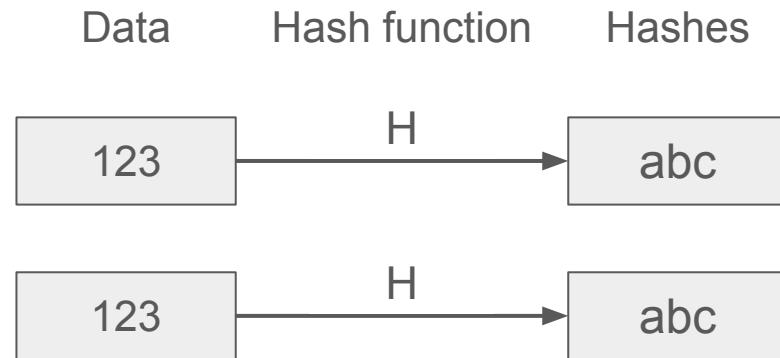
For example



Hash function

Hash function is **deterministic**:

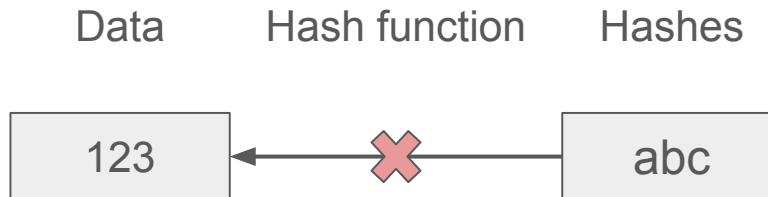
- It always produces the same hash given the same input
- Verifies the integrity of the data
 - E.g., to compare that a downloaded file is the same as the original file



Hash function

Hash function is an one-way function (**irreversible**):

- It is impossible to recover the original data from the hash
- Technically possible, but the computational power needed makes it infeasible



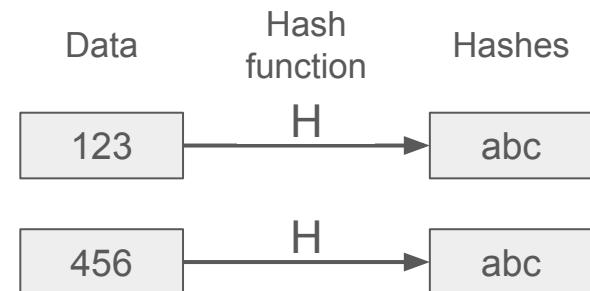
Hash collision

Hash function is unaware of its set of inputs.

- Performs arithmetic operations on the input passed to it
- Produces hashes of a fixed size

Problem? Hash collision is likely to happen.

- Hash collision happens when two pieces of data in a hash table share the same hash value
- Similar to the idea of birthday collision



Hash collision

Given a hash function that produces n hashes, it requires $1.2\sqrt{n}$ distinct values for the probability of a hash collision to be larger than 50%.

For example:

- A hash function with 100 hashes, we only need to hash 12 values to get 50% chance of having a hash collision.
- A hash function with 10,000 hashes, we only need 120 values.

Digital signature

Digital signatures verify the authenticity

- Detect the identity of the sender/signer

Digital signatures check the integrity

- Verify that the message was not changed

Digital signatures ensure non-repudiation

- Verify that the signature is not fake

Basic of asymmetric encryption

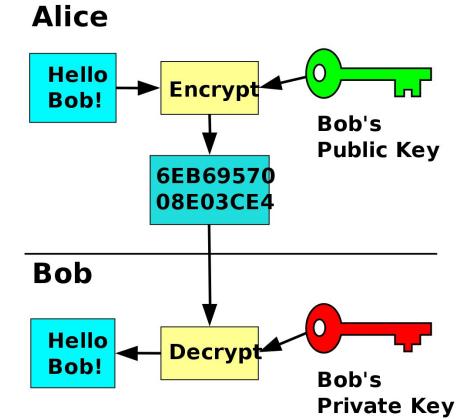
Asymmetric encryption uses a pair of related keys for encryption and decryption:

- One key for encryption, the other for decryption
- The keys are mathematically related
- File encrypted with one key can only be decrypted with the other key

Basic of asymmetric encryption

When Alice and Bob need to communicate, they publish one of their keys (the public key)

- The public key (made public) is used for encryption
 - While the private key (kept private) is used for decryption
 - We cannot derive the private key from the public key.
- ... more later



Digital signature

How is hashing used in digital signatures?

- To create a digital signature, the sender first applies a hash function to the message, then encrypts the hash with his/her private key.
- To verify the signature, the receiver decrypts the hash with the (sender's) public key, and compares it to the hash of the message. If they match, the signature is valid.

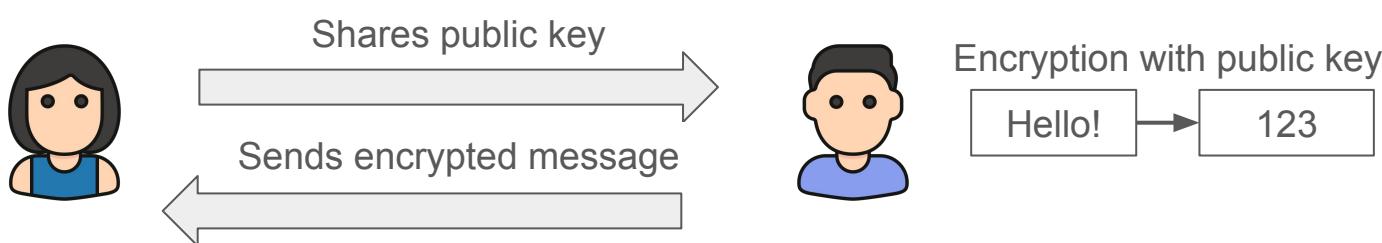
Public key: anyone can use to verify the signature

Private key: only the sender knows

Basic asymmetric encryption

Alice needs to talk to Bob

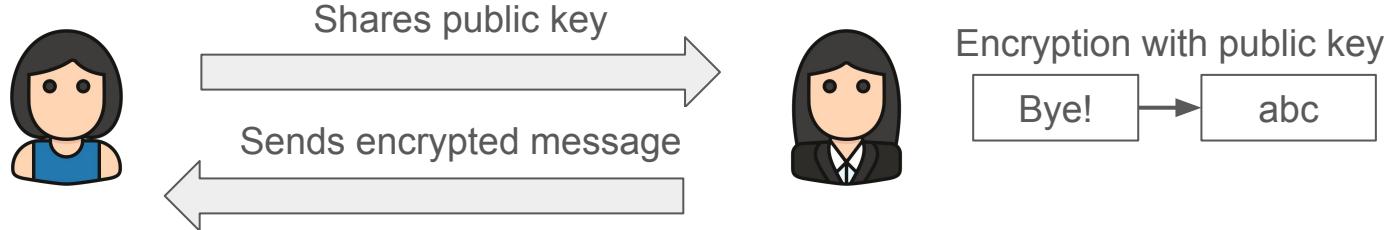
- Alice shares her public key.
- Bob uses the public key to encrypt his message, and sends back the encrypted message “123” back to Alice.
- Alice uses her private key to open the message.



Basic asymmetric encryption

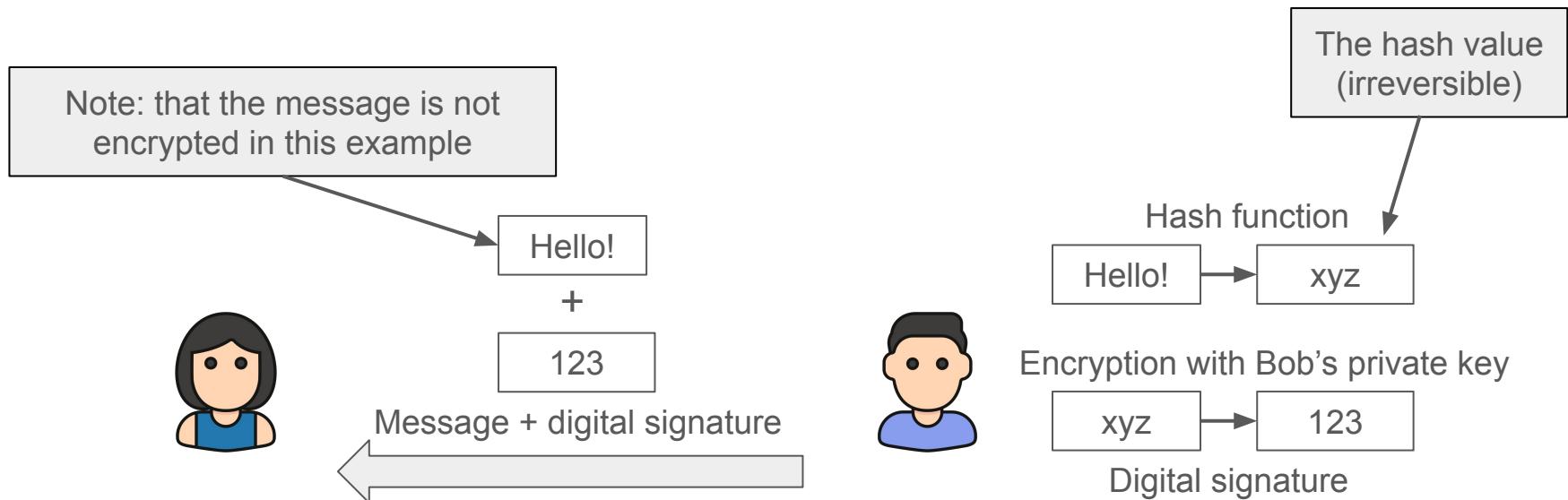
What happens if there is a third person, Eve:

- Alice shares her public key.
- Eve pretends to be Bob and sends back the message “Bye!” back to Alice.
- Alice uses her private key to open the message.
- Alice: “Does it come from Bob or someone else?”



With digital signature

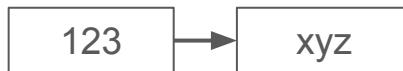
- Bob converts the message “Hello” into a hash “xyz”.
- Bob encrypts the hash with his private key, and sends it back together with the message “Hello” to Alice.



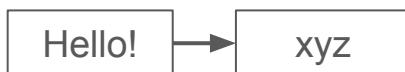
With digital signature

- Alice uses Bob's public key to decrypt the message.
- Alice creates a hash of the message by herself.
- Alice verifies whether the hash matches what Bob sends.

Decryption with Bob's public key



Hash function



Compare hash



Hello!

+

123

Message + digital signature



Digital signature



What happens if Eve pretends to be Bob?

Lecture 14

Hash Function and Digital Signature

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

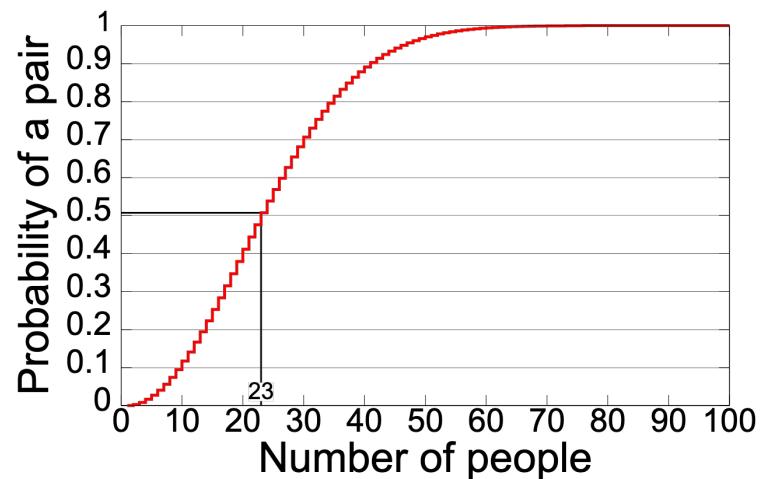
Schedule for today

- Key concepts from last class
- Digital Signature
 - Irreversibility of hash function
 - Why is hash collision a problem
- Hash function: SHA256
 - Hash collision
 - Applications in practice
- Next class: salting
- TODOs

Birthday problem

Let $Q = (1 - P)$, the probability that two people have the same birthday, then:

- $P_{10} = 88.31\%$ $Q = 1 - P = 11.69\%$
- $P_{25} = 43.13\%$ $Q = 1 - P = 56.87\%$
- $P_{50} = 2.96\%$ $Q = 1 - P = 97.04\%$
- $P_{78} = 00.01\%$ $Q = 1 - P = 99.99\%$



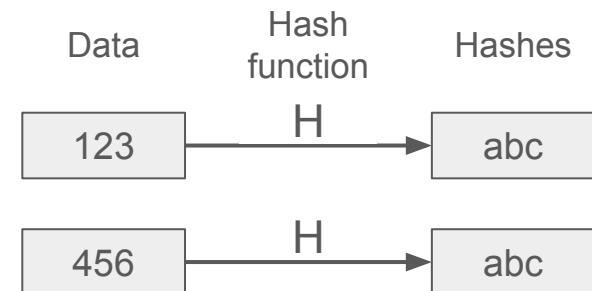
Hash collision

Hash function is unaware of its set of inputs.

- Performs arithmetic operations on the input passed to it
- Produces hashes of a fixed size

Problem? Hash collision is likely to happen.

- Hash collision happens when two pieces of data in a hash table share the same hash value
- Similar to the idea of birthday collision



Digital signature

Digital signatures verify the authenticity

- Detect the identity of the sender/signer

Digital signatures check the integrity

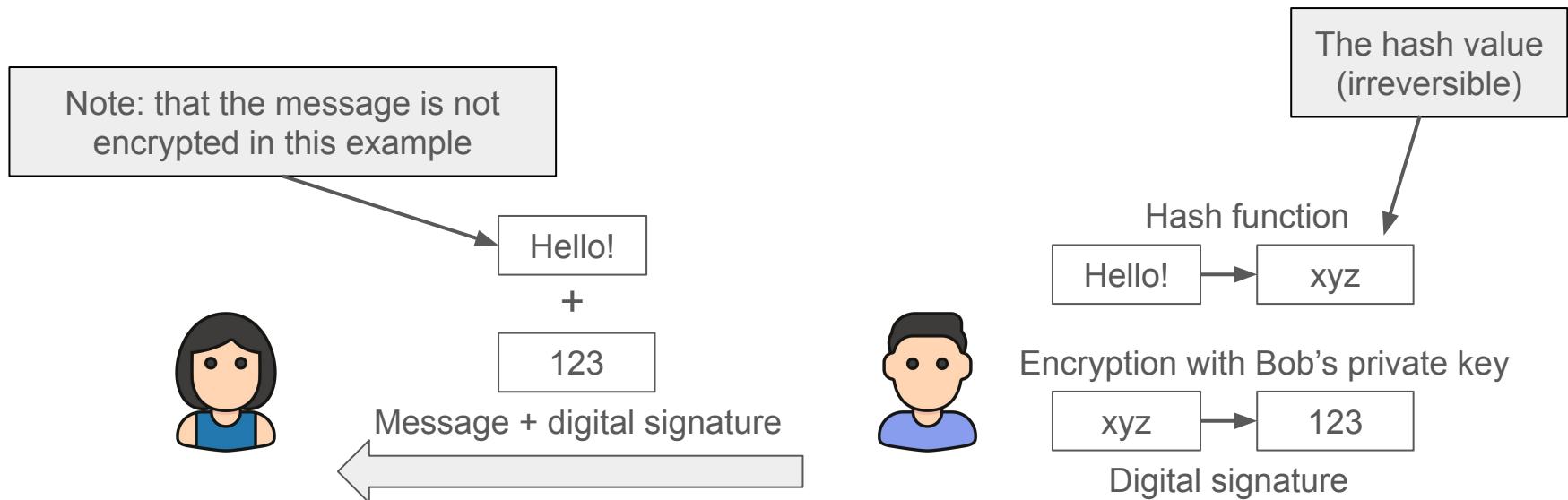
- Verify that the message was not changed

Digital signatures ensure non-repudiation

- Verify that the signature is not fake

With digital signature

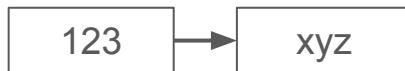
- Bob converts the message “Hello” into a hash “xyz”.
- Bob encrypts the hash with his private key, and sends it back together with the message “Hello” to Alice.



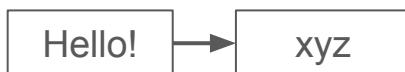
With digital signature

- Alice uses Bob's public key to decrypt the message.
- Alice creates a hash of the message by herself.
- Alice verifies whether the hash matches what Bob sends.

Decryption with Bob's public key



Hash function



Compare hash



Hello!

+

123

Message + digital signature



Digital signature



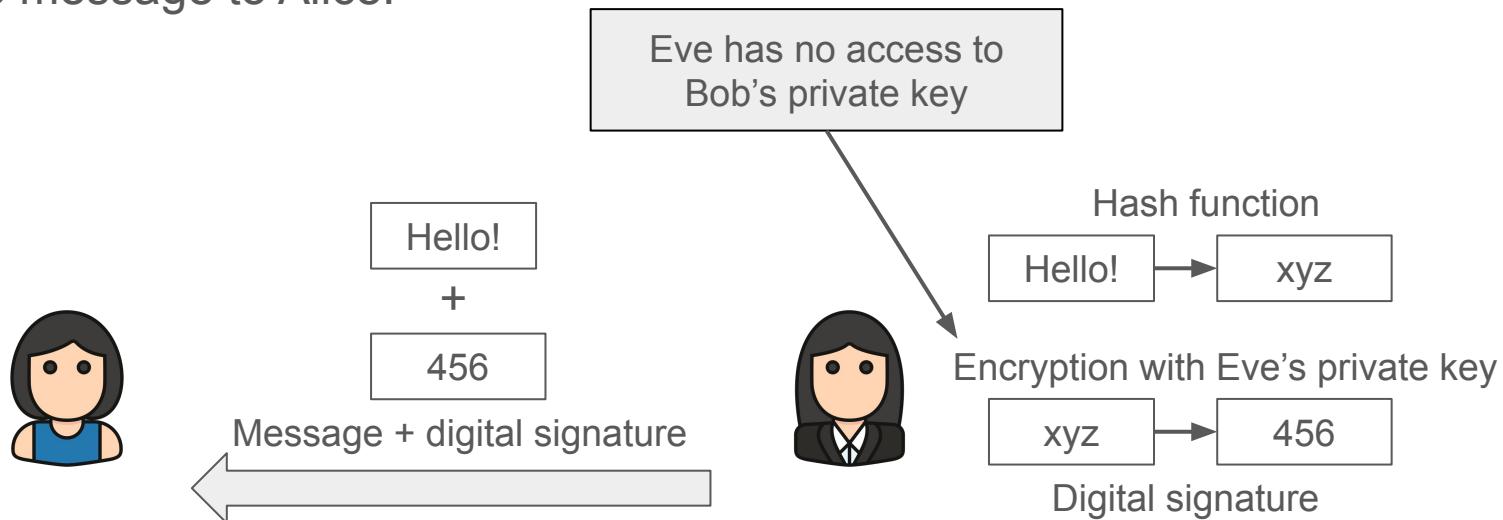
What happens if Eve pretends to be Bob?

Digital signature



What happens if Eve pretends to be Bob?

- Eve converts the message “Hello!” into the hash “xyz”.
- Eve encrypts the hash with her private key, and sends it back together with the message to Alice.



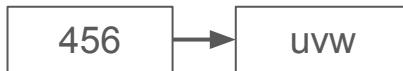
Digital signature



What happens if Eve pretends to be Bob?

- Alice uses Bob's public key to decrypt the message.
- At the same time, Alice also converts the message "Hello!" into its hash.
- Alice verifies whether the hash values match.

Decryption with Bob's public key



Hash function



Compare hash



Hello!

+

456

Message + digital signature



Irreversibility

Are hash functions still secure if they are publicly available?

- Publicly available as in:
 - Which hash function is used?
 - How did we implement it?
- Yes, because hash functions are **irreversible**.



The irreversibility is similar to the birthday problem:

- Easy to guess someone's birthday
- Hard to tell who is having the birthday

Analogous to jigsaw puzzle

Analogy to jigsaw puzzles

- Data = a blank sheet of paper
- Hash function = cutting the paper into one million pieces of jigsaw puzzle and shuffling it
- Hash = pieces of jigsaw puzzle

We can tell how the hash function works, but it is impossible to transform the hash value back to the data.

Why is hash collision a problem?

Hash function is **unaware** of its set of inputs.

- Produces hashes of a fixed size
- Problem? Hash collision is likely to happen

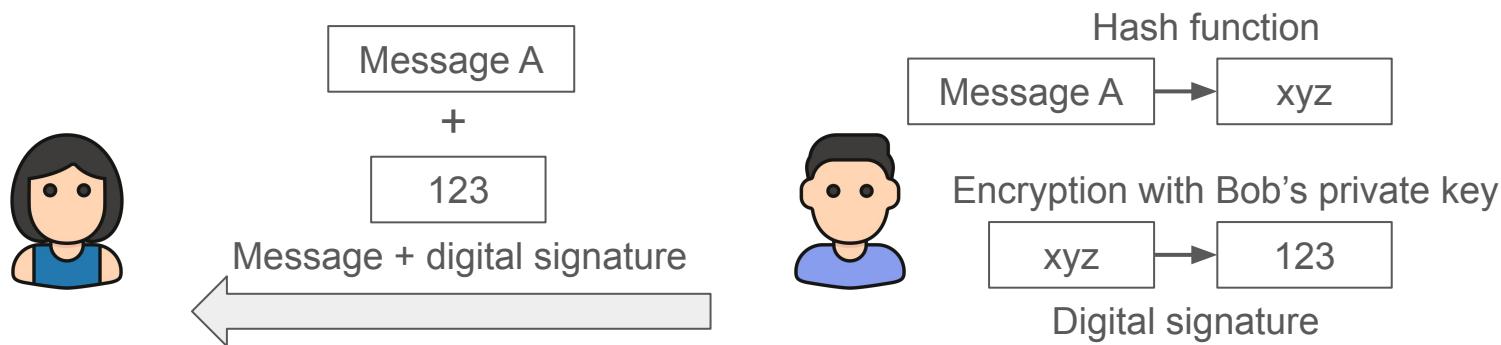
Hash collision makes systems vulnerable to **collision attack**.

- Collision attack tries to find two inputs that produce the same hash value
- Applications of collision attack
 - Digital signature: two messages with the same hash value
 - File integrity checks: two files with the same hash value

Example of collision attack: digital signature

Collision attack, scenario 1:

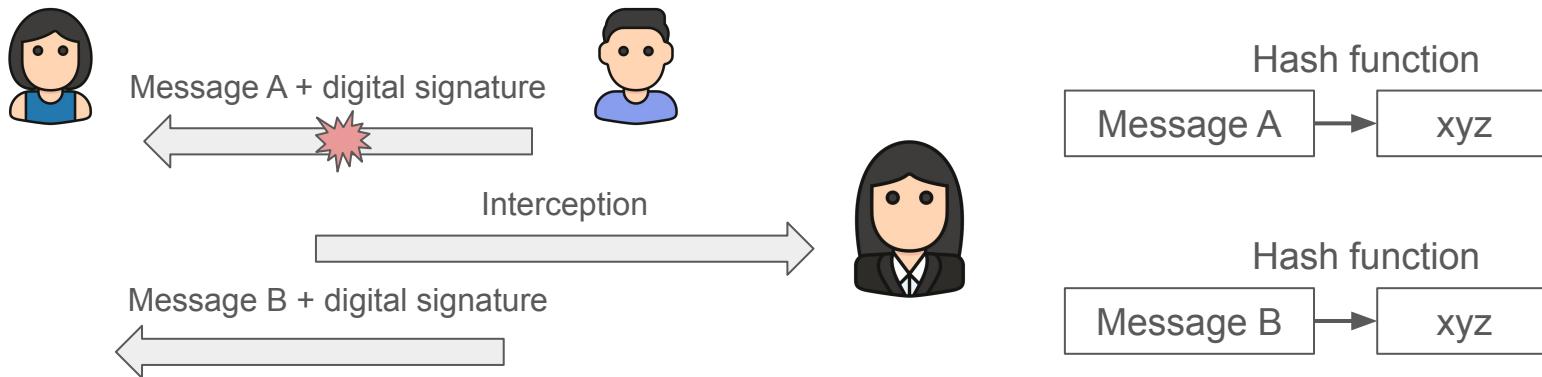
- Bob sends a message A to Alice with a digital signature.



Example of collision attack: digital signature

Collision attack, scenario 1:

- Eve intercepts the message.
- Eve changes message A into message B that produces the same hash value
- Eve sends the changed message together with Bob's signature to Alice.



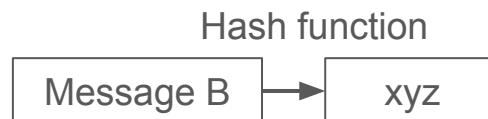
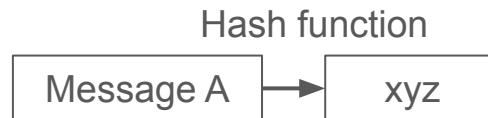
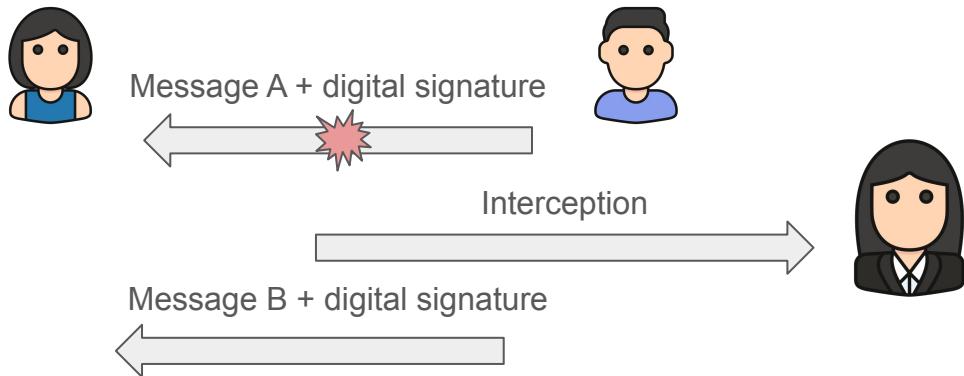
Example of collision attack: digital signature



Collision attack, scenario 1:

- Eve intercepts the message.
- Eve changes message A into message B that produces the same hash value
- Eve sends the changed message together with Bob's signature to Alice.

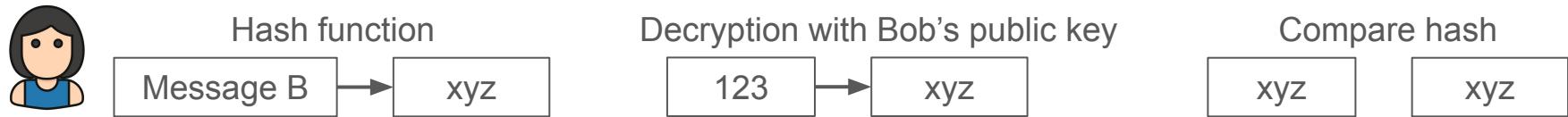
What problem can this cause?



Example of collision attack: digital signature

Collision attack, scenario 1:

- Alice receives a message B with Bob's digital signature.
- Alice: "The hashes match, it must be coming from Bob".



Hash collision presents a threat to **digital signature**.

- Despite the message was changed, its hash value matches
- Integrity of the message is broken: changed message

Example of collision attack: integrity checks

Collision attack, scenario 2:

- Eve shares a file A online that has the same hash as another malicious program B.
- Before downloading the file, Bob asks for the hash to verify the file's integrity.
- Eve provides the malicious file B instead.
- Bob: “The hashes match, it must be the same file”.

Hash collision presents a threat to **file integrity checks**.

- Despite the files are different, their hashes match
- Integrity of the file is broken: changed file

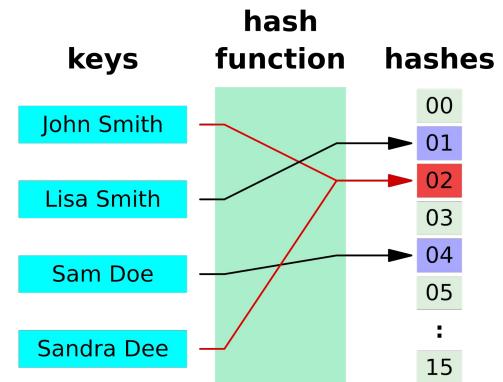
Hash collisions are unavoidable

To note that it is impossible to design a hash function that avoid collisions.

- Hash function: converts an input from a large domain to a smaller domain
- Hash collisions are unavoidable by nature
- The goal is to minimize collisions, not eliminate them

A good hash function must satisfies two properties:

- Fast hashing
- Minimal collisions



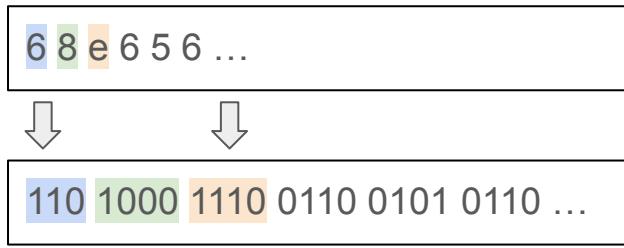
Schedule for today

- Key concepts from last class
- Digital Signature
 - Irreversibility of hash function
 - Why is hash collision a problem
- Hash function: SHA256
 - Hash collision
 - Applications in practice
- Next class: salting
- TODOs

Example of hash function: SHA256

SHA256 hash is hash function that can generates unique 256-bit hashes.

- Developed by US government's National Security Agency (NSA) in 2001
- Part of the SHA-2 (Secure Hash Algorithm 2)
- Produces hash of 64 hexadecimal characters / 256 bits
 - The output is in fact in binary, but hexadecimal is used to visually represent the hash value.



Decimal	HEX	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	a	1010
11	b	1011
12	c	1100
13	d	1101
14	e	1110
15	f	1111

Example of SHA256

Input 1: “Hi ECE 422”

SHA256 hash:

```
911fe59b33e0cf049ba953138c05178c9ffd4e57a0bd43be4c88cbf39dd7959a
```

Input 2: “Hi ECE422”

SHA256 hash:

```
f7046b0935cab27f82cd40e4c2ff6a422239d11e6f7c60da9f3d82f1b13099d0
```

...try it by yourself: [SHA256 online generation tool](#)

Hash collision in SHA256

Theory: Given a hash function that produces n hashes, it requires $1.2\sqrt{n}$ distinct values for the probability of a hash collision to be larger than 50%.

In practice: SHA256 produces 2^{256} hashes.

- 256 bits, each bit presents the possibility between 0 and 1.
- $2^{256} = (2^{32})^8 = (4.3 \text{ billions})^8 = (1.16 \times 10^{77})$
- The probability of a hash collision in SHA256 is 1 out of $(4.3 \text{ billions})^8$

We need to test at least $1.2 * (4.3 \text{ billions})^4$ hashes for a 50% chance of a collision.

SHA256 in practice

In practice, SHA256 has many different applications.

Example 1: Verify the downloaded file

- E.g., [sha256sum](#) for Linux distributions
- Compare the hashes for integrity
- Check digital signature for authenticity and non-repudiation properties

On reliability: Data loss during transmission may be another reason for file integrity check

First open a terminal and go to the correct directory to check a downloaded **iso** file:

```
cd download_directory
```

Then run the following command from within the download directory.

```
sha256sum ubuntu-9.10-dvd-i386.iso
```

sha256sum should then print out a single line after calculating the hash:

```
c01b39c7a35ccc3b081a3e83d2c71fa9a767ebfeb45c69f08e17dfe3ef375a7b *ubuntu-9.10-dvd-i386.iso
```

Compare the hash (the alphanumeric string on left) that your machine calculated with the corresponding hash in the SHA256SUMS file.

SHA256 in practice

Example 2: Password storage

- E.g., storing password as a (salted) hash in database systems
- When a user tries to login with the password, we compare the hashes
- The actual password is never stored anywhere

Next class: salting

Salting is a technique to protect passwords stored in databases by adding characters before hashing.

- Every password gets a random salt
- Extends the length of the original password
- Every hash value is different
- The salt is stored with the password

While salting does not stop the reverse-engineering, it slows down the brute force process.

TODOs

- Final report due Friday, February 9, 23:59 MST
 - One submission per team on eClass
 - Make sure to commit all your files on GitHub before the final report deadline
- Sign up for the demo time slots before February 10 (19 teams signed up)
 - Each demo is expected to be 10-15 minutes
 - All group members must attend
 - One booking per team on eClass
- Demo sessions will be held in DICE 11-251
 - Review the demo guide
 - Please be on time

Lecture 15

Authentication

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Authentication
 - Password-based authentication
 - Magic links
 - SMS-based authentication
 - Authenticator apps
 - TOTP
 - HOTP
 - Biometric authentication
 - Multi-factor authentication

The CIA triad

The CIA triad:

- Confidentiality: only the authorized user can **access** particular resources
- Integrity: ensure data are **trustworthy, complete, and have not been modified** by unauthorized parties
- Availability: ensure data are **accessible** when needed

Both security and reliability are concerned with these three concepts

- Difference: the presence or lack of a malicious adversary

Integrity

Integrity

- Ensure data are trustworthy, complete, and have not been modified by unauthorized parties

Methods to achieve integrity:

- **Hashing**: transforms any given data into fixed-size values
 - E.g., Comparing stored data
- **Digital signature**: verifies the authenticity of data
 - E.g., Emails, software application codes

Hash function

Hash function: transforms any given data into fixed-size values

- Deterministic
 - Same input = same hash value
- Irreversible
 - One-way function (input to hash)
 - The data is secure even if the hash function is public

Problem: hash collision may happen

- Unavoidable by nature
- More possible inputs than outputs

Applications of hash function

- Digital signature
 - Creating a digital signature = Hash of the message + encryption with private key
- File integrity check
 - Compare hashes to verify it is the right file
 - E.g., verify file download
- Password storage
 - Store passwords as hashes
 - Implication 1: actual password is hidden
 - Implication 2: same password is stored as different hashes

Workaround for hash function?

- Password storage
 - Potential problem: reverse-engineering the password (e.g., by brute force)
- Brute force solution: [SHA256 generation tool](#), [SHA256 database](#)
 - Input 1: ECE422!(hello*@
 - Hash: 53ffef3c775a544b9cb5866932d74084919d279f0cbab0687d11b53d1df8900e
 - Input 2: ECE422
 - Hash: b3af9c50da07cc8f7f7ed00f86b2c6ae7e41c75e5c84dce9b70c6ac8cf8454cc

Hash: 53ffef3c775a544b9cb5866932d74084919d279f0cb
Type: auto

decrypt **Encrypt**

Result:
Not Found, it is being cracked by our background system.
Please wait up to 5 days. A notification email will be sent to you when it is cracked successful , otherwise it is cracked failure.

Hash: b3af9c50da07cc8f7f7ed00f86b2c6ae7e41c75e5c8
Type: auto

decrypt **Encrypt**

Result:
Found.But this is a payment record. [Purchase](#)

Workaround for hash function?

Note that hash function **does not encrypt** the data, it generates a hash using the input as a seed instead.

- Hashing: one-way function
- Encryption: two-way function

Hash: 53ffef3c775a544b9cb5866932d74084919d279f0cb
Type: auto

decrypt **Encrypt**

Result:
Not Found, it is being cracked by our background system.
Please wait up to 5 days. A notification email will be sent to you when it is cracked successful , otherwise it is cracked failure.

Hash: b3af9c50da07cc8f7f7ed00f86b2c6ae7e41c75e5c8
Type: auto

decrypt **Encrypt**

Result:
Found.But this is a payment record. [Purchase](#)

Example of hash function: SHA256

Example of hashing: SHA256

- Hash of 64 hexadecimal characters / 256-bit hashes
- Produces a total of 2^{256} hashes
- Hash collision: 1 out (4.3 billions)⁸

Input 1: “Hi ECE 422”

SHA256 hash:

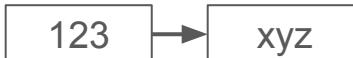
```
911fe59b33e0cf049ba953138c05178c9ffd4  
e57a0bd43be4c88cbf39dd7959a
```

Digital signature

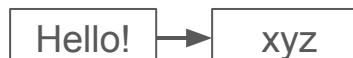
Digital signature verifies the authenticity of the sender through hashing and decryption. For example:

- Alice uses Bob's public key to decrypt the message.
- Alice creates a hash of the message by herself.
- Alice verifies whether the hash matches what Bob sends.

Decryption with Bob's public key



Hash function



Compare hash



Confidentiality

Confidentiality

- Only the authorized user can access particular resources

Methods to achieve confidentiality:

- Encryption: encoding/decoding of the plaintext
 - E.g., Symmetric/asymmetric encryption
- Access controls: restricted access
 - E.g., Our library website
- **Authentication:** credentials check
 - E.g., Mobile authentication for faculty and staff



Schedule for today

- Key concepts from last class
- Authentication
 - Password-based authentication
 - Magic links
 - SMS-based authentication
 - Authenticator apps
 - TOTP
 - HOTP
 - Biometric authentication
 - Multi-factor authentication

Identification vs authentication vs authorization

Identification happens when a user claims an identity.

- Username, student ID card, CCID

Authentication provides access control for systems by checking to see if a user's credentials match the credentials in a database.

- Username + password, smart card, driver license + fingerprint
- Something the user knows/has/is

Authorization provides access to different resources based on the user identity.

- Admin privileges, student account access, library resources

Question



Q1) What happens when a user's password has been verified?

- A. Identification
- B. **Authentication**
- C. Authorization
- D. Identity verification

Authentication

- Password-based authentication
 - Username + password
- Magic links
 - Links through email or mobile device
- SMS-based authentication
 - Text messages
- Authenticator apps
 - Push notifications, or one-time password (OTP)
- Biometric authentication
 - Biometric data (e.g., fingerprint and face recognition)
- Multi-factor authentication (MFA)
 - Two or more independent authentication factors

Password-based authentication

In password-based authentication, the user enters the right credentials to gain access to the system.

Use case:

- Enter the email address and password in the login page
- Check against the hashed (and salted) password in the database
- If they match, the user is granted access

Security risks:

- Can be stolen or guessed
- People forget passwords

Password storage

Salting is a technique to protect passwords stored in databases by adding characters before hashing.

- Stored password = Hash (password + salt)

Example:

- Password: ECE422
- Salt: !(hello*@
- Password + salt: ECE422!(hello*@
- Hash: 53ffef3c775a544b9cb5866932d74084919d279f0cbab0687d11b53d1df8900e

Magic link

In magic link authentication, the user leverages URLs with embedded tokens to gain access to the system.

Use case:

- Enter the email address in the login page
- Receive an email with a (magic) link
- Click on the link to login

Security risks:

- As secure as the email account
- Rely on the email service providers

Example of magic link in practice

On the login page:

- The user sends the email address to the server.

On the server-side:

- The database checks if the email exists.
- The application generates a link embedding a login token and stores the token in the database.
- The application sends the user an email with the link.

Example of magic link in practice

Within the email:

- The user clicks on the link.

On the server-side:

- The application verifies the token and grants user the access.

Short message services (SMS)

In SMS authentication, the user provides a code that is sent to their phone as proof of identify.

Use case:

- Login with username and password
- An SMS code sent to the phone
- Enter the SMS code into the login interface

Security risks:

- Phishing messages
- SMS messages can be intercepted

Schedule for today

- Key concepts from last class
- Authentication
 - Password-based authentication
 - Magic links
 - SMS-based authentication
 - **Authenticator apps**
 - TOTP
 - HOTP
 - Biometric authentication
 - Multi-factor authentication

Authenticator apps

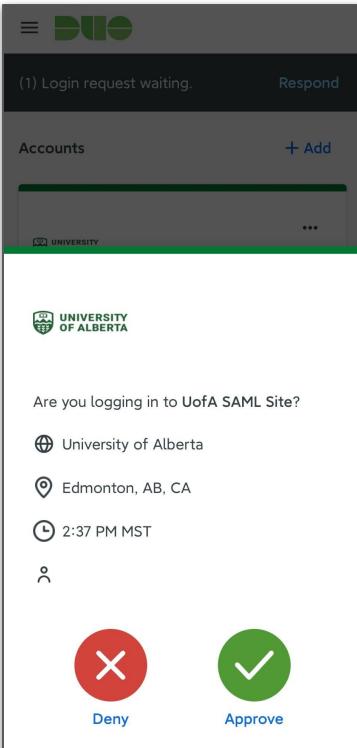
An authenticator app allows users to prove their identity through a specialized application.

- Usually on a mobile device
- Notifies users every time there is an attempt to log in
- Allows users to deny access, stopping the attacker in their tracks

Two common forms of authentication:

- Push notification
- One-time password (OTP)

Push notification



In push notification, the user approves a notification that is sent to their mobile device as proof of identify.

Use case:

- Login with username and password
- A notification sent to the mobile device
- Approve in the application

Security risks:

- Applications can be vulnerable
- Third-party applications can have access

One-time password (OTP)

Authenticator apps can also leverage one-time password to verify the identity of the user.

- Generally used within 2FA and MFA systems

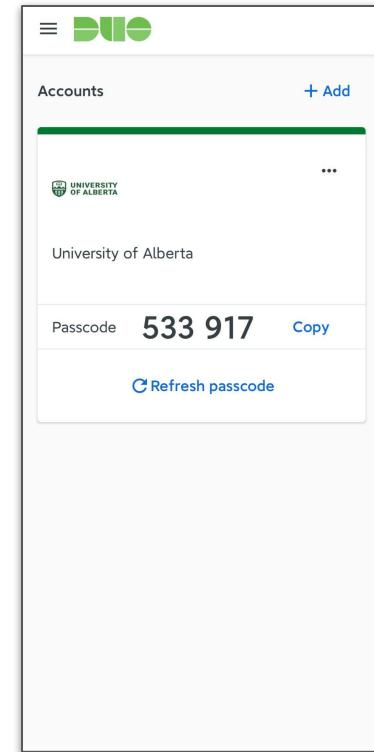
Example of OTP:

- Time-based one-time passwords (TOTP)
 - Passcode valid within a set interval of time
 - Input: secret key + time
- HMAC-based one-time passwords (HOTP)
 - Passcode that can only be used once
 - Input: secret key + counter

TOTP

Time-based one-time password (TOTP) uses a public algorithm to generate the one-time password.

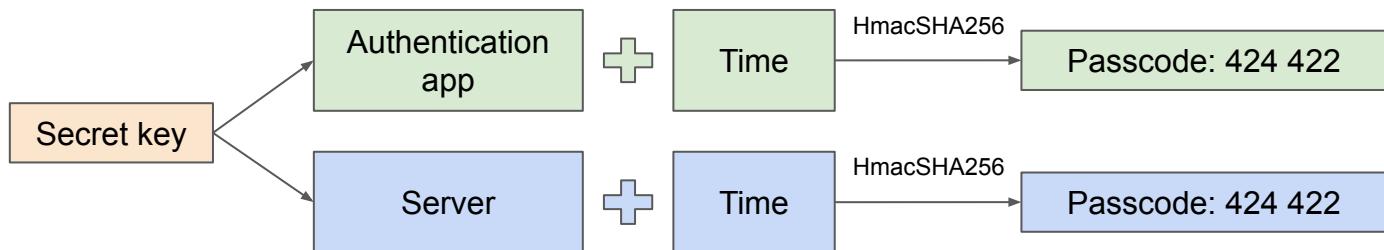
- Generate unique passcodes based on the current interval of time
- Time interval is generally 30 seconds
- No delivery of the one-time passcode is required
 - Generation algorithm shared ahead of time
- One-time passcode generated through a shared secret key and the current time



TOTP algorithm

Algorithm behind TOTP:

- Both the user application and the server generates the passcode based on the current time and secret key.
- Typical generation algorithm: HMAC-SHA-1 and HMAC-SHA-2
 - E.g., HmacSHA256, a **hash-based message authentication code (HMAC)** function
 - HMAC takes as input: (time + secret key)



Example of TOTP in practice

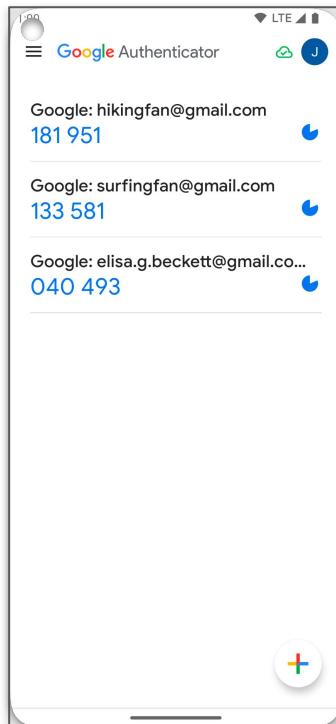
When the user installs the application:

- Time synchronization via the Network Time Protocol (NTP)
 - NTP is designed for sending time over the Internet (accurate to a few milliseconds)
- Share a unique secret key

When the user authenticates:

- The user uses the key and time to generate the passcode
- The user sends the passcode to the server
- The server computes the passcode and compares
- If the passcode matches, then the user gains access

HOTP



HMAC-based one-time passwords (HOTP) also uses a public algorithm to generate the one-time password.

- Generate unique passcodes based on the current counter
- Counter is a variable stored on the server and the application, increases each time a passcode is generated.
- No delivery of the one-time passcode is required
 - Generation algorithm shared ahead of time
- HMAC takes as input: (counter + secret key)

Example of HOTP in practice

When the user installs the application:

- Share a unique secret key

When the user authenticates:

- The user uses the key and counter to generate the passcode
- The user sends the passcode to the server
- The server computes the passcode and compares
- If the passcode matches, then the user gains access
- The server synchronizes on the counter

TOTP vs HOTP

- OTP expiration
 - TOTP only valid for a period of time until they expire
 - HOTP valid until usage, no time expiration
- Convenience
 - TOTP must be used before expiration
 - HOTP request anytime, use them later
- Security
 - TOTP is more secure, time as a moving factor
 - HOTP is vulnerable to brute force attacks

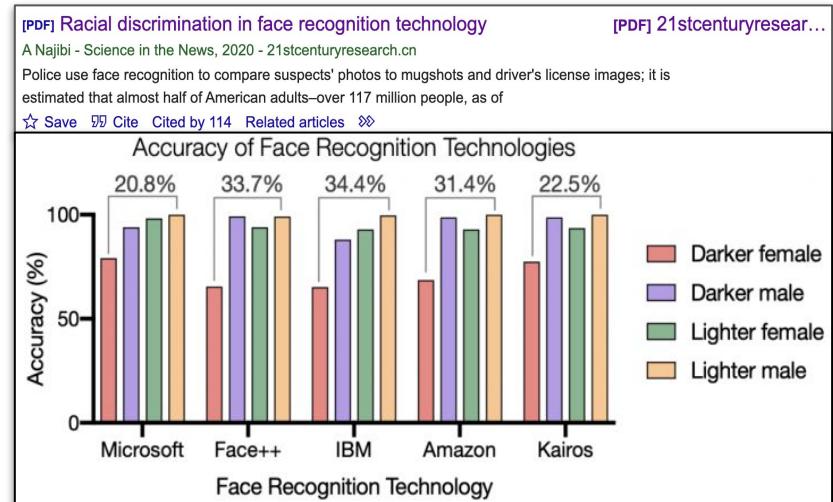
Schedule for today

- Key concepts from last class
- Authentication
 - Password-based authentication
 - Magic links
 - SMS-based authentication
 - Authenticator apps
 - TOTP
 - HOTP
 - Biometric authentication
 - Multi-factor authentication

Biometric authentication

Examples of biometric authentication:

- Facial recognition
 - Accuracy of facial recognition can depend on age, race and gender.
- Fingerprint
 - Nearly 99% accuracy
- Retina recognition
 - Used in government and military organization
- Voice recognition
 - Noisy environment can be a problem



Multi-factor authentication (MFA)

Multi-factor authentication (MFA) requires **two or more factors** to verify user's identify.

- Knowledge factor: Something only the user knows
 - Password, PIN code
- Possession factor: Something only the user has
 - Access card, key, authorized device
- Biologically factor: Something only the user is
 - Physical trait: fingerprint, retinal pattern
 - Behavioral process: voice recognition, keystroke dynamics

Multi-factor authentication (MFA)

- Location factor: Some location information the user should have
 - IP address, geolocation
- Time factor: Some time information the user should have
 - Weekdays, hours

Authentication

- Password-based authentication
 - Username + password
- Magic links
 - Links through email or text messages
- SMS-based authentication
 - Text messages
- Authenticator apps
 - Push notifications, or one-time password (OTP)
- Biometric authentication
 - Biometric data (e.g., fingerprint and face recognition)
- Multi-factor authentication (MFA)
 - Two or more independent authentication factors

Multi-factor authentication (MFA)



Q1) An example of multi-factor authentication is a username and password.

- A. True
- B. False

Q2) Which one of the following does NOT help upgrade the security of a current password-based authentication system to a multi-factor authentication system?

- A. Access card
- B. Retinal scan
- C. Authenticator apps
- D. Security questions

TODOs

- Demo sessions will be held in DICE 11-242 instead.
- Review session next Friday

Lecture 15

Access Control

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Access Control
 - Threats, vulnerabilities and attacks
 - Types of control
 - Access control lists (ACLs)
- Models of access control
 - Discretionary access control (DAC)
 - Role-based access control (RBAC)
 - Mandatory access control (MAC)
 - Attribute-based access control (ABAC)

Confidentiality

Confidentiality

- Only the authorized user can access particular resources

Methods to achieve confidentiality:

- Encryption: encoding/decoding of the plaintext
 - E.g., Symmetric/asymmetric encryption
- Access controls: restricted access
 - E.g., Our library website
- **Authentication:** credentials check
 - E.g., Mobile authentication for faculty and staff



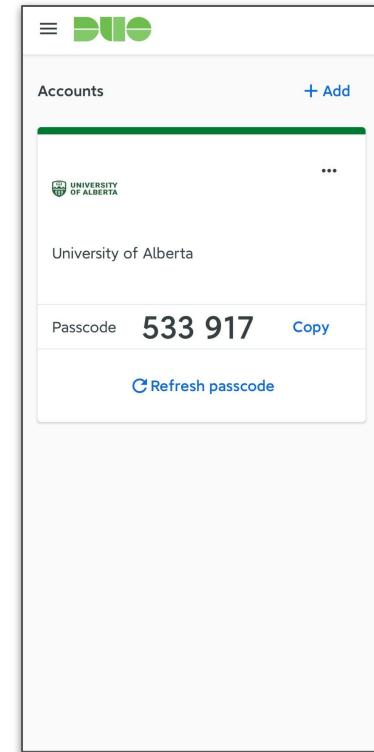
Authentication

- Password-based authentication
 - Username + password
- Magic links
 - Links through email or mobile device
- SMS-based authentication
 - Text messages
- Authenticator apps
 - Push notifications, or one-time password (OTP)
- Biometric authentication
 - Biometric data (e.g., fingerprint and face recognition)
- Multi-factor authentication (MFA)
 - Two or more independent authentication factors

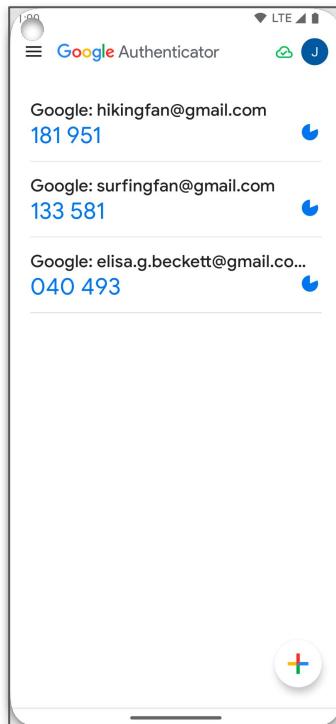
TOTP

Time-based one-time password (TOTP) uses a public algorithm to generate the one-time password.

- Generate unique passcodes based on the current interval of time
- Time interval is generally 30 seconds
- No delivery of the one-time passcode is required
 - Generation algorithm shared ahead of time
- One-time passcode generated through a shared secret key and the current time



HOTP



HMAC-based one-time passwords (HOTP) also uses a public algorithm to generate the one-time password.

- Generate unique passcodes based on the current counter
- Counter is a variable stored on the server and the application, increases each time a passcode is generated.
- No delivery of the one-time passcode is required
 - Generation algorithm shared ahead of time
- HMAC takes as input: (counter + secret key)

Multi-factor authentication (MFA)

Multi-factor authentication (MFA) requires **two or more factors** to verify user's identify.

- Knowledge factor: Something only the user knows
 - Password, PIN code
- Possession factor: Something only the user has
 - Access card, key, authorized device
- Biological factor: Something only the user is
 - Physical trait: fingerprint, retinal pattern
 - Behavioral process: voice recognition, keystroke dynamics

Schedule for today

- Key concepts from last class
- **Access Control**
 - Threats, vulnerabilities and attacks
 - Types of control
 - Access control lists (ACLs)
- Models of access control
 - Discretionary access control (DAC)
 - Role-based access control (RBAC)
 - Mandatory access control (MAC)
 - Attribute-based access control (ABAC)

Confidentiality

Confidentiality

- Only the authorized user can access particular resources

Methods to achieve confidentiality:

- Encryption: encoding/decoding of the plaintext
 - E.g., Symmetric/asymmetric encryption
- **Access controls**: restricted access
 - E.g., Our library website
- Authentication: credentials check
 - E.g., Mobile authentication for faculty and staff



Access control

Access control determines who is allowed to access certain data, apps, and resources (i.e., protects digital spaces).

Goal of access control:

- Protect confidential information
 - Customer data
 - Intellectual property
- Prevent security risks (through authentication and authorization)
 - Data exfiltration
 - Network security threats (e.g., phishing, ransomware)
- Minimize the impacts
 - Policies on access management

Access control

Assets may vary from:

- Hardware: computer systems, devices
- Software: operating system, utilities, applications
- Data: final report, class projects, emails

While hardware and software may be expensive, unique data cannot be recovered if it is lost.

Access control and authentication



Access control identifies a user based on their credentials and then, for example, authorizes the appropriate level of access once they are authenticated.

- Passwords, magic link, SMS code are all examples of credentials used to identify and authenticate a user.
- Multi-factor authentication contains an extra layer of security by having more than just one authentication factor.

Once a user's identity has been authenticated, access control policies grant specific permissions and enable the user to proceed with the resource.

Schedule for today

- Key concepts from last class
- Access Control
 - Threats, vulnerabilities and attacks
 - Types of control
 - Access control lists (ACLs)
- Models of access control
 - Discretionary access control (DAC)
 - Role-based access control (RBAC)
 - Mandatory access control (MAC)
 - Attribute-based access control (ABAC)

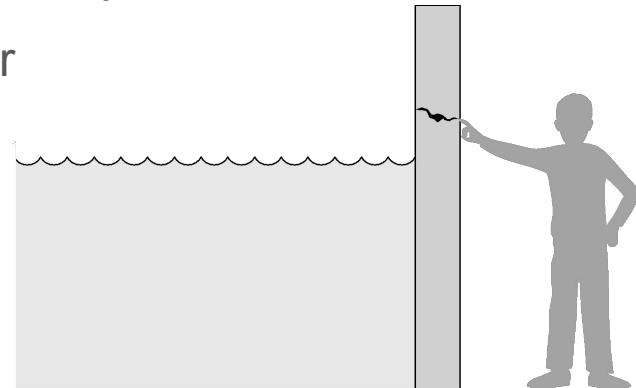
Threats, vulnerabilities and attacks

Access control is what eliminates/reduces/mitigates a vulnerability.

- Threat: an **event** that may cause harm to protected assets.
- Vulnerability: a **weakness** which could be exploited to cause harm to protected assets.
- Attack: an **action** that exploit a vulnerability in an attempt to cause harm.

Example: attempt to repair a fixture in a water reservoir

- Threat: water overflowing
- Vulnerability: crack in the reservoir
- Attack: add more water into the reservoir
- Access control: someone's finger (... for now)



Threats

Threats can be caused both by human and other sources.

- Nonhuman threats

- E.g., fires, loss of electrical power

- Human threats

- Benign (non malicious) intent

- E.g., typo, software bugs

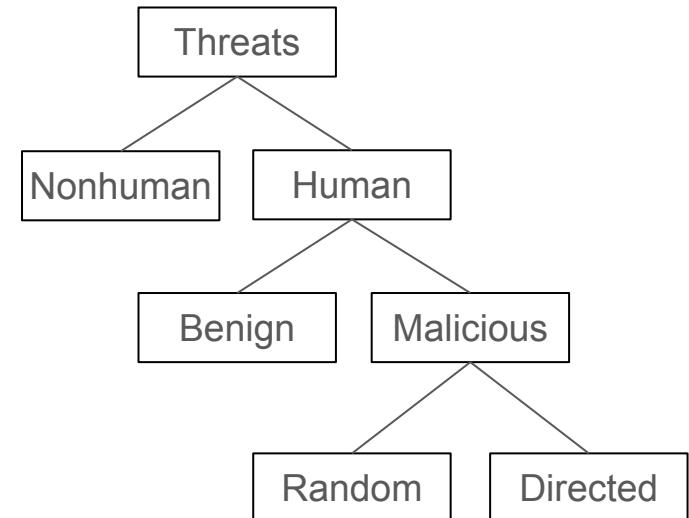
- Malicious intent

- Random (any computer/organization/individual)

- E.g., malicious code on github

- Directed (specific)

- E.g., impersonation



Types of control

There are three types of access controls:

- Physical control
- Procedural control
- Technical control

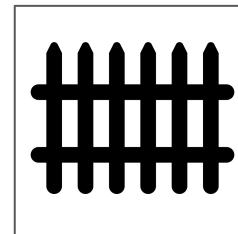
Physical control

Physical controls are implemented by physical infrastructure.

- Prevent and detect unauthorized access to physical spaces, systems or assets

Example of physical infrastructures:

- Fences
- Access cards
- CCTV



Procedural control

Procedural controls are implemented by people and practices.

- Security awareness training can also falls under procedural controls

Example of procedural control:

- Guards
- Security awareness program



Technical control

Technical controls are implemented using systems.

- Both hardware and software mechanisms are used to protect assets

Example of technical control:

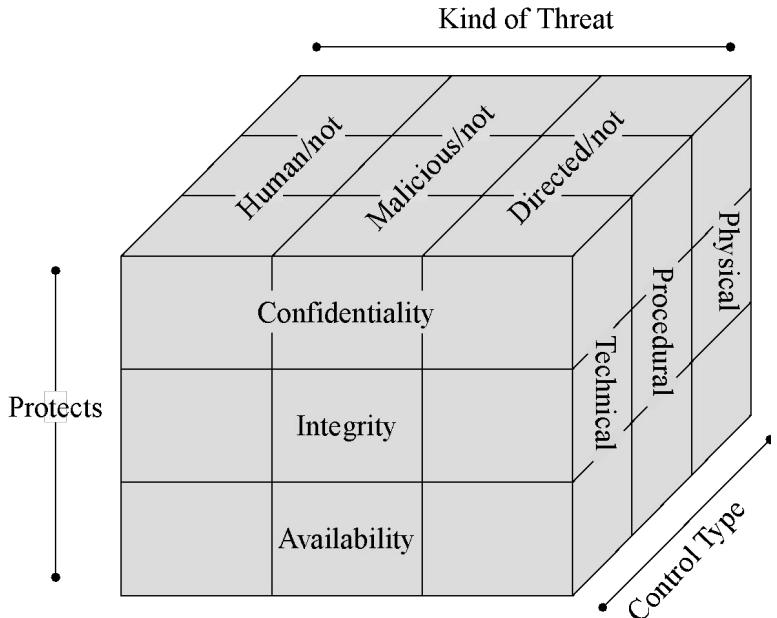
- Authentication
- Firewall
- Antivirus software

Two common forms of technical controls:

- **Access control lists** (ACLs)
- **Encryption**



Threats, vulnerabilities, and control types



- CIA are the basic security principles.
- Vulnerabilities are weaknesses in a system that affect the CIA triad.
- Threats exploit those weaknesses in the system.
- Controls protect those weaknesses from exploitation.

Schedule for today

- Key concepts from last class
- Access Control
 - Threats, vulnerabilities and attacks
 - Types of control
 - Access control lists (ACLs)
- Models of access control
 - Discretionary access control (DAC)
 - Role-based access control (RBAC)
 - Mandatory access control (MAC)
 - Attribute-based access control (ABAC)

Access control list (ACL)

An access control list is a set of instructions that either allow access to a computer environment or deny it.

- Restrict access to unauthorized users
- Control traffic by limiting the number of users

It is analogous to a guest list to a wedding.

- Only those on the lists are authorized to entries



Examples of application: filesystem ACLs

Filesystem ACLs:

- Inside an operating system (e.g., Unix-based systems)
- Inform the operating system of the access privileges that a user has to a system object (e.g., what resource, what access rights)

Example: Linux access control lists using the *getfacl* command

```
[root]# getfacl /random_folder
user::rwx
group::rwx
other::---
```

Linux filesystem:

Access class: **user**, **group**, **other**

Types of access: **read**, **write**, **execute**

```
[root]# setfacl -d -m random_user:rwx
/random_folder
```

```
[root]# getfacl /random_folder
```

user::rwx

group::rwx

other::---

default:user::rwx

default:user:random_user:rwx

Examples of application: network ACLs

Network ACLs:

- For a web server, DNS server or VPN systems
- Allow to filter requests for a single or group of IP addresses
- Protect against server attack

Example: Web access control

- Block web requests that do not meet specific conditions
- Conditions can be criteria or metrics
 - Criteria: IP address origin, country of origin, size of the request
 - Metrics: number of requests in any 10-minute period

Schedule for today

- Key concepts from last class
- Access Control
 - Threats, vulnerabilities and attacks
 - Types of control
 - Access control lists (ACLs)
- Models of access control
 - Discretionary access control (DAC)
 - Role-based access control (RBAC)
 - Mandatory access control (MAC)
 - Attribute-based access control (ABAC)

Access control models

Users receive access based on access control models.

- Different systems have different access control requirements.

There are four models of access controls:

- Discretionary access control (DAC)
- Role-based access control (RBAC)
- Mandatory access control (MAC)
- Attribute-based access control (ABAC)

Discretionary access control (DAC)

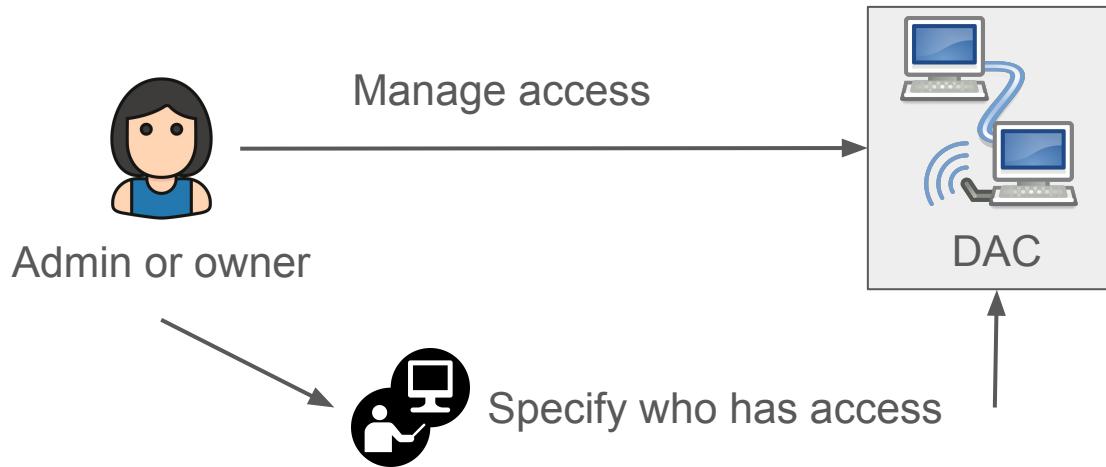
In a discretionary access control (DAC) model, each resource has an owner and owners grant access to users or user groups at their own discretion.

- Implemented using access control lists
- Case-by-case control over resources

Discretionary access control (DAC)

Administrator or owner of the resource controls the access:

- Defines a profile for each resource (i.e., resource profile)
- Updates the access control list for the profile



Benefits of DAC

- Flexible (decentralized control)
 - Owners may grant/remove access to resources at any time
- Simple (access control via ACLs)
 - An ACL defines the user privilege
- With fine-granularity
 - Access based on individual needs

However, with decentralization and simplicity:

- Difficult to tell how the resources are accessed and interconnected
- Problems: possible to have **over-privileged** users or **conflicting permissions**

Role-based access control (RBAC)

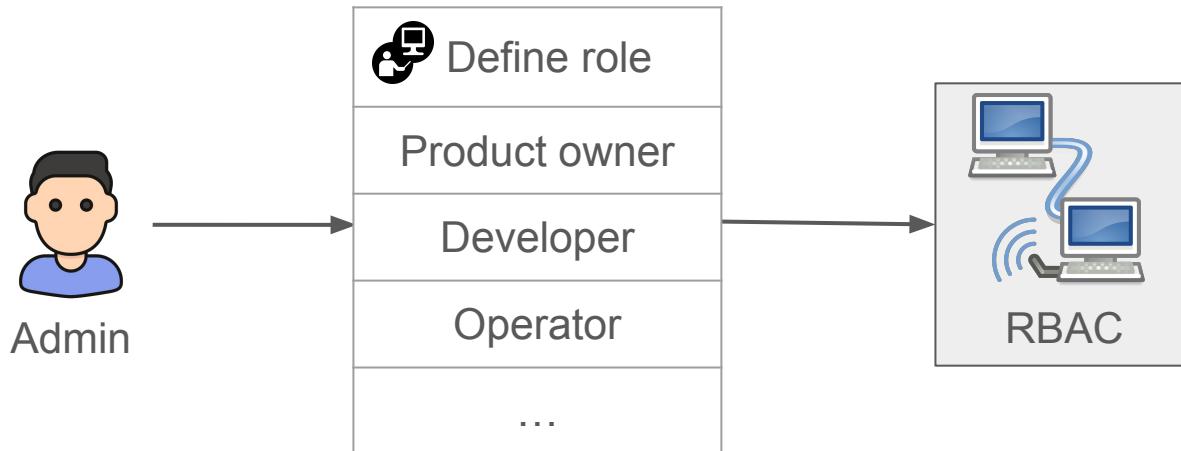
In a role-based access control (RBAC) model, the system restricts access to users based on their role(s).

- Normally within an organization
- Users are given minimum access to fulfill their job requirements

Role-based access control (RBAC)

Administrator controls the access based on the role of the user:

- Defines a role for each user
- Users can only access the resource when their role entitles the access



Benefits of RBAC

- Secure
 - Principle of least privilege
- Easy to use
 - Mapping users to data, minimal overhead in authorization management
- Compliance ready
 - Easy to handle the security and confidentiality standards

However, it requires collaboration between departments:

- Difficult to assign roles, organization is constantly growing
- Problems: new roles may **contradict to existing policies**, or **privilege creep** (unnecessary accumulation of privileges that happens during role change)

Mandatory access control (MAC)

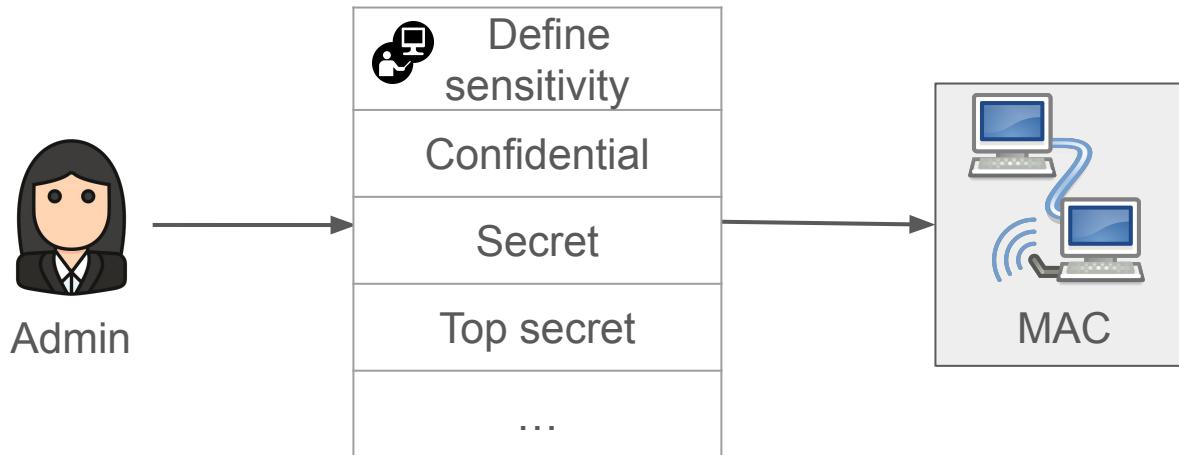
In mandatory access control (MAC), the system restricts access based on the sensitivity (or security clearance) levels of the resource.

- Commonly used in government and military contexts
- Label each resources with sensitivity levels (e.g., confidential, secret or top secret)
- Each user has a sensitivity level access
 - Defined by the administrator
 - Only the administrator can change and see the sensitivity level access

Mandatory access control (MAC)

Administrator controls the access based on the sensitivity of each resource:

- Define the sensitivity of each resource
- Users can only access the resource when their security labels entitles the access



Benefits of MAC

- Secure (most secure among the models)
 - Security levels cannot be changed
- Control over one authority
 - Centralized control
- Privacy
 - Only the administrator can see the access control (i.e., list of users and their privileges)

However, the access control at resource-level:

- Difficult to maintain the access control when data are being added and deleted constantly
- Problems: **less flexible**, thus the system **requires regular updates**

Attribute-based access control (ABAC)

In attribute-based access control (ABAC), the system restricts access based on a combination of attributes and environmental conditions.

- Attributes such as role, sensitivity level, or resource properties (e.g., ownership, types of resource)
- Environmental conditions such as time or location

Administrator or owner of the resource controls the access based on the attributes and environmental conditions:

- Assign role and sensitivity attributes to each resource
- Users can only access the resource when both their role and sensitivity label entitle them to access

Benefits of ABAC

- With fine-granularity (most precise among the models)
 - Create precise attributes without the need of additional roles
- Flexible
 - When there is a resource or user change, re-assign the attributes
- Secure
 - Security without requiring any collaboration

However, it requires time, effort and resources to implement:

- Difficult to define good attributes
- Problems: **time-consuming**, something that we implement over time

Access control models



Q1) What access control model is where an owner can assign permissions to users for resources they control?

Answer: ?

Q2) What access control model is where a user's clearance must exceed a resource's sensitivity label?

Answer: ?

Q3) For access control, what does authentication mean?

Answer: ?

Access control models



Q1) What access control model is where an **owner** can assign permissions to users for resources they control?

Answer: Discretionary access control (DAC)

Q2) What access control model is where a user's **security clearance** must exceed a resource's **sensitivity label**?

Answer: Mandatory access control (MAC)

Q3) For access control, what does authentication mean?

Answer: The credential of the user has been verified. Specific permissions are granted based on this credential.

Lecture 17

Encryption

ECE 422: Reliable and Secure Systems Design



UNIVERSITY OF
ALBERTA

Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Encryption
- Symmetric encryption
 - Caesar cipher
- Man-in-the-middle attack
- Asymmetric encryption
 - Rivest–Shamir–Adleman (RSA) algorithm

Confidentiality

Confidentiality

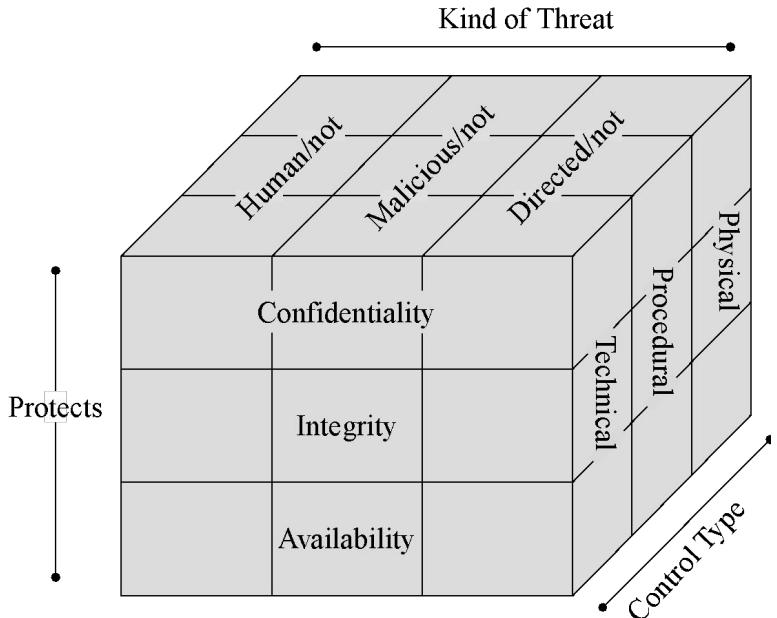
- Only the authorized user can access particular resources

Methods to achieve confidentiality:

- Encryption: encoding/decoding of the plaintext
 - E.g., Symmetric/asymmetric encryption
- Access controls: restricted access
 - E.g., Our library website
- **Authentication:** credentials check
 - E.g., Mobile authentication for faculty and staff



Threats, vulnerabilities, and control types



- CIA are the basic security principles.
- Vulnerabilities are weaknesses in a system that affect the CIA triad.
- Threats exploit those weaknesses in the system.
- Controls protect those weaknesses from exploitation.

Access control list (ACL)

An access control list is a set of instructions that either allow access to a computer environment or deny it.

- Restrict access to unauthorized users
- Control traffic by limiting the number of users

It is analogous to a guest list to a wedding.

- Only those on the lists are authorized to entries



Access control models

Users receive access based on access control models.

- Different systems have different access control requirements.

There are four models of access controls:

- Discretionary access control (DAC)
- Role-based access control (RBAC)
- Mandatory access control (MAC)
- Attribute-based access control (ABAC)

Schedule for today

- Key concepts from last class
- Encryption
- Symmetric encryption
 - Caesar cipher
- Man-in-the-middle attack
- Asymmetric encryption
 - Rivest–Shamir–Adleman (RSA) algorithm

Encryption

Encryption is used to provide confidentiality.

- Encryption: from a plain text to the ciphertext
- Decryption: from the ciphertext back to the plain text



There are two types of encryption:

- Symmetric encryption
- Asymmetric encryption

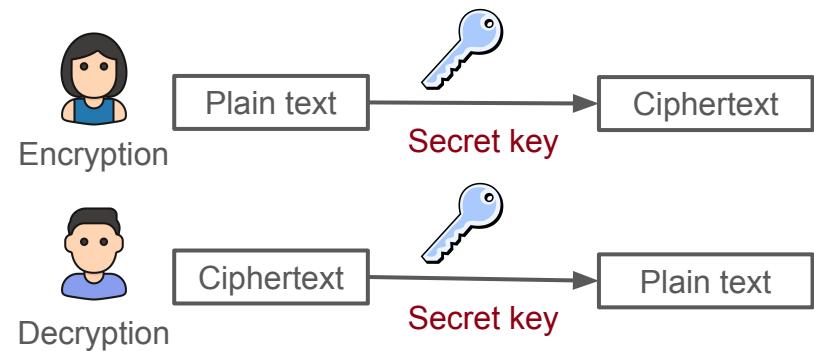
Symmetric encryption

Symmetric encryption uses a single key to encrypt and decrypt.

- Same key for encryption and decryption
- The key is kept secret
- Example of secret key: number, word, random string

Benefits of symmetric encryption include:

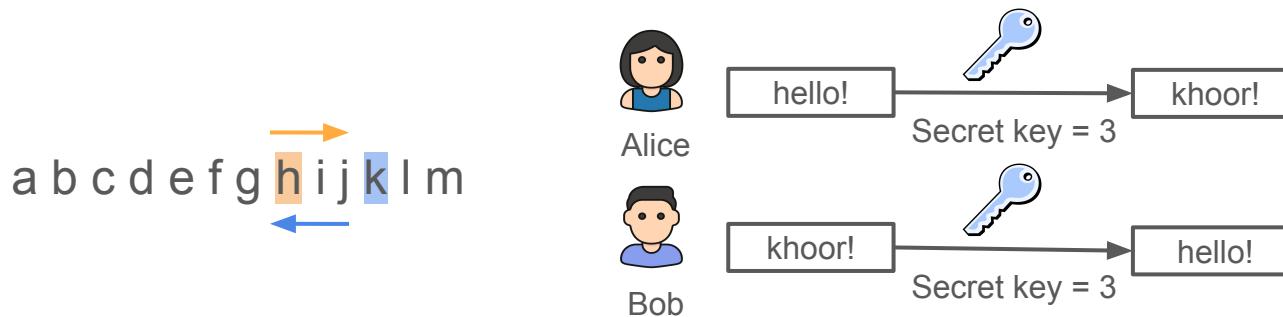
- Fast encryption and decryption
 - Inexpensive to process, single key
- Easy to implement, easy to use
 - straightforward encryption and decryption



Example of symmetric encryption

Symmetric encryption with **Caesar cipher** with a secret key, $e = 3$:

- Alice wants to send a message, $m = \text{"hello!"}$.
- Alice encrypts the message “hello!” into “khoor!”
 - by shifting the letters by 3 positions based on the secret key.
- Alice sends the ciphertext, $c = \text{"khoor!"}$, to Bob.
- Bob decrypts “khoor” with the same secret key.



Man-in-the-middle attack

In a man-in-the-middle attack, the attacker secretly intercepts and relays messages between two parties.

- Allow the attacker to eavesdrop the communication
- Sometimes worst, the attacker can intercept and then control the entire conversation

Common types of man-in-the-middle attack:

- IP spoofing (impersonate another computer system)
- Email hijacking (access to emails)
- Wifi eavesdropping
 - avoid public wifi where login is not required

Man-in-the-middle attack

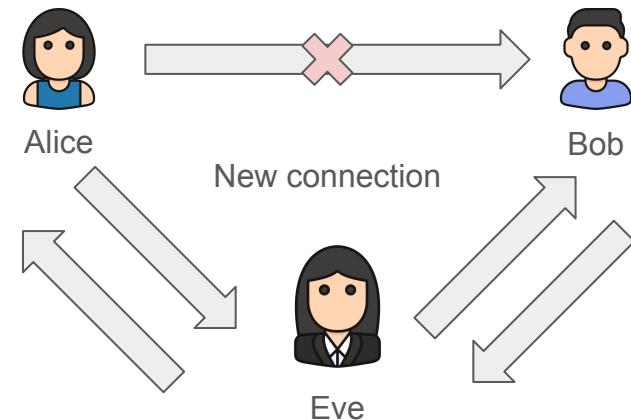
Alice wants to communicate with Bob

- Eve can intercept the messages from Alice and claim to be Bob
- At the same time, Eve can convince Bob that she is Alice

Eve can then intercept, manipulate, and relay the messages.

Solutions?

- Encryption (confidentiality)
- Digital signature (authentication)



Symmetric encryption

Problem: what happens if Eve tries to eavesdrop (adversary)?

- Eve will not be able to understand the encrypted message
- But with enough messages, Eve can come up with the right decryption algorithm (brute force)
 - E.g., based on the frequency of the letter

Another attempt ...

- We can change the encryption key everytime we send an encrypted message

A more severe problem: how do we securely distribute the secret key?

- The problem is still not solved.

Symmetric encryption

Disadvantages of symmetric encryption:

- Not scalable
 - E.g., 100 people, 100 unique keys to manage
- Key management is challenging
 - Key may be compromised, security at risk
 - Key may be lost, reissuing is expensive and time-consuming
- Lack of authentication
 - It does not verify the identity of the sender

Schedule for today

- Key concepts from last class
- Encryption
- Symmetric encryption
 - Caesar cipher
- Man-in-the-middle attack
- Asymmetric encryption
 - Rivest–Shamir–Adleman (RSA) algorithm

Asymmetric encryption

Asymmetric encryption uses a public key to encrypt and a private key to decrypt.

- Public key: anyone can see and use this key
- Private key: kept private
- Private and public keys come in pairs
- Data encrypted with the public key can only be decrypted with the private key

Suppose Alice needs to send a message to Bob

- Alice will use **Bob's public key** to encrypt the message
- Bob will use **his own private key** to decrypt the message

Asymmetric encryption vs digital signature

Note that it is the other way around in digital signature:

- Alice will use her own private key to encrypt the message
 - Only Alice can close the envelope with her encryption key
- Bob will use Alice's public key to decrypt the message
 - Everyone is welcome to open the enclosed envelope from Alice with her public key.

In asymmetric encryption:

- Alice will use **Bob's public key** to encrypt the message
- Bob will use **his own private key** to decrypt the message
 - Only Bob can open the envelope with his private key

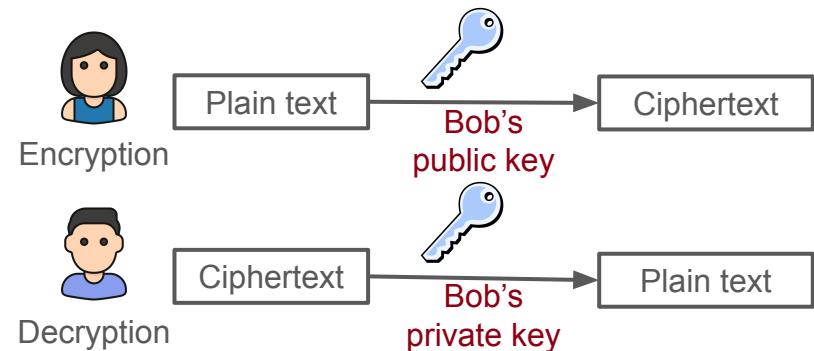
Example of asymmetric encryption

Bob generates two keys:

- public key for encryption
- private key for decryption

Alice wants to send a message to Bob

- Alice encrypts the message “hello!” with Bob’s public key.
- Alice sends the ciphertext c to Bob.
- Bob decrypts c with his private key.



Although the encryption key is public, it is impossible to eavesdrop:

- Bob is the only one with decryption key

RSA algorithm

Rivest–Shamir–Adleman (RSA) algorithm is one of the oldest widely used for secure data transmission.

- Utilize private and public key pair
 - Private key kept secret
 - Public key is available to everyone
- Either one of the keys can be public, while the other key can be private
- Based on the factorization of large prime numbers



From left to right: Adi Shamir, Ron Rivest, and Len Adleman

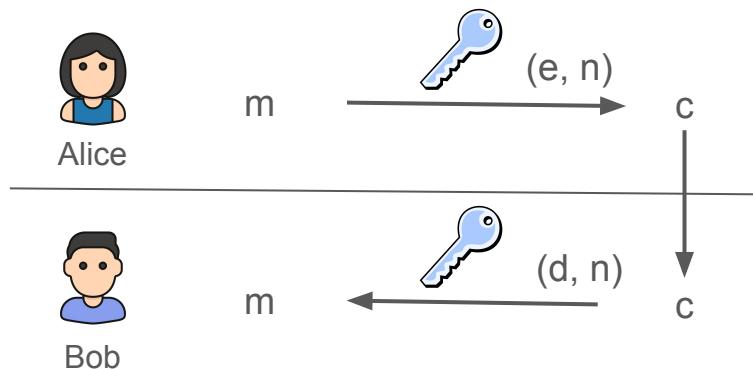
RSA algorithm

Encryption: encrypt the message m with the public key (e, n)

- $m^e \text{ mod}(n) = c$

Decryption: decrypt the ciphertext c with the private key (d, n)

- $c^d \text{ mod}(n) = m$



- Plain text m
- Encryption key: (e, n)
- Decryption key: (d, n)
- Ciphertext c

RSA algorithm

(e, n)



Encryption (3, 33), Plaintext = "b" = 2

Alice wants to send the number 2:

- $m^e \text{ mod}(n) = c \rightarrow 2^3 \text{ mod } (33) = 8$
- The ciphertext c is 8

(d, n)

Decryption (7, 33)

How does Bob decrypt the ciphertext 8?

Encryption key (e, n)

- $m^e \text{ mod}(n) = c$

Decryption key (d, n)

- $c^d \text{ mod}(n) = m$

RSA algorithm



Encryption (3, 33), Plaintext = "b" = 2

Alice wants to send the number 2:

- $m^e \text{ mod}(n) = c \rightarrow 2^3 \text{ mod } (33) = 8$
- The ciphertext c is 8

Decryption (7, 33)

Bob wants to decrypt the ciphertext:

- $c^d \text{ mod}(n) = m \rightarrow 8^7 \text{ mod } (33) = 2$
- The plaintext is 2

Encryption key (e, n)

- $m^e \text{ mod}(n) = c$

Decryption key (d, n)

- $c^d \text{ mod}(n) = m$

RSA algorithm

RSA algorithm as a three-part process:

Part I: Bob's public and private key setup

Part II: Alice encrypts m for Bob

Part III: Bob receives and decrypts c

RSA algorithm

Part I: Bob's public and private key setup

Example

- Chooses two prime numbers, p and q
 - Calculate the product $n = pq$
- $p = 11, q = 3$
 - $n = pq = 33$

n will eventually be published as part of the keys

- *Encryption key (e, n)*
- *Decryption key (d, n)*

However, p and q values remain secret.

RSA algorithm

Part I: Bob's public and private key setup

Example

- Chooses two prime numbers, p and q
- Calculate the product $n = pq$
- Solve $\varphi(n) = (p-1)(q-1)$

Euler's totient function

Euler's totient function (also called Phi function), $\varphi(n)$, counts the number of integers less than n that are coprime to n .

E.g., $\varphi(6)$ from [1, 2, 3, 4, 5, 6]

$\varphi(6)$ from [1, 5], $\varphi(6) = 2$ or $\varphi(2 \times 3) = (p-1)(q-1) = 1 \times 2 = 2$

RSA algorithm

Part I: Bob's public and private key setup

- Chooses two prime numbers, p and q
- Calculate the product $n = pq$
- Solve $\varphi(n) = (p-1)(q-1)$
- Choose numbers e and d so that ed has a remainder of 1 when divided by $\varphi(n)$
 - $1 < e < \varphi(n)$, where e must be an integer
 - e and $\varphi(n)$ must be coprime
 - $e^*d \pmod{\varphi(n)} = 1$

Example

- $p = 11, q = 3$
- $n = pq = 33$
- $\varphi(n) = 10 \times 2 = 20$
- Pick e and d so that $ed = 20+1$
 - e.g.,: $e = 3, d = 7$
 - $1 < 3 < 20$
 - 3 and 20 are coprime

RSA algorithm

Part I: Bob's public and private key setup

- Chooses two prime numbers, p and q
- Calculate the product $n = pq$
- Solve $\varphi(n) = (p-1)(q-1)$
- Choose numbers e and d so that ed has a remainder of 1 when divided by $\varphi(n)$
 - $1 < e < \varphi(n)$, where e must be an integer
 - e and $\varphi(n)$ must be coprime
- Publish the public key (e, n)

Example

- $p = 11, q = 3$
- $n = pq = 33$
- $\varphi(n) = 10 \times 2 = 20$
- Pick e and d so that $ed = 20+1$
 - e.g.,: $e = 3, d = 7$
 - $1 < 3 < 20$
 - 3 and 20 are coprime
- Publish $(e, n) = (3, 33)$

RSA algorithm

Part II: Alice encrypts m for Bob

- Get Bob's public key (e, n)
- Encryption, calculate the ciphertext
 - $m^e \text{ mod}(n) = c$
- Send the ciphertext to Bob

Example

- $(e, n) = (3, 33)$
- Encrypting $m = 2$
 - $2^3 \text{ mod } (33) = 8$
- Send $c = 8$ to Bob

Encryption key (e, n)

- $m^e \text{ mod}(n) = c$

RSA algorithm

Part III: Bob receives and **decrypts** c

- Get his private key (d, n)
- Decryption, calculate the plaintext
 - $c^d \text{ mod}(n) = m$
- The plaintext should match what Alice sent

Example

- $(d, n) = (7, 33)$
- Decrypting c = 8
 - $8^7 \text{ mod } (33) = 2$
- $m = 2$, Alice's original message

Decryption key (d, n)

- $c^d \text{ mod}(n) = m$

RSA algorithm

Rivest–Shamir–Adleman (RSA) algorithm is one of the oldest widely used for secure data transmission.

- Utilize private and public
 - Private key kept secret
 - Public key is available to anyone
- Either one of the key can be public, while the other key can be private
- Based on the factorization of large prime numbers

What does this mean?



From left to right: Adi Shamir, Ron Rivest, and Len Adleman

What happens if we swap the public and private keys?



Encryption ~~(3, 33)~~ (7, 33), Plaintext = "b" = 2

Alice wants to send the number 2:

- $m^e \bmod(n) = c$

Decryption ~~(7, 33)~~ (3, 33)

Bob wants to decrypt the ciphertext:

- $c^d \bmod(n) = m$

Public and private key swap



Encryption ~~(3, 33)~~ (7, 33), Plaintext = "b" = 2

Alice wants to send the number 2:

- $m^e \bmod(n) = c \rightarrow 2^7 \bmod(33) = 29$
- The ciphertext c is 29

Decryption ~~(7, 33)~~ (3, 33)

Bob wants to decrypt the ciphertext:

- $c^d \bmod(n) = m \rightarrow 29^3 \bmod(33) = 2$
- The ciphertext c is 2

... Still works!

RSA algorithm in practice

When our internet browser shows a URL beginning with https, the RSA algorithm is being used to protect our privacy.

For example, log in to Facebook:

- Our computer plays the role of Alice
- Facebook server plays the role of Bob, encrypting and decrypting the information passed back and forth.

In practice, the primes p and q are chosen to be very big numbers.

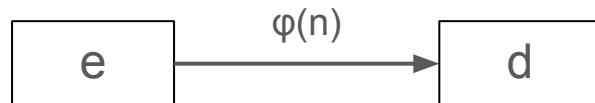
- Because the security of the RSA cryptosystem lies in the difficulty of factoring an integer that is the product of two large prime numbers.

Confidentiality

- Bob calculates e , d , n using p and q
 - Private: d , p and q
- Bob shares the information: encryption key (e , n)
 - Public: e , n
- Alice encrypts with (e , n), and shares the ciphertext
 - Public: ciphertext
- Bob decrypts with: decryption key (d , n), ciphertext
 - Public: n
 - Private: d

Confidentiality

To calculate the value of d, we need to use e:



$$e^*d \pmod{\varphi(n)} = 1$$

To calculate $\varphi(n)$, we need to use p and q:



$$\varphi(n) = (p-1)(q-1)$$

To calculate p and q, we need to use n:



$$n = pq$$

However, prime factorization is hard. We typically use 1024-bit or 2048-bit RSA keys. To factor the n value, we can only use a brute force solution.

Plaintext into numeric digits

Computers represent text as long numbers (01 for “A”, 02 for “B” and so on), so an email message is just a very big number.

Suppose we want to convert “hello world” into digits

- Convert “Hello world” into a sequence of bytes (hexadecimal)
 - “Hello world” -> “48 65 6C 6C 6F 20 57 6F 72 6C 64”
- Convert the sequence from hexadecimal to decimal
 - “48 65 6C 6C 6F 20 57 6F 72 6C 64” -> “87521618088882533792115812”
 - $(48 \ 65 \ 6C \ 6C \ 6F \ 20 \ 57 \ 6F \ 72 \ 6C \ 64)_{16} = (4 \times 16^{21}) + (8 \times 16^{20}) + (6 \times 16^{19}) + (5 \times 16^{18}) + (6 \times 16^{17}) + (12 \times 16^{16}) + (6 \times 16^{15}) + (12 \times 16^{14}) + (6 \times 16^{13}) + (15 \times 16^{12}) + (2 \times 16^{11}) + (0 \times 16^{10}) + (5 \times 16^9) + (7 \times 16^8) + (6 \times 16^7) + (15 \times 16^6) + (7 \times 16^5) + (2 \times 16^4) + (6 \times 16^3) + (12 \times 16^2) + (6 \times 16^1) + (4 \times 16^0) = (87521618088882533792115812)_{10}$

U.S. Patent on RSA algorithm

U.S. Patent 4,405,829 on RSA public key encryption

United States Patent [19]

Rivest et al.

[11] 4,405,829

[45] Sep. 20, 1983

[54] CRYPTOGRAPHIC COMMUNICATIONS
SYSTEM AND METHOD

[75] Inventors: Ronald L. Rivest, Belmont; Adi Shamir, Cambridge; Leonard M. Adleman, Arlington, all of Mass.

[73] Assignee: Massachusetts Institute of
Technology, Cambridge, Mass.

[21] Appl. No.: 860,586

[22] Filed: Dec. 14, 1977

[51] Int. Cl.³ H04K 1/00; H04I 9/04

[52] U.S. Cl. 178/22.1; 178/22.11

[58] Field of Search 178/22, 22.1, 22.11,
178/22.14, 22.15

Primary Examiner—Sal Cangialosi
Attorney, Agent, or Firm—Arthur A. Smith, Jr.; Robert J. Horn, Jr.

[57] ABSTRACT

A cryptographic communications system and method. The system includes a communications channel coupled to at least one terminal having an encoding device and to at least one terminal having a decoding device. A message-to-be-transferred is enciphered to ciphertext at the encoding terminal by first encoding the message as a number M in a predetermined set, and then raising that number to a first predetermined power (associated with the intended receiver) and finally computing the remainder, or residue, C, when the exponentiated number

Lecture 18

Midterm Review

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

How was the midterm?

A quick survey: [Click here for the survey link](#)

- How hard was the midterm?
- Did you know what to study for your midterm?
- Did you have enough support (e.g., materials, sample questions) to prepare for the midterm?

Question on (7, 4) Hamming code

Multiple Choice Question: Given that 0001011 is a codeword in (7, 4) Hamming code, which of the following cannot be the valid codeword in the codespace?
(Hint: a (7, 4) Hamming code can correct and detect any single-bit error.)

- a. 0011101
- b. 0101100
- c. 0011010
- d. 1110100

Code distance in error correction

To correct d bit errors, the code distance for the codewords must be larger or equal to $2d+1$.

Example

Code: {000, 101}

$$C_d = 2$$

Transmitting codeword: 000

A single-bit error happens, 000 becomes 001

We cannot tell how to correct 001 (000 or 101).

Analogy

Dictionary: {accept, except}

Distance = 2

Typed word: accept

A typo happens, accept becomes except

We cannot tell which word is mistyped.

Question on (7, 4) Hamming code

Multiple Choice Question: Given that 0001011 is a codeword in (7, 4) Hamming code, which of the following **cannot be the valid codeword** in the codespace?
(Hint: a (7, 4) Hamming code can correct and detect any single-bit error.)

- a. 0011101
- b. 0101100
- c. 0011010
- d. 1110100

To correct d bit errors, the code distance for the codewords **must be larger or equal to $2d+1$** .

0	0	0	1	0	1	1
---	---	---	---	---	---	---

0	0	1	1	1	0	1
---	---	---	---	---	---	---

→ distance = 3

0	1	0	1	1	0	0
---	---	---	---	---	---	---

→ distance = 5

0	0	1	1	0	1	0
---	---	---	---	---	---	---

→ distance = 2

1	1	1	0	1	0	0
---	---	---	---	---	---	---

→ distance = 7

Question on Spectrum-based Fault Localization

Multiple Choice Question: Which of the following statements is the most likely to be suspicious based on Spectrum-based Fault Localization?

- a. S_1
- b. S_2
- c. S_3
- d. S_4

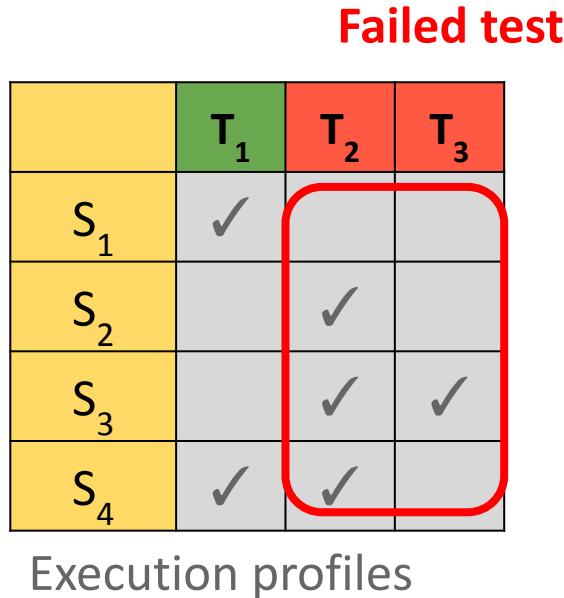
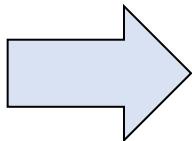
	T_1	T_2	T_3
S_1	✓		
S_2		✓	✓
S_3	✓		
S_4	✓	✓	✓
Result	P	F	F

$$Ochiai(element) = \frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}}$$

Lecture 6: Spectrum-based fault localization

Source
file A

Run tests



SBFL

Statement	Suspiciousness score
S ₁	0.00
S ₂	0.71
S ₃	1.00
S ₄	0.50

Hint: program elements that are covered by more failing tests but less passing tests are more suspicious.

Question on Spectrum-based Fault Localization

Multiple Choice Question: Which of the following statements is the most likely to be suspicious based on Spectrum-based Fault Localization?

- a. S_1
- b. S_2
- c. S_3
- d. S_4

	T_1	T_2	T_3
S_1	✓		
S_2		✓	✓
S_3	✓		
S_4	✓	✓	✓
Result	P	F	F

S_2 : 2 failing tests
 S_4 : 2 failing tests, 1 passing test

S_2 is the most suspicious statement

$$Ochiai(element) = \frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}}$$

Question on RSA algorithm



Alice wants to send a message ($m = 5$) to Bob. Assume that the two prime numbers used to generate the keys are $p = 5$, $q = 11$, and Alice must choose a value $e < 6$.

Question: What is the ciphertext? (show your calculations and assumptions)

[2 points]

Encryption key (e, n)

- $m^e \text{ mod}(n) = c$

Question on RSA algorithm

Alice wants to send a message ($m = 5$) to Bob. Assume that the two prime numbers used to generate the keys are $p = 5$, $q = 11$, and Alice must choose a value $e < 6$.

Question: What is the ciphertext? (show your calculations and assumptions)

Thought process: We have the values m , p , q , asked to calculate c

- We need to calculate the public key (e, n) to find c
 - Step 1: calculate n
 - Step 2: calculate $\varphi(n)$
 - Step 3: find e that satisfies the conditions
 - Step 4: calculate c with (e, n)

Lecture 17: RSA algorithm

Part I: Bob's public and private key setup

- Chooses two prime numbers, p and q
- Calculate the product $n = pq$
- Solve $\varphi(n) = (p-1)(q-1)$
- Choose numbers e and d so that ed has a remainder of 1 when divided by $\varphi(n)$
 - $1 < e < \varphi(n)$, where e must be an integer
 - e and $\varphi(n)$ must be coprime
- Publish the public key (e, n)

Example

- $p = 11, q = 3$
- $n = pq = 33$
- $\varphi(n) = 10 \times 2 = 20$
- Pick e and d so that $ed = 20+1$
 - e.g.,: $e = 3, d = 7$
 - $1 < 3 < 20$
 - 3 and 20 are coprime
- Publish $(e, n) = (3,$

Question on RSA algorithm

Solution: We need to calculate the public key (e, n) to find c

Step 1: calculate n

- If $p = 5, q = 11$, then $n = pq = 55$ [0.5 pts]

Step 2: calculate $\phi(n)$

- $\phi(n) = (p-1)(q-1) = 4 \times 10 = 40$ [0.5 pts]

Question on RSA algorithm

Solution (cont.):

Step 3: find e that satisfies the conditions

- Condition: $1 < e < \varphi(n)$, and given that $e < 6$
 - $1 < e < 6$
- Condition: e and $\varphi(n)$ must be coprime
 - e and 40 must be coprime
 - e cannot be a divisor of 40 including 2, 4, 5
- Since e must be less than 6, it must be 3
- Public key: (3, 55)

Question on RSA algorithm

Solution (cont.):

Step 4: calculate c with (e, n)

Version A: If $e = 3$, $m = 5$

- $m^e \bmod(n) = c$
- $5^3 \bmod(55) = 15$ [1 pt]

Version B: If $e = 3$, $m = 7$

- $m^e \bmod(n) = c$
- $7^3 \bmod(55) = 13$ [1 pt]

Question on RSA algorithm



In previous question, Alice used the RSA algorithm to send the message ($m = 5$) to Bob so that others could not read the message. Suppose that you have been listening to their communication channel:

Question: Do you know Alice's public key? If yes, what is it? (1~2 sentences)

[1 point]

Question on RSA algorithm

In previous question, Alice used the RSA algorithm to send the message ($m = 5$) to Bob so that others could not read the message. Suppose that you have been listening to their communication channel:

Question: Do you know Alice's public key? If yes, what is it? (1~2 sentences)

Thought process: Alice wants to send a message to Bob

- Alice should use Bob's public key to encrypt the message
 - Analogy: only Bob can open the envelope with his private key

Lecture 17: Asymmetric encryption

Asymmetric encryption uses a public key to encrypt and a private key to decrypt.

- Public key: anyone can see and use this key
- Private key: kept private
- Private and public keys come in pairs
- Data encrypted with the public key can only be decrypted with the private key

Suppose Alice needs to send a message to Bob

- Alice will use **Bob's public key** to encrypt the message
- Bob will use **his own private key** to decrypt the message

Question on RSA algorithm

In previous question, Alice used the RSA algorithm to send the message ($m = 5$) to Bob so that others could not read the message. Suppose that you have been listening to their communication channel:

Question: Do you know Alice's public key? If yes, what is it? (1~2 sentences)

Thought process: Alice wants to send a message to Bob

- Alice should use Bob's public key to encrypt the message
 - Analogy: only Bob can open the envelope with his private key
- Do we need Alice's public key? No

Solution: No, the public and private key needed for encryption belong to Bob. Nothing is known about Alice's public key.

Question on RSA algorithm



In previous question, Alice used the RSA algorithm to send the message ($m = 5$) to Bob so that others could not read the message. Suppose that you have been listening to their communication channel:

Question: Do you know Bob's public key? If yes, what is it? (1~2 sentences)

[1 point]

Question on RSA algorithm

In previous question, Alice used the RSA algorithm to send the message ($m = 5$) to Bob so that others could not read the message. Suppose that you have been listening to their communication channel:

Question: Do you know Bob's public key? If yes, what is it? (1~2 sentences)

Solution: Yes, Bob's public key: $(e, n) = (3, 55)$

Question on RSA algorithm



In previous question, Alice used the RSA algorithm to send the message ($m = 5$) to Bob so that others could not read the message. Suppose that you have been listening to their communication channel:

Question: If Bob uses a digital signature, then what is its purpose? (i.e., what does the digital signature do?)

[1 point]

Question on RSA algorithm

In previous question, Alice used the RSA algorithm to send the message ($m = 5$) to Bob so that others could not read the message. Suppose that you have been listening to their communication channel:

Question: If Bob uses a digital signature, then what is its purpose? (i.e., what does the digital signature do?)

Solution: The digital signature certifies that the public key belongs to Bob.

Digital signature

Digital signatures verify the authenticity

- Detect the identity of the sender/signer

Digital signatures check the integrity

- Verify that the message was not changed

Digital signatures ensure non-repudiation

- Verify that the signature is not fake

Question on cyclic codes



Suppose $g(x) = 1101$ for a $(7, 4)$ cyclic code. Bob receives a codeword 0010111 from Alice.

Question: Is there an error? If yes, justify why. If not, what is the original data?
(show your steps)

[2 points]

Question on cyclic codes

Suppose $g(x) = 1101$ for a $(7, 4)$ cyclic code. Bob receives a codeword 0010111 from Alice.

Question: Is there an error? If yes, justify why. If not, what is the original data? (show your steps)

Thought process: We have the values $g(x)$ and $c(x)$, asked whether there is a remainder in $c(x)/g(x)$. If yes, then there is an error.

- Solve $d(x) = c(x)/g(x)$
 - Step 1: convert $c(x)$ and $g(x)$ into polynomials
 - Step 2: calculate polynomial division $c(x)/g(x)$, check for remainder

Question on cyclic codes

Suppose $g(x) = 1101$ for a (7, 4) cyclic code. Bob receives a codeword 0010111 from Alice.

Question: Is there an error? If yes, justify why. If not, what is the original data?
(show your steps)

Solution: Solve $d(x) = c(x)/g(x)$

Step 1: convert $c(x)$ and $g(x)$ into polynomials

- $c(x) = x^2 + x^4 + x^5 + x^6$ (0.5 pts)
- $g(x) = 1 + x + x^3$ (0.5 pts)

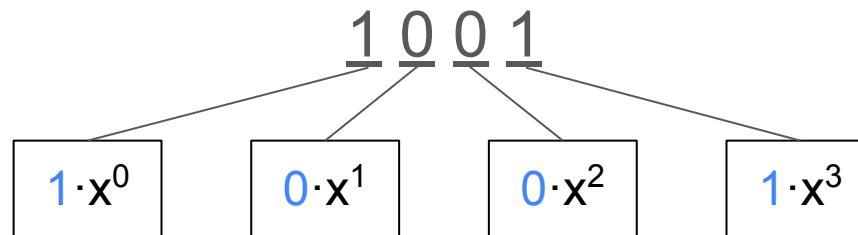
Lecture 9: Step 1: data polynomial

Data = {1001}

The data can be represented as a polynomial:

$$a(x) = a_0 \cdot x^0 + a_1 \cdot x^1 + \dots + a_{n-1} \cdot x^{n-1}$$

The data can also be visualized as:



So we represent 1001 as:

$$d(x) = 1 \cdot x^0 + 0 \cdot x^1 + 0 \cdot x^2 + 1 \cdot x^3 = 1 + x^3$$

Lecture 9 Slide 19
Lecture 10 Slide 9

Question on cyclic codes

Solution (cont.):

Step 2: calculate polynomial division $c(x)/g(x)$, check for remainder

- $(x^2 + x^4 + x^5 + x^6)/(1 + x + x^3) = x^2 + x^3$, no remainder

$$\begin{array}{r} x^6 + x^5 + x^4 + x^2 \\ \underline{-} x^6 + x^4 + x^3 \\ \hline x^5 + x^3 + x^2 \\ \underline{-} x^5 + x^3 + x^2 \\ \hline 0 \end{array} \quad \left| \begin{array}{r} x^3 + x + 1 \\ \hline x^3 + x^2 \end{array} \right.$$

No remainder
No error

Lecture 10: Polynomial division (1)

$$\begin{array}{r} x^6 + x^5 + x^4 + x^3 + x^2 + x + 1 \\ \underline{-} x^6 + x^4 + x^3 \\ \hline x^5 + x^2 + x + 1 \\ \underline{-} x^5 + x^3 + x^2 \\ \hline x^3 + x + 1 \\ \underline{-} x^3 + x + 1 \\ \hline 0 \end{array} \quad \left| \begin{array}{r} x^3 + x + 1 \\ \hline x^3 + x^2 + 1 \end{array} \right.$$

No error

Question on cyclic codes

Suppose $g(x) = 1101$ for a (7, 4) cyclic code. Bob receives a codeword 0010111 from Alice.

Question: Is there an error? If yes, justify why. **If not, what is the original data?** (show your steps)

Solution: The original data is 0011 (or 0011000) (1 pt)

- Partial mark: $x^3 + x^2$ (0.5 pts)

Question on extended Hamming code



Question: Calculate the final code after encoding the code 11110100100 into 16-bit even parity extended Hamming code. (show your steps, e.g., P1: {0000001}, odd parity, P1 = 1) (Hint: 1 111 010 0100)

[3 points]

Question on extended Hamming code



Question: Calculate the final code after encoding the code 11110100100 into 16-bit even parity extended Hamming code (show your steps, e.g., P1: {0000001}, odd parity, P1 = 1) (Hint: 1 111 010 0100)

[3 points]

(16, 11) Extended Hamming code



			1
	1	1	1
	0	1	0
0	1	0	0

Lecture 8: Extended Hamming codes

Extended Hamming code is a linear code that can detect and correct single-bit errors, and also detect double-bit errors.

- Uses an extra parity check for the whole block of bits
- For example, parity check on {1100 111 0010 0100}

1	1	1	0
0	1	1	1
0	0	1	0
0	1	0	0

Extended Hamming code (even parity)

Question on extended Hamming code

			1
1	1	1	
0	1	0	
0	1	0	0

Parity: ?

Parity bit at
position 1

			1
1	1	1	
0	1	0	
0	1	0	0

Parity: ?

Parity bit at
position 2

			1
1	1	1	
0	1	0	
0	1	0	0

Parity: ?

Parity bit at
position 4

			1
1	1	1	
0	1	0	
0	1	0	0

Parity: ?

Parity bit at
position 8

Question on extended Hamming code

	0		1
	1	1	1
	0	1	0
0	1	0	0

Parity: 0

Parity bit at
position 1

P_1 : {1110010}
even parity

$$P_1 = 0$$

(0.6 pts)

	0	0	1
	1	1	1
	0	1	0
0	1	0	0

Parity: 0

Parity bit at
position 2

P_2 : {1111000}
even parity

$$P_2 = 0$$

(0.6 pts)

	0	0	1
0	1	1	1
	0	1	0
0	1	0	0

Parity: 0

Parity bit at
position 4

P_3 : {1110100}
even parity

$$P_3 = 0$$

(0.6 pts)

	0	0	1
0	1	1	1
	0	1	0
0	1	0	0

Parity: 0

Parity bit at
position 8

P_4 : {0100100}
even parity

$$P_4 = 0$$

(0.6 pts)

Question on extended Hamming code

	0	0	1
0	1	1	1
0	0	1	0
0	1	0	0

Parity: 0

Parity bit at
position 0

P_5 : {001011100100100}

even parity

$P_5 = 0$

(0.6 pts)

Question: Calculate the final code after encoding the code 11110100100 into 16-bit even parity extended Hamming code. (show your steps, e.g., P1: {0000001}, odd parity, $P_1 = 1$) (Hint: 1 111 010 0100)

Solution: The final code is:

0001 0111 0010 0100

Secure File System Project

Project description and marking guide available on eClass

- Week 6: February 12, 2024
- Same groups of 3 people
- Programming language of your choice (e.g., Python, Java, and C++)
- Following the agile methodology

Final report (6-10 pages)

- Expands on the deliverable, based on the finished product

Demonstration (10-15 minutes)

- After the final report submission, scheduled with the TAs.

Course projects

Project 2: Secure File System

A secure file system that allows its internal users to store data on an untrusted file server.

Project deliverable (10%)

- Due Friday, March 15
- More than two weeks from now

Final report and demo (15%)

- Due Monday, April 8
- Three weeks from the submission of the deliverable

Project deliverable

Project 2: Secure File System Deliverable (3-5 pages)

- Due Friday, March 15

Design

- Class diagram

Tools and technologies

- What technologies you plan to use? Why?

User stories

- Three user stories (persona + need + purpose)
- Each user story should be broken down into sub-tasks

Planning

- Timeline of the subtasks

Lecture 19

Encryption - Part II

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
- Diffie-Hellman Key Exchange
 - Primitive roots
 - Discrete logarithm problem
- TODOs

Confidentiality

Confidentiality

- Only the authorized user can access particular resources

Methods to achieve confidentiality:

- **Encryption:** encoding/decoding of the plaintext
 - E.g., Symmetric/asymmetric encryption
- Access controls: restricted access
 - E.g., Our library website
- Authentication: credentials check
 - E.g., Mobile authentication for faculty and staff



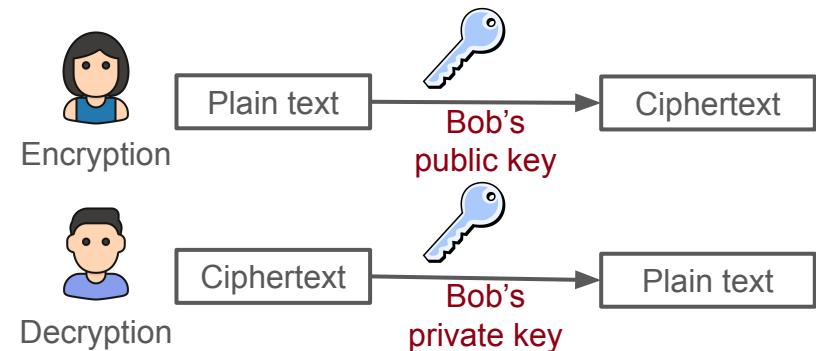
Example of asymmetric encryption

Bob generates two keys:

- public key for encryption
- private key for decryption

Alice wants to send a message to Bob

- Alice encrypts the message “hello!” with Bob’s public key.



Although the encryption key is public, it is impossible to eavesdrop:

- Bob is the only one with decryption key

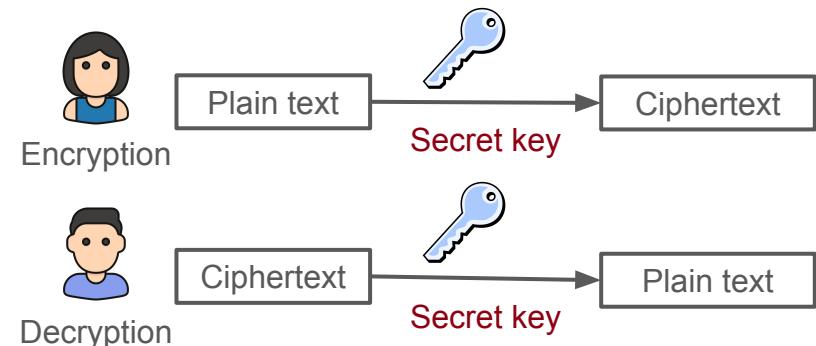
Example of symmetric encryption

Alice generates one key:

- The secret key is used for both encryption and decryption

Alice wants to send a message to Bob

- Alice shares the secret key with Bob.



With man-in-the-middle attack

- Eve can intercept the messages between Alice and Bob.
- But, worst than that, Eve can also intercept the secret key.

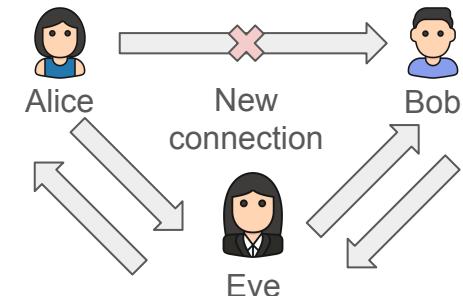
Key exchange in symmetric encryption

Contrary to asymmetric encryption, in symmetric encryption, the key exchange becomes vulnerable to man-in-the-middle attacks.

Alice wants to communicate with Bob

- Alice sends her key to Bob
- Eve can intercept the key from Alice and claim to be Bob
- At the same time, Eve can convince Bob that she is Alice

How can Alice and Bob agree on a shared secret key without letting Eve, who is always listening, also obtain the key?



Schedule for today

- Key concepts from last class
- Diffie-Hellman Key Exchange
 - Primitive roots
 - Discrete logarithm problem
- TODOs

Diffie-Hellman algorithm

Diffie-Hellman is a **key exchange algorithm** used with symmetric encryption.

- It allows both parties to agree on an identical secret key
- Without having to share the actual key in the communication channel
- This is used for key exchange, and not encryption/decryption

Overall idea behind DF key exchange algorithm:

- Neither parties choose the key explicitly
- Both parties contribute in calculating the secret key together
- The calculated secret key can then be used in symmetric encryption.

Diffie-Hellman algorithm

Introduced by Whitfield Diffie and Martin E. Hellman (2015 Turing Award) from Stanford University in their paper: [New Directions in Cryptography](#), 1976

[\[PDF\] New directions in cryptography](#)

W Diffie, ME Hellman - Democratizing **Cryptography**: The Work of ..., 2022 - dl.acm.org

... In turn, such applications create a need for **new** types of **cryptographic** systems which minimize the necessity of secure key distribution channels and supply the equivalent of a ...

☆ Save 芻 Cite Cited by 23568 Related articles All 145 versions

644 IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. IT-22, NO. 6, NOVEMBER 1976

New Directions in Cryptography

Invited Paper

WHITFIELD DIFFIE AND MARTIN E. HELLMAN, MEMBER, IEEE

Abstract—Two kinds of contemporary developments in cryptography are examined. Widening applications of teleprocessing have given rise to a need for new types of cryptographic systems, which minimize the need for secure key distribution channels and supply the equivalent of a written signature. This paper suggests ways to solve these currently open problems. It also discusses how

The best known cryptographic problem is that of privacy: preventing the unauthorized extraction of information from communications over an insecure channel. In order to use cryptography to insure privacy, however, it is currently necessary for the communicating parties to share



From left to right:
Hellman and Diffie

Analogous to finding a shared secret color

Analogy: Alice and Bob wants to agree on a secret color without letting Eve know
Solution as a 4-steps process:

Step 1: Alice and Bob agree on a starting color

Step 2: Alice and Bob pick their secret color to come up with a mixed color

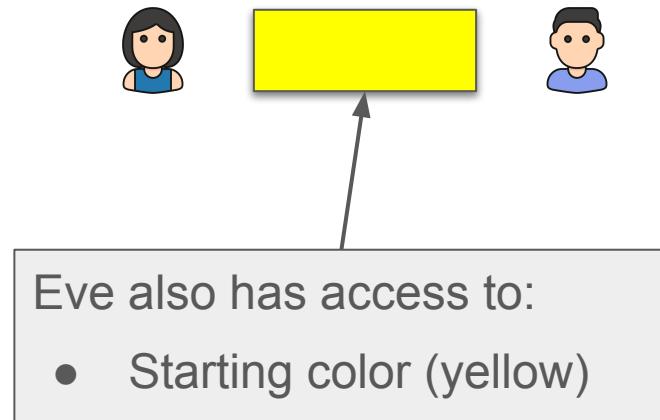
Step 3: Alice and Bob share their mixed color

Step 4: Alice and Bob get a new mixture based on each other's mixed color
and their own secret color

Analogous to finding a shared secret color

Analogy: Alice and Bob wants to agree on a secret color without letting Eve know

Step 1: Alice and Bob agree on a starting color, yellow



Analogous to finding a shared secret color

Analogy: Alice and Bob wants to agree on a secret color without letting Eve know

Step 1: Alice and Bob agree on a starting color, yellow



Step 2: Alice and Bob pick their secret color and mix it with the starting color

- Alice picks blue, the mixture gives green (yellow + blue)



- Bob picks red, the mixture gives orange (yellow + red)



Analogous to finding a shared secret color

Analogy: Alice and Bob wants to agree on a secret color without letting Eve know
Step 3: Alice and Bob exchange the mixed color



Eve also has access to:

- Starting color (yellow)
- Mixed colors (green and orange)

Analogous to finding a shared secret color

Analogy: Alice and Bob wants to agree on a secret color without letting Eve know

Step 4: Alice and Bob mix their secret color with the mixed color to get the shared secret color

- Alice mixes orange with blue, the mixture gives brown (yellow + red + blue)



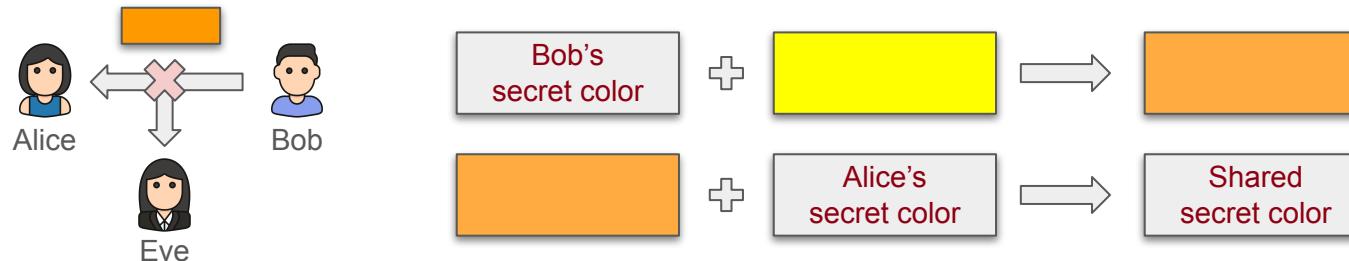
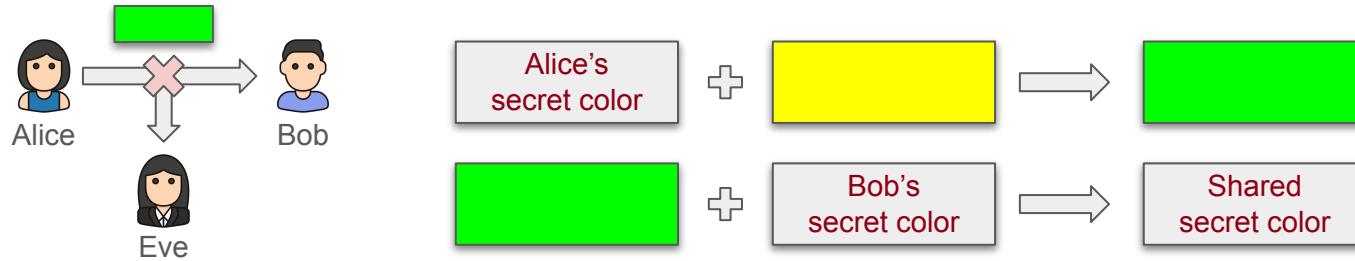
- Bob mixes green with red, the mixture gives brown (yellow + blue + red)



Analogous to finding a shared secret color

Can Eve come up with the shared secret color, brown?

- Eve has access to the starting color (yellow), and the mixed colors (green and orange), but not the secret colors (red and blue).



DF algorithm

DF algorithm as a 4-steps process:

Step 1: Alice and Bob agree on some **public parameters** (starting color)

Step 2: Alice and Bob pick their **secret integer** (secret color) to come up with a **public integer** (mixed color)

Step 3: Alice and Bob **exchange their own public integer** (mixed color)

Step 4: Alice and Bob **compute the shared secret key** based on the exchanged public integer (mixed color) and their own secret integer (secret color)

DF algorithm

Step 1: Alice and Bob agree on some **public parameters**

- a (large) prime number p
- an integer g

Example

- $p = 17$
- $g = 3$



The integer g must be a primitive root to the prime number p .

Primitive roots

An integer g is a **primitive root mod p** if every integer a coprime to p is congruent to a power of $g \bmod p$.

- Example: 2 is a primitive root mod 5, because for every number a that is coprime to 5 (e.g., 1, 2, 3, 4), there is an integer z such that $2^z \bmod (5) = a$.

For g to be a primitive root mod p , there are two conditions:

- $1 < g < p$
- The following modular results must be distinct:
 - $g^1 \bmod p$
 - $g^2 \bmod p$
 - \dots
 - $g^{p-1} \bmod p$

Primitive roots

Example: To verify that 2 is a primitive root of prime number 5:

- For every integer coprime to 5, there must be a power of 2 that is congruent
- Integers that are coprimes to 5: 1, 2, 3, 4
- $z = 1, 2^1 \text{ mod}(5) = 2$
- $z = 2, 2^2 \text{ mod}(5) = 4$
- $z = 3, 2^3 \text{ mod}(5) = 3$
- $z = 4, 2^4 \text{ mod}(5) = 1$
- Verified: For 1, 2, 3, 4, there is a power of 2 that is congruent

Primitive roots



Question: Is 3 a primitive root of prime number 7?

Primitive roots



Question: Is 3 a primitive root of prime number 7?

Answer: Yes, 3 is a primitive root of 7.

- For every integer coprime to 7, there is a power of 3 that is congruent.
- Integers that are coprimes to 7: 1, 2, 3, 4, 5, 6
- $3^1 \text{ mod}(7) = 3$
- $3^2 \text{ mod}(7) = 2$
- $3^3 \text{ mod}(7) = 6$
- $3^4 \text{ mod}(7) = 4$
- $3^5 \text{ mod}(7) = 5$
- $3^6 \text{ mod}(7) = 1$
- Verified: For 1, 2, 3, 4, 5, 6, there is a power of 3 that is congruent

DF algorithm

Step 1: Alice and Bob agree on some **public parameters**

- a (large) prime number p
- an integer g

Example

- $p = 17$
- $g = 3$



The integer g must be a primitive root to the prime number p .

DF algorithm

Step 2: Alice and Bob pick their **secret integer** to come up with a **public integer**

Alice's private computation

- Select a secret integer a , $a < p$
- Compute the public integer $A = g^a \text{ mod}(p)$

Example

Alice's private computation

- $a = 15$, $15 < 17$
- $A = 3^{15} \text{ mod } (17) = 6$

The public integer A will be shared, but the integer a remains secret.



DF algorithm

Step 2: Alice and Bob pick their **secret integer** to come up with a **public integer**

Alice's private computation

- Select a secret integer a , $a < p$
- Compute the public integer $A = g^a \text{ mod}(p)$

Bob's private computation

- Select a random number b , $b < p$
- Compute the public integer $B = g^b \text{ mod}(p)$

Example

Alice's private computation

- $a = 15$, $15 < 17$
- $A = 3^{15} \text{ mod } (17) = 6$

Bob's private computation

- $b = 13$, $13 < 17$
- $B = 3^{13} \text{ mod } (17) = 12$

DF algorithm

Step 3: Alice and Bob **exchange their own public integer**

Bob sends B to Alice. Alice has access to

- a , her secret integer
- B , Bob's public integer

Example

Alice has:

- $a = 15, A = 6$
- $B = 12$

DF algorithm

Step 3: Alice and Bob **exchange their own public integer**

Bob sends B to Alice. Alice has access to

- a , her secret integer
- B , Bob's public integer

Alice sends A to Bob. Bob has access to:

- b , his secret integer
- A , Alice's public integer

Example

Alice has:

- $a = 15, A = 6$
- $B = 12$

Bob's private computation

- $b = 13, B = 12$
- $A = 6$

DF algorithm

Step 4: Alice and Bob **compute the shared secret key**

Alice's private computation

- Compute the shared secret key
 $K = B^a \text{ mod}(p)$

Example

Alice computes:

- $K = 12^{15} \text{ mod } (17) = 10$

The diagram illustrates the computation process. On the left, under 'Alice's private computation', there is a list of steps. An arrow points from the text 'Compute the shared secret key' to a box containing the derivation of the shared secret key. The box is divided into three sections: 'Shared secret key', 'Bob's public integer B', and 'Therefore:'. The 'Shared secret key' section contains the formula $K = g^{ab} \text{ mod}(p)$. The 'Bob's public integer B' section contains the formula $B = g^b \text{ mod}(p)$. The 'Therefore:' section contains the final formula $K = (g^b)^a \text{ mod}(p) = B^a \text{ mod }(p)$.

Shared secret key

- $K = g^{ab} \text{ mod}(p)$

Bob's public integer B

- $B = g^b \text{ mod}(p)$

Therefore:

- $K = (g^b)^a \text{ mod}(p) = B^a \text{ mod }(p)$

DF algorithm

Step 4: Alice and Bob **compute the shared secret key**

Alice's private computation

- Compute the shared secret key
 $K = B^a \text{ mod}(p)$

Bob's private computation

- Compute the shared secret key
 $K = A^b \text{ mod}(p)$

Example

Alice computes:

- $K = 12^{15} \text{ mod } (17) = 10$

Bob computes:

- $K = 6^{13} \text{ mod } (17) = 10$

DF algorithm cheat sheet

DF algorithm as a 4-steps process:

Step 1: Alice and Bob agree on some **public parameters**

Step 2: Alice and Bob come up with a **public integer**

Step 3: Alice and Bob **exchange their own public integer**

Step 4: Alice and Bob **compute the shared secret key**

Public integers

- $A = g^a \text{ mod}(p)$
- $B = g^b \text{ mod}(p)$

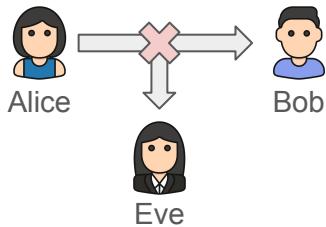
Shared secret key

- $K = g^{ab} \text{ mod}(p) = A^b \text{ mod}(p)$
- $K = g^{ba} \text{ mod}(p) = B^a \text{ mod}(p)$

DF algorithm

Can Eve come up with the shared secret key?

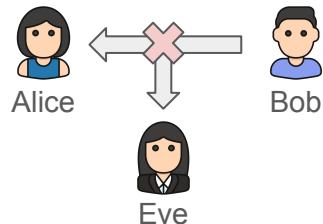
- Eve has access to the public parameters (p, g), and the public integers (A, B), but not the secret integers (a and b), which are required to find the secret key.



$$A = g^a \text{ mod}(p)$$

$$A = g^? \text{ mod}(p)$$

A Discrete Logarithm Problem (DLP)



$$B = g^b \text{ mod}(p)$$

$$B = g^? \text{ mod}(p)$$

A Discrete Logarithm Problem (DLP)

Discrete Logarithm Problem (DLP)

A discrete log problem is defined as:

Given $g^a \bmod(p) = b$, solve for a
where a is a primitive root and p is a prime number.

Solving a discrete log problem is hard, because of the primitive root property

- Given any exponent, the solution to $g^a \bmod(p)$ is uniformly distributed
- Each solution is equally likely to happen

DF algorithm

Step 1: Alice and Bob agree on some **public parameters**

- a (large) prime number p
- an integer g

Example

- $p = 17$
- $g = 3$

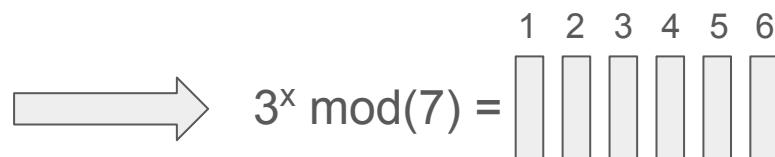


The integer g must be a primitive root to the prime number p .

Primitive roots

Example: To verify that 3 is a primitive root of prime number 7:

- Coprimes of 7: 1, 2, 3, 4, 5, 6
- $3^1 \text{ mod}(7) = 3$ • $3^7 \text{ mod}(7) = 3$
- $3^2 \text{ mod}(7) = 2$ • $3^8 \text{ mod}(7) = 2$
- $3^3 \text{ mod}(7) = 6$ • $3^9 \text{ mod}(7) = 6$
- $3^4 \text{ mod}(7) = 4$ • $3^{10} \text{ mod}(7) = 4$
- $3^5 \text{ mod}(7) = 5$ • $3^{11} \text{ mod}(7) = 5$
- $3^6 \text{ mod}(7) = 1$ • $3^{12} \text{ mod}(7) = 1$

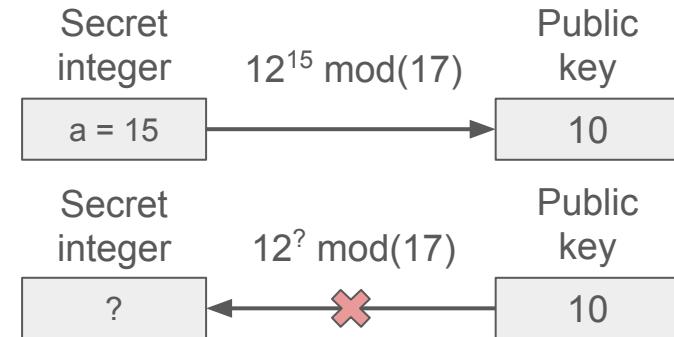


Property of a primitive root: when raised to an exponent, the solution is distributed uniformly.

Discrete Logarithm Problem (DLP)

For Eve to find the secret integer, she needs to solve the discrete logarithm problem, however

- It is easy to calculate the public key
- Hard to find the secret integer (many solutions)



The idea is similar to the shared secret color solution:

- Easy to mix a secret color with the starting one to get a mixture
- Hard to find the secret color from the mixed color (many color combinations)

Question on DF algorithm



Suppose that Alice and Bob agree on $g = 2$ and $p = 19$.

Question: What is the secret key if Alice chooses 6 and Bob chooses 8 as their respective secret integers? (show the calculations for both Alice and Bob)

Public integers

- $A = g^a \text{ mod}(p)$
- $B = g^b \text{ mod}(p)$

Shared secret key

- $K = g^{ab} \text{ mod}(p) = A^b \text{ mod}(p)$
- $K = g^{ba} \text{ mod}(p) = B^a \text{ mod}(p)$

Question on DF algorithm



Suppose that Alice and Bob agree on $g = 2$ and $p = 19$.

Question: What is the secret key if Alice chooses 6 and Bob chooses 8 as their respective secret integers? (show the calculations for both Alice and Bob)

Intuition: Given the value of g , p , a and b , calculate K

Public integers

- $A = g^a \text{ mod}(p)$
- $B = g^b \text{ mod}(p)$

Shared secret key

- $K = g^{ab} \text{ mod}(p) = A^b \text{ mod}(p)$
- $K = g^{ba} \text{ mod}(p) = B^a \text{ mod}(p)$

Question on DF algorithm



Suppose that Alice and Bob agree on $g = 2$ and $p = 19$.

Question: What is the secret key if Alice chooses 6 and Bob chooses 8 as their respective secret integers? (show the calculations for both Alice and Bob)

Answer: Calculate the public integers A and B to find out K.

For Alice:

- $A = g^a \text{ mod}(p) = 2^6 \text{ mod } (19) = 7$
- $K = B^a \text{ mod}(p) = 9^6 \text{ mod } (19) = 11$

For Bob:

- $B = g^b \text{ mod}(p) = 2^8 \text{ mod } (19) = 9$
- $K = A^b \text{ mod}(p) = 7^8 \text{ mod } (19) = 11$

Public integers

- $A = g^a \text{ mod}(p)$
- $B = g^b \text{ mod}(p)$

Shared secret key

- $K = g^{ab} \text{ mod}(p) = A^b \text{ mod}(p)$
- $K = g^{ba} \text{ mod}(p) = B^a \text{ mod}(p)$

TODOs

- Project 2 Hints and FAQs are posted under Week 6
 - Make sure to check it out, send your questions on Slack
- No class on Monday, March 4
 - [Students' Union Election Forum](#)

Lecture 20

The Dining Philosophers Problem

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last classes
- The Dining Philosophers Problem
 - Race condition
 - Deadlocks
 - Starvation
- A solution to the Dining Philosophers Problem
 - Atomic locks for resource reservation
 - Critical section
- TODOs
 - Distributing the midterm

Diffie-Hellman algorithm

Diffie-Hellman is a **key exchange algorithm** used with symmetric encryption.

- It allows both parties to agree on an identical secret key
- Without having to share the actual key in the communication channel
- This is used for key exchange, and not encryption/decryption

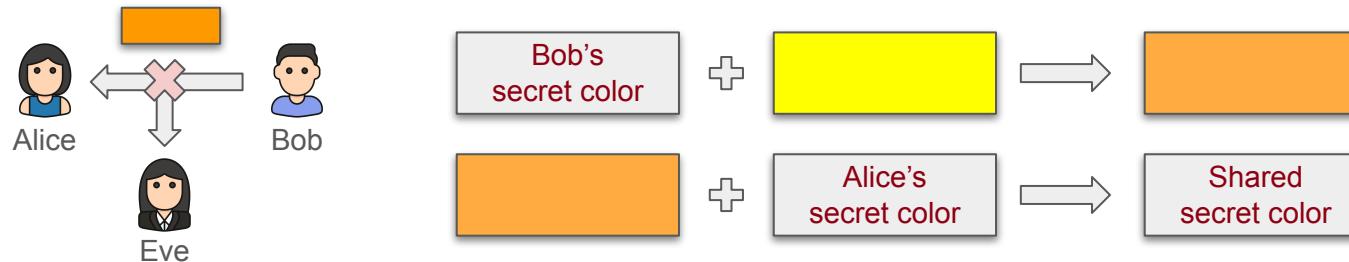
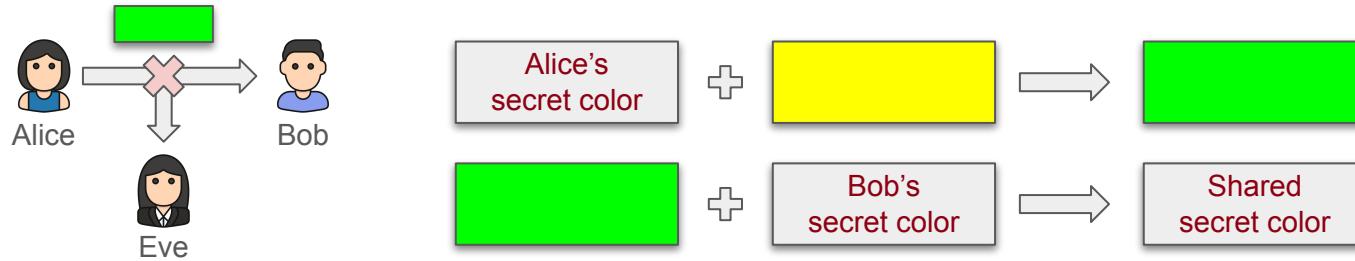
Overall idea behind DF key exchange algorithm:

- Neither parties choose the key explicitly
- Both parties contribute in calculating the secret key together
- The calculated secret key can then be used in symmetric encryption.

Analogous to finding a shared secret color

Can Eve come up with the shared secret color, brown?

- Eve has access to the starting color (yellow), and the mixed colors (green and orange), but not the secret colors (red and blue).



DF algorithm cheat sheet

DF algorithm as a 4-steps process:

Step 1: Alice and Bob agree on some **public parameters**

Step 2: Alice and Bob come up with a **public integer**

Step 3: Alice and Bob **exchange their own public integer**

Step 4: Alice and Bob **compute the shared secret key**

Public integers

- $A = g^a \text{ mod}(p)$
- $B = g^b \text{ mod}(p)$

Shared secret key

- $K = g^{ab} \text{ mod}(p) = A^b \text{ mod}(p)$
- $K = g^{ba} \text{ mod}(p) = B^a \text{ mod}(p)$

Schedule for today

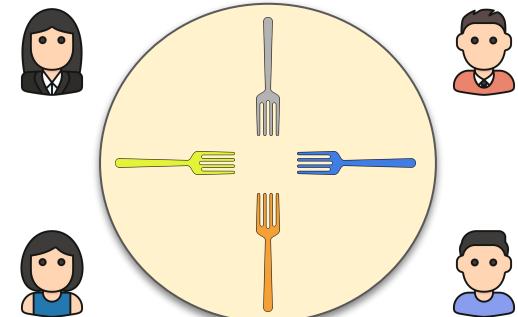
- Key concepts from last classes
- The Dining Philosophers Problem
 - Race condition
 - Deadlocks
 - Starvation
- A solution to the Dining Philosophers Problem
 - Atomic locks for resource reservation
 - Critical section
- TODOs
 - Distributing the midterm

The Dining Philosophers Problem

Dining Philosophers Problem is a classical synchronization problem in the operating system.

- Philosophers can either **think** or **eat**
- To **eat**, philosophers needs both their left and right forks
 - Two forks will only be available when the two nearest neighbors are thinking, not eating
 - If not available, they start **thinking**
 - After they are done **eating**, they will put down both forks
- To **think**, philosophers does nothing but thinking

Goal: Designing a solution (algorithm) so that no philosopher **starves**.



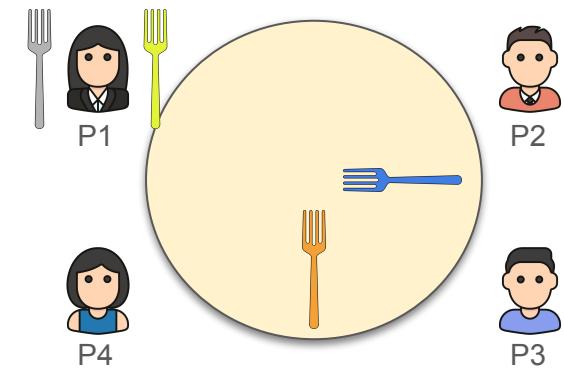
Basic protocol

```
while (true) {  
    grab left fork;  
    grab right fork;  
    eat;  
    release left fork;  
    release right fork;  
    think;  
}
```

Example for P1:

- P1 grabs the left and right fork
- P1 eats
- P1 put the left and right fork down
- P1 thinks

What problems
can happen here?



Challenges in the Dining Philosophers Problem

There are two challenges in solving the Dining Philosophers Problem:

- Race condition
- Deadlock

Race condition

A **race condition** may happen when philosophers attempt to grab the same fork at the same time:

- However, the correctness of the process depends on timing
- The action of one philosopher can intervene during another philosopher's final decision (grabbing or not grabbing) that involves the shared fork
- Therefore, we want to **avoid race condition**

Example: Between checking the availability of the fork and grabbing the fork, another philosopher intervenes, making the check out of date and the action incorrect

Example of race condition

P1 decides to eat

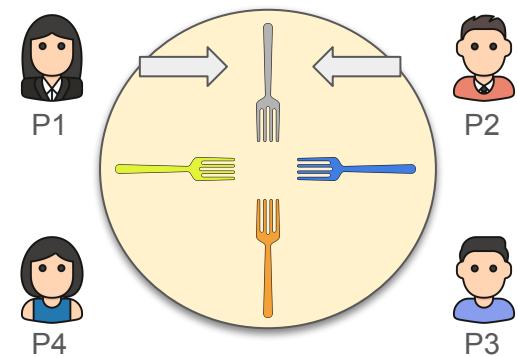
- P1 checks if the grey fork is available
- It is available, therefore P1 will try to grab the grey fork

At the same time, P2 decides to eat

- P2 checks if the grey fork is available
- Because the check happens before P1 actually grabs the grey fork, the fork will also be available to P2

Race condition detected!

- P2 cannot grab the fork as P1 will grab the fork before



Deadlock

Assume that the action of the philosophers are **perfectly interleaved**:

- Each philosopher grabs the fork on their right
- Then, they try to grab the fork on their left

Deadlock

Assume that the action of the philosophers are **perfectly interleaved**:

- Each philosopher grabs the fork on their right
- Then, they try to grab the fork on their left

However, as all philosophers grab their right fork at the same time, their left is gone.

- Each philosopher waits for the person on their left to release the fork, so they can start eating
- But it will never happen because of a circular chain where the person next to them is also waiting
- This is the problem of **deadlock**

Example of deadlock

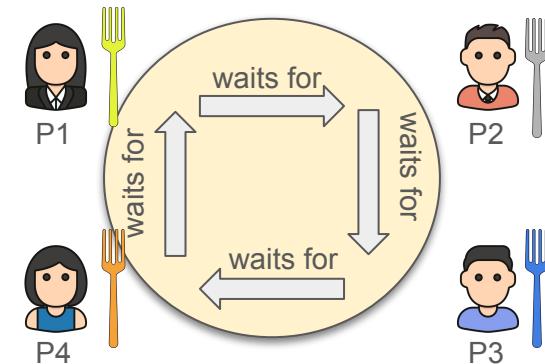
Step 1: Each philosopher grabs their right fork at the same time

- P1 grabs the yellow fork; P2 grabs the grey fork; ...

Step 2: Each philosopher grabs their left fork at the same time

- P1 tries to grab the grey fork which is taken, so P1 starts to wait (think);
- P2 tries to grab the blue fork which is taken, so P2 starts to wait (think);
- ...

All philosophers hold a fork but are unable to eat.
They will eventually all **starve**.



A process synchronization problem

The Dining Philosophers Problem illustrates synchronization issues in concurrent processes.

- Philosophers = Processes, programs or threads
- Forks = Shared resources (e.g., files, memory)
- Solution = Algorithm for resource allocation

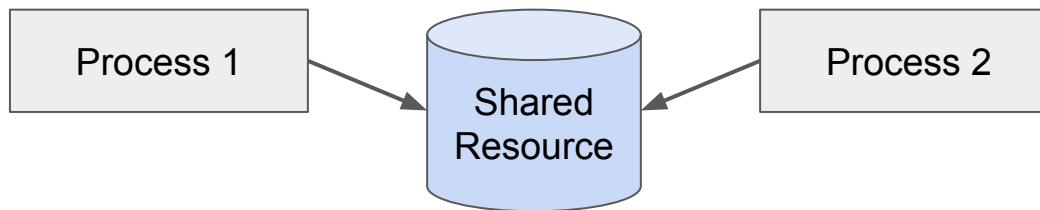
Lack of forks is an analogy to limited (shared) resources in computer programming

- Avoid race condition where some processes may never access the resource
- Avoid deadlocks where processes are mutually locking each other out

Race condition

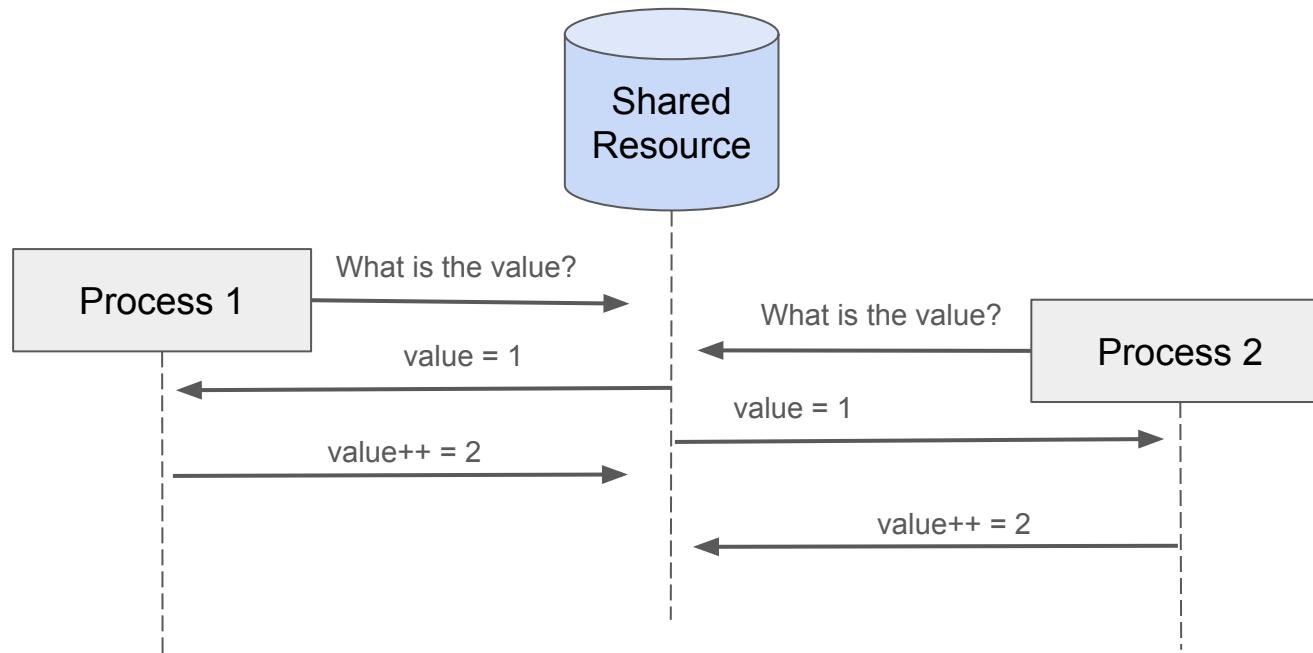
A **race condition** happens when two processes attempt to access the same resource at the same time. However, the timing or ordering of events affects a program's correctness.

- E.g., Two processes that attempt to increase a shared value by 1
- Instead of increasing the value twice, the value is only increased by 1 where the last modification is preserved



Cause of race condition

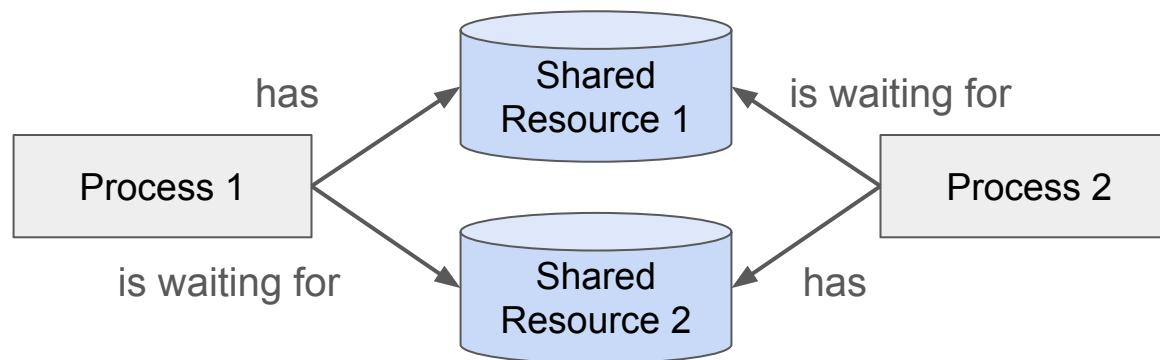
Between checking on a critical resource and acting on that check, another process intervenes, making that check out of date, and the action becomes incorrect.



Deadlock

Deadlock describes a situation in which two processes, sharing the same resources, are preventing each other from accessing the resources.

- Each process is holding the resource the other is waiting for
- But no one is releasing the resource until the other releases it



Necessary conditions for deadlock

Deadlock can only occur in systems where all following conditions are met:

- **Mutual Exclusion**
 - A resource cannot be used by more than one process at a time
- **Hold and Wait**
 - At least one process holds one resource while waiting for another
- **No Preemption**
 - No other process can force one process to release the resource
- **Circular Wait**
 - Two or more processes form a circular chain where each process waits for a resource that the next process in the chain holds

Questions on deadlock and race condition



Multiple-choice question: A program containing a race condition will always/sometimes/never result in some incorrect behavior?

True/false question: Every deadlock is always starvation but every starvation is not a deadlock.

Questions on deadlock and race condition



Multiple-choice question: A program containing a race condition will always/sometimes/never result in some incorrect behavior?

Answer:

True/false question: Every deadlock is always starvation but every starvation is not a deadlock.

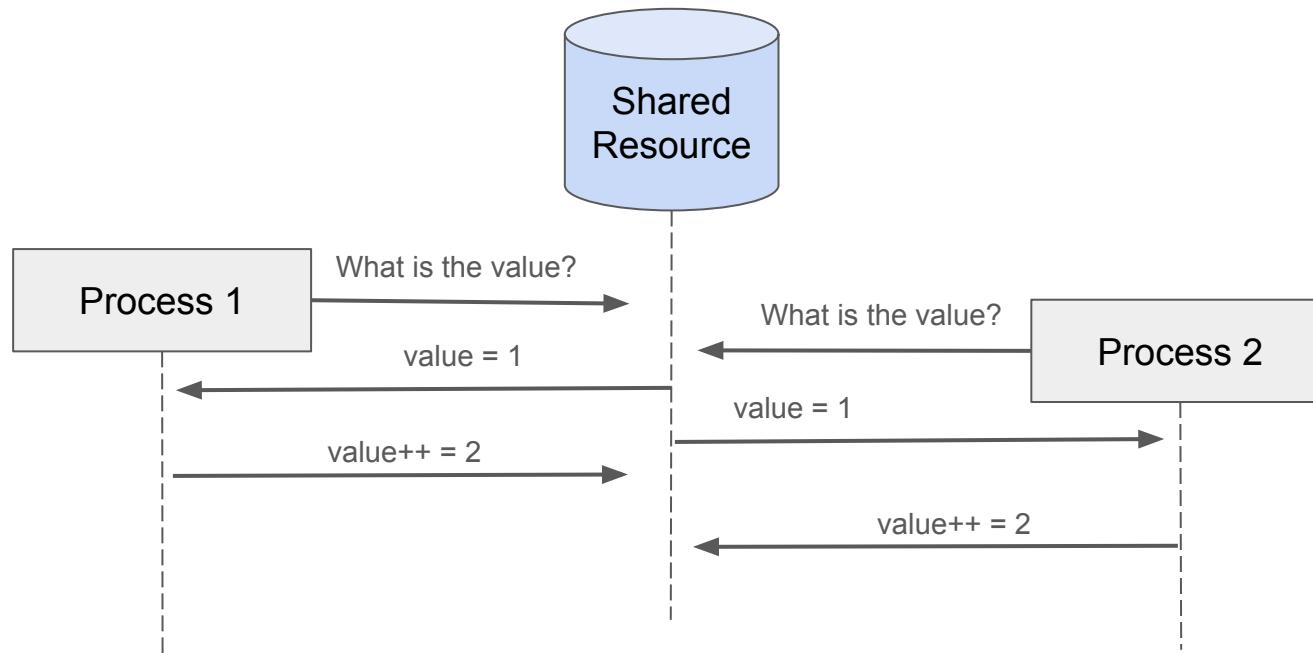
Answer:

Schedule for today

- Key concepts from last classes
- The Dining Philosophers Problem
 - Race condition
 - Deadlocks
 - Starvation
- A solution to the Dining Philosophers Problem
 - Atomic locks for resource reservation
 - Critical section
- TODOs
 - Distributing the midterm

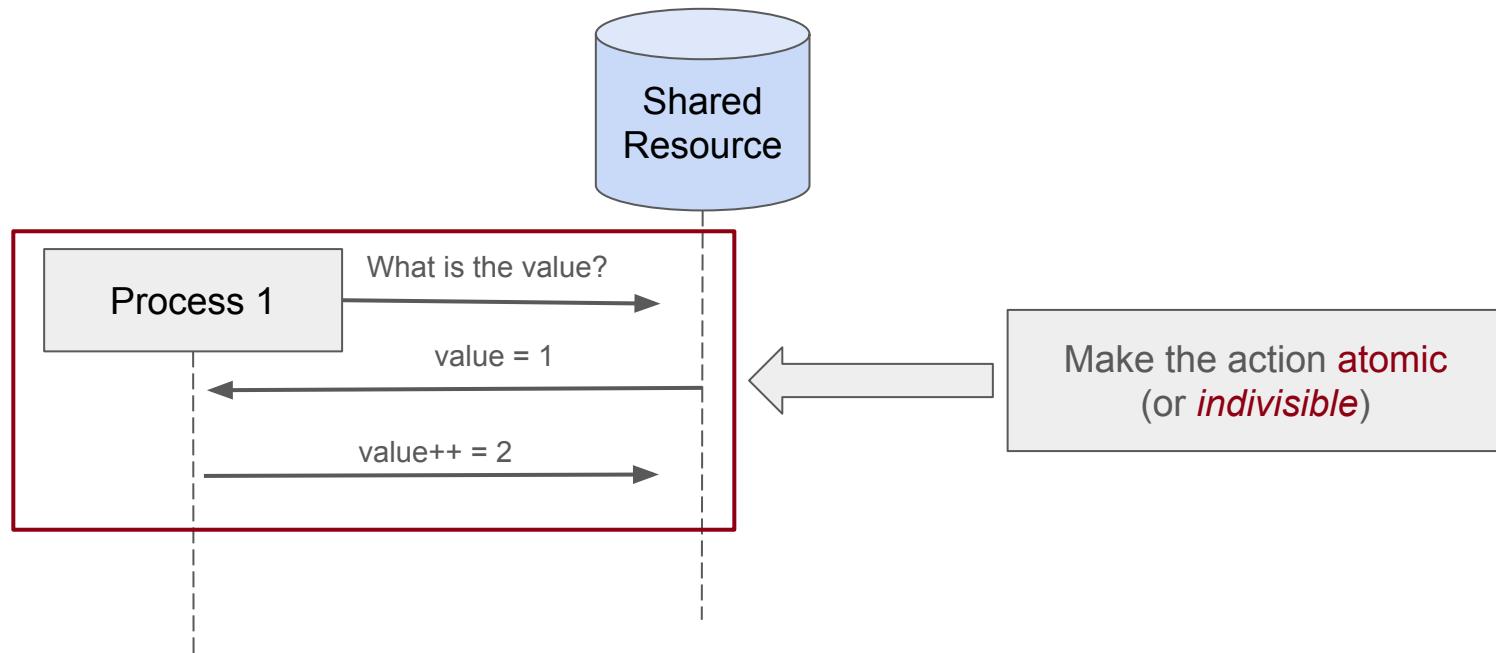
Cause of race condition

One process may intervene during another process' execution, making the check on the critical resource out of date. This is why the action becomes incorrect.



Avoiding race conditions

We must guarantee that no process can intervene during another process' manipulation that involves shared resources.



Deadlock Prevention

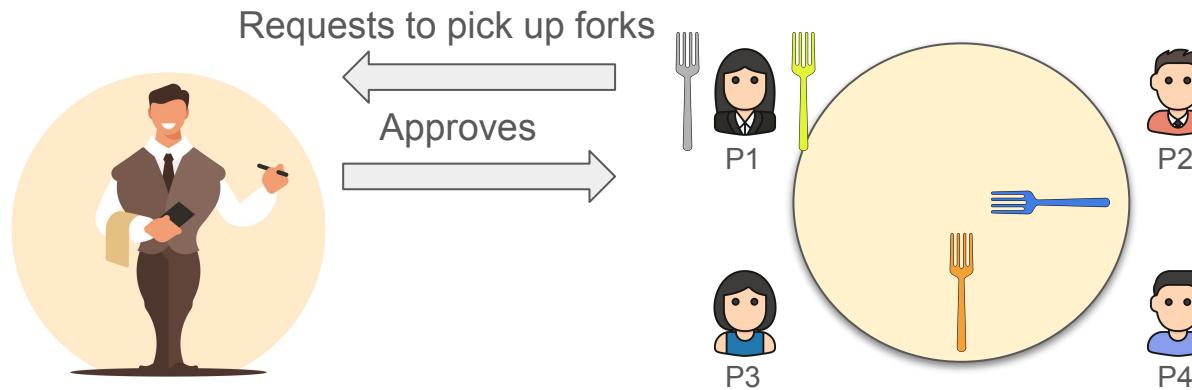
To prevent deadlock, we only need to ensure one of the conditions does not hold:

- Mutual Exclusion
 - Make the resource sharable
 - However, not all resources are sharable, e.g., editing files, analogous to forks
- Hold and Wait
 - Make processes request all resources at once and make them release all resources once done
- No Preemption
 - Make processes release all resources if the request cannot be proceed immediately
 - However, not all resources can be easily preempted, e.g., printing jobs
- Circular Wait
 - Impose an ordering on the resources and processes can only request them in order
 - However, waste of resources for such turn-based solution

Solution: “locking” forks

Introducing a waiter to monitor which forks are been used:

- Before eating, philosophers will ask for permission to pick up **both forks**
- If both forks are available, the waiter will give the permission to do so

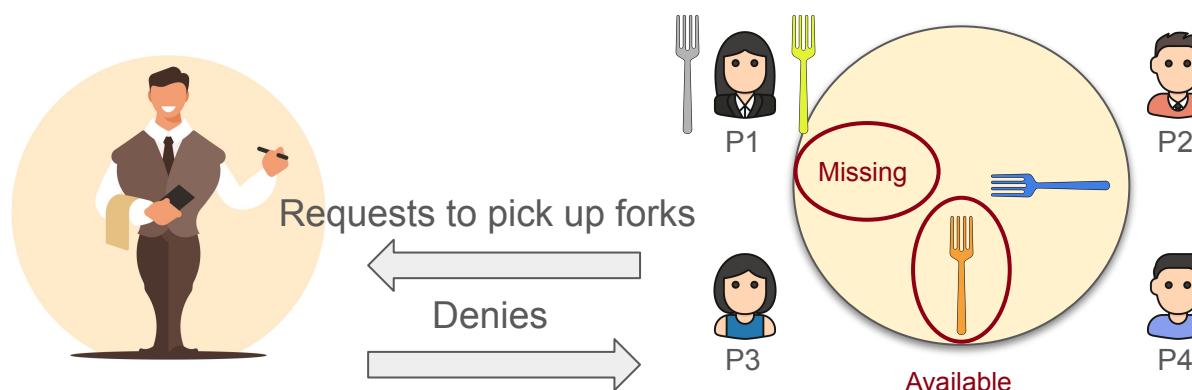


Intuition: Make philosophers request both forks at once,
at the same time, each request must be atomic.

Example of “locking” forks

Given that P1 picked up the forks:

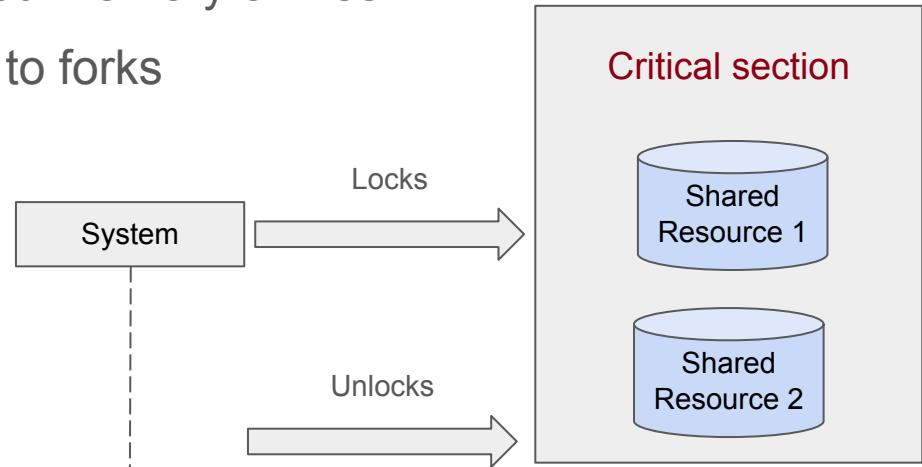
- If P3 (or P2) requests to pick up their forks, their request will be denied
- Because one of the forks is unavailable



Locks for resource reservation

Locks provide a mechanism to prevent multiple processes from accessing the resources in the critical section at the same time.

- **Critical section** is the part of a program which must be executed by only one process at a time to avoid race conditions
- Example of critical section: shared memory or files
- The critical section is analogous to forks



Locks for resource reservation

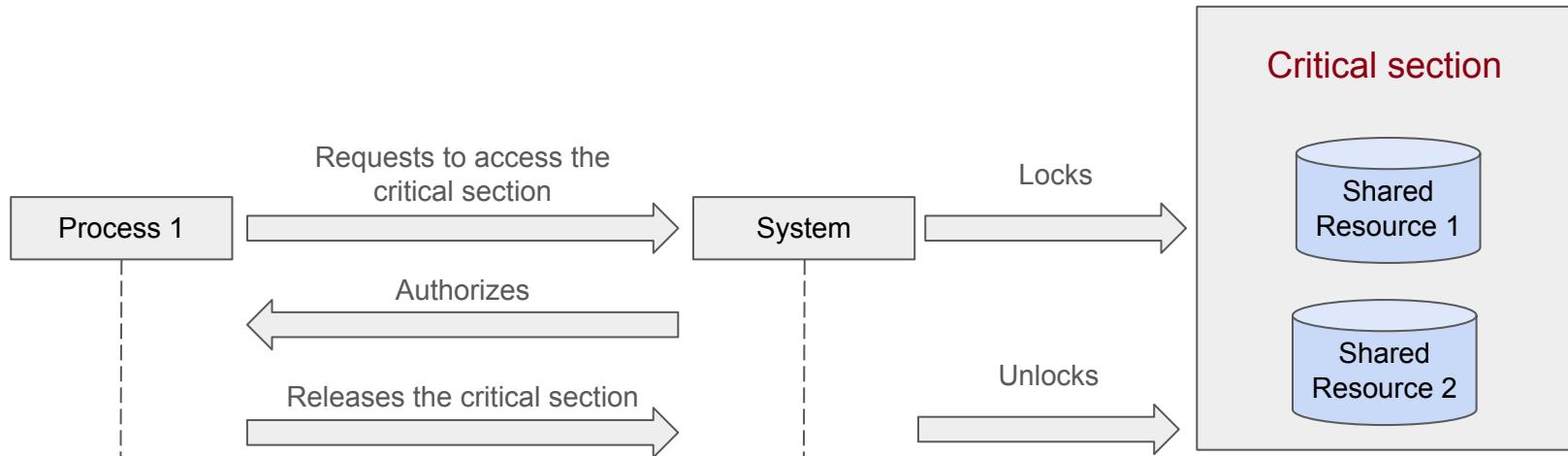
Generic implementation of locks:

- Lock before entering a critical section
- Unlock when leaving a critical section
- Wait and retry if the critical section is locked
- Lock is initially set free

Solution: atomic locking

Introducing atomic locks to the resources:

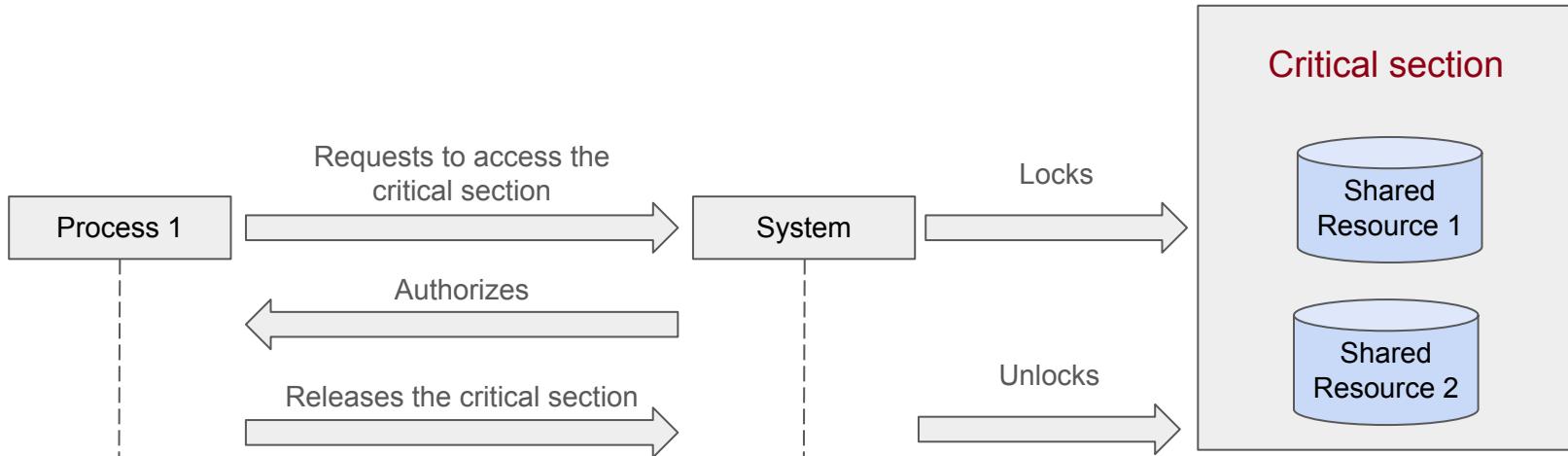
- Ensure that the resource is accessed by only one process at a time
- If the lock is free, the system will allow the process to access the resources and lock the critical section



Solution: atomic locking

Introducing **atomic locks** to the resources:

The atomic property guarantees that the resource remains locked until Process 1 finishes all its execution.



Other solutions?

- Resource hierarchy solution
- Arbitrator solution
- Chandy/Misra solution
- ... and more

Question: Why can't the philosophers just get more forks?

Answer: But more resources (forks) are expensive. The Dining Philosophers is used to illustrate a **synchronization problem using the same resources**.

One more question on deadlock



Multiple-choice question: A system that meets the four deadlock conditions will always/sometimes/never result in deadlock?

Next class: introducing the resource allocation graph to answer this question.

TODOs

- Midterm distribution
 - Resources = Classroom tables for distributing the midterms
 - Processes = Students picking up their midterms
 - To have atomic actions = Each student searches for his/her midterm without interference
 - To avoid race condition = Multiple students cannot access the tables at the same time
 - To avoid starvation = Avoid having 73 students waiting on 1 student

Solution? More resources, more instances

- Version A, Last name starting with A to M (Resource 1, instance 1)
- Version A, Last name starting with N to Z (Resource 1, instance 2)
- Version B, Last name starting with A to M (Resource 2, instance 1)
- Version B, Last name starting with N to Z (Resource 2, instance 2)

Lecture 21

Deadlocks

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

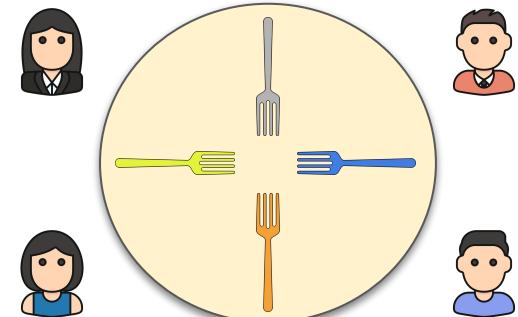
- Key concepts from last classes
- Race condition on reliability
- Deadlock avoidance
 - Resource allocation graph
 - Resource allocation table
 - Resource availability

The Dining Philosophers Problem

Dining Philosophers Problem is a classical synchronization problem in the operating system.

- Philosophers can either **think** or **eat**
- To **eat**, philosophers needs both their left and right forks
 - Two forks will only be available when the two nearest neighbors are thinking, not eating
 - If not available, they start **thinking**
 - After they are done **eating**, they will put down both forks
- To **think**, philosophers does nothing but thinking

Goal: Designing a solution (algorithm) so that no philosopher **starves**.



Example of race condition

P1 decides to eat

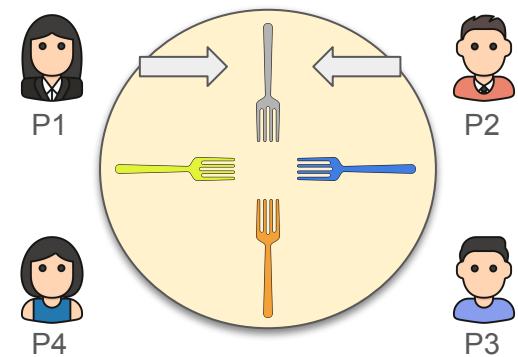
- P1 checks if the grey fork is available
- It is available, therefore P1 will try to grab the grey fork

At the same time, P2 decides to eat

- P2 checks if the grey fork is available
- Because the check happens before P1 actually grabs the grey fork, the fork will also be available to P2

Race condition detected!

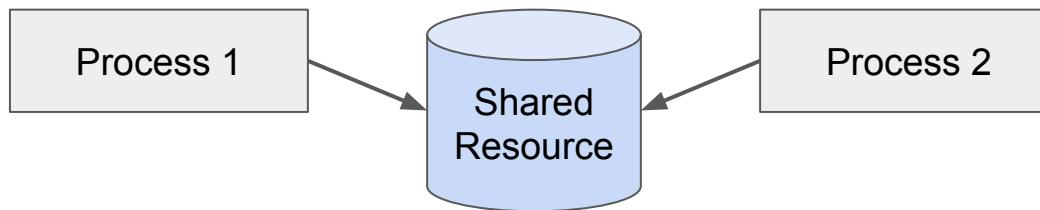
- P2 cannot grab the fork as P1 will grab the fork before



Race condition

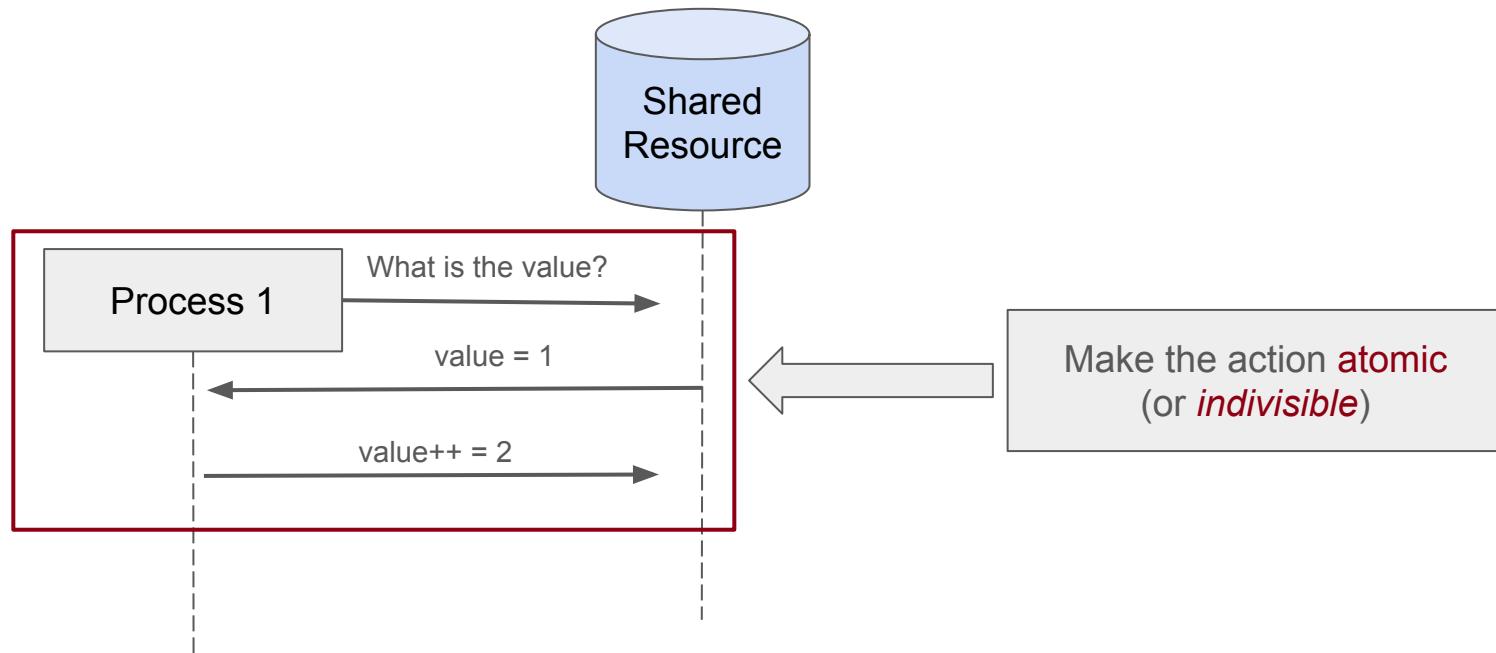
A **race condition** happens when two processes attempt to access the same resource at the same time. However, the timing or ordering of events affects a program's correctness.

- E.g., Two processes that attempt to increase a shared value by 1
- Instead of increasing the value twice, the value is only increased by 1 where the last modification is preserved



Avoiding race conditions

We must guarantee that no process can intervene during another process' manipulation that involves shared resources.



Race condition on reliability

Detecting and identifying race condition in practice is difficult but important.

- With race condition vulnerabilities, the system may suffer from **denial-of-service attack**

For example:

- Hackers could ask the program to execute multiple operations concurrently
- If a deadlock occurs, then the system risks of stop responding and eventually crashing

Example of reliability issue: Northeast Blackout of 2003

- Power outage throughout Northeastern United States and most parts of Ontario
 - 55 million people affected
 - Duration: 2 hours–4 days, depending on location
 - At the time, it was the world's second most widespread blackout in history
- Race condition in the energy management system
 - Some unique combination of events and alarm conditions triggered the race condition
 - Because of the race condition, the system was sending an system alarm event into an infinite loop
 - Within thirty minutes, the server went down as there was too much events
 - Then, the backup server also went down



Example of deadlock

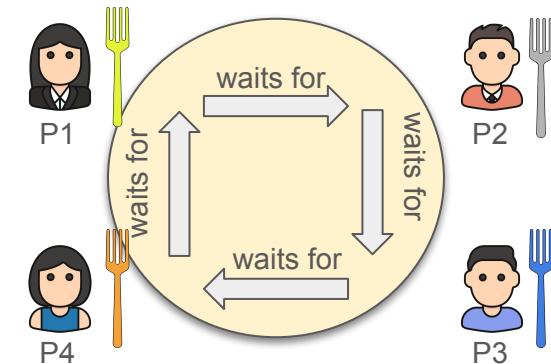
Step 1: Each philosopher grabs their right fork at the same time

- P1 grabs the yellow fork; P2 grabs the grey fork; ...

Step 2: Each philosopher grabs their left fork at the same time

- P1 tries to grab the grey fork which is taken, so P1 starts to wait (think);
- P2 tries to grab the blue fork which is taken, so P2 starts to wait (think);
- ...

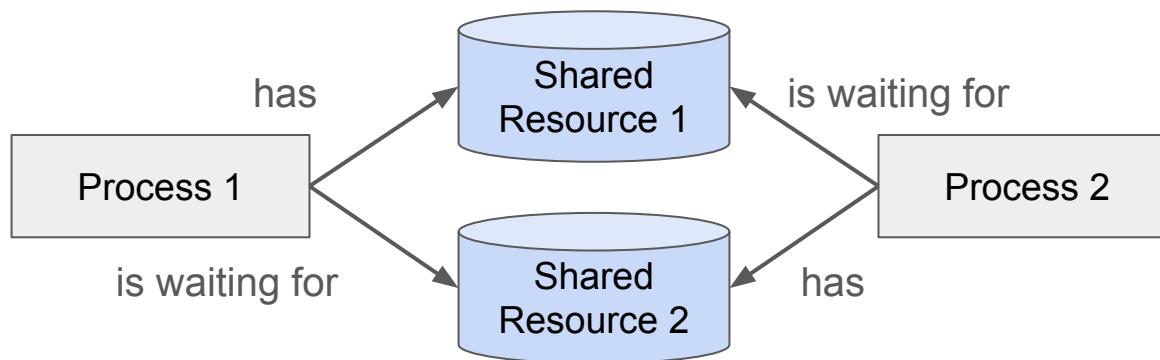
All philosophers hold a fork but are unable to eat.
They will eventually all **starve**.



Deadlock

Deadlock describes a situation in which two processes, sharing the same resources, are preventing each other from accessing the resources.

- Each process is holding the resource the other is waiting for
- But no one is releasing the resource until the other releases it



Deadlock Prevention

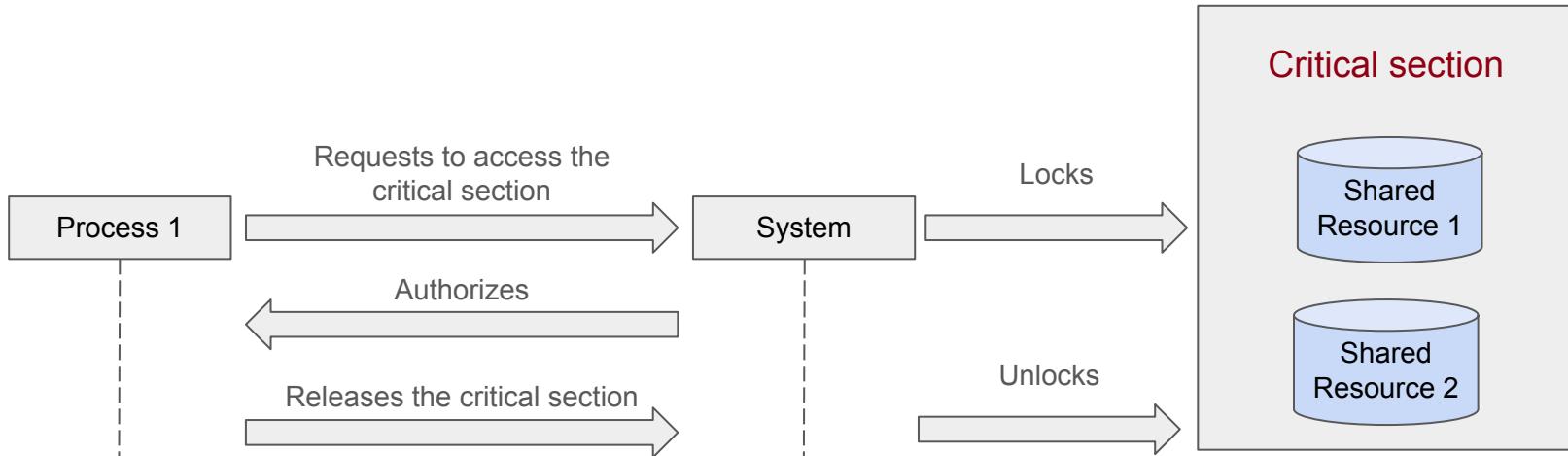
To prevent deadlock, we only need to ensure one of the conditions does not hold:

- Mutual Exclusion
 - Make the resource sharable
 - However, not all resources are sharable, e.g., editing files, analogous to forks
- Hold and Wait
 - Make processes request all resources at once and make them release all resources once done
- No Preemption
 - Make processes release all resources if the request cannot be proceed immediately
 - However, not all resources can be easily preempted, e.g., printing jobs
- Circular Wait
 - Impose an ordering on the resources and processes can only request them in order
 - However, waste of resources for such turn-based solution

Solution: atomic locking

Introducing **atomic locks** to the resources:

The atomic property guarantees that the resource remains locked until Process 1 finishes all its execution.



Schedule for today

- Key concepts from last classes
- Race condition on reliability
- **Deadlock avoidance**
 - Resource allocation graph
 - Resource allocation table
 - Resource availability

Deadlock avoidance

To **avoid deadlock**, we can examine the processes and resources to guarantee there can never be a circular-wait condition.

- System has access to information in advance about what resources will be needed by which processes
- System only approves resource requests if the process can obtain all resources it needs in future requests

However, it is tough to avoid deadlock in practice

- Hard to determine all the needed resources in advance
- Good problem statement in theory, but difficult to apply in practice

Deadlock avoidance

Deadlock avoidance requires that the system knows **a priori information** about the requests and needed resources.

There are two methods for deadlock avoidance:

- Resource Allocation Graph (RAG)
- Banker's Algorithm

Resource Allocation Graph (RAG)

Resource Allocation Graph (RAG) represents the state of the system as graphs. The graph contain a set of vertices and edges to describe the state of the system:

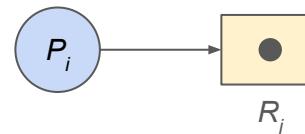
- Vertices (nodes) can either represent a process or resource
 - Process: $P = \{P_1, P_2, \dots, P_n\}$ is the set of processes in the system
 - Resource: $R = \{R_1, R_2, \dots, R_m\}$ is the set of resources in the system
- Edges can either represent a request or assignment
 - Request edge (directed edge): $P_i \rightarrow R_j$
 - Assignment edge (directed edge): $R_j \rightarrow P_i$

Resource Allocation Graph

- P_i , Process



- P_i requests instance of R_j

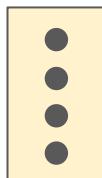


- R_j , Resource with instances

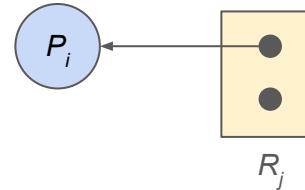
- Resource with a single instance



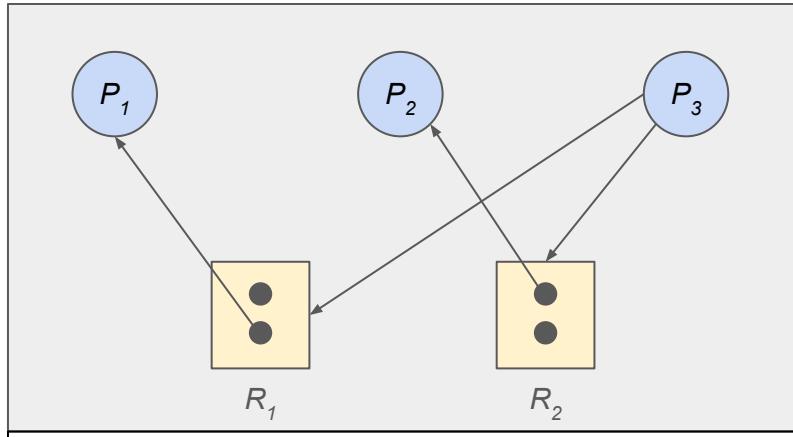
- Resource with 4 instances



- P_i is holding an instance of R_j



Example of Resource Allocation Graph



Vertices

- P_i , Process
- R_j , Resource with instances

Edges

- P_i requests instance of R_j
- P_i is holding an instance of R_j

Dependencies:

- P_1 is holding an instance of R_1
- P_2 is holding an instance of R_2
- P_3 is waiting for R_1 and R_2 .

Basic facts

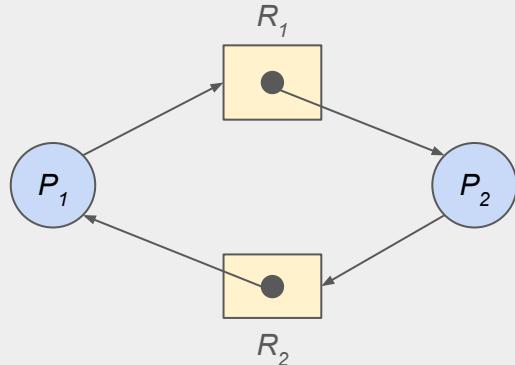
To determine if a resource allocation graph contains deadlock:

- If graph contains no cycle, then no deadlock
- If graph contains at least one cycle
 - If only one instance per resource, then deadlock
 - If several instances per resource, then possible deadlock

Basic facts

To determine if a resource allocation graph contains deadlock:

- If graph contains no cycle, then no deadlock
- If graph contains at least one cycle
 - If only one instance per resource, then deadlock
 - If several instances per resource, then possible deadlock

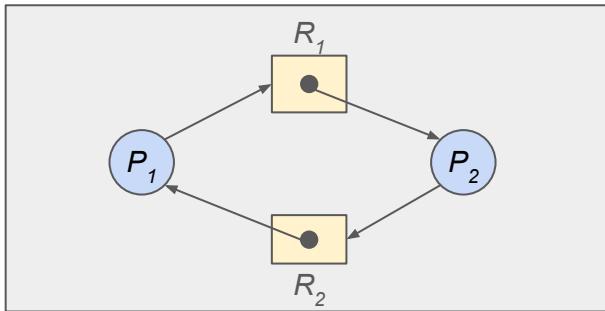


Example: processes in deadlock

- P1 holds R2
- P1 waits for R1
- P2 holds R1
- P2 waits for R2

Resource allocation table

The **resource allocation table** provides an overview of the allocated and requested resources.

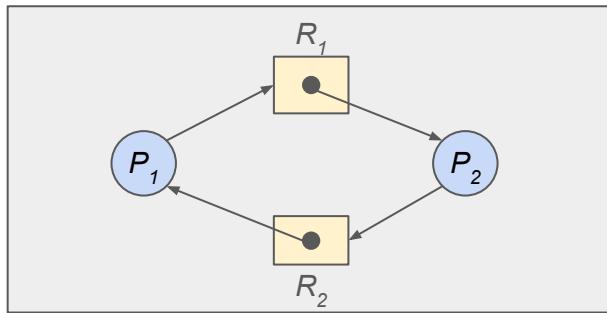


	Allocated		Requested	
	R1	R2	R1	R2
P1				
P2				

Example: Check for deadlock

Resource allocation table

The **resource allocation table** provides an overview of the allocated and requested resources.



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1		
P2	1	0		

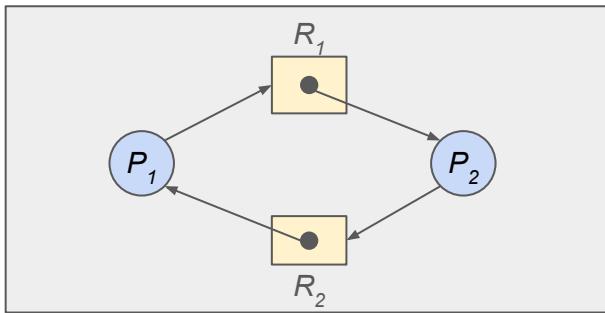
Example: Check for deadlock

Step 1: Fill in the allocated resources

- P1 is holding R2
- P2 is holding R1

Resource allocation table

The **resource allocation table** provides an overview of the allocated and requested resources.



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	1	0
P2	1	0	0	1

Example: Check for deadlock

Step 1: Fill in the allocated resources

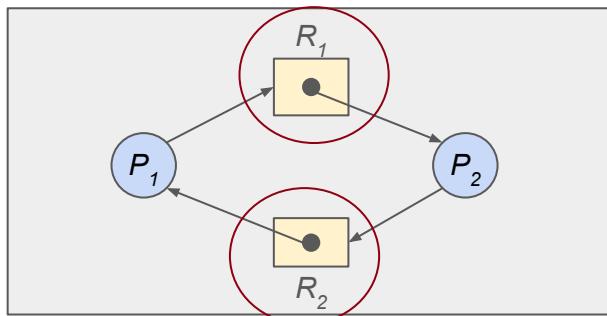
- P1 is holding R2
- P2 is holding R1

Step 2: Fill in the requested resources

- P2 is requesting for R2
- P1 is requesting for R1

Resource allocation table

The **resource allocation table** provides an overview of the allocated and requested resources.



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	1	0
P2	1	0	0	1

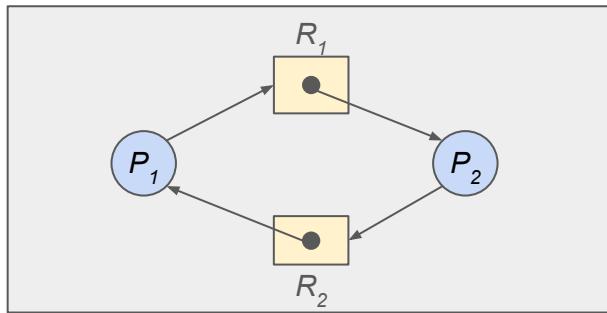
Example: Check for deadlock

Step 3: Check the availability of the resources

- Availability (R_1, R_2) = (0, 0)

Resource allocation table

The **resource allocation table** provides an overview of the allocated and requested resources.



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	1	0
P2	1	0	0	1

Example: Check for deadlock

Step 3: Check the availability of the resources

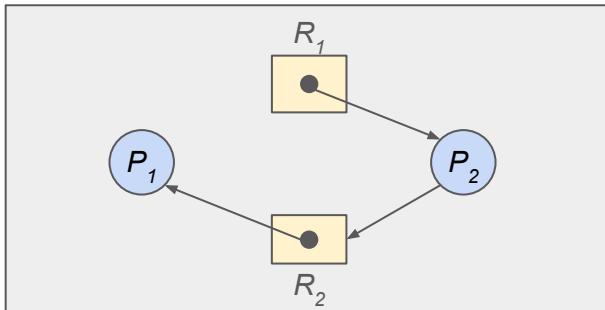
- Availability (R1, R2) = (0, 0)

Step 4: Free the requested resources (if applicable)

- With current availability, none of the requests can be fulfilled:
 - P1 Request on Availability (R1, R2) = (1, 0)
 - P2 Request on Availability (R1, R2) = (0, 1)
- Therefore, there is a **deadlock**

Another resource allocation table

One more example: Check for deadlock



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	0	0
P2	1	0	0	1

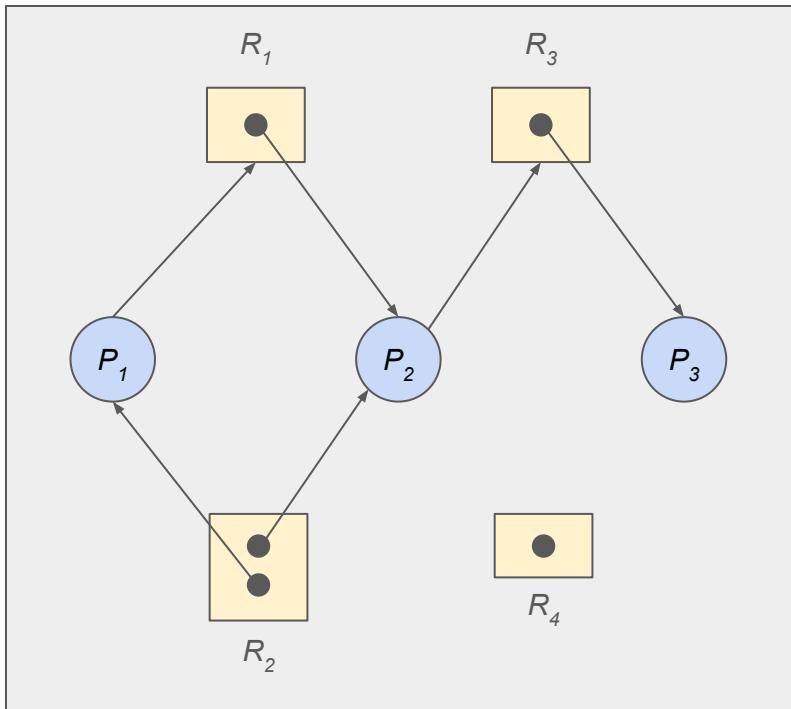
Step 3: Check the availability of the resources

- Availability (R1, R2) = (0, 0)

Step 4: Free the requested resources (if applicable)

- With current availability, none of the requests can be fulfilled:
 - P1 Request on Availability (R1, R2) = (0, 0), free P1
 - New Availability (R1, R2) = (0, 1)
 - P2 Request on Availability (R1, R2) = (0, 1), free P2
- Therefore, there is no **deadlock**

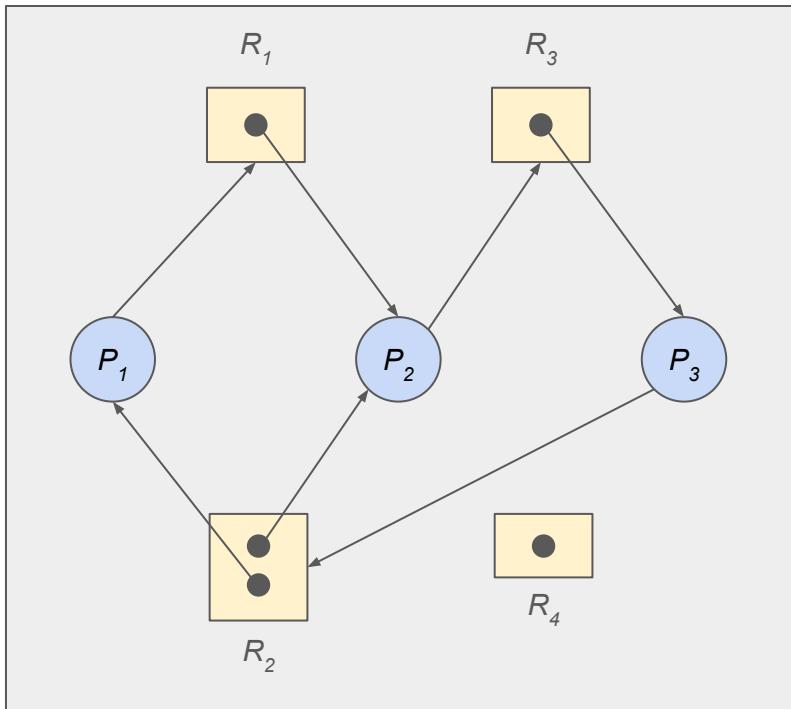
Example 1: Resource Allocation Graph



Question: Is there a deadlock?

- Check for cycles
 - If none, then no deadlock
- Check for the number of instances per resource in the cycle
 - If only one instance per resource, then deadlock
- Solve the resource allocation table

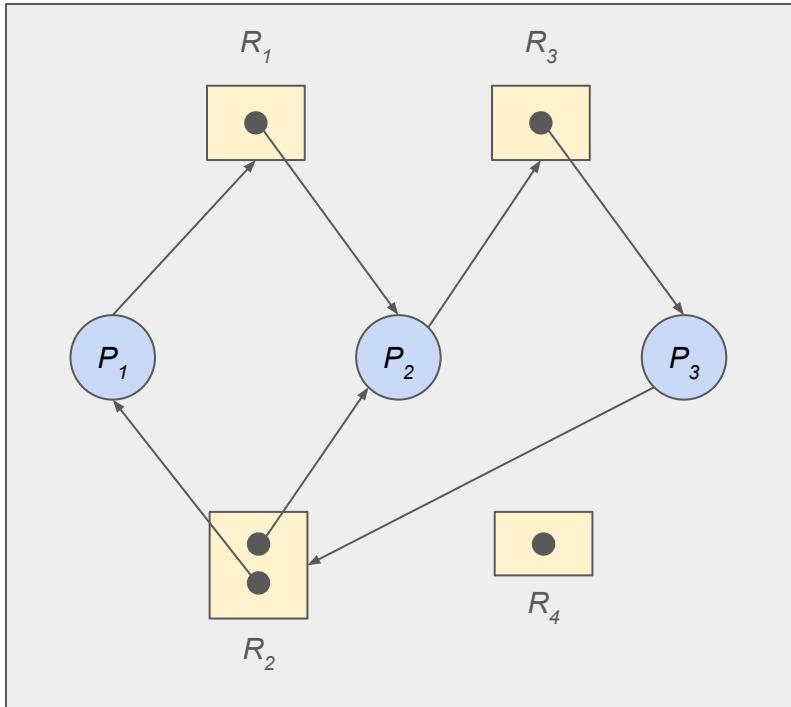
Example 2: Resource Allocation Graph



Question: Is there a deadlock?

- Check for cycles
 - If none, then no deadlock
- Check for the number of instances per resource in the cycle
 - If only one instance per resource, then deadlock
- Solve the resource allocation table

Example 2: Resource Allocation Graph



Question: Is there a deadlock?

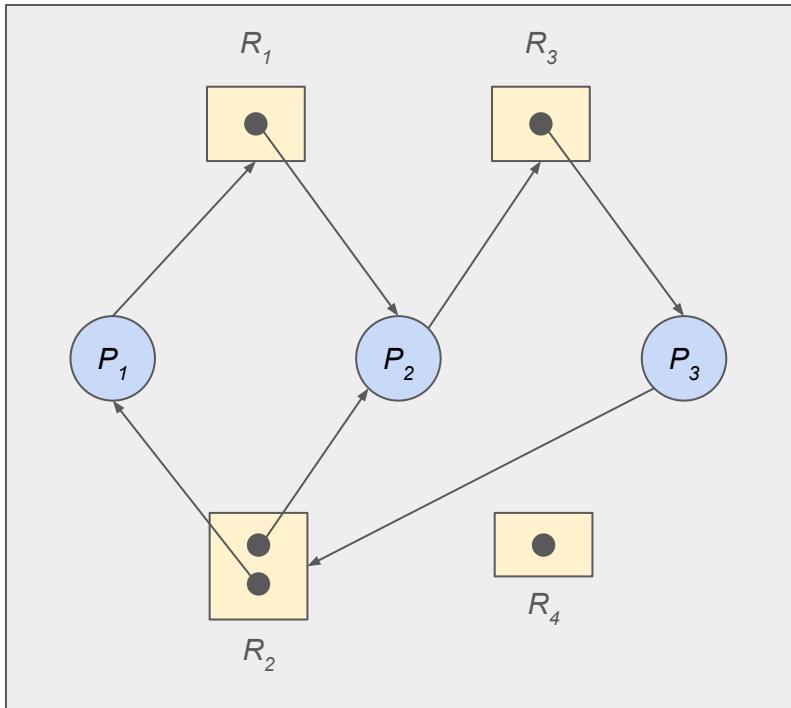
Step 1: Fill in the allocated resources

Step 2: Fill in the requested resources

	Allocated			Requested		
	R1	R2	R3	R1	R2	R3
P1						
P2						
P3						

Hint: allocated = $R \rightarrow P$; requested = $P \rightarrow R$

Example 2: Resource Allocation Graph



Question: Is there a deadlock?

Step 3: Check the availability of the resources

Example 2: Resource Allocation Graph



From Step 1 & 2:

	Allocated			Requested		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	1	0	0
P2	1	1	0	0	0	1
P3	0	0	1	0	1	0

From Step 3:

- Availability = (0, 0, 0)

Question: Is there a deadlock?

Step 4: Free the requested resources (if applicable)

Example 2: Resource Allocation Graph

From Step 1 & 2:

	Allocated			Requested		
	R1	R2	R3	R1	R2	R3
P1	0	1	0	1	0	0
P2	1	1	0	0	0	1
P3	0	0	1	0	1	0

From Step 3:

- Availability = (0, 0, 0)

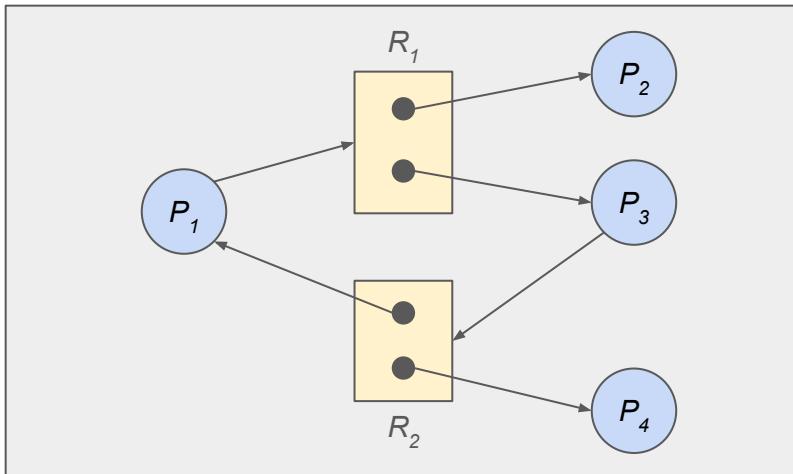
Question: Is there a deadlock?

Step 4: Free the requested resources (if applicable)

- Cannot solve any process with Availability = (0, 0, 0)
- Therefore, all three processes are deadlocked

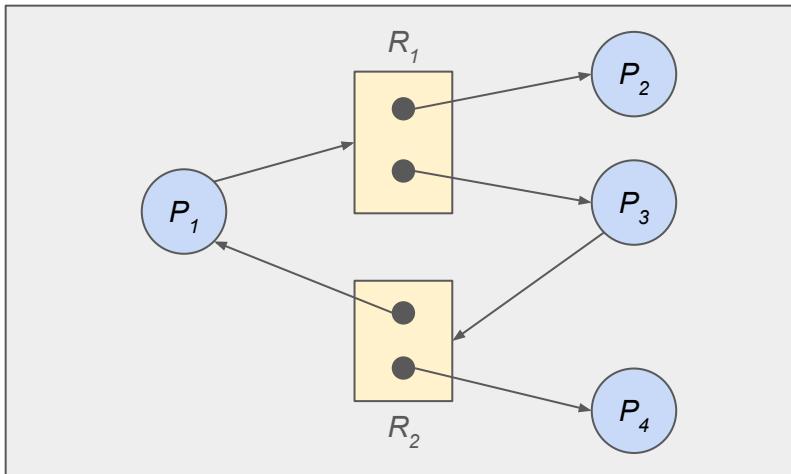


Can we have a cycle but no deadlock?



	Allocated		Requested	
	R1	R2	R1	R2
P1				
P2				
P3				
P4				

Can we have a cycle but no deadlock?



	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	1	0
P2	1	0	0	0
P3	1	0	0	1
P4	0	1	0	0

Step 3: Availability = (R1, R2) = (0, 0)

Step 4: P2 and P4 break the cycle

- Availability after P2 = (1, 0)
- Availability after P4 = (1, 1)

Cycle with no deadlock

Question on deadlock from last lecture



Multiple-choice question: A system that meets the four deadlock conditions will always/sometimes/never result in deadlock?

Answer: Sometimes, meeting four deadlock conditions is necessary for a deadlock to happen, but not sufficient.

- In the last example, the **Circular Wait** condition happened but no deadlock.

More examples of system failure due to race conditions

- Therac-25 x-ray machine (developed in Canada)
 - Between 1985 and 1987
 - Race condition caused 100 times the normal dose of radiation
 - Consequences: six patients injured, three deaths
- NASA Mars-Rover
 - In January, 2004
 - Race condition identified in the file system
 - Consequences: infinite reboot loop

An Engineering Disaster: Therac-25

[Introduction](#)
[Therac-25](#)
[Conclusions](#)
[References](#)



Lecture 23

Cookies and Sessions

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last classes
- Cookies and sessions
- Ambient authority
- Signing cookies
- Cookie attributes
 - Domain attribute
 - Path attribute
 - Expires attribute

Deadlock avoidance

To **avoid deadlock**, we can examine the processes and resources to guarantee there can never be a circular-wait condition.

- System has access to information in advance about what resources will be needed by which processes
- System only approves resource requests if the process can obtain all resources it needs in future requests

However, it is tough to avoid deadlock in practice

- Hard to determine all the needed resources in advance
- Good problem statement in theory, but difficult to apply in practice

Resource Allocation Graph (RAG)

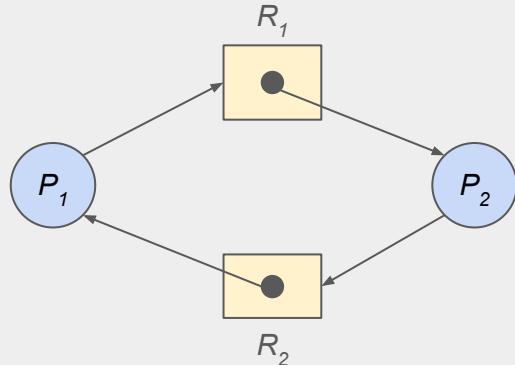
Resource Allocation Graph (RAG) represents the state of the system as graphs. The graph contain a set of vertices and edges to describe the state of the system:

- Vertices (nodes) can either represent a process or resource
 - Process: $P = \{P_1, P_2, \dots, P_n\}$ is the set of processes in the system
 - Resource: $R = \{R_1, R_2, \dots, R_m\}$ is the set of resources in the system
- Edges can either represent a request or assignment
 - Request edge (directed edge): $P_i \rightarrow R_j$
 - Assignment edge (directed edge): $R_j \rightarrow P_i$

Basic facts

To determine if a resource allocation graph contains deadlock:

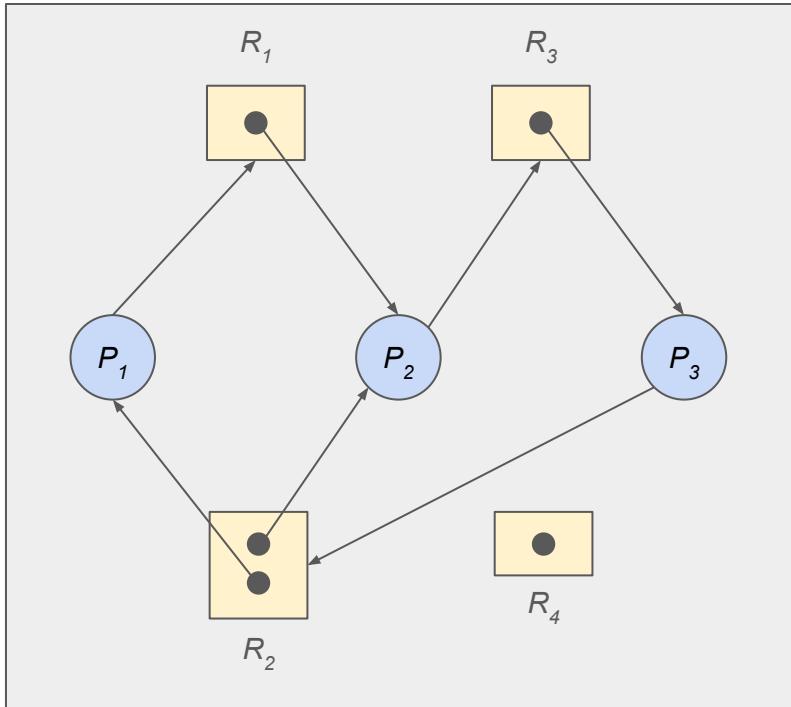
- If graph contains no cycle, then no deadlock
- If graph contains at least one cycle
 - If only one instance per resource, then deadlock
 - If several instances per resource, then possible deadlock



Example: processes in deadlock

- P1 holds R2
- P1 waits for R1
- P2 holds R1
- P2 waits for R2

Example 2: Resource Allocation Graph



Question: Is there a deadlock?

Step 1: Fill in the allocated resources

Step 2: Fill in the requested resources

	Allocated			Requested		
	R1	R2	R3	R1	R2	R3
P1						
P2						
P3						

Hint: allocated = $R \rightarrow P$; requested = $P \rightarrow R$

Schedule for today

- Key concepts from last classes
- Cookies and sessions
- Ambient authority
- Signing cookies
- Cookie attributes
 - Domain attribute
 - Path attribute
 - Expires attribute

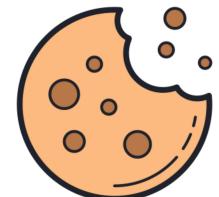
Cookies

Cookies are small piece of data that a server sends to a user's web browser.

- Web browser may store the cookie and send it back to the same server with later requests

Goals of cookies:

- Personalization: provide experiences based on personal preferences
 - E.g., preferred language to automatically deliver the right content
- Session management: make it easier for users to access accounts
 - E.g., stored login information to avoid login every time
- Tracking: track and analyze how users interact with the website
 - E.g., user behavior to enhance performance



Example to illustrate cookies

Example of how cookies work: <https://cookie-tracking.glitch.me/>

- First visit: Welcome to the website
- From next visit: Welcome back!

After clearing the cookies:

- Back to: Welcome to the website

Welcome to the website!



Welcome back!

Session cookies

A session cookie (also known as non-persistent cookie) is created by the web server and used to store information about the user's browsing session.

- Session cookies are used by the server to implement sessions
- Deleted once the session ends (e.g., logged out)

Example of session cookies:

- Logins
- Shopping carts
- User tracking

Creating session cookies

After receiving an HTTP request, a server can send **Set-Cookie** headers with the response:

- Use the Set-Cookie response header to send cookies from the server to the user agent.
- Set-Cookie: <cookie-name>=<cookie-value>
- E.g., Set-Cookie: theme=dark

Creating session cookies

After receiving an HTTP request, a server can send **Set-Cookie** headers with the response:

- Use the Set-Cookie response header to send cookies from the server to the user agent.
- **Set-Cookie: <cookie-name>=<cookie-value>**
- E.g., **Set-Cookie: theme=dark**

Then, with every subsequent request to the server, a client can send **Cookie** headers with a request:

- Use the Cookie request header to send cookies back to the server.
- **Cookie: <cookie-name>=<cookie-value>**
- E.g., **Cookie: theme=dark**

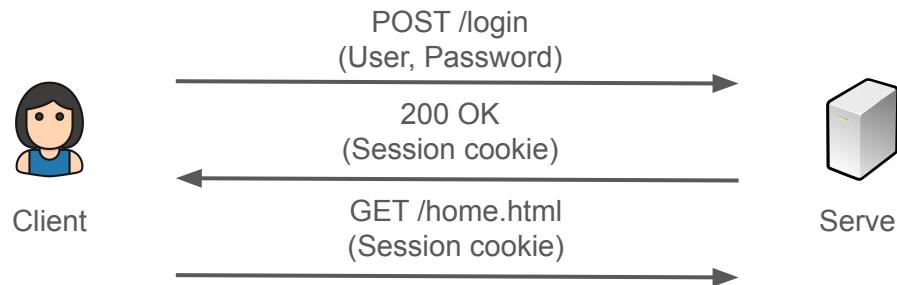
Login example with session cookies

Alice needs to login into her account:

- Alice sends her credentials to the server
- Server verifies the identity and authentication of Alice
- If credentials match, server returns 200 OK and a session cookie

For all future requests:

- Alice uses her session cookie to authenticate



Schedule for today

- Key concepts from last classes
- Cookies and sessions
- Ambient authority
- Signing cookies
- Cookie attributes
 - Domain attribute
 - Path attribute
 - Expires attribute

Ambient authority

Ambient authority is an access control based on global and persistent properties of the requester.

- Involve implicit trust and privileges to a user

In the context of web application security, the authority is automatically granted to a user based on their session state

For example:

- User logs in to a web application
- Server sends the session ID back to the user
- Ambient authority: web application implicitly authenticates and authorizes all the actions performed by the user based on the **session ID**

Ambient authority

There are four types of ambient authority on the web:

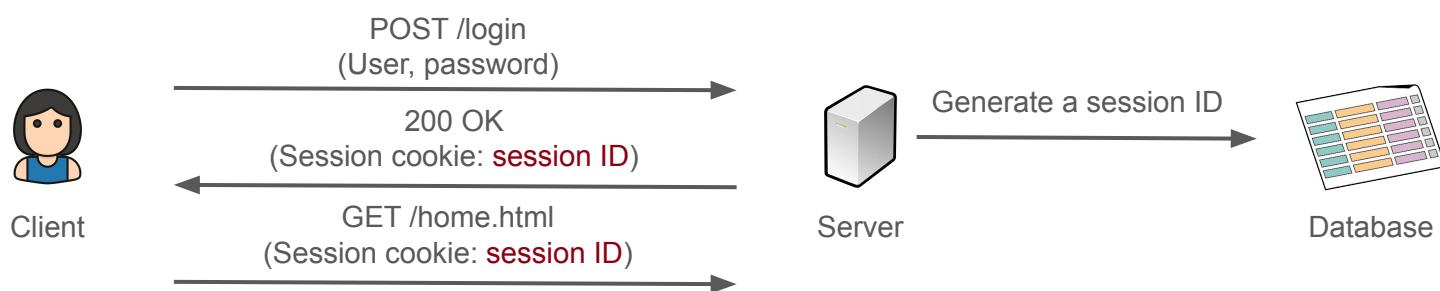
- **(Session) Cookies**: most common, most versatile method
- **IP checking**: used at some of the UoA printing services
 - Problem: Possible to change the IP address by having a server on campus
- **Client certificates**: rarely used
 - Verify the owner's identify with more than just a signature
 - Problem: Require a client application to submit a client certificate for authentication
- **Basic (HTTP) authentication**: rarely used
 - Problem: Authentication headers can be seen, which makes it easier to capture user credentials
 - In 2022, Microsoft disabled Basic authentication on Outlook

Session ID

A session ID (also known as session token) is a unique identifier that a web server assigns to a user for the duration of the user's current session.

- Randomly generated token
- Analogy: Coat tickets at a restaurant

After the server verifies the credentials, it will generate a login event and returns a session ID in the form of a cookie.



Example of session ID: eClass

You can inspect the network traffic from eClass login to see an example:

- Enable Preserve log checkbox in Network tab
- Login into eClass
- loginuserpass.php will appear in the log
 - Switch to Cookies tab
 - SimpleSAMLSessionID is the session ID

The screenshot shows a browser developer tools Network tab with the following details:

- Request:** A POST request to "loginuserpass.php?AuthState=...".
- Response:** A 200 OK response.
- Cookies:** The "Cookies" tab is selected, showing the following table:

Name	Value	Domain	Path	Expires	Size	Http...	Secure	Same...	Partition...	Priority
SimpleSAMLAuthToken	_9c00b1b10b72418294bf0387aa44f4d8bd...	login...	/	Session	62	✓	✓	None	Medium	
SimpleSAMLSessionID	a3fa17b58cbda4c2a11373ae0719d54a	login...	/	Session	51	✓	✓	None	Medium	
_gsas	ID=cdb8b38tc80d3c08fT=1709538731:RT=...	uab...	/	2025...	89				Medium	
ce.s	v=feb46cd52ee3btf55bf0ff4de804ed5087e...	uab...	/	2025...	225			Strict	Medium	
_click	109j72o%7C2%7Cfjw%7C0%7C1509	uab...	/	2025...	33				Medium	
_fbp	fb.1.1708262876798.728361074	uab...	/	2024...	32			Lax	Medium	
_ga	GA1.1.554714979.1708280304	uab...	/	2025...	29				Medium	
gs.00QGX5M2CJ	GS1.1.1708640075.1.0.1708640075.0.0.0	uab...	/	2025...	51				Medium	

Example of database with login events

The screenshot shows a software application window titled "Sessions". The top menu bar includes "File", "Edit", "View", "Search", "Actions", "View", and "Detach". The main area has a title bar "Sessions" with a blue icon. Below it is a search bar with placeholder text "Use the search tool to find Sessions." and a dropdown menu "Search". The search criteria include fields for Application ID, User ID, Username, Session ID, Internal Session ID, Authentication Status, Client Type, IP Address, Device ID, Processing Status, Alert Level, and Login Time. There are also "Advanced", "Saved Search", and "Search Sessions" buttons. The bottom section is titled "Search Results" and displays a table with columns: Session ID, Internal Session ID, Alerts, Application ID, Username, Device ID, IP Address, Location, and Auther Status. The table lists 15 rows of session data, each with a small blue info icon next to the "Alerts" column.

Session ID	Internal Session ID	Alerts	Application ID	Username	Device ID	IP Address	Location	Auther Status
3760	26_7ea78ccfc3818fcc	High Alerts: (1)	bharosaUIOGrp	gdfg	3201	141.144.80.177	(i) denmark, kobenh Succes	
3759	21_bf6d8d53789551		bharosaUIOGrp	kama1216	3201	141.144.80.177	(i) denmark, kobenh Wrong	
3758	16_904940b56c76f8c		bharosaUIOGrp	bodurai	3201	141.144.80.177	(i) denmark, kobenh Succes	
3757	12_34922f8a765f3d1		bharosaUIOGrp	bodurai	3201	141.144.80.177	(i) denmark, kobenh Wrong	
3756	8_0603273ecbd5389i		bharosaUIOGrp	bodura345	3201	141.144.80.177	(i) denmark, kobenh Wrong	
3755	3_1f0679f07658b635		bharosaUIOGrp	bodura345	3201	141.144.80.177	(i) denmark, kobenh Pendin	
3754	43_4806da7b0f14afe		bharosaUIOGrp	dg	3201	141.144.80.177	(i) denmark, kobenh Succes	
3753	36_f2fa9d36922cc6d	Medium Alerts: (1)	bharosaUIOGrp	oaam_two	3203	10.143.234.152	(i) Private, Private, I Blocker	
3752	32_e0a40bf79c258f8		bharosaUIOGrp	oaam_test	3203	10.143.234.152	(i) Private, Private, I Blocker	
3751	28_48ad2f18822eb19		bharosaUIOGrp	oaam_two	3203	10.143.234.152	(i) Private, Private, I Wrong	
3750	22_16f17f069b8378c		bharosaUIOGrp	surai	3201	141.144.80.177	(i) denmark, kobenh Succes	

Login example in details

An actual example of session cookie:

s%3AI3ozSdvQ83TtC5RvJ.CibaQoHtaY0H3QOB1kqR8H2A

Signed/unsigned cookie

Session ID

Signature

- s%3A states it is a signed cookie
- I3ozSdvQ83TtC5RvJ indicates the session ID
- CibaQoHtaY0H3QOB1kqR8H2A shows the signature

Signing is necessary to confirm that the cookie originated from the server.

Digital signature

Digital signatures verify the authenticity

- Detect the identity of the sender/signer/requester

Digital signatures check the integrity

- Verify that the message was not changed

Digital signatures ensure non-repudiation

- Verify that the signature is not fake

Session fixation attack

A third party is able to determine a user's session identifier (i.e., by reading it or setting it), and therefore interact with the server as that user.

- Stealing cookies is one way to do this

A more advanced technique:

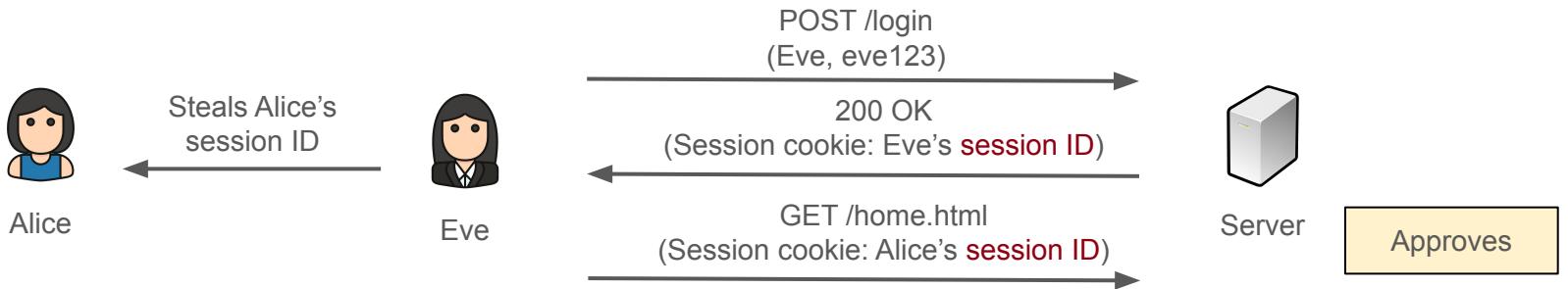
- When the user visits a page on the parent domain (or another subdomain), the application may share the existing value sent in the user's cookie
- This could allow an attacker to hijack a session after the user logs in

Without signing cookies

Without signature, another client can access someone else's session. For example:

- Eve can copy over Alice's session ID to her browser
- Server validates the session ID (which is valid) and approves the request
- **Confidentiality** of the system is broken!

Signature ensures the integrity of the cookie

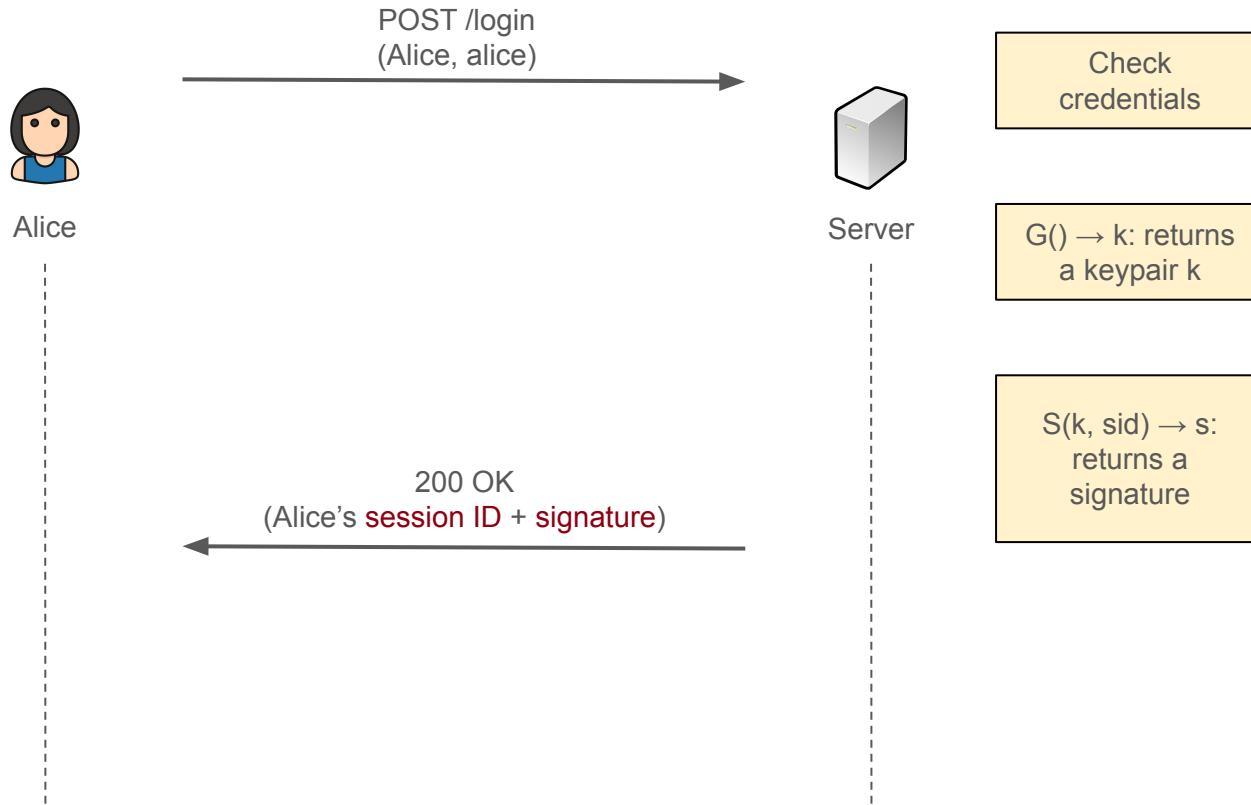


Signing cookies

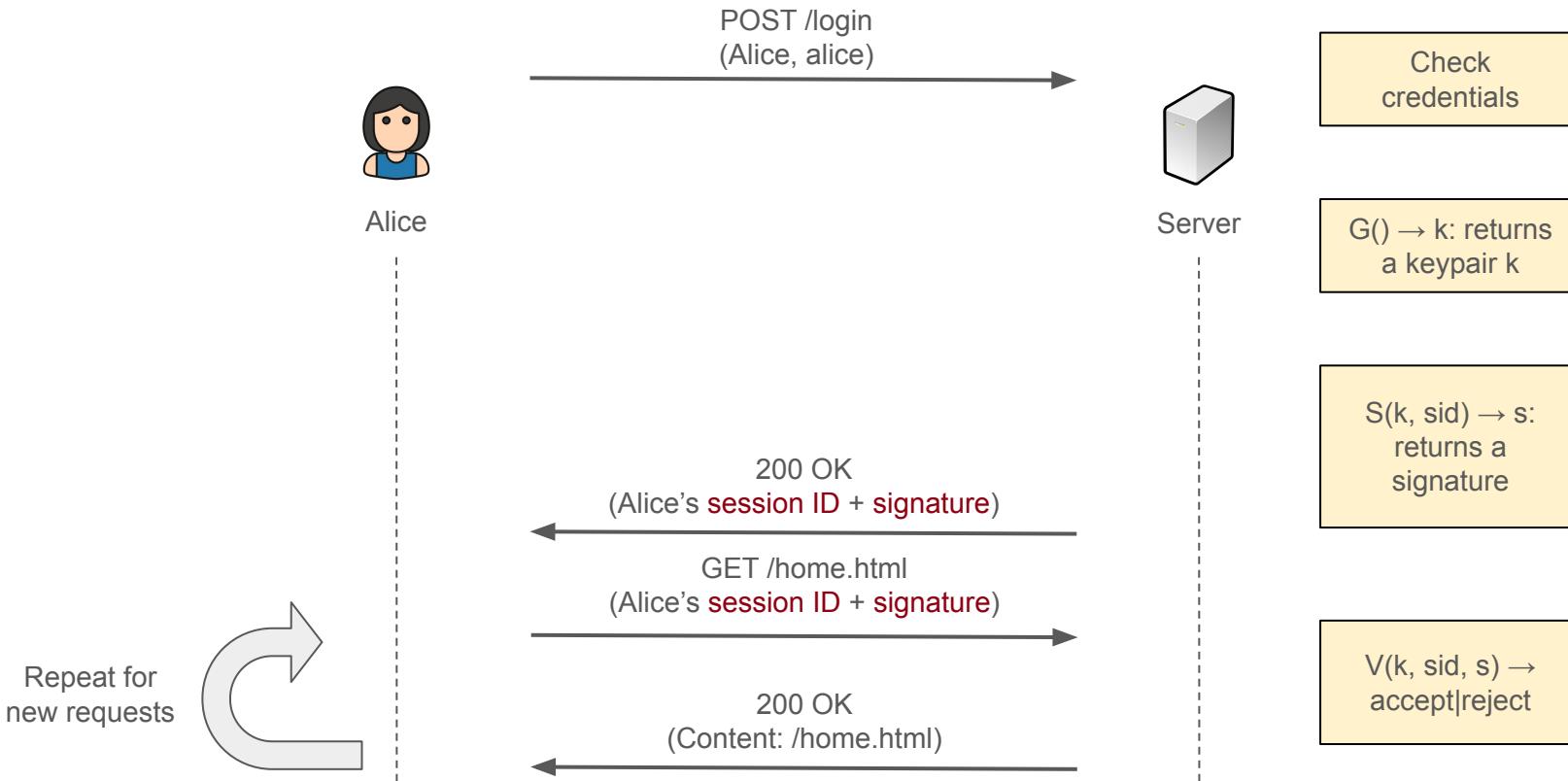
A signature scheme consists of a triple of algorithms (G, S, V):

- **G = key generation algorithm**
 - Generate a public key for verification
 - Generate a private key for signing
 - $G() \rightarrow k$: returns a keypair k
- **S = signing algorithm**
 - Output a signature
 - $S(k, sid) \rightarrow s$: returns a signature for keypair k and session ID sid
- **V = verification algorithm**
 - Output a value stating the validity of the signature (e.g., accept/reject)
 - $V(k, sid, s) \rightarrow \text{accept/reject}$: checks validity of signature for given key pair k and session ID sid

Signing cookies



Signing cookies



Signing cookies

By signing the cookie, we make sure that the cookie has not been modified.

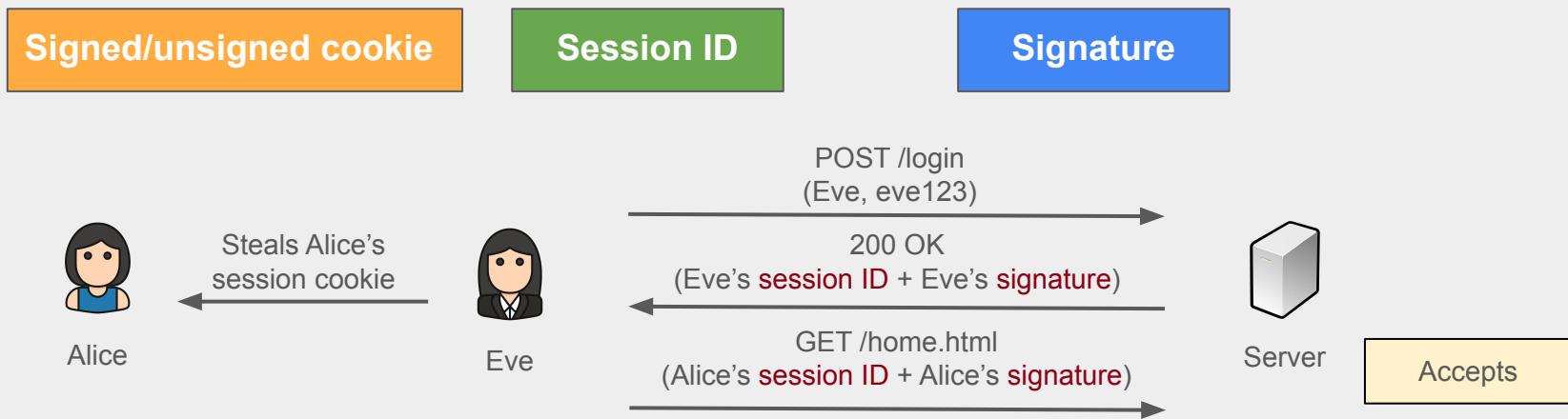
- Eve can copy Alice's session ID over to her browser
- Server validates the signature with the session ID (which is invalid)
- Server denies the request as the signature does not match the session ID
- **Confidentiality** of the system is satisfied



Another attempt to break the system

What if the whole session cookie was stolen?

s%3AI3ozSdvQ83TtC5RvJ.CibaQoHtaY0H3QOB1kqR8H2A



Confidentiality of the system is broken again!

Password reset on leaked session cookie



Can we fix this use case by resetting the password?

Password reset on leaked session cookie

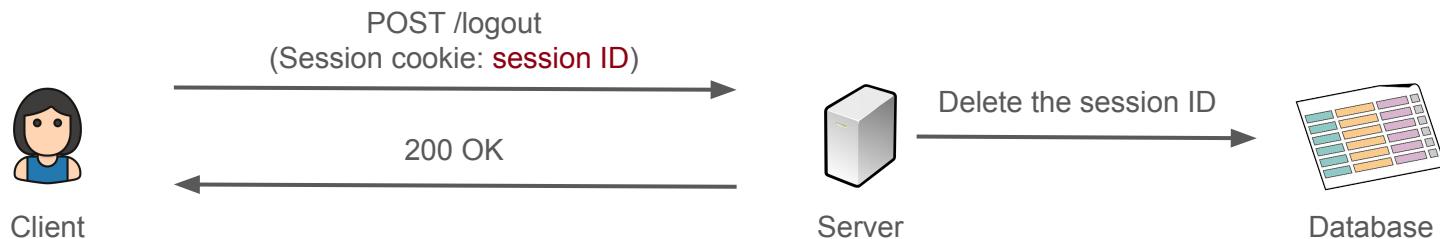


Can we fix this use case by resetting the password?

- Password may be reset, but the session ID and signature remain valid for authentication

Solution? We must destroy the session in a meaningful way

- Delete the session ID when the user resets the password, or regenerate and resend cookie when the user authenticate (even if already exist one)
- Analogy: Coat tickets at a restaurant



Cookie attributes

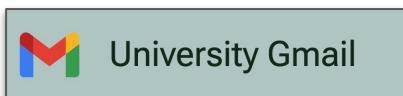
Cookie attributes are what protect the cookies from malicious users:

- Domain attribute: allows the cookie to be set to a broader domain
- Path attribute: scopes the cookie to a path prefix
- Expires attribute: specifies an expiration date
 - Expires date and time are relative to client the cookie is being set on, not the server
 - Defining lifetime of a cookie is necessary to avoid session fixation attacks

Domain attribute

Domain attribute allows the cookies to be scoped to a broader domain.

- E.g., Gmail Apps@UAlberta could set a cookie for Apps@UAlberta
- Try it yourself in Incognito mode:
 - Login into your Gmail: apps.ualberta.ca/appslink/auth/feature/gmail
 - Open apps.ualberta.ca/, you should be logged in



Login into Gmail in
Apps@UAlberta



Also logged in at
Apps@UAlberta

If the domain attribute is set too **loosely**, then the server may be vulnerable to **session fixation attack** (e.g., allowing a third party to access the session id).

Path attribute

Path attribute scopes the cookie to a path prefix, which works in conjunction with the domain attribute to a particular request path prefix.

- E.g., cookies in **path=/docs** will match **path=/docs/admin**
- Try it yourself in Incognito mode:
 - Login into eClass: eclass.srv.ualberta.ca/course
 - Open eclass.srv.ualberta.ca/course/view.php?id=2187, you should be logged in



If the path attribute is set too **loosely**, it could leave the application vulnerable to **attacks by other applications** on the same server.

Basic facts

Desired properties for sessions:

- Browser remembers user
- User cannot modify session cookie to login as another user
- Session cookies only last as long as the browser is open
- Sessions can be managed (e.g., session ID can be deleted) by the server
- Sessions expire after some time

Lecture 24

Cross Site Scripting

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last classes
- Cookie attributes
- Cross-Site Request Forgery
 - Demo with eClass
- Same Origin Policy
- Cross Site Scripting (XSS)
 - Reflected XSS
- TODOs

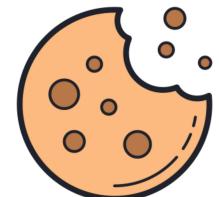
Cookies

Cookies are small piece of data that a server sends to a user's web browser.

- Web browser may store the cookie and send it back to the same server with later requests

Goals of cookies:

- Personalization: provide experiences based on personal preferences
 - E.g., preferred language to automatically deliver the right content
- Session management: make it easier for users to access accounts
 - E.g., stored login information to avoid login every time
- Tracking: track and analyze how users interact with the website
 - E.g., user behavior to enhance performance



Creating session cookies

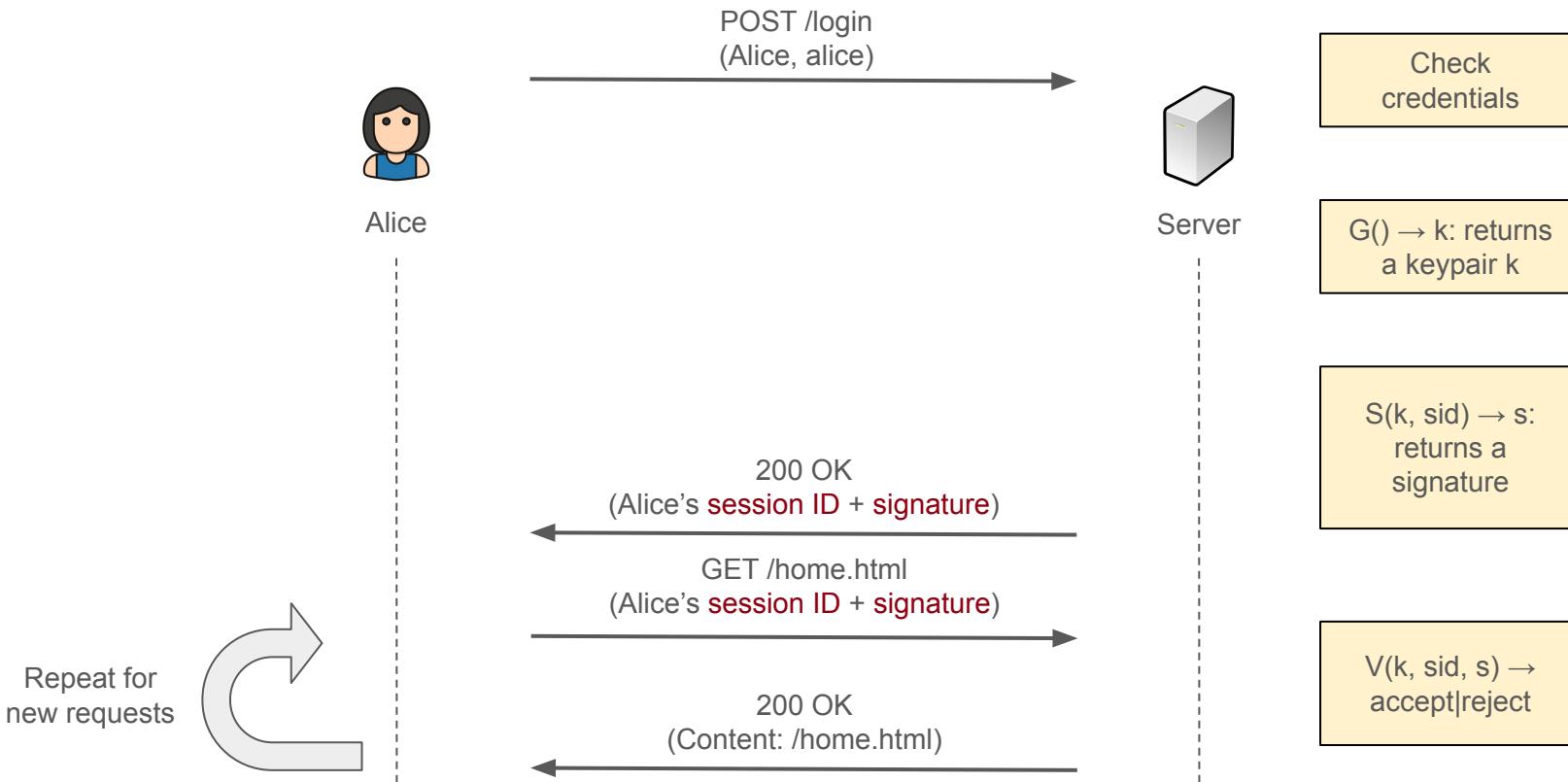
After receiving an HTTP request, a server can send **Set-Cookie** headers with the response:

- Use the Set-Cookie response header to send cookies from the server to the user agent.
- **Set-Cookie: <cookie-name>=<cookie-value>**
- E.g., **Set-Cookie: theme=dark**

Then, with every subsequent request to the server, a client can send **Cookie** headers with a request:

- Use the Cookie request header to send cookies back to the server.
- **Cookie: <cookie-name>=<cookie-value>**
- E.g., **Cookie: theme=dark**

Signing cookies



To avoid session fixation attack

Session fixation attack attempts to fixate (find) another user's session identifier to gain access to the account by:

- Stealing cookies
- Guessing session identifiers

Solution: Session IDs should be signed, transient, revocable and unpredictable

- Signed: integrity of the cookies
- Transient: timeout after a certain period of time
- **Revocable:** ready for worst-case scenario (e.g., reset password = clean up)
- Unpredictable: randomness in session ID generation
 - E.g., HMAC hash for one-way function

Question on sessions and cookies



Which of the following statements is true?

- A. A web cookie is a small piece of data sent from a website and stored in user's web browser while a user is browsing a website.
- B. A web cookie is a small piece of data sent from user and stored in the server while a user is browsing a website.
- C. A web cookie is a small piece of data sent from root server to all servers.
- D. None of these

Cookie attributes

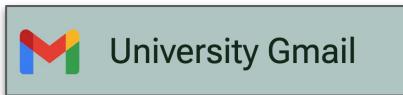
Cookie attributes are what restrict/protect the cookies from malicious users:

- Domain attribute: allows the cookie to be set to a broader domain
- Path attribute: scopes the cookie to a path prefix
- Expires attribute: specifies an expiration date
 - Expires date and time are relative to client the cookie is being set on, not the server
 - Defining lifetime of a cookie is necessary to avoid session fixation attacks

Domain attribute

Domain attribute allows the cookies to be scoped to a broader domain.

- E.g., Gmail Apps@UAlberta could set a cookie for Apps@UAlberta
- Try it yourself in Incognito mode:
 - Login into your Gmail: apps.ualberta.ca/appslink/auth/feature/gmail
 - Open apps.ualberta.ca/, you should be logged in



Login into Gmail in
Apps@UAlberta



Also logged in at
Apps@UAlberta

If the domain attribute is set too **loosely**, then the server may be vulnerable to **session fixation attack** (e.g., allowing a third party to access the session id).

Path attribute

Path attribute scopes the cookie to a path prefix, which works in conjunction with the domain attribute to a particular request path prefix.

- E.g., cookies in **path=/docs** will match **path=/docs/admin**
- Try it yourself in Incognito mode:
 - Login into eClass: eclass.srv.ualberta.ca/course
 - Open eclass.srv.ualberta.ca/course/view.php?id=2187, you should be logged in



If the path attribute is set too **loosely**, it could leave the application vulnerable to **attacks by other applications** on the same server.

Overall idea

Desired properties for sessions:

- Browser remembers user
 - Cookies for better user experience
- User cannot modify session cookie to login as another user
 - Signed and unpredictable session IDs to avoid session fixation attack
- Session cookies only last as long as the browser is open
 - Desired, but different in practice
- Sessions can be managed by the server
 - Worst-case scenario: revoke the session IDs
- Sessions expire after some time
 - Avoid session fixation attack

Schedule for today

- Key concepts from last classes
- Cookie attributes
- Cross-Site Request Forgery
 - Demo with eClass
- Same Origin Policy
- Cross Site Scripting (XSS)
 - Reflected XSS
- TODOs

Ambient authority

Ambient authority is an access control based on global and persistent properties of the requester.

- Involve implicit trust and privileges to a user

In the context of web application security, the authority is automatically granted to a user based on their session state

For example:

- User logs in to a web application
- Server sends the session ID back to the user
- Ambient authority: web application implicitly authenticates and authorizes all the actions performed by the user based on the **session ID**

Problem with ambient authority

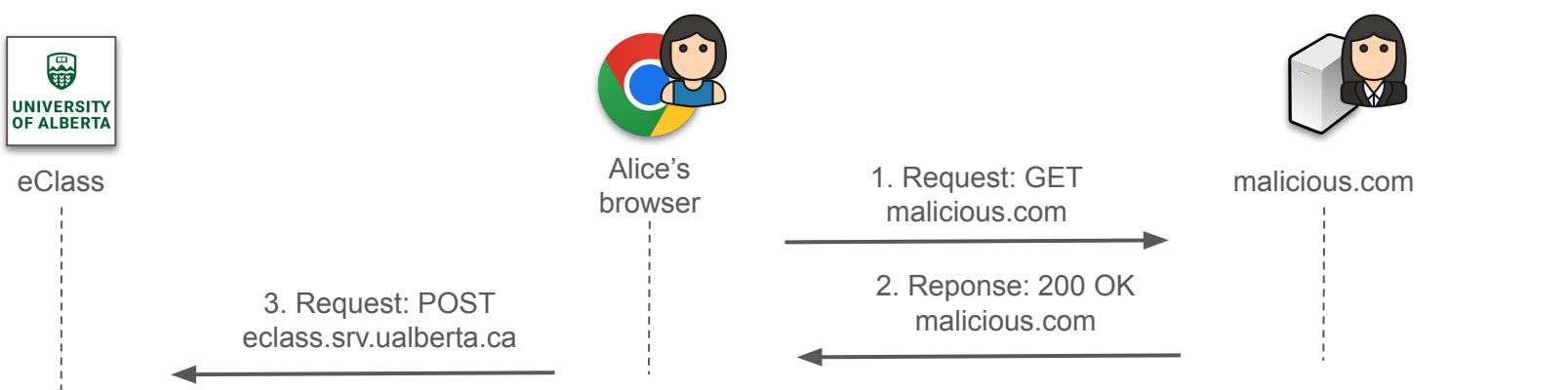
Recall: Ambient authority can be implemented with cookies

- If some properties (e.g., session ID) are valid, grant privileges to users

Consider the following scenario:

- Eve sets up malicious.com with an embedded HTML tag:

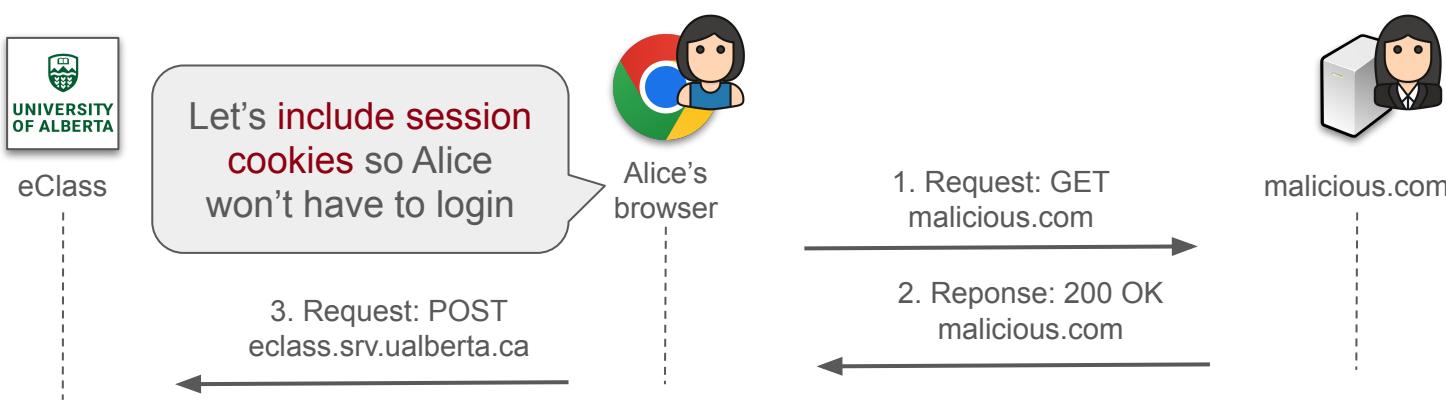
```
<img src='https://https://eclasse.srv.ualberta.ca/ece422/change\_grade?user=eve&grade=100' />
```



Problem with ambient authority

Browser helpfully includes the session cookies in all requests to eClass

- When an instructor (logged in) goes to malicious.com, the HTTP request will be triggered with his/her session cookies
- Since the session ID is valid, the request is accepted
- However, the request originated from malicious.com!



Problem with ambient authority

While ambient authority improves the user experience and facilitates session management, it also introduces security risks.

- Attackers can use a victim's logged-in session to perform any action
- E.g., Eve uses Alice's logged-in eClass session to change the grade

Problem: It is unclear who initiated the request

This is an example of **Cross-Site Request Forgery (CSRF)**

- CSRF attack tricks the victim into submitting a malicious request
- It inherits the identity of the victim



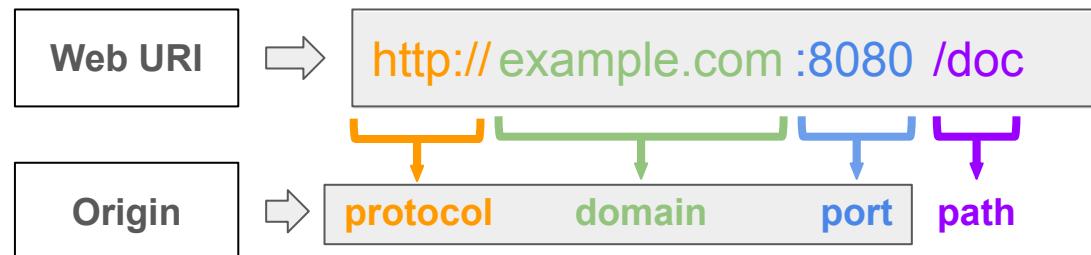
Same Origin Policy (SOP)

Same Origin Policy (SOP) is a web security mechanism that restricts how documents and scripts on one origin can interact with resources on another origin.

- Stop one website from interfering with another website

The policy checks if the **origin** matches the one from the website:

- Only when the **origin (protocol + domain + port)** is the same, the website allows read and write



Example: origin matching



Check if the Same Origin Policy can be satisfied:



Requests coming from:



Example: origin matching



Check if the Same Origin Policy can be satisfied:



Requests coming from:

Request 1	→	http://example.com	Default port for HTTP: 80	
Request 2	→	https://example.com/doc	Different protocol	
Request 3	→	http://event.example.com	Different subdomain	
Request 4	→	http://example.com:8080	Different port	

Same Origin Policy prevents Cross-Site Request Forgery

Browser will not be allowed to include the session cookies:

- When an instructor (logged in) goes to malicious.com, the HTTP request will be triggered
- The browser permits cross-origin request, but prevents reading data from another origin
- Alice's session cookies remain safe

Same Origin Policy: prevents scripts running under one origin to read data from another origin



eClass

Different origin, let's
not share the
session cookies!

3. Request: POST
eclass.srv.ualberta.ca



Alice's
browser

1. Request: GET
malicious.com
2. Response: 200 OK
malicious.com



malicious.com

Schedule for today

- Key concepts from last classes
- Cookie attributes
- Cross-Site Request Forgery
- Same Origin Policy
- Cross Site Scripting (XSS)
 - Reflected XSS
 - Stored XSS
- TODOs

Samy worm

In 2005, Samy Kamkar released the **Samy worm** onto MySpace.

- Samy worm was a self-propagating cross-site scripting worm that caused the victim to send a friend request to Kamkar without knowing it
- Problem: each user who viewed his profile will have the same script planted on their own profile
- Consequences:
 - Within 20 hours, over one million users was affected
 - MySpace was forced shut down



Samy Kamkar at Black Hat conference (2010)

History of Samy



Cross-site scripting (XSS)

Cross-site scripting (XSS) is an attack in which attackers injects malicious executable scripts into the website.

- This happens when untrusted user data unexpectedly becomes code

How it works?

- Attackers can inject malicious code to a vulnerable website
- Website then returns malicious code to users
- When the malicious code executes inside a victim's browser, the attacker can fully control and compromise their interaction

Without proper **data sanitization**, the website may execute the malicious code on other users' system

XSS as a code injection vulnerability

In cross-site scripting (XSS), malicious JavaScript scripts are injected into an HTML document.

```
<html> Welcome back,  
      <script> ... </script>  
<html>
```

XSS: untrusted user data unexpectedly becomes code

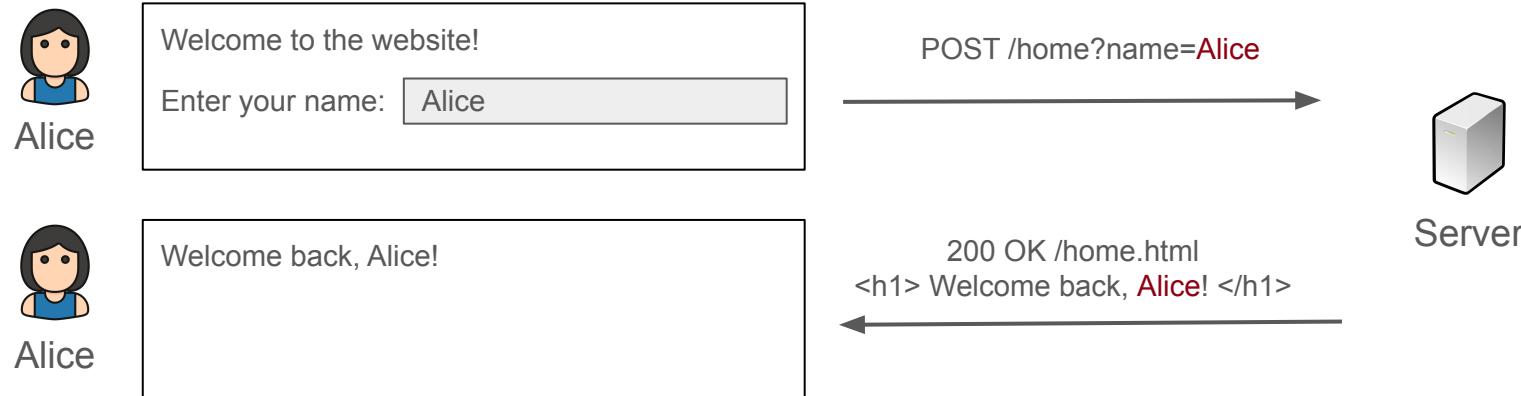
With XSS, attackers can perform session fixation attacks:

- View user's cookies
- Send HTTP requests with the user's cookies

Without XSS

Suppose Alice sends her name to the web server:

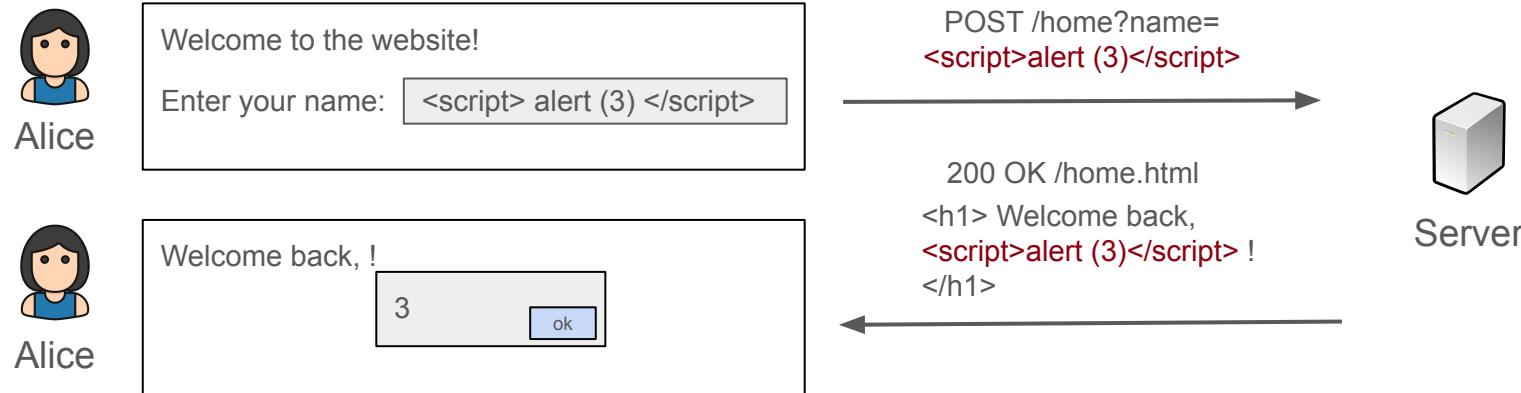
- Alice sends a request with the parameter `name` as: **Alice**
- Server responds with the home page html with the `name` parameter
- Input **Alice** is being reflected back in the response



with XSS

Now suppose Alice injects a script in the request:

- Alice sends a request with the parameter **name** as:
`<script> alert (3) </script>`
- Server responds with the home page html with the **name** parameter
- Input `<script> ... </script>` is being **reflected** back in the response



Types of XSS

There are 3 common types of XSS:

- **Reflected (non-persistent) XSS**
 - Malicious script is reflected off of a web application to the victim's browser
- **Stored (persistent) XSS**
 - Malicious script is stored on the target server (e.g., database)
- **DOM-based XSS**
 - Malicious script only exists in client-side code, usually by writing the data back to the DOM

Reflected XSS

Reflected (non-persistent) XSS arises where the malicious script is reflected off in the victim's browser.

- The script is activated through a link, which sends a request to a website with a vulnerability that enables execution of malicious scripts

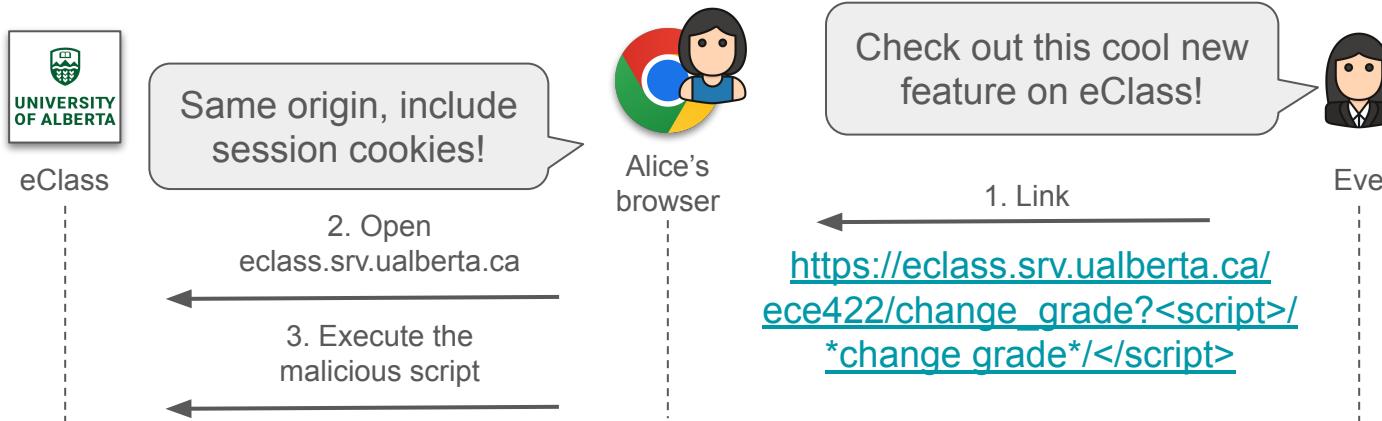
Attackers need to:

- Find a webpage vulnerable to XSS
 - Find a URL that they can make victims visit that can include the attack scripts
- Send the URL with injected scripts to the victims

Example of reflected XSS

Eve sends an eClass link with some malicious script to Alice

- The script is activated once Alice clicked on the link
- The link sends a request to eClass which executes a malicious script that changes Eve's grade using Alice's session
- But the Same Origin Policy is never triggered (same origin)



CSRF vs XSS



eClass

Same origin, include session cookies!



Alice's browser

2. Open eclass.srv.ualberta.ca
3. Execute the malicious script

Check out this cool new feature on eClass!



Eve

1. Link
https://eclass.srv.ualberta.ca/ece422/change_grade?<script>*change grade*</script>

XSS



eClass

Let's **include session cookies** so Alice won't have to login



Alice's browser

3. Request: POST eclass.srv.ualberta.ca

1. Request: GET malicious.com



malicious.com

2. Response: 200 OK malicious.com

CSRF

with XSS

Note that, while `alert(3)` is a typical payload we use to find XSS vulnerabilities:

- Avoid using it in Capture The Flag ([CTFTime](#), [Google CTF](#)) competitions
- Avoid using it in Bug Bounty Programs
- [Google Bug Hunters: Don't use alert\(1\)](#)



Alice

Welcome to the website!

Enter your name:

POST /home?name=
`<script>alert (3)</script>`



Server



Alice

Welcome back, !

3

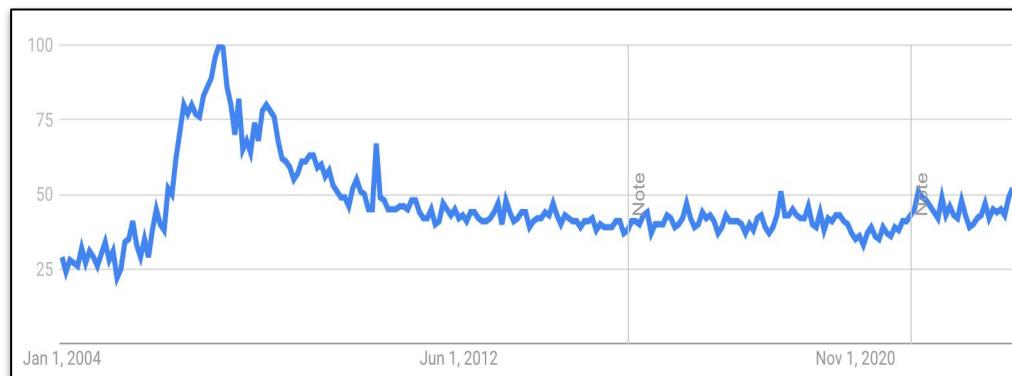
ok

200 OK /home.html
`<h1> Welcome back,
<script>alert (3)</script> !
</h1>`

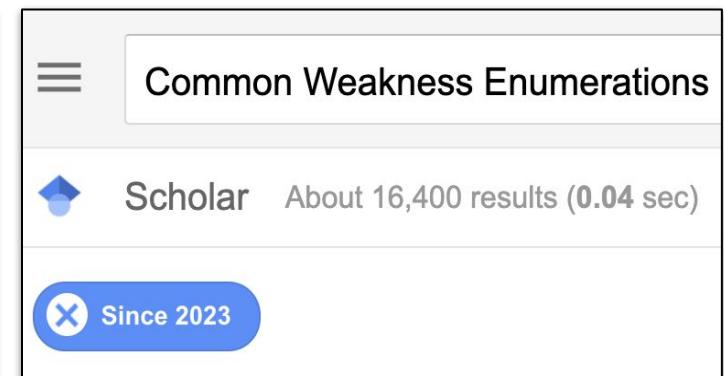
Google Trends on XSS

Although XSS became popular in 2005, it is still relevant today.

- [OWASP's Top Web App Security Risks](#): XSS attacks remain in the Top 3
- 33 Common Weakness Enumerations (CWEs) mapped into this category have the second most occurrences in applications



"XSS" on Google Trends



"Common Weakness Enumerations" on Google Scholar

TODOs

Next class:

- Stored (persistent) XSS + Defences against XSS
- Deliverable due on Friday, March 15, 23:59 MST.
- Poll for materials from Week 12 - 13 (March 25 ~ April 5)
 - 2-minute overview on each advanced subject, same thing with Database and Networks modules
 - Poll remains open until Monday, March 18 so you have more time to look up each subject
- No new material on Week 14 (April 8)
- No class on Week 15 (April 15)
- Final exam during Week 16, on Wednesday, April 24

Lecture 25

Cross Site Scripting Prevention

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

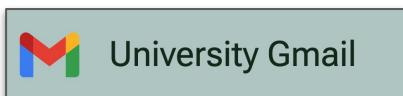
Schedule for today

- Key concepts from last classes
- Cross Site Scripting (XSS)
 - Stored XSS
- More details: Injection methods
- Cross Site Scripting Prevention
 - HttpOnly flag
 - HTML Escaping
 - Content Security Policy (CSP)
- Polls for Week 12 - 13 materials
 - Advanced Topics: Bitcoin, LLMs, Program Analysis, Automated Testing (Selenium)

Domain attribute

Domain attribute allows the cookies to be scoped to a broader domain.

- E.g., Gmail Apps@UAlberta could set a cookie for Apps@UAlberta
- Try it yourself in Incognito mode:
 - Login into your Gmail: apps.ualberta.ca/appslink/auth/feature/gmail
 - Open apps.ualberta.ca/, you should be logged in



Login into Gmail in
Apps@UAlberta



Also logged in at
Apps@UAlberta

If the domain attribute is set too **loosely**, then the server may be vulnerable to **session fixation attack** (e.g., allowing a third party to access the session id).

Path attribute

Path attribute scopes the cookie to a path prefix, which works in conjunction with the domain attribute to a particular request path prefix.

- E.g., cookies in **path=/docs** will match **path=/docs/admin**
- Try it yourself in Incognito mode:
 - Login into eClass: eclass.srv.ualberta.ca/course
 - Open eclass.srv.ualberta.ca/course/view.php?id=2187, you should be logged in



If the path attribute is set too **loosely**, it could leave the application vulnerable to **attacks by other applications** on the same server.

Problem with ambient authority

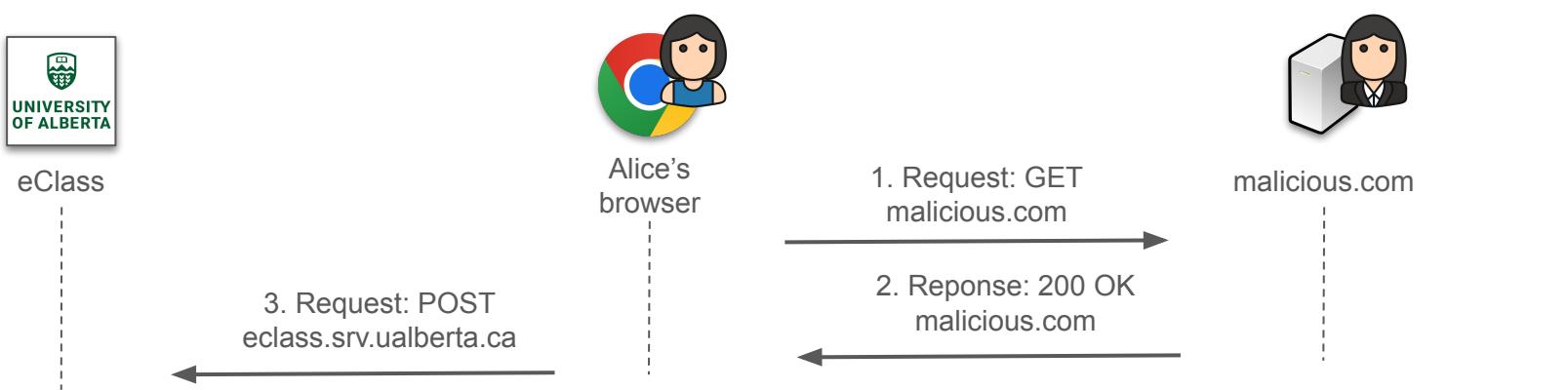
Recall: Ambient authority can be implemented with cookies

- If some properties (e.g., session ID) are valid, grant privileges to users

Consider the following scenario:

- Eve sets up malicious.com with an embedded HTML tag:

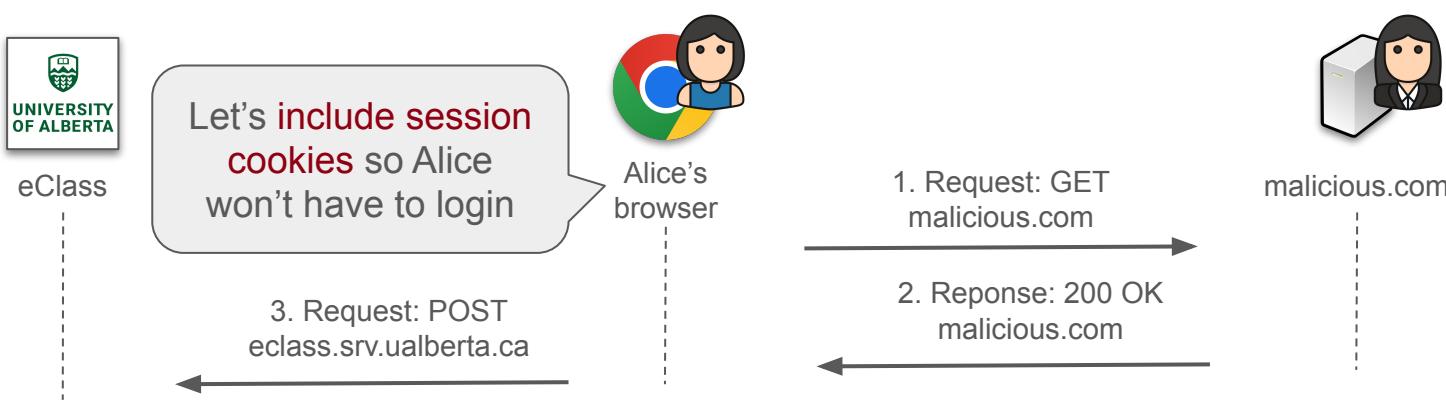
```
<img src='https://https://eclasse.srv.ualberta.ca/ece422/change\_grade?user=eve&grade=100' />
```



Problem with ambient authority

Browser helpfully includes the session cookies in all requests to eClass

- When an instructor (logged in) goes to malicious.com, the HTTP request will be triggered with his/her session cookies
- Since the session ID is valid, the request is accepted
- However, the request originated from malicious.com!



Recap on the demo

Demo: We tried to have access to eClass when we are logged in into Gmail

Result: It works because of it is coming from the same origin

- In case where the HTTP request to eClass is triggered by malicious.com, the **Same-Origin Policy** will restrict the access to Gmail session cookies



A screenshot of a web browser window. At the top, there is a header bar with three dots. Below it, a green button contains the Google 'G' logo and the text "University Gmail". Underneath the button, the text "Login into Gmail in Apps@UAlberta" is displayed.

A screenshot of a web browser window. At the top, there is a header bar with three dots. Below it, a dark green bar features the University of Alberta crest and the text "UNIVERSITY OF ALBERTA Apps@UAlberta". Underneath the bar, the text "Hi Alice (alice)" is displayed. Further down, the text "Also logged in at Apps@UAlberta" is shown.

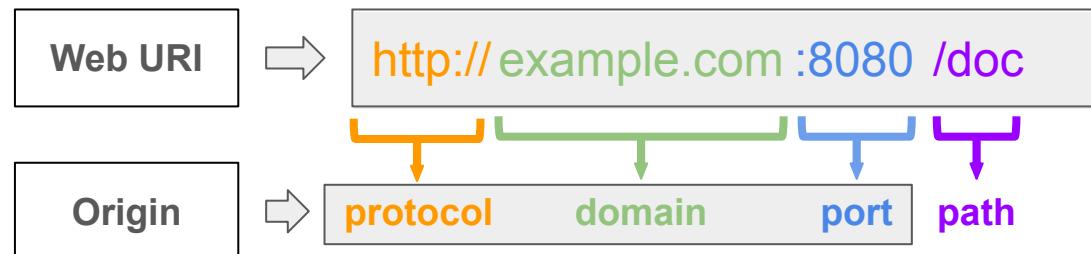
Same Origin Policy (SOP)

Same Origin Policy (SOP) is a web security mechanism that restricts how documents and scripts on one origin can interact with resources on another origin.

- Stop one website from interfering with another website

The policy checks if the **origin** matches the one from the website:

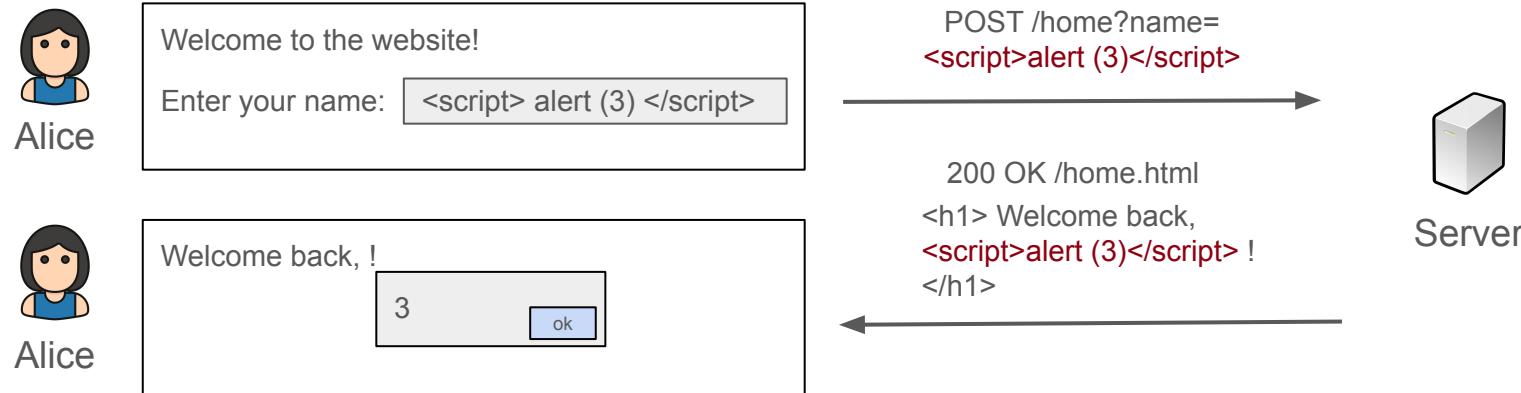
- Only when the **origin (protocol + domain + port)** is the same, the website allows read and write



with XSS

Now suppose Alice injects a script in the request:

- Alice sends a request with the parameter **name** as:
`<script> alert (3) </script>`
- Server responds with the home page html with the **name** parameter
- Input `<script> ... </script>` is being **reflected** back in the response



CSRF vs XSS



eClass

Same origin, include session cookies!



Alice's browser

2. Open eclass.srv.ualberta.ca
3. Execute the malicious script

Check out this cool new feature on eClass!



Eve

1. Link
https://eclass.srv.ualberta.ca/ece422/change_grade?<script>*change grade*</script>

XSS



eClass

Let's **include session cookies** so Alice won't have to login



Alice's browser

3. Request: POST eclass.srv.ualberta.ca

1. Request: GET malicious.com



malicious.com

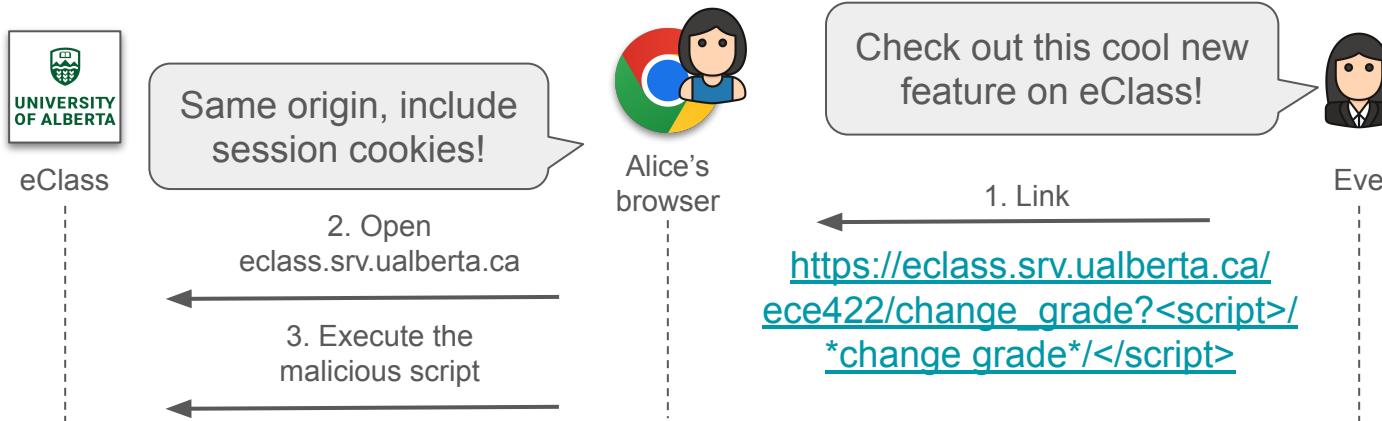
2. Response: 200 OK malicious.com

CSRF

Example of reflected XSS

Eve sends an eClass link with some malicious script to Alice

- The script is activated once Alice clicked on the link
- The link sends a request to eClass which executes a malicious script that changes Eve's grade using Alice's session
- But the Same Origin Policy is never triggered (same origin)



Question on previous materials



A security analyst reviews the following web server log:

```
[15/March/2024:12:00:00 +0100] "GET  
/profile.php?id=<script>alert('www.malicious.com/grade_update.php')</script>"
```

Question: Which vulnerability is the attacker exploiting?

- A. Ambient authority
- B. Session fixation
- C. Cross-Site Request Forgery (CSRF)
- D. Cross Site Scripting (XSS)

Schedule for today

- Key concepts from last classes
- Cross Site Scripting (XSS)
 - Stored XSS
- More details: Injection methods
- Cross Site Scripting Prevention
 - HttpOnly flag
 - HTML Escaping
 - Content Security Policy (CSP)
- Polls for Week 12 - 13 materials
 - Advanced Topics: Bitcoin, LLMs, Program Analysis, Automated Testing (Selenium)

Stored XSS

Stored (persistent) XSS arises where the malicious script is stored on the target server (e.g., database).

- Malicious script is activated when a user retrieve the data without that data being made safe to render in the browser

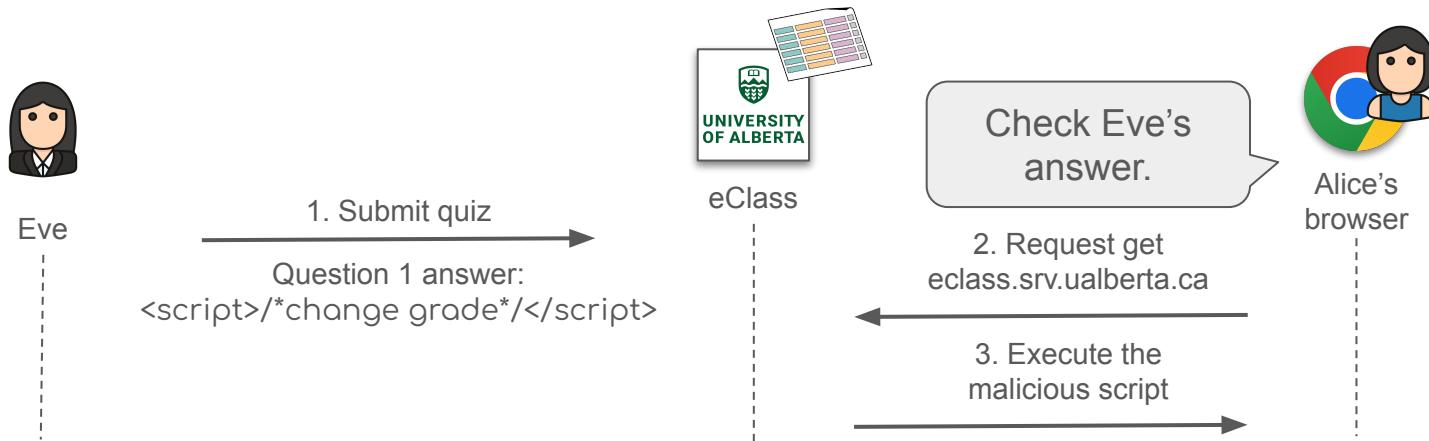
Attackers need to:

- Find a webpage vulnerable to XSS
- Send the malicious script as data to the server
- Wait for another user to retrieve the data and trigger the script

Example of stored XSS

Eve submit a malicious script to eClass as the answer to an online quiz question:

- Web server stores the scripts in the database
- Alice sends a request to eClass for Eve's answer
- The script is activated when Eve's answer is displayed on Alice's browser



Samy worm = stored XSS

Samy worm is an example of stored XSS:

- Malicious script was first uploaded by Samy on his own profile (his own profile entry in the database)
- When users visited Samy's profile, the script also injected itself to the victim's profile (another user's profile entry in the database)
- Then, the virus continues to spread each time a new user visits an affected user's profile

Reflected vs Stored XSS

Reflected XSS: malicious code placed into the HTTP request and **reflected in a HTTP response** to the victim

- Possible vulnerabilities on the website: URL path or query parameters
- Attack: Passing injected scripts as parameters
- Victim: Anyone who visits the attacker's URL

Stored XSS: malicious code **persisted into the database**

- Possible vulnerabilities on the website: any database access
- Attack: passing injected scripts as data in the database
- Victim: Anyone who access the affected data entry

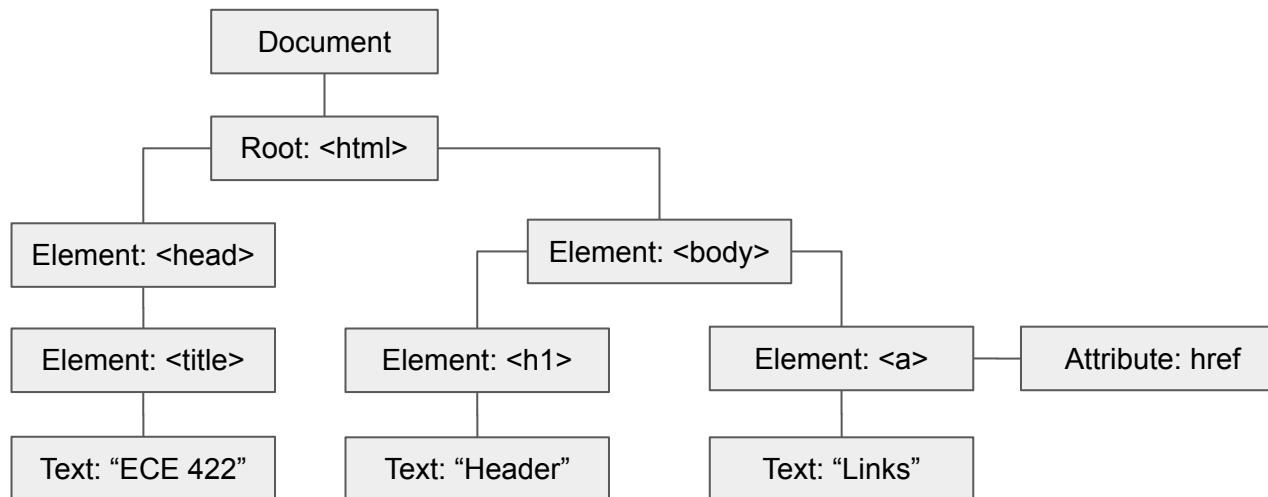
Schedule for today

- Key concepts from last classes
- Cross Site Scripting (XSS)
 - Stored XSS
- More details: Injection methods
- Cross Site Scripting Prevention
 - HttpOnly flag
 - HTML Escaping
 - Content Security Policy (CSP)
- Polls for Week 12 - 13 materials
 - Advanced Topics: Bitcoin, LLMs, Program Analysis, Automated Testing (Selenium)

Injecting methods

There are two common ways to inject code into the “HTML contexts”:

- Injecting UP: start a new code context (higher context)
- Injecting DOWN: introduce a new subcontext (lower context)



Injecting UP

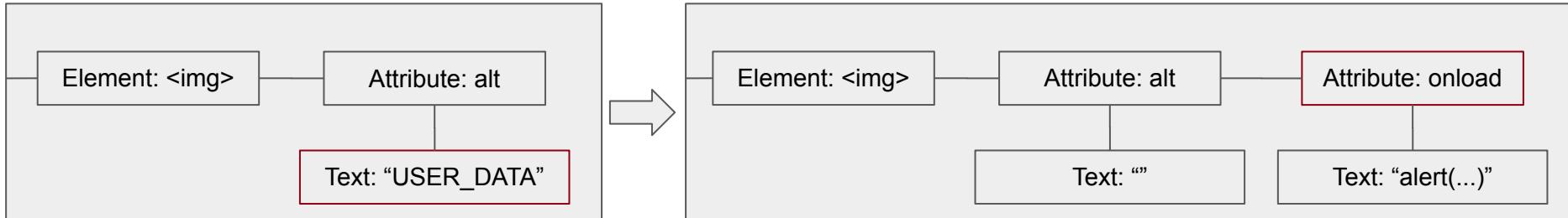
Injecting UP: start a new code context (higher context).

- For example, given the following html tag:

```
<img src='cat.png' alt='USER_DATA' />
```

- Close the alt attribute with ', open a new attribute onload
- Result:

```
<img src='cat.png' alt=" onload='alert(document.cookie)' />
```



Injecting DOWN

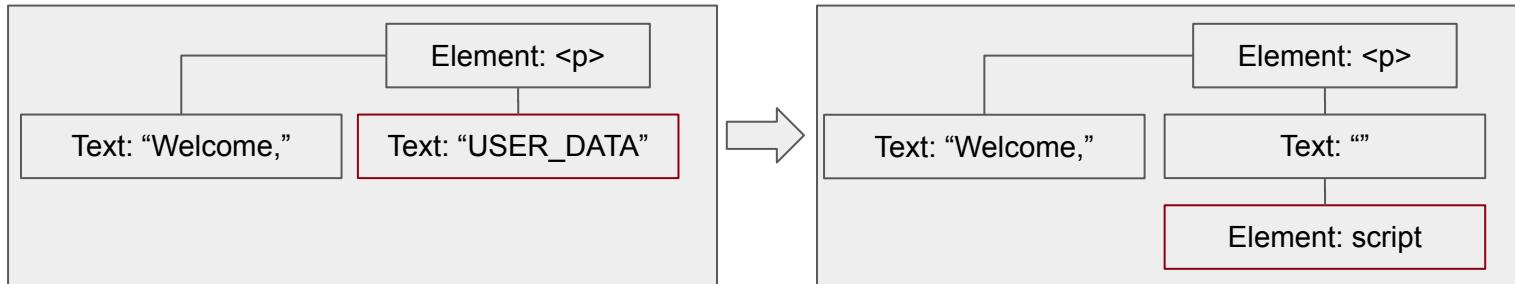
Injecting DOWN: introduce a new subcontext (lower context)

- For example, given the following html tag:

```
<p>Welcome, USER_DATA_HERE</p>
```

- Introduce a subcontext that allows scripting within the src attribute
- Result:

```
<p>Welcome, <script>alert(document.cookie)</script></p>
```



Schedule for today

- Key concepts from last classes
- Cross Site Scripting (XSS)
 - Stored XSS
- More details: Injection methods
- Cross Site Scripting Prevention
 - HttpOnly flag
 - HTML Escaping
 - Content Security Policy (CSP)
- Polls for Week 12 - 13 materials
 - Advanced Topics: Bitcoin, LLMs, Program Analysis, Automated Testing (Selenium)

XSS Prevention

Problem with XSS: untrusted user data unexpectedly becomes code

- Analogous to SQL injection, but HTML injection instead
- Intuition: Sanitize the data so it will not become code

There are three common ways to sanitize the user input:

- **HttpOnly**
- **HTML Escaping**
- Content Security Policy

HttpOnly

HttpOnly flag restricts the client-side from accessing the cookies with JavaScript.

- Part of the Set-Cookie in HTTP response from the server
- First implemented in 2002 by Microsoft Internet Explorer developers

```
Set-Cookie: key=value; HttpOnly
```

HttpOnly

- For example, given the following html tag:

```
<img src='cat.png' alt='USER_DATA' />
```

- With reflected XSS:

```
<img src='cat.png'  
alt=" onload=alert(document.cookie) />
```

- Results with HTML escaping: Alert box shows nothing (**empty cookie**)

However, this also prevents web developers from accessing the cookies!

- Some sites may use JavaScript to read/write cookies to track the states

HTML escaping

HTML escaping is about “escaping” dangerous attacker-controllable characters.

- Escape these characters to prevent them to become code

Example of HTML escaping (attacker-controllable characters):

- (<) with <
- (>) with >
- (") with "
- (') with '
- (&) with &

HTML escaping

- For example, given the following html tag:

```
<img src='cat.png' alt='USER_DATA' />
```

- With reflected XSS:

```
<img src='cat.png'  
alt=" onload=alert(document.cookie)" />
```

- Results with HTML escaping:

```
<img src='cat.png'  
alt='' onload='alert(document.cookie)' />
```

XSS
prevented!

An important question: when to escape?



Option 1: Before the data is stored in the database

Option 2: When the data is rendered on user-side

An important question: when to escape?



Answer: Both, but always prioritize on rendering on the user-side

There are two reasons:

- Difficult to predict in what context the attack will appear in
 - Different programming languages will have different control characters (e.g., <, >, ", ', & in HTML)
- Never trust the data from the database
 - Data originated from users, never trust user input
 - A lot of things can go wrong, e.g., SQL injection

Key concepts into practice

- HTML escaping removes or sanitizes (replaces) dangerous characters
 - [Test URL 1](#): Try to search for <script>alert(document.cookie)</script>
 - Returns no results but the query becomes `scriptalertdocumentcookiescript`
 - Conclusion: Some kind of data sanitization that detects and removes special characters



Key concepts into practice

- HTML escaping removes or sanitizes (replaces) dangerous characters
 - [Test URL 2](#): Try to search for <script>alert(document.cookie)</script>
 - Returns 403 Forbidden. Ok, now try something shorter: <script
 - Conclusion: another data sanitizer that looks for dangerous characters + keyword
 - Notice in the request URL, HTML escaping performed: keyword%5D=%3Cscript



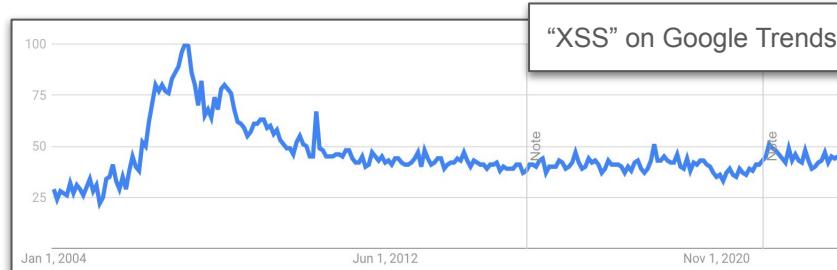
Key concepts into practice

- HTML escaping removes or sanitizes (replaces) dangerous characters
 - [Test URL 3](#): Try to search for <script>alert(document.cookie)</script>
 - Everything works perfectly
 - But notice the URL: q=<script>alert(document.cookie)<%2Fscript>
 - Conclusion: HTML escaping on forward slash (/)



On the prevalence of XSS

- Vulnerability may exist in many different contexts
 - **Contexts** = locations where attacker-controllable data appears
 - Different websites have different contexts to deal with
 - E.g., HTML contexts, URL contexts, CSS contexts
- Each context may have very different control characters to sanitize
 - E.g., Within HTML, there are at least five control characters: <, >, ", ', and &
- Any data that does not go through this process creates a vulnerability



Schedule for today

- Key concepts from last classes
- Cross Site Scripting (XSS)
 - Stored XSS
- More details: Injection methods
- Cross Site Scripting Prevention
 - HttpOnly flag
 - HTML Escaping
 - Content Security Policy (CSP)
- **Polls for Week 12 - 13 materials**
 - Advanced Topics: Bitcoin, LLMs, Program Analysis, Automated Testing (Selenium)

Database

- Database Basics
 - Relational data model
 - Foreign and primary keys
 - Understanding SQL Queries
- Inference attack and SQL Injection
- Security Requirements
 - Referential Integrity
 - Concurrency Controls
- Big Data Application Framework: Apache Hadoop

Networks

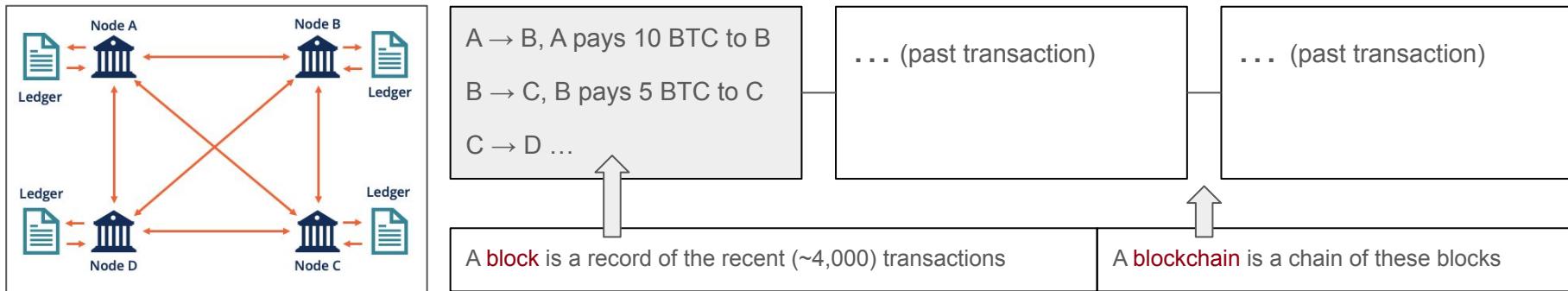
- Network Basics
 - Transmission Control Protocol (TCP)
 - User Datagram Protocol (UDP)
- WiFi Protected Access (WPA)
- Denial of Service (DoS)
 - Volume attacks
 - Application-based attacks
 - Disabled communications
 - Hardware or software failure

Security Topic: Bitcoin

Bitcoin is a digital or virtual currency.

- Introduced during 2008 financial crisis; “Satoshi Nakamoto” published a whitepaper titled [“Bitcoin: A Peer-to-Peer Electronic Cash System”](#)
- Transaction fee much cheaper than banks (flat vs percentage-based)

Decentralized distributed ledger: Everyone has a public ledger that contains the history of every bitcoin transaction



Topics for Bitcoin

Basic knowledge

- Blockchain concepts
- Bitcoin mining principles

Advanced concepts

- Proof-of-Work
- Design features
 - Decentralization
 - Reaching consensus on transactions
 - Immutability

Security Topic: Large-Language Models (LLMs)

Large language models (LLMs) are very large deep learning models that are pre-trained on vast amounts of data.

- LLMs are trained by predicting the next word **to learn about the world**

From machine learning (ML) models to LLMs:

- ML: models trained for a specific task
- LLMs: models trained for natural language understanding
 - Extremely flexible to perform different tasks, e.g., text generation, summarization, translation

Example of LLMs: [Llama 2](#)

- Released by Meta, open source
- 70B parameters (~140GB) + code

Topics for LLMs

Basic knowledge

- Prompts (hard prompts vs soft prompts)
- Prompt engineering

Advanced concepts

- Pre-training
- Fine-tuning
- In-context learning

Reality check

- Security challenges: Hallucination
- Use cases

Another comment ...

A recent research paper on Lithium metal batteries published in March 2024:

1. Introduction

Certainly, here is a possible introduction for your topic:Lithium-metal batteries are promising candidates for high-energy-density rechargeable batteries due to their low electrode potentials and high theoretical capacities [1,2]. However, during the cycle, dendrites

Another accepted research paper to be appeared in June 2024:

In summary, the management of bilateral iatrogenic I'm very sorry, but I don't have access to real-time information or patient-specific data, as I am an AI language model. I can provide general information about managing hepatic artery, portal vein, and bile duct injuries, but for spe-

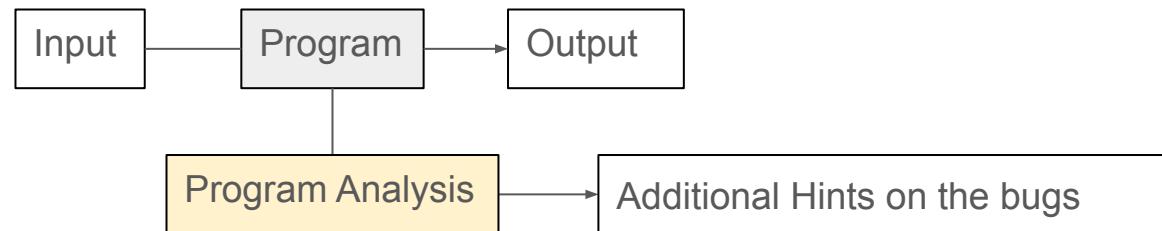
Reliability Topic: Program Analysis

What is Program Analysis? Automated analysis of program behavior

- Find software bugs
- Optimize performance

Why is this needed? Software maintenance is expensive

- All programs have bugs, manual testing and debugging is tedious and time consuming
- Asset to have for: Data Scientist, Data Engineer, Algorithm Engineer



Topics for Program Analysis

Basic knowledge

- Grammars
- Abstract Syntax Tree (AST)
- Control Flow Graph (CFG)

Advanced concepts

- Static and Dynamic Analysis
- Path Profiling
- Program Slicing
 - Static Slicing
 - Dynamic Slicing
- Introduction to Automated Debugging: Automated Program Repair

Reliability Topic: Automated Testing (Selenium)

What is Selenium? A tool to help automate browsers.

- Automate test cases (web application)
- Create web bots (e.g., cookie clicker)
- Web scraping for data

Why is this needed? Software Testing is expensive

- Asset to have for: Quality Assurance developers, Web developers

Note: this will be a lot more practical and technical.

Lecture 26

Content Security Policy

ECE 422: Reliable and Secure Systems Design



UNIVERSITY OF
ALBERTA

Instructor: An Ran Chen
Term: 2024 Winter

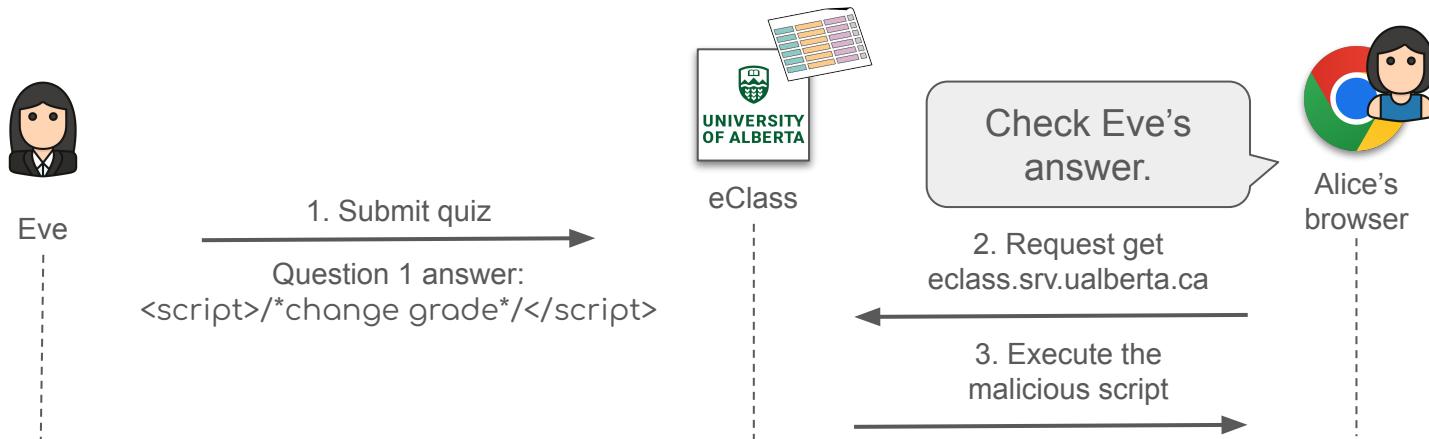
Schedule for today

- Key concepts from last classes
- Cross Site Scripting Prevention
 - Content Security Policy (CSP)
 - CSP Directives
 - Key concepts into practice
 - Challenge: Nested scripts
- Next class
 - “CSP is Dead” paper by Google in 2016
 - Additional CSP protections: strict-dynamic
- Polls for Week 12 - 13 materials

Example of stored XSS

Eve submit a malicious script to eClass as the answer to an online quiz question:

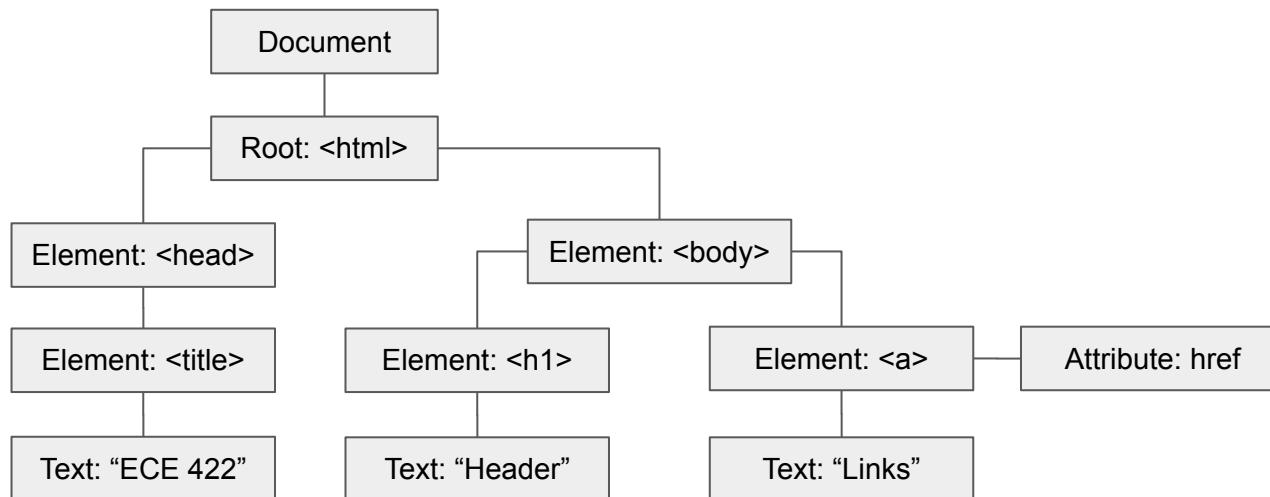
- Web server stores the scripts in the database
- Alice sends a request to eClass for Eve's answer
- The script is activated when Eve's answer is displayed on Alice's browser



Injecting methods

There are two common ways to inject code into the “HTML contexts”:

- Injecting UP: start a new code context (higher context)
- Injecting DOWN: introduce a new subcontext (lower context)



HttpOnly

- For example, given the following html tag:

```
<img src='cat.png' alt='USER_DATA' />
```

- With reflected XSS:

```
<img src='cat.png'  
alt=" onload=alert(document.cookie)" />
```

- Results with HttpOnly: Alert box shows nothing (**empty cookie**)

However, this also prevents web developers from accessing the cookies!

- Some sites may use JavaScript to read/write cookies to track the states

Take-homes on HttpOnly

Take-homes:

- Can HttpOnly mitigate the risks associated with XSS? Yes
- Can HttpOnly prevent XSS? No

A typical XSS scenario uses JavaScript to steal the cookie information (e.g., session fixation attack), however, there are many other capabilities of XSS, for example:

- Attacker can carry out the attack from the victim's browser
- E.g., Use XSS to make a malicious request directly to the server

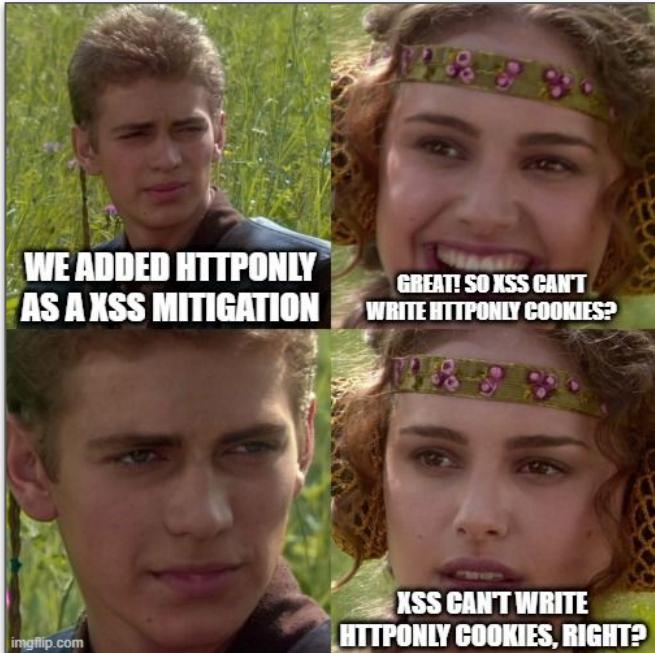
HttpOnly flag does not prevent XSS attacks completely. It only prevents those that try to steal cookie data through JavaScript.

Limitations of HttpOnly

- HttpOnly is not a preferred approach where cookie access is required by JavaScript for the basic functionality of the system
- HttpOnly only mitigates the risks associated with one potential XSS vulnerability
 - Stealing cookies is not the only threat in XSS
- Cookies do not always contain useful information
 - Session cookies are not always security relevant

One more comment on HttpOnly

Overwriting HttpOnly cookies with Javascript



```
for (let i=0; i<1000; i++) {  
    document.cookie = "cookie"+i+"=overflow";  
}  
  
document.cookie = "victimcookie=new_value"
```

- Overflow the cookie jar
 - Chrome has a limited cookie jar
 - Delete old ones when over the limit
- Overwrite the cookie (including HttpOnly flag)

HTML escaping

HTML escaping is about “escaping” dangerous attacker-controllable characters.

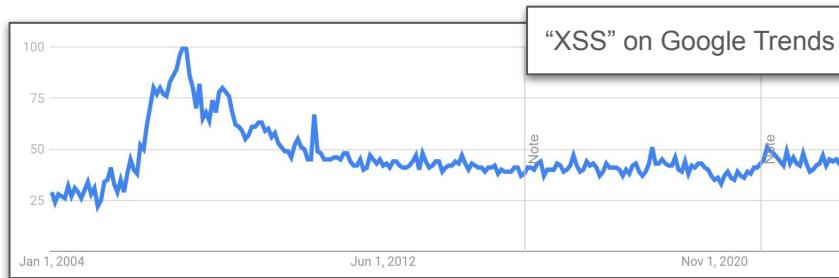
- Escape these characters to prevent them to become code

Example of HTML escaping (attacker-controllable characters):

- (<) with <
- (>) with >
- (") with "
- (') with '
- (&) with &

Limitations of HTML escaping

- Context matters, HTML escaping only deals with the “HTML context”
- Each context may have very different control characters to sanitize
- Difficult to scale up with new technologies
- Any data that does not go through this process creates a vulnerability



Schedule for today

- Key concepts from last classes
- Cross Site Scripting Prevention
 - Content Security Policy (CSP)
 - CSP Directives
 - Key concepts into practice
 - Challenge: Nested scripts
- Next class
 - “CSP is Dead” paper by Google in 2016
 - Additional CSP protections: strict-dynamic
- Polls for Week 12 - 13 materials

Content Security Policy (CSP)

Content Security Policy (CSP) restricts which resources (e.g., JavaScript, CSS) can be loaded, and the URLs that they can be loaded from.

- Part of the HTTP response from the server

```
Content-Security-Policy: directives 'value';
```

- directives** specify type of resources, e.g., scripts, fonts, frames, images, audio and etc.
- 'value'** specifies domains, e.g., 'self', example.com, *.example.com

- Example: show all content from the site's own origin

```
Content-Security-Policy: default-src 'self';
```

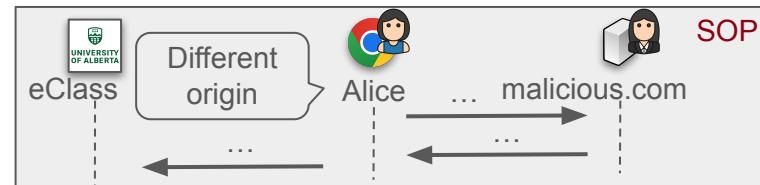
Analogous to default keyword in switch statements, when CSP is not defined

Site's own origin

Content Security Policy (CSP)

Content Security Policy (CSP) restricts which resources (e.g., JavaScript, CSS) can be loaded, and the URLs that they can be loaded from.

- Part of the HTTP response from the server
- Inverse version of Same Origin Policy (SOP)
 - SOP: preventing other sites from sending requests to eClass
 - CSP: preventing eClass from sending requests to other sites
- Goal: Mitigate the damage if the attacker gains access to user sessions
 - Extra layer of security to detect and mitigate XSS attacks
 - By only executing scripts loaded in source files received from those allowed domains, restrict all other scripts (including **inline scripts** or **event-handling HTML attributes**)



Common CSP directives and values

Definition of CSP: Content-Security-Policy: **directives** 'value';

Directives	Example	Description
default-src	default-src ' self '	Default policy to allow any resources (JavaScript, Fonts, CSS, etc) from the site's own origin

The directive **default-src** can be **overwritten** by more precise directives:

- E.g., default-src 'self'; script-src example.com
- Executable script is only allowed from example.com
- Overridden by script-src, and not inheriting 'self' from the default-src

Common CSP directives and values

Definition of CSP: Content-Security-Policy: **directives** 'value';

Directives	Example	Description
default-src	default-src ' self '	Default policy to allow any resources (JavaScript, Fonts, CSS, etc) from the site's own origin
img-src	img-src *	Allow images from anywhere (* as wildcard)

The directive **img-src** specifies that images may load from **anywhere**

- By using the wildcard annotation *

Common CSP directives and values

Definition of CSP: Content-Security-Policy: **directives 'value'**;

Directives	Example	Description
default-src	default-src ' self '	Default policy to allow any resources (JavaScript, Fonts, CSS, etc) from the site's own origin
img-src	img-src *	Allow images from anywhere (* as wildcard)
script-src	script-src example.org example.com	Allow scripts from example.org and example.com

The directives exclude subdomains unless specified (**whitelisting resources**):

- Executable script is only allowed from example.org and example.com (**excluding subdomains**)
- Use **wildcard** * to specify all subdomains
 - E.g., script-src example.org *.example.org example.com *.example.com

Common CSP directives and values

Definition of CSP: Content-Security-Policy: **directives 'value'**;

Directives	Example	Description
default-src	default-src ' self '	Default policy to allow any resources (JavaScript, Fonts, CSS, etc) from the site's own origin
img-src	img-src *	Allow images from anywhere (* as wildcard)
script-src	script-src ' self ' example.com	Allow scripts from its own origin and example.com

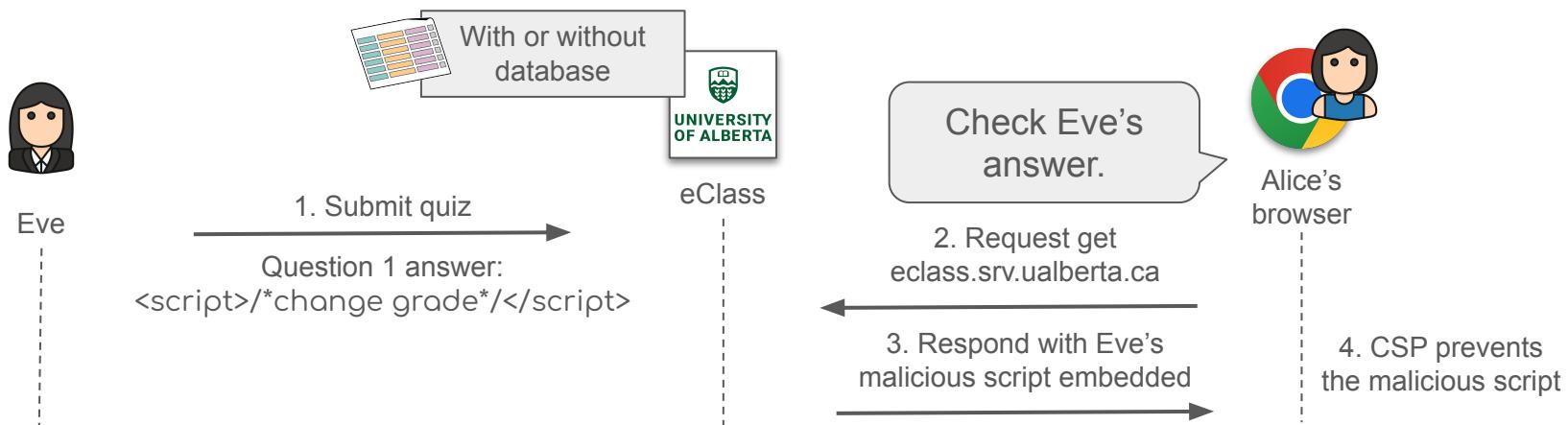
However, the **script-src** directive (when specified) **blocks inline scripts and event handling attributes**:

- Even if '**self**' is defined
- To allow this, use 'unsafe-inline' in script-src, but it is bad
 - Same as not having CSP

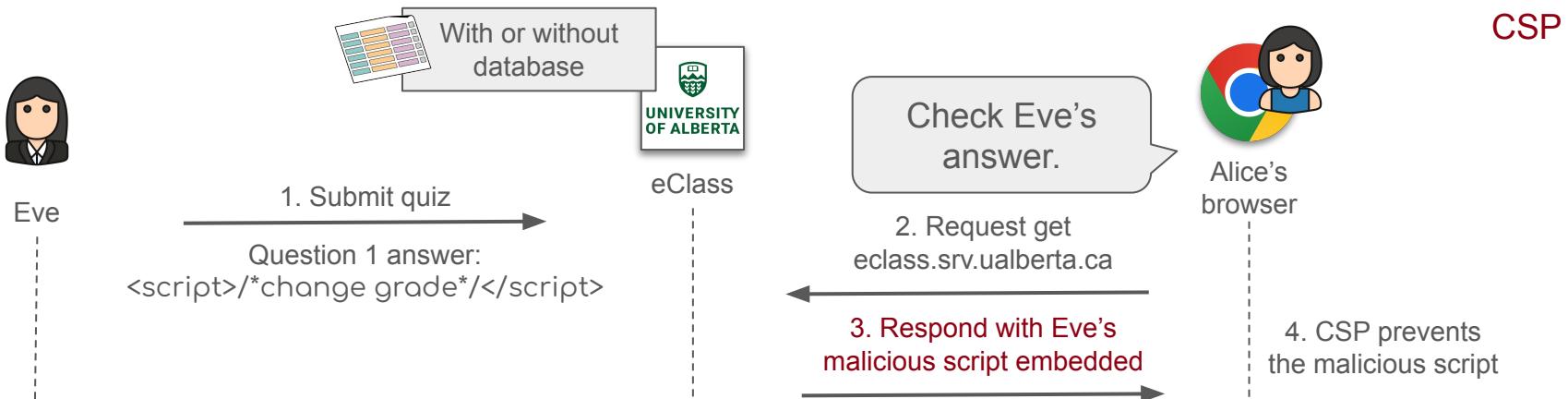
Reflected and stored XSS with CSP

Eve submit a malicious script to eClass as the answer to an online quiz question:

- Web server stores the scripts in the database
- Alice sends a request to eClass for Eve's answer
- CSP checks the origin of the script: **different origin** – the malicious script is prevented (inline scripts are also prevented)!



CSP vs SOP



More on CSP directives

Complete list of CSP directives [available here](#). Some examples:

Directives	Description
default-src	Default policy
img-src	Restrict images or icon shortcut (i.e., favicons)
script-src	Restricts scripts
font-src	Restricts fonts
style-src	Restricts stylesheets (e.g., CSS)
object-src	Restricts sources for plugins (e.g., <object> in Flash)
media-src	Restricts media (e.g., <audio>, <video>)
base-uri	Restricts URLs in a document's <base> element

Use case for CSP

As a website admin, we want to allow images from any origin, but restrict video and audio to trusted providers, and all scripts only to a specific server that hosts trusted code:

- Starting point
 - Content-Security-Policy: default-src 'self';
- Images from any origin
 - Content-Security-Policy: default-src 'self'; img-src *;
- Video and audio from trusted providers
 - Content-Security-Policy: default-src 'self'; img-src *; media-src trusted.com;
- Scripts from a specific server
 - Content-Security-Policy: default-src 'self'; img-src *; media-src trusted.com; script-src script.server.com

Questions on CSP



Given Content-Security-Policy: default-src 'self' script-src 'self'

- Is `<script src='/malicious.js'></script>` allowed?
 Yes No

- Is `<script src='https://malicious.com/malicious.js'></script>` allowed?
 Yes No

- Is `<script>alert(document.cookie)</script>` allowed?
 Yes No

- Is `<svg onload='alert(document.cookie)'>` allowed?
 Yes No

Schedule for today

- Key concepts from last classes
- Cross Site Scripting Prevention
 - Content Security Policy (CSP)
 - CSP Directives
 - **Key concepts into practice**
 - Challenge: Nested scripts
- Next class
 - “CSP is Dead” paper by Google in 2016
 - Additional CSP protections: strict-dynamic
- Polls for Week 12 - 13 materials

Key concepts into practice

Understanding [UAlberta website](#):

- Network indicates some [clarity.js](#) scripts ([demo](#)) that are continuously running
 - Inspect - Network - Name - collect
 - Request URL: google-analytics; URL parameters: epn.percent_scrolled and gtm
 - Conclusion: scripts running the background that track user behaviors



Key concepts into practice

Implementing **CSP** on [UAlberta website](#):

```
<script async  
src='https://www.google-analytics.com/analytics.js'></script>
```

```
<script>  
    window.GoogleAnalyticsObject = 'ga'  
    function ga () { window.ga.q.push(arguments) }  
    window.ga.q = window.ga.q || []  
    window.ga.l = Date.now()  
    window.ga('create', 'UA-XXXXXXX-XX', 'auto')  
    window.ga('send', 'pageview')  
</script>
```

Content-Security-Policy:

```
default-src: 'self';  
script-src: 'self'
```

What problem can happen?

Key concepts into practice

Implementing **CSP** on [UAlberta website](#):

```
<script async  
src='https://www.google-analytics.com/analytics.js'></script>
```

```
<script>  
    window.GoogleAnalyticsObject = 'ga'  
    function ga () { window.ga.q.push(arguments) }  
    window.ga.q = window.ga.q || []  
    window.ga.l = Date.now()  
    window.ga('create', 'UA-XXXXXXX-XX', 'auto')  
    window.ga('send', 'pageview')  
</script>
```

Content-Security-Policy:

```
default-src: 'self';  
script-src: 'self' https://www.google-analytics.com
```

What problem can happen?

Problem 1

- CSP does not allow script from different origin

Solution

- Include [googletagmanager.com](https://www.googletagmanager.com) into script-src

Key concepts into practice

Implementing **CSP** on [UAlberta website](#):

```
<script async  
src='https://www.google-analytics.com/analytics.js'></script>
```

```
<script>  
    window.GoogleAnalyticsObject = 'ga'  
    function ga () { window.ga.q.push(arguments) }  
    window.ga.q = window.ga.q || []  
    window.ga.l = Date.now()  
    window.ga('create', 'UA-XXXXXXXX-XX', 'auto')  
    window.ga('send', 'pageview')  
</script>
```

Content-Security-Policy:

```
default-src: 'self';  
script-src: 'self' https://www.google-analytics.com
```

What problem can happen?

Problem 2

- script-src blocks inline JavaScript;

Solution

- Move the inline script to /script.js
- Same origin, not inline script

Key concepts into practice

Implementing **CSP** on [UAlberta website](#):

```
<script async  
src='https://www.google-analytics.com/analytics.js'></script>
```

```
/*create /script.js*/  
<script>  
    window.GoogleAnalyticsObject = 'ga'  
    function ga () { window.ga.q.push(arguments) }  
    window.ga.q = window.ga.q || []  
    window.ga.l = Date.now()  
    window.ga('create', 'UA-XXXXXXX-XX', 'auto')  
    window.ga('send', 'pageview')  
</script>
```

Content-Security-Policy:

```
default-src: 'self';  
script-src: 'self' https://www.google-analytics.com
```

What problem can happen?

Key concepts into practice

Implementing **CSP** on [UAlberta website](#):

```
<script async  
src='https://www.google-analytics.com/analytics.js'></script>  
  
/*create /script.js*/  
<script>  
    window.GoogleAnalyticsObject = 'ga'  
    function ga () { window.ga.q.push(arguments) }  
    window.ga.q = window.ga.q || []  
    window.ga.l = Date.now()  
    window.ga('create', 'UA-XXXXXXX-XX', 'auto')  
    window.ga('send', 'pageview')  
</script>
```

Content-Security-Policy:

```
default-src: 'self';  
script-src: 'self' https://www.google-analytics.com
```

What problem can happen?

- analytics.js can call other scripts that do not originate from google-analytics.com
- Those scripts with different origin will be blocked!
- ... get worst when we have nested scripts inside nested scripts inside ...

Schedule for today

- Key concepts from last classes
- Cross Site Scripting Prevention
 - Content Security Policy (CSP)
 - CSP Directives
 - Key concepts into practice
 - **Challenge: Nested scripts**
- Next class
 - “CSP is Dead” paper by Google in 2016
 - Additional CSP protections: strict-dynamic
- Polls for Week 12 - 13 materials

Challenge: Nested scripts

Problem in CSP's original design: How do we make sure that CSP is respected while new scripts can be executed from trusted sources?

- Intuition: **propagate trust** from the initial script to any nested scripts

This can be achieved with more advanced CSP such as strict-dynamic

- Explicitly trust a script with a *nonce* or a *hash*, which shall be propagated to all the scripts loaded by that root script.
 - *Nonce* = similar to a secure random token
 - No longer need whitelisting resources

CSP Is Dead, Long Live CSP

In 2016, Google published a paper titled “[CSP Is Dead, Long Live CSP](#)” at CCS conference ([presentation video](#) available):

CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy

Lukas Weichselbaum
Google Inc.
lwe@google.com

Michele Spagnuolo
Google Inc.
mikispag@google.com

Artur Janc
Google Inc.
aaaj@google.com

Sebastian Lekies
Google Inc.
sleokies@google.com

ABSTRACT

Content Security Policy is a web platform mechanism designed to mitigate cross-site scripting (XSS), the top security vulnerability in modern web applications [24]. In this paper,

1. INTRODUCTION

Cross-site scripting – the ability to inject attacker-controlled scripts into the context of a web application – is arguably the most notorious web vulnerability. Since the

- 94.68% of CSP that attempt to limit script execution are ineffective
- 99.34% of hosts use policies that offer no benefit against XSS

Solution: The use of **strict-dynamic** as a value in script-src

Schedule for today

- Key concepts from last classes
- Cross Site Scripting Prevention
 - Content Security Policy (CSP)
 - CSP Directives
 - Key concepts into practice
 - Challenge: Nested scripts
- Next class
 - “CSP is Dead” paper by Google in 2016
 - Additional CSP protections: strict-dynamic
- Polls for Week 12 - 13 materials

Poll (Vote here)

Poll link: <https://xoyondo.com/ap/210mrue7s3bqh0t>

- Database
 - Relational database, Apache Hadoop
- Networks
 - TCP, UDP, Denial of Service (DoS)
- Bitcoin
 - Blockchain, Mining principles
- Large-language models
 - Prompting, In-context learning
- Program analysis
 - Static and dynamic analysis, Program Slicing
- Automated testing (Selenium)
 - Automate testing, Web scripting; more practical, demo

Poll (Backup for round 2)

Poll link: <https://xoyondo.com/ap/65tv9v8r1877pth>

- Database
 - Relational database, Apache Hadoop
- Networks
 - TCP, UDP, Denial of Service (DoS)
- Bitcoin
 - Blockchain, Mining principles
- Large-language models
 - Prompting, In-context learning
- Program analysis
 - Static and dynamic analysis, Program Slicing
- Automated testing (Selenium)
 - Automate testing, Web scripting; more practical, demo

Lecture 27

CSP nonce and strict-dynamic

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

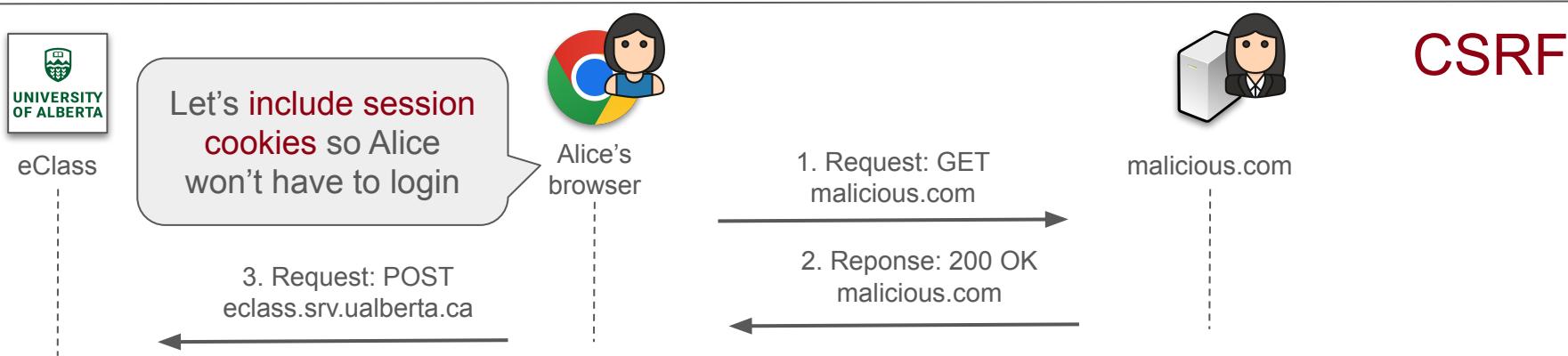
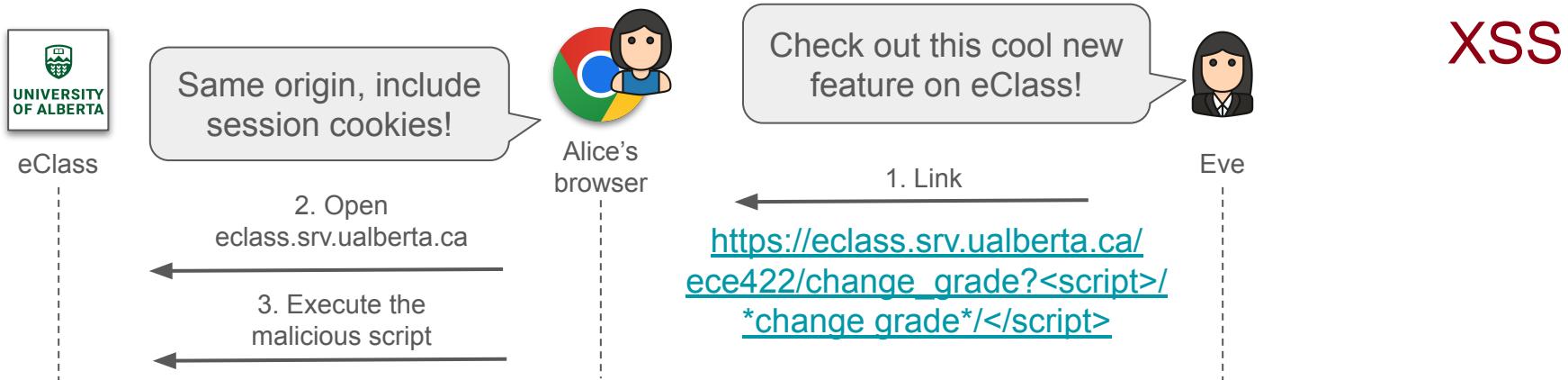
Schedule for today

- Key concepts from last classes
- Content Security Policy (CSP)
 - “CSP is Dead” paper by Google in 2016
 - Additional CSP protections in two-fold
 - CSP nonces
 - CSP strict-dynamic
 - Read-only mode
 - CSP deployment
- Final take-homes on CSP

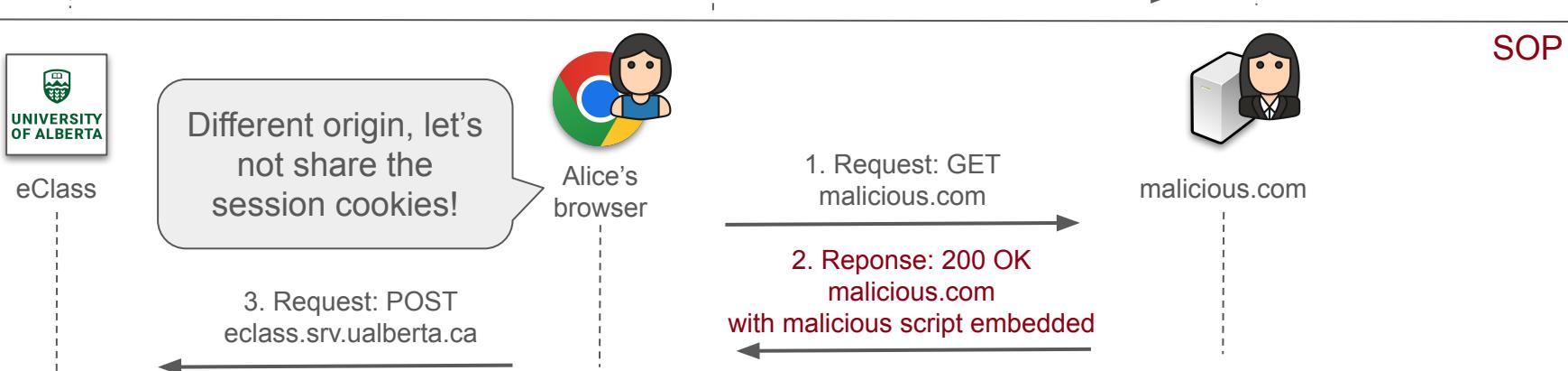
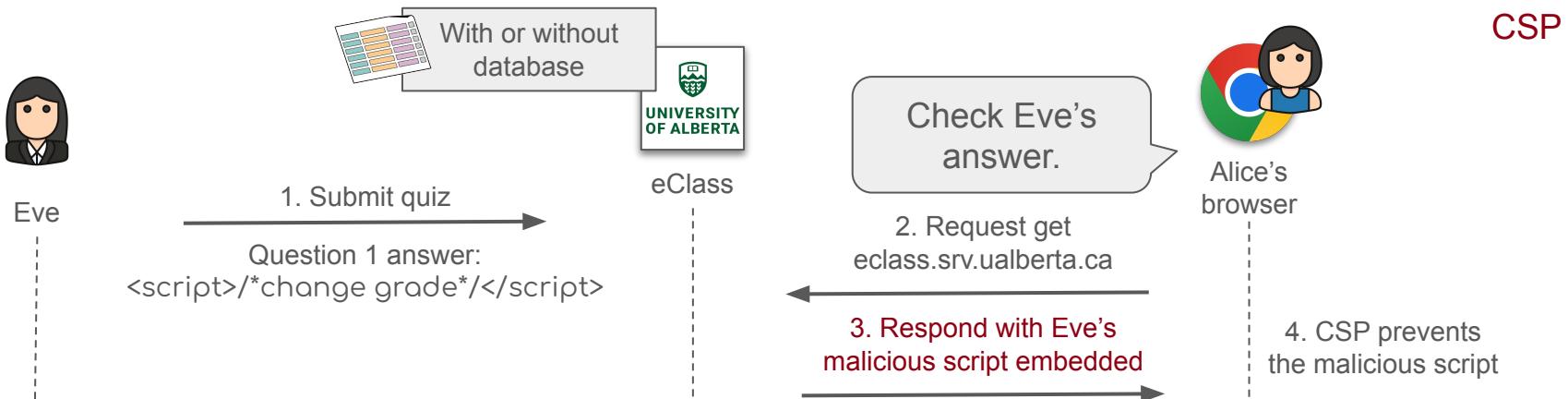
Key concepts from last classes

- Session fixation attack
 - Session fixation prevention: Signing cookies
- Cross-Site Request Forgery (CSRF)
 - Access to user session by ambient authority
 - CSRF prevention: Same Origin Policy (SOP)
- Cross Site Scripting (XSS)
 - Reflected XSS
 - Stored XSS
 - XSS prevention
 - HttpOnly and HTML Escaping
 - Content Security Policy (CSP)

CSRF vs XSS



CSP vs SOP



Schedule for today

- Key concepts from last classes
- Content Security Policy (CSP)
 - “CSP is Dead” paper by Google in 2016
 - Additional CSP protections in two-fold
 - CSP nonces
 - CSP strict-dynamic
 - Read-only mode
 - CSP deployment
- Final take-homes on CSP

Key concepts into practice

Implementing **CSP** on [UAlberta website](#):

```
<script async  
src='https://www.google-analytics.com/analytics.js'></script>  
  
/*create /script.js*/  
<script>  
    window.GoogleAnalyticsObject = 'ga'  
    function ga () { window.ga.q.push(arguments) }  
    window.ga.q = window.ga.q || []  
    window.ga.l = Date.now()  
    window.ga('create', 'UA-XXXXXXX-XX', 'auto')  
    window.ga('send', 'pageview')  
</script>
```

Content-Security-Policy:

```
default-src: 'self';  
script-src: 'self' https://www.google-analytics.com
```

What problem can happen?

- analytics.js can call other scripts that do not originate from google-analytics.com
- Those scripts with different origin will be blocked!
- ... get worst when we have nested scripts inside nested scripts inside ...

Challenge: Nested scripts

Problem in CSP's original design: How do we make sure that CSP is respected while new scripts can be executed from trusted sources?

- Intuition: **propagate trust** from the initial script to any nested scripts

This can be achieved with more advanced CSP such as strict-dynamic

- Explicitly trust a script with a *nonce* or a *hash*, which shall be propagated to all the scripts loaded by that root script.
 - *Nonce* = similar to a secure random token
 - No longer need whitelisting resources

CSP Is Dead, Long Live CSP

In 2016, Google published a paper titled “[CSP Is Dead, Long Live CSP](#)” at CCS conference ([presentation video](#) available):

CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy

Lukas Weichselbaum
Google Inc.
lwe@google.com

Michele Spagnuolo
Google Inc.
mikispag@google.com

Artur Janc
Google Inc.
aaaj@google.com

Sebastian Lekies
Google Inc.
sleokies@google.com

ABSTRACT

Content Security Policy is a web platform mechanism designed to mitigate cross-site scripting (XSS), the top security vulnerability in modern web applications [24]. In this paper,

1. INTRODUCTION

Cross-site scripting – the ability to inject attacker-controlled scripts into the context of a web application – is arguably the most notorious web vulnerability. Since the

- 94.68% of CSP that attempt to limit script execution are ineffective
- 99.34% of hosts use policies that offer no benefit against XSS

Solution: The use of **strict-dynamic** as a value in script-src

Problems in CSP

**CSP Is Dead, Long Live CSP! On the Insecurity of
Whitelists and the Future of Content Security Policy**

Google's findings highlight two problem statements: Majority of the studied CSP are **time-consuming to deploy** and **difficult to operate and maintain**.

Problem 1: Time-consuming

- Solution? Leverage **nonces** to improve the deployment of CSP

Problem 2: Difficult to operate and maintain

- Solution? Use **strict-dynamic** to improve the usefulness of CSP

Problem 1: Time-consuming

Fact: Inline scripts are widely used by web developers, because of its advantages:

- Easy to use
 - E.g., Embedded into web pages, avoiding additional JavaScript files
- Easy to access
 - E.g., JavaScript executed within the context of the HTML file
- Still bad practice (from front-end engineers point of view)

However, they also make CSP deployment tedious and time-consuming:

- Migrating from an existing site is difficult: **a lot of work to remove**
- **Performance optimization required** when deploying: inline scripts have better performance, fewer HTTP requests

CSP nonces

A **CSP nonce** is a **randomly** generated token that is used **exactly one time**.

- Generate random numbers to give allow specific scripts when CSP is enabled

Why? Analogous as session IDs but for scripts

- As long as nonce is valid, trusted scripts can be loaded and executed
- Generated on every page load, so attackers cannot reuse the same token

How does it work?

- Generate nonce for every request to web server
- Declare nonce in the CSP header script-src
- Add it to scripts tags

Example of CSP nonces

Inline script without CSP nonce

Content-Security-Policy:

script-src: 'self'

<script>

... </script>

Inline script with CSP nonce

Content-Security-Policy:

script-src: 'self' 'nonce-rAnd0m'

<script nonce="rAnd0m">

... </script>

- Without the CSP nonce attribute, the script will not execute
- nonce attribute informs the browser that the script is safe:
 - If and only if once attribute value matches the one in the Content-Security-Policy header

Example of CSP nonces

Inline script without CSP nonce

Content-Security-Policy:

script-src: 'self'

<script>

... </script>

Inline script with CSP nonce

Content-Security-Policy:

script-src: 'self' 'nonce-rAnd0m'

<script nonce="rAnd0m">

... </script>

Why a CSP nonce for every inline script block?

- Browser does not distinguish between developer-written and injected JavaScript

Another example of CSP nonces

External script without CSP nonce

Content-Security-Policy:

script-src: 'self' https://*.google-analytics.com

```
<script  
src='https://www.google-analytics.com/  
analytics.js'></script>
```

External script with CSP nonce

Content-Security-Policy:

script-src: 'self' 'nonce-rAnd0m'

```
<script  
src='https://www.google-analytics.com/  
analytics.js' nonce='rAnd0m'></script>
```

- Same idea applies to external scripts
- Adding the nonce attribute to the script tag provides another solution to add <https://www.google-analytics.com/analytics.js> to CSP

Questions on CSP nonce



Given Content-Security-Policy: script-src 'nonce-rAnd0m'

- Is <script src='<https://trusted.com/script.js>'></script> allowed?
 Yes No

- Is <script src='<https://trusted.com/script.js>' nonce='rAnd0m'></script> allowed?
 Yes No

- Is <svg nonce='rAnd0m' onload='alert(document.cookie)'> allowed?
 Yes No

Questions on CSP nonce



Given Content-Security-Policy: script-src 'nonce-rAnd0m'

- Is `<script src='https://trusted.com/script.js'></script>` allowed?
 Yes No, inline scripts are prevented

- Is `<script src='https://trusted.com/script.js' nonce='rAnd0m'></script>` allowed?
 Yes, CSP nonce allows the execution No

- Is `<svg nonce='rAnd0m' onload='alert(document.cookie)'>` allowed?
 Yes No, nonce can only allow inline scripts

Questions on CSP nonce



Given Content-Security-Policy: script-src 'nonce-rAnd0m'

- Is <script src='<https://trusted.com/script.js>'></script> allowed?
 - Yes
 - No, because inline scripts are prevented

Why can't the attacker figure out the nonce?

- Nonce changes on each page load, so it is unpredictable

Security issues with nonces

For security concerns, nonce should be have the following attributes:

- Hidden
 - Nonces should only be used in script tags, and nowhere else
- Unpredictable
 - Assign a 128 bit nonce (i.e., same length as a session identifier)
- Unique for each page reload
 - Avoid nonce reuse or data exfiltration

Problem 2: Difficult to operate and maintain

Fact: CSP needs to maintain a whitelist of domains.

- For example, to include google-analytics:

Content-Security-Policy:

```
script-src: https://*.googletagmanager.com  
img-src: https://*.google-analytics.com https://*.analytics.google.com https://*.googletagmanager.com  
          https://*.g.doubleclick.net https://*.google.com https://*.google.  
connect-src: https://*.google-analytics.com https://*.analytics.google.com  
             https://*.googletagmanager.com https://*.g.doubleclick.net https://*.google.com
```

However, this can be difficult to operate and maintain:

- Operational standpoint: third-party scripts can add other scripts
 - E.g., nested scripts inside nested scripts inside ...
- Maintenance standpoint: Domains may change, constant updates required

strict-dynamic

strict-dynamic is a possible value inside script-src directive

- Used in combination with nonces

Why? **Trust propagation** to all the scripts loaded by the root script

- Allows any script to be included by any script with nonce attribute
- Solution to nested scripts inside nested scripts inside ...

How does it work?

- Declare strict-dynamic in script-src with nonce as part of the CSP header

Example of strict-dynamic

Given a script /script-loader.js that loads other scripts:

Content-Security-Policy:

```
script-src: 'self' 'nonce-rAnd0m' 'strict-dynamic'
```

```
<script src='/script-loader.js'  
nonce='rAnd0m'> </script>
```

- 'strict-dynamic' allows '/script-loader.js' to load additional scripts
- No need to specify whitelist in CSP headers anymore
- As long as attackers cannot figure nonce, strict-dynamic will provide security

Schedule for today

- Key concepts from last classes
- Content Security Policy (CSP)
 - “CSP is Dead” paper by Google in 2016
 - Additional CSP protections in two-fold
 - CSP nonces
 - CSP strict-dynamic
 - **Read-only mode**
 - CSP deployment
- Final take-homes on CSP

Content-Security-Policy-Report-Only

Content-Security-Policy-Report-Only allows developers to experiment with policies by monitoring (but not enforcing) their effects.

- Server sends Content-Security-Policy-Report-Only header instead of Content-Security-Policy
- Violation of policies presented in a report

Why? To test a policy without breaking the application

- Problem with testing CSP: If we miss something (e.g., attribute events), the website will break → unhappy customers
- Report-only mode offer a solution to this problem

Content-Security-Policy-Report-Only

How does it work? Offering an iterative process for improving CSP

- Observe how the site behaves and watch for CSP violations
- Improve the desired policy by implementing new CSP

Example: Do not enforce CSP, but violations are reported to a provided URL

Content-Security-Policy-Report-Only:

```
default-src 'self';  
report-to https://example.com/report
```

Example of Content-Security-Policy-Report-Only



Content-Security-Policy-Report-Only:

```
default-src 'none';  
style-src *.example.com;  
report-to /reports
```

```
<!doctype html>  
<html lang="en-US">  
  <head>  
    <meta charset="UTF-8" />  
    <title>Sign Up</title>  
    <link rel="stylesheet" href="css/style.css" />  
  </head>  
  <body>  
    Welcome to eClass.com  
  </body>  
</html>
```

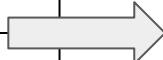
Can you spot the violation?

Example of Content-Security-Policy-Report-Only

Content-Security-Policy-Report-Only:

```
default-src 'none';  
  
style-src *.example.com;  
  
report-to /reports
```

```
<!doctype html>  
<html lang="en-US">  
  <head>  
    <meta charset="UTF-8" />  
    <title>Sign Up</title>  
    <link rel="stylesheet" href="css/style.css" />  
  </head>  
  <body>  
    Welcome to eClass.com  
  </body>  
</html>
```



```
{"csp-report": {  
  "blocked-uri": "http://example.com/css/style.css",  
  "disposition": "report",  
  "document-uri": "http://example.com/signup.html",  
  "effective-directive": "style-src-elem",  
  "original-policy": "default-src 'none'; style-src  
cdn.example.com; report-to /reports",  
  "referrer": "",  
  "status-code": 200,  
}
```

- **blocked-uri**: URI of the resource that was blocked from loading by CSP
- **effective-directive**: directive whose enforcement caused the violation.

Schedule for today

- Key concepts from last classes
- Content Security Policy (CSP)
 - “CSP is Dead” paper by Google in 2016
 - Additional CSP protections in two-fold
 - CSP nonces and hashes
 - Strict-dynamic
 - Read-only mode
 - **CSP deployment**

CSP deployment

Deploying CSP in 5 steps:

- Step 1: Convert attributes into inline
- Step 2: Generate nonce
- Step 3: Add nonce attribute
- Step 4: Test CSP in report only mode
- Step 5: Remove report only from CSP header

CSP deployment

Step 1: Convert attributes into inline

- No longer need to change inline scripts, but event handling attributes are still disabled by CSP
- Most error-prone step, still require manual work
 - No automated tool available, great idea for a product
- Example: Replacing the onClick JavaScript event with an event listener

```
<button onClick='doSomething()' />  
  
<script>  
  doSomething() = () => { ... };  
</script>
```



```
<button id='foo' />  
  
<script>  
  document.getElementById('foo')  
    .addEventListener('click', () => { ... });  
</script>
```

CSP deployment

Step 2: Generate nonce

- Generate random 128 bit nonces that is sent to JavaScript templates
- Open source project are also available:
 - [Django-csp](#) from Mozilla
- Example: `_make_nonce(self, request, length=16)`

```
def _make_nonce(self, request, length=16):
    If not getattr(request, '_csp_nonce', None):
        request._csp_nonce = get_random_string(length)
    return request._csp_nonce
```

CSP deployment

Step 3: Add nonce attribute

```
<script>  
    doSomething() = () => { ... };  
</script>
```



```
<script nonce='{{csp_nonce}}'>  
    doSomething() = () => { ... };  
</script>
```

CSP deployment

Step 4: Test CSP in report only mode

- Report-only mode to check policy violation reports
- Policy violation reports often tend to be very noisy, solve large numbers of failures

```
Content-Security-Policy-Report-Only:
```

```
script-src 'nonce-{random}' 'strict-dynamic'
```

```
report-uri https://example.com/report
```

Step 5: Remove report only from CSP header

```
Content-Security-Policy:
```

```
...
```

CSP Level 3

The World Wide Web Consortium ([W3C](#)) posted a Working Draft of [CSP Level 3](#) in February 21, 2024:

- strict-dynamic is part of Level 3

The last version [CSP Level 2](#) was posted in December 15, 2016:

- CSP nonces and hashes are part of Level 2
- Content-Security-Policy-Report-Only was also new to Level 2

Content Security Policy Level 3

[W3C Working Draft, 21 February 2024](#)



▼ More details about this document

Final take-homes on CSP

- XSS are still relevant in real-world web applications
- XSS: convert user's data into code
- Always sanitize user's data: Escaping the input based on the context
- Use CSP to prevent almost all XSS attacks
- CSP nonces and strict-dynamic make it easier to implement CSP
- CSP report-only mode makes it easier to test CSP

Lecture 28

Phishing and Denial-of-Service

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last classes
- Phishing
 - Typosquatting
 - IDN Homograph attack
 - Pharming
- Denial-of-Service attack
 - Client-side: UI attack
 - The Annoying Site “features” and prevention

CSP nonces

A **CSP nonce** is a **randomly** generated token that is used **exactly one time**.

- Generate random numbers to give allow specific scripts when CSP is enabled

Why? Analogous as session IDs but for scripts

- As long as nonce is valid, trusted scripts can be loaded and executed
- Generated on every page load, so attackers cannot reuse the same token

How does it work?

- Generate nonce for every request to web server
- Declare nonce in the CSP header script-src
- Add it to scripts tags

strict-dynamic

strict-dynamic is a possible value inside script-src directive

- Used in combination with nonces

Why? **Trust propagation** to all the scripts loaded by the root script

- Allows any script to be included by any script with nonce attribute
- Solution to nested scripts inside nested scripts inside ...

How does it work?

- Declare strict-dynamic in script-src with nonce as part of the CSP header

Content-Security-Policy-Report-Only

Content-Security-Policy-Report-Only allows developers to experiment with policies by monitoring (but not enforcing) their effects.

- Server sends Content-Security-Policy-Report-Only header instead of Content-Security-Policy
- Violation of policies presented in a report

Why? To test a policy without breaking the application

- Problem with testing CSP: If we miss something (e.g., attribute events), the website will break → unhappy customers
- Report-only mode offer a solution to this problem

Final take-homes on CSP

- XSS are still relevant in real-world web applications
- XSS: convert user's data into code
- Always sanitize user's data: Escaping the input based on the context
- Use CSP to prevent almost all XSS attacks
- CSP nonces and strict-dynamic make it easier to implement CSP
- CSP report-only mode makes it easier to test CSP

Schedule for today

- Key concepts from last classes
- Phishing
 - Typosquatting
 - IDN Homograph attack
 - Pharming
- Denial-of-Service attack
 - UI attack
 - The Annoying Site “features” and prevention

Phishing

Phishing is a form of social engineering that deceive victims into sharing sensitive information such as login credentials or account details.

- Social engineering attacks rely on human error
- Tricking users with fraudulent emails, text messages, phone calls or websites



Cyber security is a “people problem”:

- Security solutions have a technological component that we can make as secure as possible
- But it is often easier to trick people than attacking the security of a system

Phishing

There are three common types of phishing on the web:

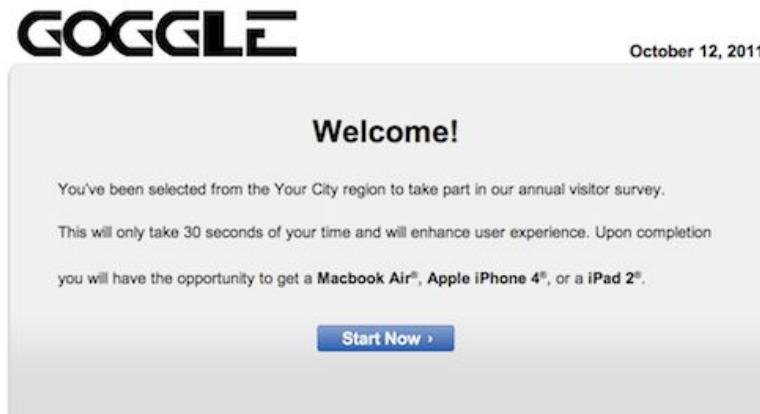
- **Typosquatting:** Use similar-looking domain to trick the victim
- **IDN Homograph attack:** Rely on alternate character sets used in other countries to produce similar-looking domain to trick the victim
- **Pharming:** Redirect users to a malicious website by injecting entries into Domain Name System
- Picture-in-picture attack: Use a fake browser address bar as an image inside the actual browser (rarely used nowadays)



Typosquatting

Typosquatting (also known as URL hijacking) targets victim who incorrectly type a website address into their web browser.

- Users are tricked into thinking that they are in fact in the real site
- Example: goggle.com vs google.com



Internationalized Domain Names (IDN)

Internationalized Domain Names (IDN) are domain names which use a wide range of Unicode characters used in different languages.

- Including letters or characters from non-Latin scripts (e.g., Arabic or Chinese characters) in domains
 - Japanese .jp domain registry services: [日本語.jp](#)
 - Starbucks Korea: [스타벅스코리아.com](#)

Why? To allow more web users to navigate in their preferred language

- Most domain names are registered in ASCII characters (A to Z, 0 to 9, and the hyphen "-")
- However, languages that require diacritics cannot be rendered in ASCII
- Also useful for brand localization

Internationalized Domain Names (IDN)

How it works? This is done by transcoding Unicode characters to **punycode**

- **Punycode** is a representation of Unicode with the limited ASCII character subset used for Internet host names
- Starting with the prefix "xn--" to signal the domain name is using punycode

Example of IDNs:

- 日本語.jp → <https://xn--wqv71a119e.jp>
- 스타벅스코리아.com → <http://xn--oy2b35ckwhba574atvuzkc.com>

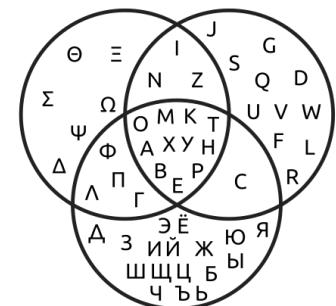
Try it yourself:

- 日本語.jp
- <https://xn--wqv71a119e.jp>

IDN homograph attack

IDN homograph attack exploits the fact that different characters from different writing systems look alike.

- Users may not notice subtle differences in characters from different writing systems (e.g., Latin, Cyrillic or Greek)
 - Often happening on IDNs that allow non-ASCII characters
 - Font family can also cause issues (e.g., m vs rn vs rri)



IDN homograph attack

Example: Can you tell which one is the phishing site?

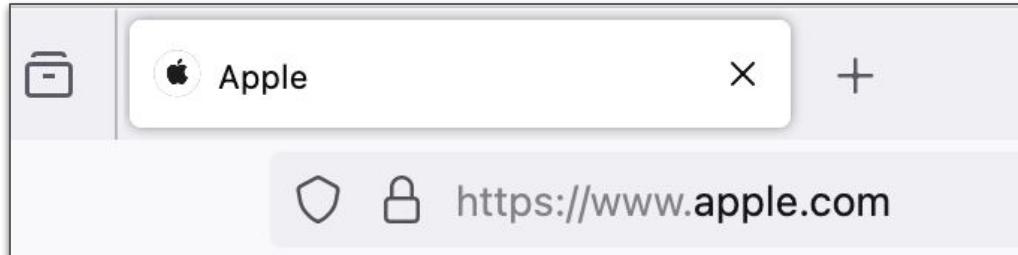
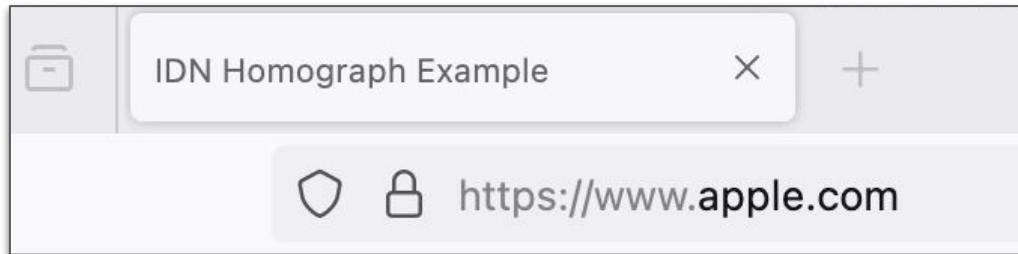
- <https://www.apple.com>
- <https://www.apple.com>

Try visiting <https://www.apple.com/> in Firefox vs Chrome

- First ‘apple.com’ uses Cyrillic characters rather the ASCII characters
 - E.g., Cyrillic ‘a’ (U+0430) vs ASCII “a” (U+0041)
- Two websites returns different hashes
 - Recall hash function: same input produces same output
 - First ‘apple.com’ produces 7a9f74
 - Second ‘apple.com’ produces 15fb0e
 - Try it yourself: [UTF-8 to SHA256](#)



IDN homograph attack



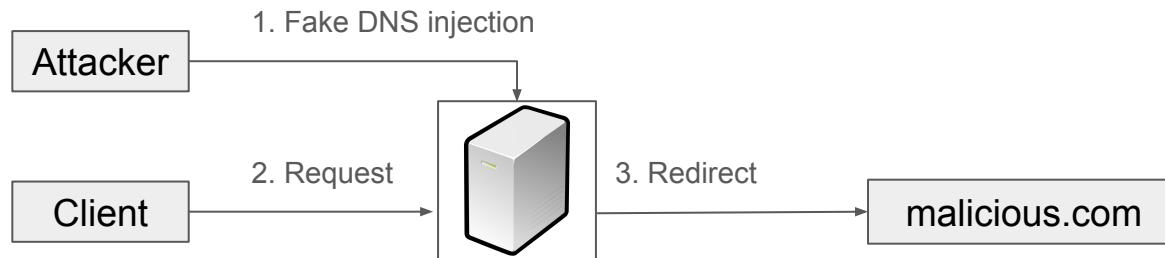
Pharming attack

Pharming attack redirects users to a fake website that mimics a real website.

- Manipulate the Domain Name System to redirect user's request without their knowledge

How it works? Attacker injects fake DNS entry on the DNS server

- Request to the DNS server is redirected to attacker's malicious website
- Attacker gets access to user credentials



Schedule for today

- Key concepts from last classes
- Phishing
 - Typosquatting
 - IDN Homograph attack
 - Pharming
- Denial-of-Service attack
 - UI attack
 - The Annoying Site “features” and prevention

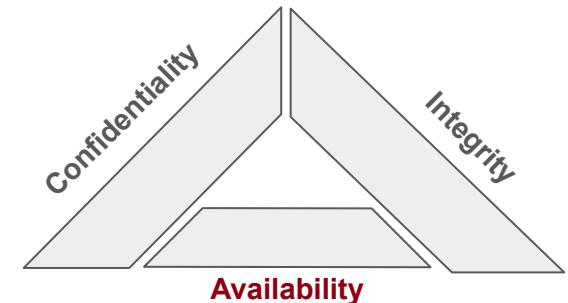
Denial-of-Service attack

Denial-of-Service attack, also known as DoS attack, is designed to render a service inaccessible to its users.

- Attack on the availability of the system

There are typically two types of DoS attack:

- Server-side DoS attack: Network or server attacks
 - Make the server or network resource unavailable by flooding it with superfluous requests to overload the system
- Client-side DoS attack: **UI attacks** (or clickjacking attacks)
 - Trick a victim into inadvertently clicking on an attacker-supplied input.



Note that DoS is different from Distributed Denial-of-Service (DDoS) attack

UI attacks

UI attacks (or clickjacking attacks) are a category of attacks that try to fool a victim into inadvertently clicking on an attacker-supplied input.

UI attacks can have different **goals**:

- Override browser defaults to disorient users on site
 - E.g., Advertisement pretending to be part of the website UI
- Scareware to intimidate the user into trapping them on an unwanted site
 - E.g., Pop-ups windows with “Warning! Virus has been detected!”
- Annoy the user, mostly harmless but annoying

UI attacks

Often, attacker can compromise a vulnerable website by:

- Including malicious advertisements
- Injecting malicious scripts into third-party widgets
- Injecting malicious scripts as user-generated content (i.e., Stored XSS)

UI attacks

Example of UI attacks: Combining UI attacks with **phishing** + **web scraping** + **XSS**

- Big idea: “steal” a click from the user, so that the user loads something malicious

Possible scenario:

- Scraping for contact information
- Phishing users into vulnerable websites with XSS and UI attacks
 - Malicious download button added through stored XSS or as a paid advertisement
 - Buttons redirect users to malicious websites
- Ask users to login and steal their credentials

Infinite alert loop

```
const messages = [  
  'Once upon a time...',  
  'There is a lecture...',  
  'About UI attack...',  
  'Where the instructor discusses about...',  
]  
  
while (true) {  
  messages.forEach(message => alert(message))  
}
```

- Block any user's input to the tab
- Force user to quite the browser

Intuition: Find a way to break the user out of infinite alert loops without needing to quit their browser.



Infinite alert loop

Different browsers have came up with different **defense mechanisms**:

- Chrome: multiprocess allowing close button on the tab from working
- Firefox: checkbox on popup, also multiprocess

But users still loses their session...

- Alternative solution available, but not for end-users
- Chrome - Source - Pause button available

Infinite Pop-up Incident

Infinite Pop-up Incident: In 2019, a 13-year-old Japanese girl was arrested for posting a infinite pop-up loop prank on a forum.

- Modern browser could close the popup
- However, majority of mobile browsers couldn't closed it

Lets-get-arrested project was launched three days later to protest against the incident:

How to get arrested

It's easy. Fork this project and branch it as gh-pages. It's all done. It would be more effective to share the url "<https://{{youraccount}}.github.io/lets-get-arrested>" on social media. When you share it in Twitter, use hash tag `#letsgetarrested4jscode`.

Not arrested?

You can surrender yourself to the police.

^_~ ババババ
(・ω・)=つミつ
(つミつ=つ
'/)
(ノノU
何回閉じても無駄ですよ～ww
m9 (*Д*) プギヤー！！
byソル (@0_Infinity_)

Allow dialogs from web.archive.org to take you to their tab

OK

Example of UI attacks

The Annoying Site is an example of harmless UI attack:

- <https://www.theannoyingsite.com> ([Github@ feross/TheAnnoyingSite.com](#))
- **Warning:** Avoid opening the link in the main browser, use an alternative one that you can force to quit

Some “features” on The Annoying Site:

- Log user out
- Embarrassing searches
- Tabnabbing
- Trigger a file download



API Level

API Level	Restrictions	Examples
Level 0	No restrictions	window.move() File download CSS
Level 1	User interaction required (e.g., click or keypress)	window.open() Copy text to clipboard
Level 2	User “engagement” required	Autoplay sound
Level 3	User permission required	Camera, microphone, USB

Log user out



```
window.onload = function() {  
  
doSites(document.getElementById("sitelist"), [  
    ["Apple", get("https://appleid.apple.com/account/signout")],  
    ["GitHub", get("https://github.com/logout")],  
    ["GMail", get("http://mail.google.com/mail/?logout")],  
];  
};
```

- Force users out of their session

What is happening behind the scenes?

Script from [SuperLogout](#)

Log user out

```
window.onload = function() {  
  
doSites(document.getElementById("sitelist"), [  
    ["Apple", get("https://appleid.apple.com/account/signout")],  
    ["GitHub", get("https://github.com/logout")],  
    ["GMail", get("http://mail.google.com/mail/?logout")],  
];  
};
```

Script from [SuperLogout](#)

- Force users out of their session

Behind the scene:

- User lands on the website
- Website sends a HTTP request from user's browser to popular sites
- User's browser helpfully attaches user session cookies
- **SOP** passes, **CSP** passes; User logged out of their own account without knowing it ...

Embarrassing searches



```
const searches = [  
  'where should i bury the body',  
  'why does my eye twitch',  
  'why is my poop green',  
  'why do i feel so empty',  
]  
  
// for each entry in search do:  
  
let searchIndex = 1  
  
window.location = 'https://www.bing.com/search?q=' +  
encodeURIComponent(searches[searchIndex]);  
  
searchIndex += 1
```



- Force users to search for something embarrassing

What is happening behind the scenes?

Embarrassing searches

```
const searches = [  
  'where should i bury the body',  
  'why does my eye twitch',  
  'why is my poop green',  
  'why do i feel so empty',  
]  
  
// for each entry in search do:  
  
window.location = 'https://www.bing.com/search?q=' +  
  encodeURIComponent(searches[searchIndex])  
};
```

- Force users to search for something embarrassing

Behind the scene:

- User lands on the website
- Website sends a HTTP request from user's browser to popular sites
- **SOP** passes, **CSP** passes; User searches for something embarrassing ...

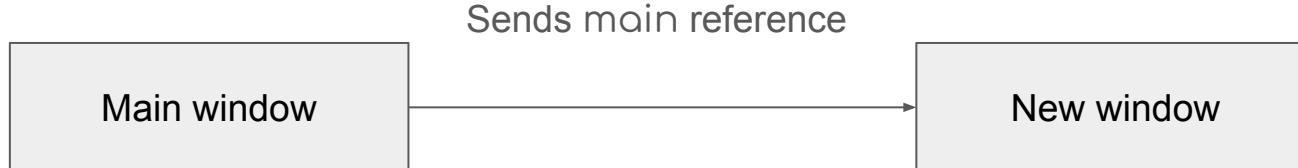
Tabnabbing

Tabnabbing is UI attack that targets the inactive tabs in victim's browser.

For example:

- When user clicks on a link on reddit.com with:
`External Website`
- malicious.com gets a reference to reddit.com window `window.opener`
- malicious.com redirects the user to a fake reddit on the main window

Window opener property returns a reference to the window that created the window



Tabnabbing prevention

To prevent tabnabbing, add rel='noopener' to all links with target='_blank'

- The opened site's window.opener will be null
- From 2021, all browsers treat target="_blank" as implying rel="noopener"
- Tabnabbing is rare today

Next class: Bitcoin



Lecture 29

Blocks and blockchain

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Blockchain technology
 - Why Bitcoin has value?
 - A form of distributed ledger technology
 - Blocks and blockchain
- Why maintain and create new blocks?
 - Block rewards (i.e., mining rewards)
 - Bitcoin Halving
- Who maintain and create new blocks?
 - Proof of Work (PoW)
 - Brief introduction into mining principles
- Next class: Mining principles

Bitcoin

Bitcoin is a cryptocurrency, a virtual currency designed to act as money

- Introduced by an anonymous developer or group of developers using the name “Satoshi Nakamoto”

Bitcoin outlines the concept of a **decentralized distributed ledger system**

- Electronic cash system
- No trusted third-party (e.g., central banks)
- Peer-to-peer network

History of Bitcoin

- On Oct. 31, 2008, “Satoshi Nakamoto” wrote an email to a cypherpunk mailing list discussing about an electronic cash system
- The concept of Bitcoin was shared through a paper titled “[Bitcoin: A Peer-to-Peer Electronic Cash System](#)”, later known as the Bitcoin white paper
- On Jan. 3, 2009, the first bitcoin was mined
- On Jan. 12, 2009, the world’s first Bitcoin transaction took place
 - Transaction of 10 Bitcoin (BTC) to a regular poster on the cypherpunk mailing list

[PDF] [**Bitcoin: A peer-to-peer electronic cash system**](#)

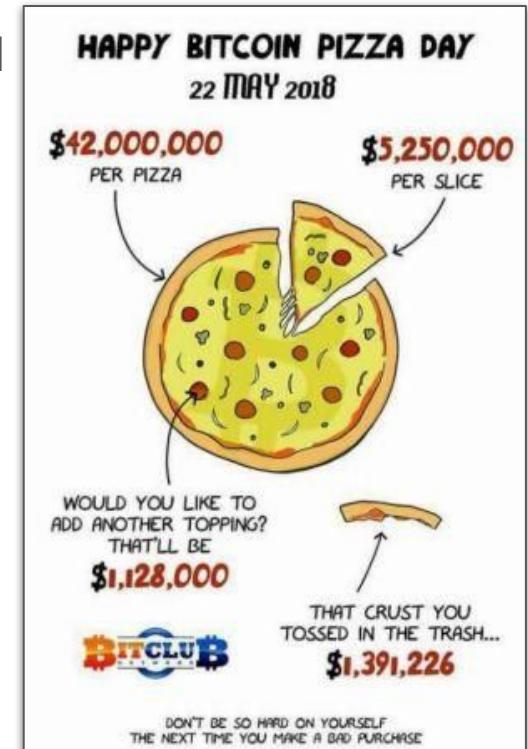
S Nakamoto - 2008 - assets.pubpub.org

... To implement a distributed timestamp server on a **peer-to-peer** basis, we will need to use a proof-of-work **system** similar to Adam Back's Hashcash [6], rather than newspaper or Usenet ...

[☆ Save](#) [⤒ Cite](#) [Cited by 32025](#) [Related articles](#) [All 1369 versions](#) [⤓](#)

Iconic Bitcoin Pizza Day

- On May 22, 2010, programmer Laszlo Hanyecz offered 10,000 Bitcoin for two large pizzas at the BitcoinTalk online forum
- Jeremy Sturdivant took up Hanyecz's offer and delivered the meal in exchange for the Bitcoin
- This marked the first real-world transaction for the currency



[Image from cryptonews](#)

The Big Bang Theory Season 11 Episode 9

The Bitcoin Entanglement



Locked USB Drive

Stefan Thomas, a German-born programmer living in San Francisco, has two guesses left (already tried eight incorrect guesses) to figure out a password that is worth, as of this week, about \$657 millions.

Thomas said that his 7,002 bitcoins were left over from a payment he received for making a video titled “What is Bitcoin?” that published on YouTube in early 2011, when a bitcoin was worth less than a dollar.

- [WIRED updates in 2023](#)
- [BCC on James Howells](#)
 - 7,500 BTC in the landfill

What is Bitcoin? (v1)



WeUseCoins
25.3K subscribers

Subscribe

10,207,650 views Mar 22, 2011
Learn about Bitcoin with the most watched Bitcoin video.

Schedule for today

- Blockchain technology
 - Why Bitcoin has value?
 - A form of distributed ledger technology
 - Blocks and blockchain
- Why maintain and create new blocks?
 - Block rewards (i.e., mining rewards)
 - Bitcoin Halving
- Who maintain and create new blocks?
 - Proof of Work (PoW)
 - Brief introduction into mining principles
- Next class: Mining principles

Why Bitcoin has value?

- Decentralized = Peer-to-peer network
 - Transactions between private users are not regulated
 - Central banks offer credibility (regulated by the government)
 - Idea: Save effort and money from the third-party authority
- Flat transaction fee = Congestion of the network and size of the transaction
 - Fee for larger transactions (\$1,000,000) = Fee for smaller transactions (\$100)
 - Central banks with percentage rate ([Why fixed costs matter for Bitcoin](#) by Bank of Canada)
 - Idea: Depending on how many people are making transactions
- Scarcity = Limited to a total of 21 million Bitcoin
 - Over 19.5 million Bitcoins currently in circulation, leaving 1.5 million yet to be mined
 - Central banks regulate the money supply based inflation and economic growth = unlimited

Why Bitcoin has value?

- Decentralized → **Blockchain technology**
 - Transactions between private users are not regulated
 - Central banks offer credibility (regulated by the government)
 - Idea: Save effort and money from the third-party authority
- Flat transaction fee → **Block rewards**
 - Fee for larger transactions (\$1,000,000) = Fee for smaller transactions (\$100)
 - Central banks with percentage rate ([Why fixed costs matter for Bitcoin](#) by Bank of Canada)
- Scarcity → **Bitcoin halving**
 - Over 19.5 million Bitcoins currently in circulation, leaving 1.5 million yet to be mined
 - Central banks regulate the money supply based inflation and economic growth = unlimited

Message from Government of Canada

“Crypto assets are very risky.

*Unlike the Canadian dollar, crypto assets are not legal tender in Canada. A government or **central bank doesn’t issue or oversee them.***

*Crypto assets are also quickly **evolving, unstable and complex**. You should learn more about crypto assets and their risks before investing or using them. You may also want to consult a financial advisor.”*

- [Risks of using crypto assets](#) from Government of Canada
- [Avoid crypto investment fraud](#) from Competition Bureau Canada
- [Warnings about crypto assets](#) from Canadian Securities Administrators

Schedule for today

- Blockchain technology
 - Why Bitcoin has value?
 - A form of **distributed ledger technology**
 - Blocks and blockchain
- Why maintain and create new blocks?
 - Block rewards (i.e., mining rewards)
 - Bitcoin Halving
- Who maintain and create new blocks?
 - Proof of Work (PoW)
 - Brief introduction into mining principles
- Next class: Mining principles

Distributed ledger technology

Distributed ledger technology (DLT) is a system for recording digital transactions without the need for a centralized authority.

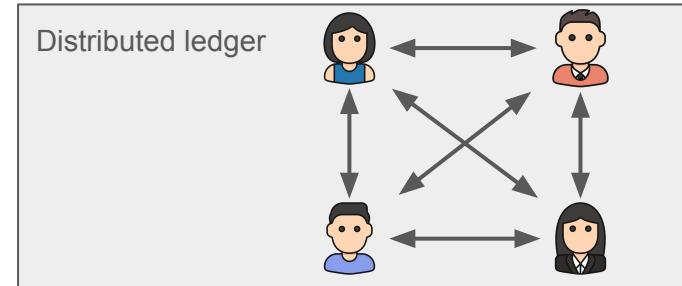
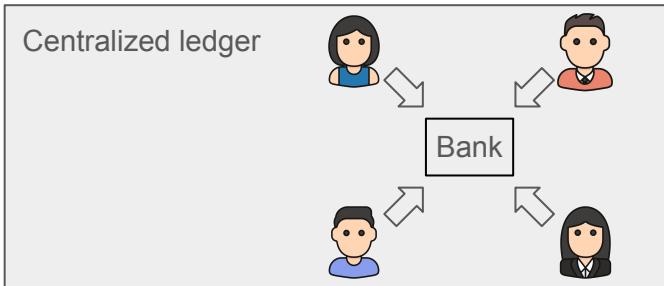
- Database spread across several nodes or computing devices
- Each node replicates and saves an identical copy of a public ledger
- Each participant node of the network updates itself independently

Distributed ledger technology

Distributed ledger technology (DLT) is a system for recording digital transactions without the need for a centralized authority.

- Database spread across several nodes or computing devices
- Each node replicates and saves an identical copy of the ledger
- Each participant node of the network updates itself independently

The distributed ledger technologies focuses on reducing the cost of trust.



Blockchain technology

Blockchain technology is a way to implement a distributed ledger system which employs a chain of blocks (i.e., **blockchain**) to store transactions.

- **Blockchain** stores transactions in **blocks** that are linked together in a chain
- As the chain provides a chronological consistency, ledgers are immutable

As a form of peer-to-peer network:

- Node requests a transaction
- Transaction is validated and recorded by other nodes in the network
- When the record reaches ~4,000 transactions, they forms a **block**
- Block is added to the existing chain of blocks, also known as a **blockchain**

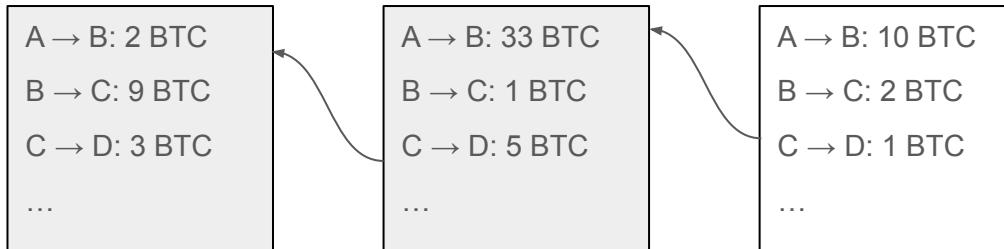
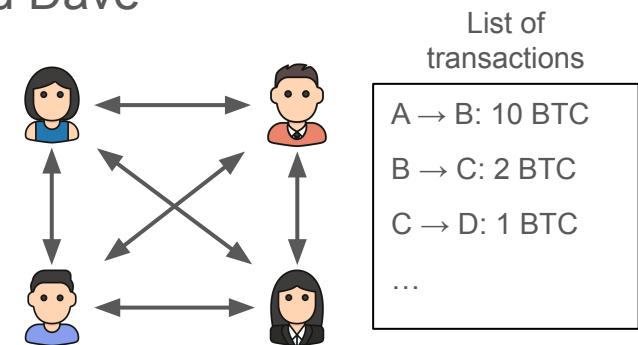
Peer-to-peer network

Assume the network is between Alice, Bob, Carol and Dave

- Alice pays 10 BTC to Bob
- Bob pays 2 BTC to Charlie
- Charlie pays 1 BTC to Dave ...

Once the list reach ~4,000 transactions

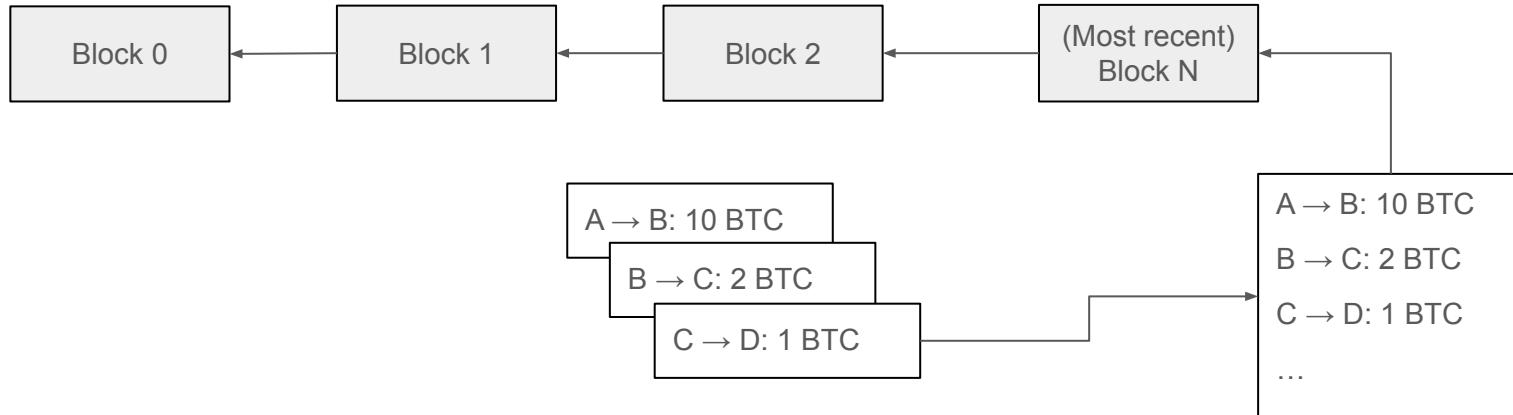
- List of transactions is packaged into a **block**
- The block is connected to the chain of all prior transactions (i.e., **blockchain**)



A basic protocol

Design: Decentralized network without a centralized authority

- Everyone keeps a copy of the blockchain on their own public ledger
- Every time new transactions are packaged into blocks, the public ledger is updated



Challenges in a distributed ledger system

There are two main challenges in the basic protocol:

- **Why maintain and create new blocks?**
 - Fact: Everyone uses the system to make transitions
 - What is the incentive of recording transactions for other people?
 - E.g., roommate agreement, Alice and Bob only make transactions with each other
 - Why should they help create the sticky note (with other roommates' transactions)?
- **Who maintain and create new blocks?**
 - Fact: Network delays exist, everyone has a ledger with different transactions order
 - Whose ledger do we rely on?
 - E.g., roommate agreement, every roommate has their own ledger
 - Whose notebook do we use to update the sticky note?

Schedule for today

- Blockchain technology
 - Why Bitcoin has value?
 - A form of **distributed ledger technology**
 - Blocks and blockchain
- **Why maintain and create new blocks?**
 - Block rewards (i.e., mining rewards)
 - Bitcoin Halving
- Who maintain and create new blocks?
 - Proof of Work (PoW)
 - Brief introduction into mining principles
- Next class: Mining principles

Why maintain and create new blocks?

Creating rewards for block creators (i.e., Bitcoin miners):

- Reward 1: **Transaction fee**
 - Each Bitcoin transaction includes a fixed transaction fee for the block creators
 - From senders to block creators
 - Same transaction fee that central banks charge
- Reward 2: **Block reward** (i.e., mining reward)
 - Each creation of new blocks is rewarded with a block reward
 - From the system to block creators
 - Only one block creator per block

Block rewards

The design of block rewards is called **Bitcoin halving**

- Initial block reward: 50 BTC (in 2009)
- Bitcoin halving: Block rewards decrease by half every four years
- Next Bitcoin halving: April 2024
 - Block reward falls from 6.25 to 3.125 BTC

Block rewards is the only way for new bitcoins to enter circulation, produced by block creators, or commonly known as Bitcoin “miners”

Bitcoin halving

The design of block rewards is called **Bitcoin halving**

- Initial block reward: 50 BTC (in 2009)
- Block rewards decrease by half every four years
 - Block time: 10 minutes between every new block (more on this later ...)

Calculating the total number of BTC:

- Number of blocks over 4 years = 6 blocks per hour x 24 hours per day x 365 days per year x 4 years
= 210,240
- 2009 to 2012 = 50 BTC per block
- 2013 to 2016 = 25 BTC per block
- 2017 to 2020 = 12.5 BTC per block ...
- Total number of BTC = $210,240 (50 + 25 + 12.5 + 6.25 + \dots) \approx 21,000,000$ BTC

This ensures the
scarcity of Bitcoin

Challenges in a distributed ledger system

There are two main challenges in the basic protocol:

- Why maintain and create new blocks?
 - Answer: Rewards available (\$\$\$) for the block creators
 - Note that the rewards reduce by half every four years

Why Bitcoin has value?

- Decentralized → **Blockchain technology**
 - Transactions between private users are not regulated
 - Central banks offer credibility (regulated by the government)
- Flat transaction fee → **Block rewards**
 - Fee for larger transactions (\$1,000,000) = Fee for smaller transactions (\$100)
 - Central banks with percentage rate ([Why fixed costs matter for Bitcoin](#) by Bank of Canada)
- Scarcity → **Bitcoin halving**
 - Over 19.5 million Bitcoins currently in circulation, leaving 1.5 million yet to be mined
 - Central banks regulate the money supply based inflation and economic growth = unlimited

Challenges in a distributed ledger system

There are two main challenges in the basic protocol:

- Why maintain and create new blocks?
 - Answer: Rewards available (\$\$\$) for the block creators
 - Note that the rewards reduce by half every four years
- Who maintain and create new blocks?
 - Fact: Network delays exist, everyone has a ledger with different transactions order
 - Whose ledger do we rely on?
 - In addition, everyone wants to be the block creators
 - Who gets to create new blocks?

Schedule for today

- Blockchain technology
 - Why Bitcoin has value?
 - A form of **distributed ledger technology**
 - Blocks and blockchain
- Why maintain and create new blocks?
 - Block rewards (i.e., mining rewards)
 - Bitcoin Halving
- Who maintain and create new blocks?
 - Proof of Work (PoW)
 - Brief introduction into mining principles
- Next class: Mining principles

Who maintain and create new blocks?

Proof of Work (PoW) is a consensus algorithm which is used to verify transactions and create new blocks.

- In PoW, participants (miners) compete to solve complex mathematical puzzles
 - Puzzles are difficult to solve but easy to verify the solution
- First one to find a valid solution gets the rights to create new blocks (and its block creation rewards)

This process of “mining” for the solution is referred to as **Bitcoin mining**.



Review on hash functions

Proof-of-Work uses hash functions to associate the amount of work done with a block of transactions.

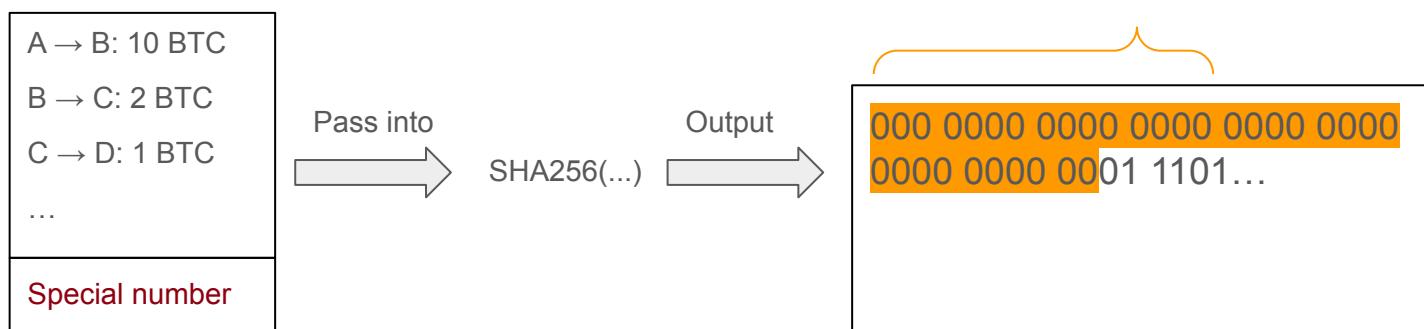
To recall on hash functions:

- Hash functions are irreversible
 - Analogy to jigsaw puzzles: cutting the paper into one million pieces of jigsaw puzzle and shuffling it
- Easy to apply the hash function, hard to find the original data
 - Analogy to birthday problem: hard to guess the person based on a birthday
- SHA256 produces a hash of 64 hexadecimal characters / 256 bits
 - $\text{SHA256}(?) = 110\ 1000\ 1110\ 0110\ 0101\ 0110\ \dots$
 - Brute force is the only solution

Mining principle

Proof-of-Work is about finding a **special number**:

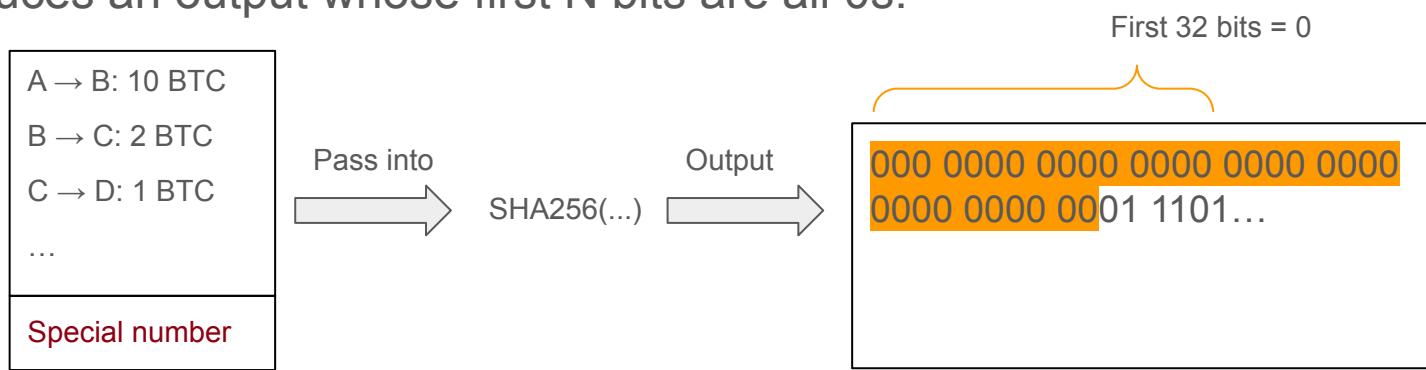
- Combined with the other information from the block and applied SHA256 produces an output whose first N bits are all 0s.



Mining principle

Proof-of-Work is about finding a **special number**:

- Combined with the other information from the block and applied SHA256 produces an output whose first N bits are all 0s.



However, this is **very difficult!**

- 32 fixed bits, each bit presents the possibility between 0 and 1.
- Probability: $2^{32} = (4 \text{ billions}) \rightarrow \text{Random guess} = 1 \text{ out of } 4 \text{ billions}$

Announcements

- No class on Friday
- Reminder to contribute to projects
 - Peer evaluation will be available
 - Team participation required for the demo

Lecture 30

Mining Principles

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
 - A basic protocol
 - Challenges in a distributed ledger system
- Who maintain and create new blocks?
 - Proof-of-Work (PoW)
- Mining principles
 - Role of miners
 - Difficulty adjustment + The Longest Chain Rule
 - The 51% Attack

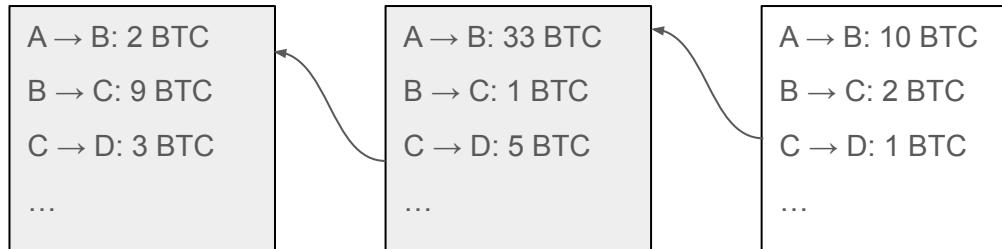
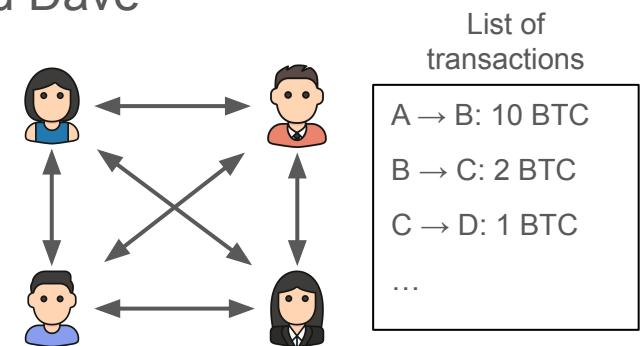
Peer-to-peer network

Assume the network is between Alice, Bob, Carol and Dave

- Alice pays 10 BTC to Bob
- Bob pays 2 BTC to Charlie
- Charlie pays 1 BTC to Dave ...

Once the list reach ~4,000 transactions

- List of transactions is packaged into a **block**
- The block is connected to the chain of all prior transactions (i.e., **blockchain**)



Challenges in a distributed ledger system

There are two main challenges in the basic protocol:

- **Why maintain and create new blocks?**
 - Fact: Everyone uses the system to make transitions
 - What is the incentive of recording transactions for other people?
 - E.g., roommate agreement, Alice and Bob only make transactions with each other
 - Why should they help create the sticky note (with other roommates' transactions)?
- **Who maintain and create new blocks?**
 - Fact: Network delays exist, everyone has a ledger with different transactions order
 - Whose ledger do we rely on?
 - E.g., roommate agreement, every roommate has their own ledger
 - Whose notebook do we use to update the sticky note?

Who maintain and create new blocks?

Proof-of-Work (PoW) is a consensus algorithm which is used to verify transactions and create new blocks.

- In PoW, participants (miners) compete to solve complex mathematical puzzles
 - Puzzles are difficult to solve but easy to verify the solution
- First one to find a valid solution gets the rights to create new blocks (and its block creation rewards)

This process of “mining” for the solution is referred to as **Bitcoin mining**.



Schedule for today

- Key concepts from last class
 - A basic protocol
 - Challenges in a distributed ledger system
- Who maintain and create new blocks?
 - Proof of Work (PoW)
- **Mining principles**
 - Role of miners
 - Difficulty adjustment + The Longest Chain Rule
 - The 51% Attack

Proof-of-Work as a consensus problem

Byzantine General Problem is a consensus problem:

- Problem: Message may be sent by traitors
- Consistency: All royal lieutenants must execute the same order
- Validity: All royal lieutenants must obey the order of commanding general

Proof-of-Work also needs to deal with a consensus problem:

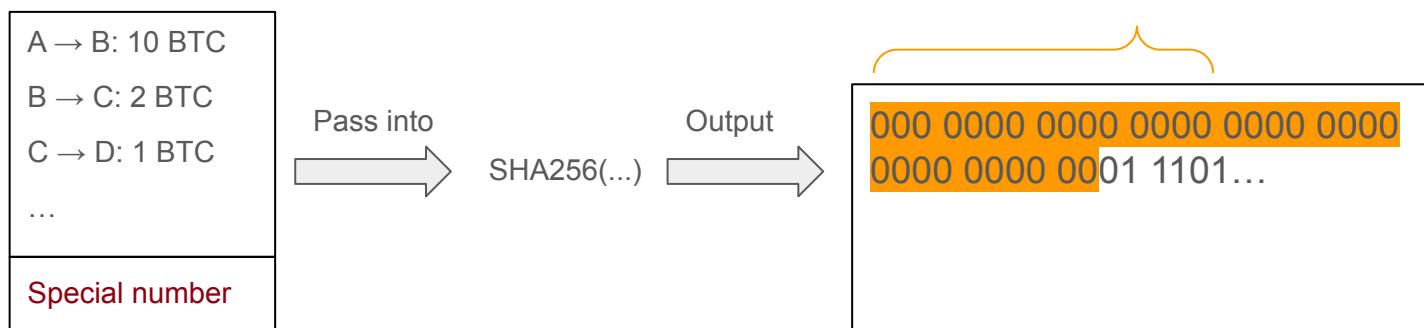
- Problem: Block with fake transactions may be sent by malicious users
- Consistency: All honest nodes record the same blockchain
- Validity: All honest nodes record the blockchain coming from a honest node

Consensus rule: Trust the node that has done the most of work

Mining principle

Proof-of-Work is about finding a **special number**:

- Combined with the other information from the block and applied SHA256 produces an output whose first N bits are all 0s.



However, this is **very difficult!**

Review on hash functions

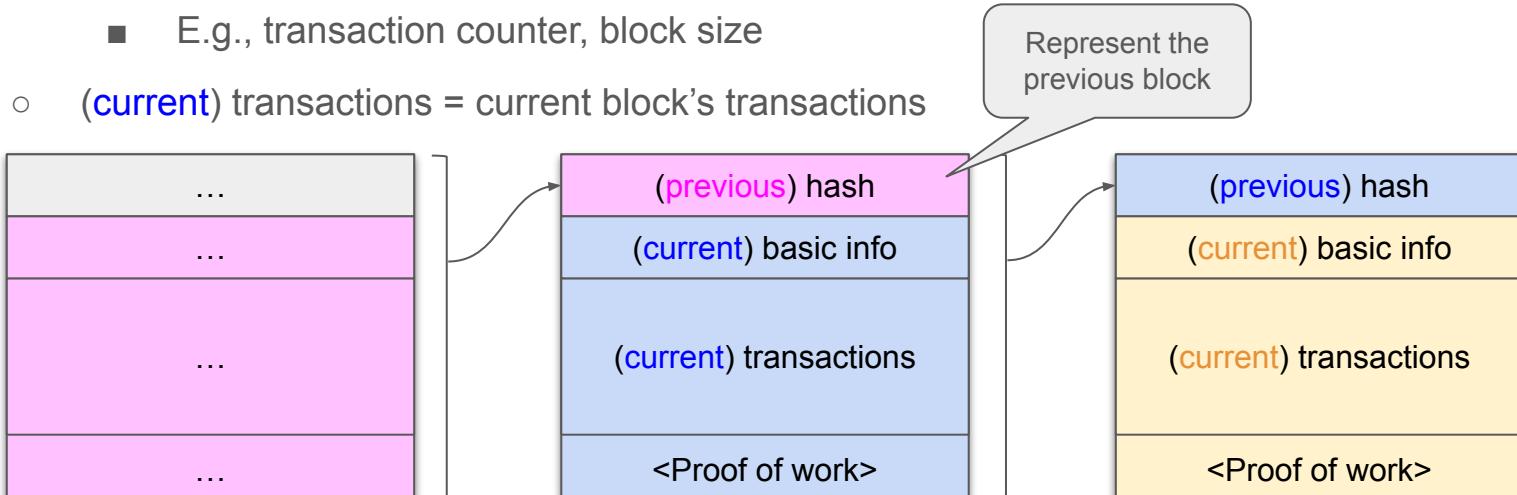
Proof-of-Work uses hash functions to associate the amount of work done with a block of transactions.

- Hash functions are irreversible
 - Analogy to jigsaw puzzles: cutting the paper into one million pieces of jigsaw puzzle and shuffling it
- Easy to apply the hash function, hard to find the original data
 - Analogy to birthday problem: hard to guess the person based on a birthday
- SHA256 produces a hash of 64 hexadecimal characters / 256 bits
 - $\text{SHA256}(?) = 110\ 1000\ 1110\ 0110\ 0101\ 0110\ \dots$
 - Brute force is the only solution

Finding the special number

Step 1: Package current block's data into a string

- `string = (previous) hash + (current) basic info + (current) transactions`
 - **(previous)** hash: previous block's hash value
 - **(current)** basic info: current block's basic information
 - E.g., transaction counter, block size
 - **(current)** transactions = current block's transactions



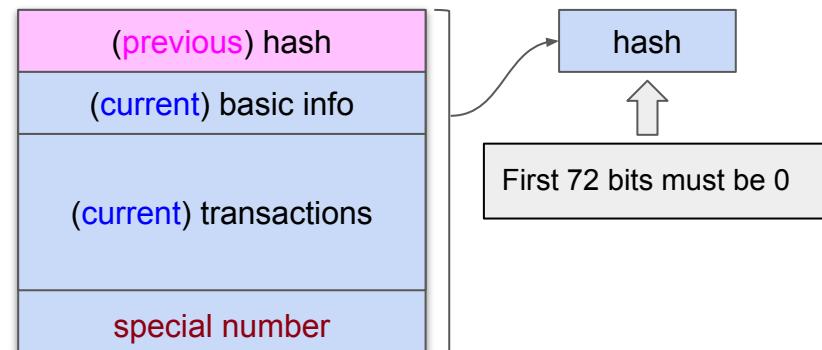
Finding the special number

Step 1: Package current block's data into a string

- **string** = (previous) hash + (current) basic info + (current) transactions

Step 2: Find a **special number**

- Add the special number to the string
- Calculate SHA-256(**string** + **special number**) = 256-bits number
- Requirement: First 72 bits must be all 0s



Finding the special number

Step 1: Package current block's data into a string

- **string** = (previous) hash + (current) basic info + (current) transactions

Step 2: Find a **special number**

- Requirement: First 72 bits must be all 0s

These two steps are very difficult to complete, but very easy to verify:

- Compute SHA-256(string + special number)
- Check if the has gives 72 leading 0s

This proof of work is **tied to the list of transactions**

- Any change to the transactions also changes the hash

Mining difficulty

The difficulty of finding the special number is very high:

- Probability of first digit as a 0 is = $\frac{1}{2}$
- Probability of first two digits as 0s is = $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$
- ...
- Probability of first 72 digits as 0s is = $1/2^{72} = 1 \text{ out of } (2^{36})^2$
 $\approx 1 \text{ out of } (69 \text{ billions})^2$

SHA-256(string + special number) { 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 1010 1110 ...

Schedule for today

- Key concepts from last class
 - A basic protocol
 - Challenges in a distributed ledger system
- Who maintain and create new blocks?
 - Proof-of-Work (PoW)
- Mining principles
 - **Role of miners**
 - Difficulty adjustment + The Longest Chain Rule
 - The 51% Attack

Bigger picture on mining

From the **Bitcoin miner's** perspective:

- Mining is similar to a lottery system
- Everyone try to find the special number first
- Once found, broadcast the blockchain (i.e., ledger) to Bitcoin users

From the **Bitcoin user's** perspective:

- No need to listen or record other people's transactions
- Listen for block broadcasts from miners
- Compute the hash value to verify the “work done”
- Update their own copy of the blockchain (i.e., ledger)

Example on mining



Bob as a Bitcoin user

Step 1) As a Bitcoin user, Bob makes transactions to Alice

Step 4) Bob verifies the block and updates his copy of the blockchain



Carol as a lucky miner

Step 2) Carol captures Bob's transactions

Step 3) Carol finds the special number and broadcasts the blockchain

Step 5) Carol receives a small transaction fee

Schedule for today

- Key concepts from last class
 - A basic protocol
 - Challenges in a distributed ledger system
- Who maintain and create new blocks?
 - Proof-of-Work (PoW)
- Mining principles
 - Role of miners
 - **Difficulty adjustment + The Longest Chain Rule**
 - The 51% Attack

Difficulty adjustment

With more participants and more computing power, the difficulty of the hash problem increases accordingly.

- Bitcoin automatically adjusts the difficulty after every 2,016 new blocks (in other words, every two weeks)
- New difficulty based on the number of participants in the Mining network and their combined computational power

Week 1

Requirement: First **72** bits must all be 0s



Week 3

Requirement: First **73** bits must all be 0s



Example of difficulty adjustment

Suppose there are 10,000 mining nodes on the network

- Assuming a processing time of 1.4×10^{13} checks/sec per mining node
- In 10 minutes, total computational power = 8×10^{19} checks
 - 1.4×10^{13} checks/sec \times 10,000 nodes \times 600 seconds = 8×10^{19} checks
- Given $n = 66$, 1 out of (2^{66}) = 7×10^{19} hash checks
- Therefore, the difficulty is adjusted to $n = 66$

New requirement: First **66**
bits must be all 0s

Difficulty adjustment

The actual hash difficulty is not about the leading zeros.

- It is about matching a target hash that is updated by the network every two weeks
- To ensure the block time maintains at a constant 10 minutes regardless of the network's computational power

Such adjustment ensures the network's **security** and **stability** by regulating the rate at which new blocks are added to the blockchain:

- When more miners join the network, the computational power increases and the difficulty level is adjusted to keep the block generation time constant
- If many miners leave the network, lowering the hash rate, the difficulty decreases

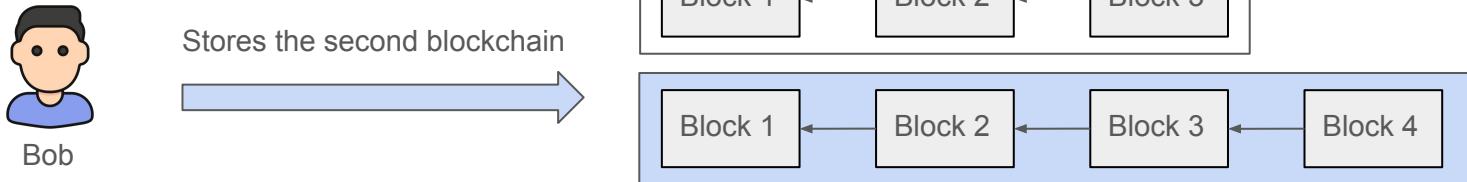
The Longest Chain Rule

The Longest Chain Rule allows every node on the network to agree on the same blockchain (e.g., same transaction history).

- Solving a consensus problem
- Protecting the immutability of the blockchain

Example of application:

- As a Bitcoin user, Bob receives two conflicting blockchains from miners
- Bob must always use the longest blockchain (i.e., with the most work)



The Longest Chain Rule

The Longest Chain Rule allows every node on the network to agree on the same blockchain (e.g., same transaction history).

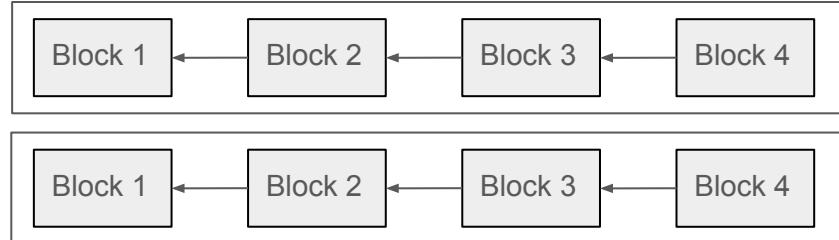
- Solving a consensus problem
- Protecting the immutability of the blockchain

Example of application:

- For blockchains with the same length, Bob waits for the next block that makes one of blockchain longer



Waits for the next update



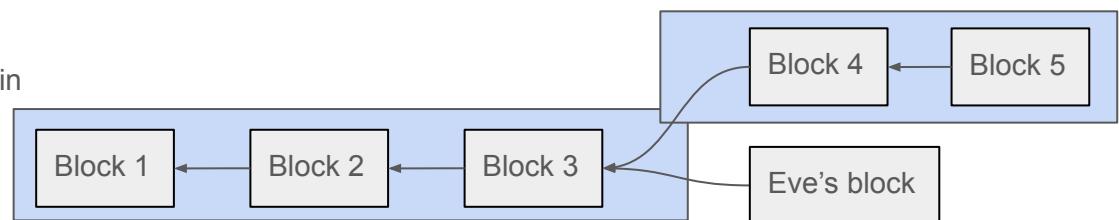
Why Proof-of-Work works?

Suppose Eve tries to send a block with fraudulent transactions:

- Eve first needs to find the special number based on the fraudulent transactions before everyone else, and broadcasts the blockchain
- Bob verifies the blockchain and copies it over
- **However**, Bob continues to listen to the broadcast
 - Any longer blockchain will replace the current one
 - For Bob to keep Eve's blockchain, Eve needs to keep extending the blockchain



Holds and waits for longer blockchain



Schedule for today

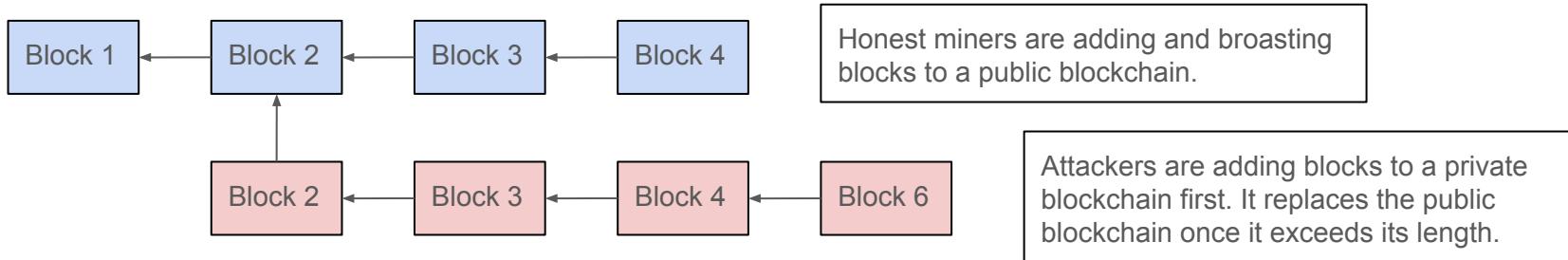
- Key concepts from last class
 - A basic protocol
 - Challenges in a distributed ledger system
- Who maintain and create new blocks?
 - Proof-of-Work (PoW)
- Mining principles
 - Role of miners
 - Difficulty adjustment + The Longest Chain Rule
 - **The 51% Attack**

The 51% Attack

Now assuming Eve owns 51% of the total computational power on the network

- Theoretically, Eve has the power to alter the blockchain
- By always creating the longest blockchain
- This assumption is known as **The 51% Attack**

A **51% Attack** is an attack performed by a group of miners who control more than 50% of the network's **mining power**.



The 51% Attack

How it works? Attackers with majority network control the blockchain

- Interrupt the recording of new blocks
- Rewrite parts of the blockchain and reverse their own transactions

However, in real-life settings

- Only smaller networks can be targets for 51% attacks

While possible, this is **incredibly costly** for the attacker:

- Great amounts of computing power (cost of electricity)
- Honest miners will stop mining (no rewards)

Recap on mining principle

Mining is about creating a new block and verifying the transactions:

- First miner to find the special number gets to create the new block
 - Each block is represented by **SHA-256(string + special number)**
- Transactions are considered verified once the miner solves the hash problem
 - **Proof-of-Work**
- Difficulty of the hash problem is based on the total computational power in the network
 - **Difficulty adjustment**

What happens when a malicious node (pretending to be a Bitcoin user) broadcasts a fake transaction?

Next class: Related concepts to Bitcoin

- Next class: Wednesday, April 3
 - Happy Good Friday and Easter Monday
- Half lecture on Bitcoin
 - Proof-of-Work as a consensus problem
 - Byzantine Fault-Tolerance
 - Transactions recordings as an integrity problem
 - Digital signatures
- Half lecture on Automated Testing
 - Overview of learning objectives + Demo

Lecture 31

Digital Signature & Double Spending Problem

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Key concepts from last class
 - Proof-of-Work (PoW)
 - Role of miners
 - Difficulty adjustment + The Longest Chain Rule
- Transactions and digital signatures
- Consensus for recording transactions
 - Balance checking
 - The Double Spending Problem
- Selenium brief introduction & IDE demo

Difficulty adjustment

With more participants and more computing power, the difficulty of the hash problem increases accordingly.

- Bitcoin automatically adjusts the difficulty after every 2,016 new blocks (in other words, every two weeks)
- New difficulty based on the number of participants in the mining network and their combined computational power

Week 1

Requirement: First **72** bits must all be 0s



Week 3

Requirement: First **73** bits must all be 0s



The Longest Chain Rule

The Longest Chain Rule allows every node on the network to agree on the same blockchain (e.g., same transaction history).

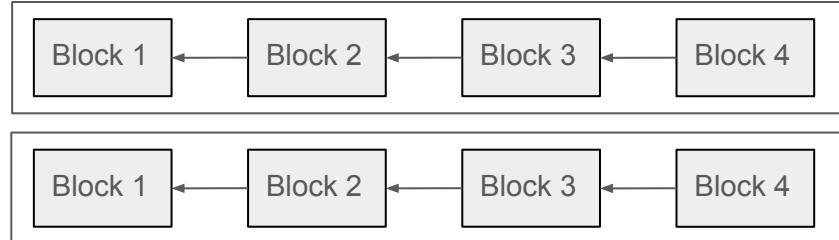
- Solving a consensus problem
- Protecting the immutability of the blockchain

Example of application:

- For blockchains with the same length, Bob waits for the next block that makes one of blockchain longer



Waits for the next update



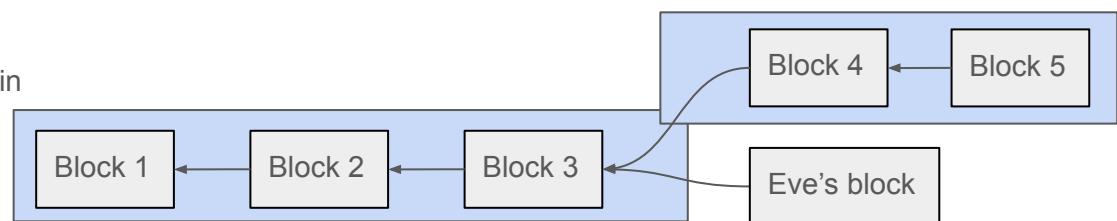
Why Proof-of-Work works?

Suppose Eve tries to send a block with fraudulent transactions:

- Eve first needs to find the special number based on the fraudulent transactions before everyone else, and broadcasts the blockchain
- Bob verifies the blockchain and copies it over
- **However**, Bob continues to listen to the broadcast
 - Any longer blockchain will replace the current one
 - For Bob to keep Eve's blockchain, Eve needs to keep extending the blockchain



Holds and waits for longer blockchain



Recap on mining principle

Mining is about creating a new block and verifying the transactions:

- First miner to find the special number gets to create the new block
 - Each block is represented by **SHA-256(string + special number)**
- Transactions are considered verified once the miner solves the hash problem
 - **Proof-of-Work**
- Difficulty of the hash problem is based on the total computational power in the network
 - **Difficulty adjustment**

What happens when a malicious node (pretending to be a Bitcoin user) broadcasts a fake transaction?

Schedule for today

- Key concepts from last class
 - Proof-of-Work (PoW)
 - Role of miners
 - Difficulty adjustment + The Longest Chain Rule
- **Transactions and digital signatures**
- Consensus for recording transactions
 - Balance checking
 - The Double Spending Problem
- Selenium brief introduction & IDE demo

Digital signature in blockchains

Digital signatures are used in blockchains to verify the authenticity of transactions.

- **Objective:** Every node must prove that they are authorized to spend the funds (i.e., having enough balance)
- At the same time, they must prevent other nodes from spending their funds

In Bitcoin, Bitcoin miners verify the authenticity of transactions:

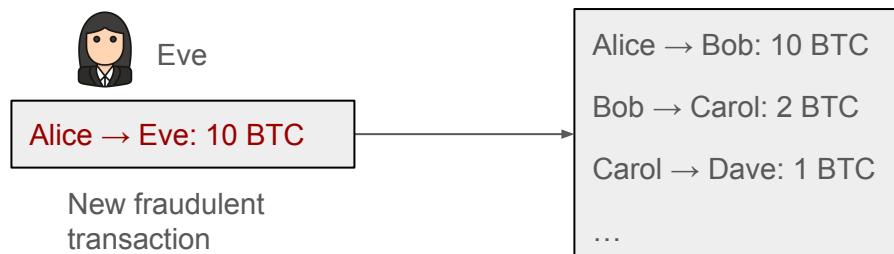
- **Strategy:** Verify the digital signature
- Check all other miners' work to agree on a correct state

The authenticity is based on three properties: **Ownership + Integrity + Association**

Fraudulent transaction

What happens when a malicious node broadcasts a fake transaction?

- In a decentralized ledger system, anyone can add a new transaction
 - E.g., Eve, a malicious user, can broadcast a new transaction: Alice → Eve: 10 BTC
 - What prevents Eve writing a new transaction without Alice's approval?



Each transaction is **approved by a digital signature**

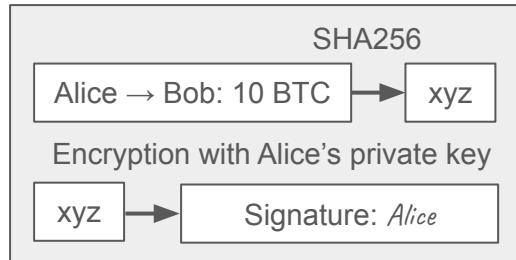
Digital signature

Each transaction is signed by the **owner (or owners)** of the source funds

Different transactions generate unique signatures:

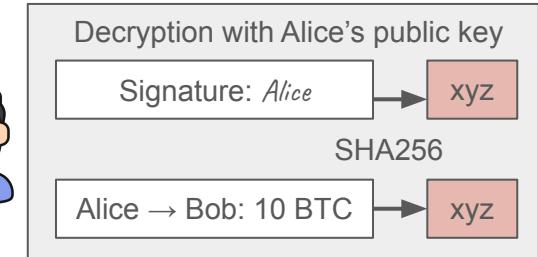
- Signed using a secret/private key (sk)
 - $\text{Sign}(\text{transaction}, \text{sk}) \rightarrow \text{signature}$
- Verified using a public key (pk)
 - $\text{Verify}(\text{transaction}, \text{pk}, \text{signature}) \rightarrow \text{accept} | \text{reject}$

$\text{Sign}(\text{transaction}, \text{sk})$



New transaction
Alice → Bob: 10 BTC
Signature: *Alice*

$\text{Verify}(\text{transaction}, \text{pk}, \text{signature})$



Digital signature in blockchains

Digital signature verifies the following properties:

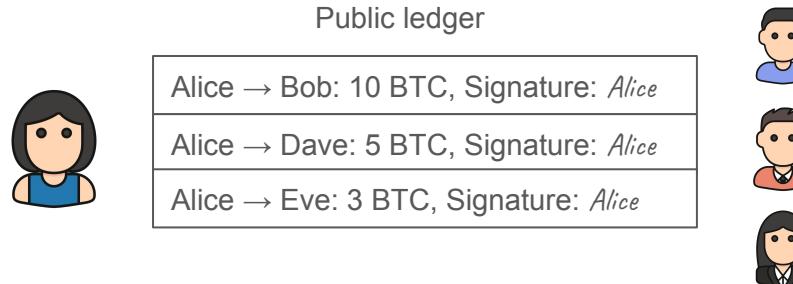
- **Ownership:** Digital signature belongs to Alice
 - Public and private keys for encryption and decryption
- **Integrity:** Transaction has not been modified
 - $\text{Decryption}(\text{Signature}) = \text{SHA256}(\text{Transaction})$

Potential threat to the current protocol



Suppose the following protocol:

- For each transaction, Alice adds her signature
- Alice broadcasts each transaction to everyone else
- What can go wrong?
 - Assume that the network is secure (no transaction interception)
 - Hint: Public ledger (anyone can add new transaction) + Session fixation attack



Another fraudulent transaction

Problem: Malicious users may broadcast fraudulent transactions

- By replicating a valid transaction + signature
- Analogous to Session Fixation attack
 - Malicious user can copy over the signature + session ID

Public ledger
Alice → Eve: 3 BTC, Signature: <i>Alice</i>
Alice → Eve: 3 BTC, Signature: <i>Alice</i>
Alice → Eve: 3 BTC, Signature: <i>Alice</i>

Solution: Having each transaction generate a unique signature

- By adding transaction IDs as part of the transaction (generate a different hash value)

Public ledger
1. Alice → Eve: 3 BTC, Signature: <i>Alice1</i>
2. Alice → Eve: 3 BTC, Signature: <i>Alice2</i>
3. Alice → Eve: 3 BTC, Signature: <i>Alice3</i>

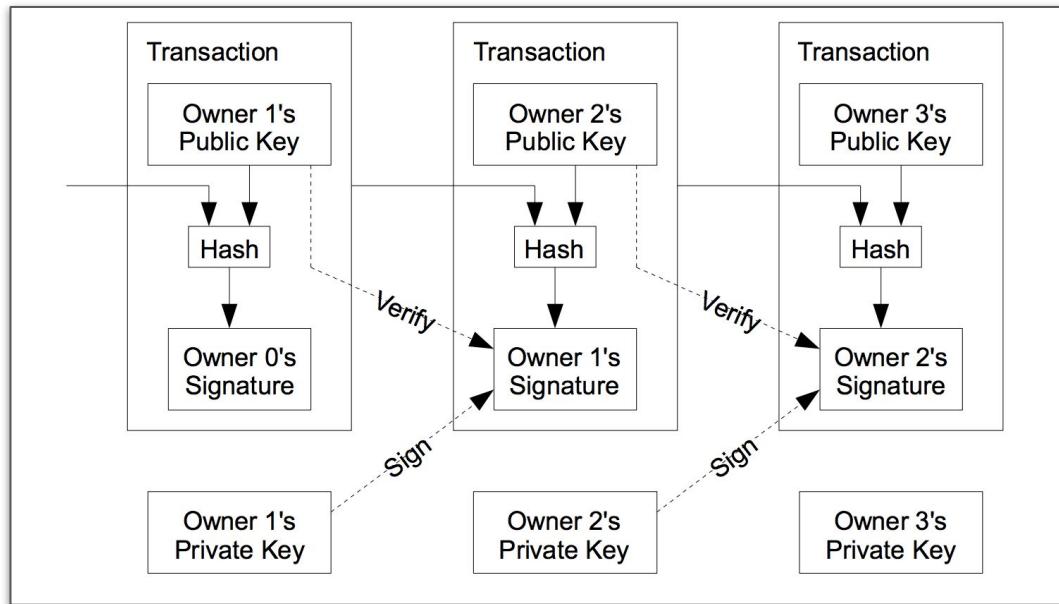
Digital signature in blockchains

Digital signature verifies the following properties:

- Ownership: Digital signature belongs to Alice
 - Public and private keys for encryption and decryption
- Integrity: Transaction has not been modified
 - $\text{Decryption}(\text{Signature}) = \text{SHA256}(\text{Transaction})$
- Association: Digital signature is associated with a particular transaction
 - Unique transaction ID

Bitcoin Script

There are multiple implementations of digital signature for Bitcoin in the real world. It depends on the particular scripting system used for transactions.



For more information:

- [Bitcoin Script](#)
- [White paper](#)
- [Schnorr Signature](#)
- [Elliptic Curve Digital Signature](#)

Another common implementation of digital signature from the white paper

Schedule for today

- Key concepts from last class
 - Proof-of-Work (PoW)
 - Role of miners
 - Difficulty adjustment + The Longest Chain Rule
- Transactions and digital signatures
- **Consensus for recording transactions**
 - Balance checking
 - The Double Spending Problem
- Selenium brief introduction & IDE demo

Balance checking

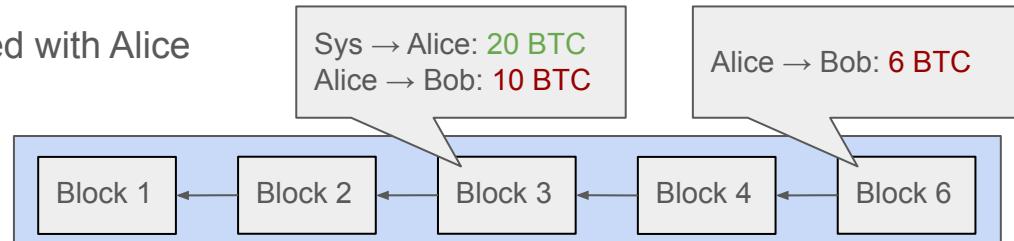
Bitcoin uses blockchain as a means to transparently record the transactions as well as **the balance**.

- Blockchain holds the complete transaction history
- Everyone can compute someone else's balance

Example: Alice → Bob: 5 BTC

- Alice (Bitcoin user) broadcasts the transaction
- Bitcoin miners verify the transaction

- Identify every transactions associated with Alice
- Calculate the remaining balance
- Accept or reject

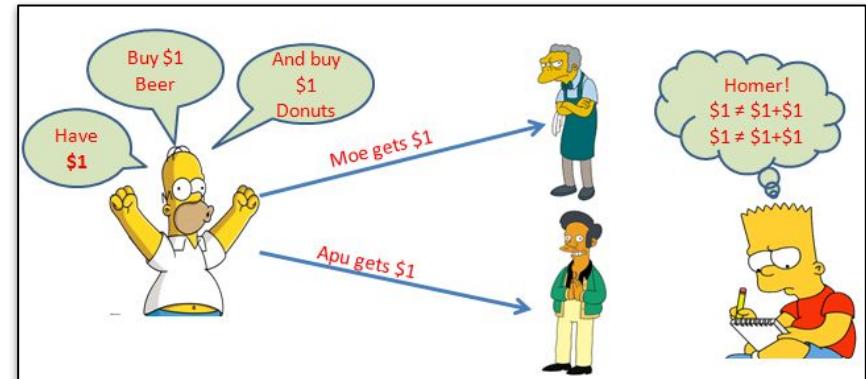


The Double Spending Problem

The Double Spending Problem describes a problem in transactions where the same BTC are spent more than once.

- Public ledger: Anyone can broadcast transactions
- Analogous to **The Byzantine Generals problem**

- ❑ **Challenge 1:** Agreeing on a single truth with decentralized systems
- ❑ **Challenge 2:** Reaching consensus even if some of them are faulty or malicious (Byzantine Fault Tolerance)



The Double Spending Problem

Suppose that Eve has 5 BTC

- Eve broadcasts two transactions at the same time
 - Eve → Bob: 5 BTC; Eve → Alice: 5 BTC;
- Bitcoin miners record the first transaction that they receive (network delays)
 - Carol (miner) records: Eve → Bob: 5 BTC
 - Dave (miner) records: Eve → Alice: 5 BTC
- **However**, both miners will wait until a block is created to confirm the transaction
 - Carol finds the special number and broadcasts the blockchain
 - Dave gives up his block and records Carol's block, following the Longest Chain Rule

The Double Spending Problem

Satoshi Nakamoto did not solve the Byzantine Generals Problem, but proposed a workaround using Proof-of-Work.

- Building trust based on The Longest Chain Rule

Challenge 1: Agreeing on a single truth with decentralized systems

- Trust the blockchain with the most work done (longest blockchain)

Challenge 2: Reaching consensus even if some of them are faulty or malicious

- Malicious nodes must do more work than the rest of the network to make the honest nodes trust their blockchain

Schedule for today

- Key concepts from last class
 - Proof-of-Work (PoW)
 - Role of miners
 - Difficulty adjustment + The Longest Chain Rule
- Transactions and digital signatures
- Consensus for recording transactions
 - Balance checking
 - The Double Spending Problem
 - **BTC Explorer**
- Selenium brief introduction & IDE demo

BTC Explorer

Latest Blocks

Height	Timestamp	Transactions	Size (KB)
837498	2024-04-03 00:41:07	3894	1616.358
837497	2024-04-03 00:11:13	4408	1732.096
837496	2024-04-03 00:07:47	4013	1747.949

BTC Explorer

- Height: Block number
- Timestamp: Time that the block was mined
- Transaction: Number of transactions in the block
- Size: Size of the block

Coinbase transaction

The screenshot shows a transaction details page from a blockchain explorer. At the top, it says "25 of 3894 Transactions". Below that is the transaction ID: "22aa9a7d089f911c89944141db8071020cf8d2f463c7af4bdff6673715099087". To the right is a "DETAILS" button with a "+" sign. The transaction has two outputs:

- Output #0: Coinbase (Address: bc1qxhmdufsvnuaaaer4ynz88fspdsxq2h9e9ctdj) - Amount: 6.53444409 BTC
- Output #1: OP_RETURN (Address: OP_RETURN) - Amount: 0 BTC

At the bottom, it shows "3 CONFIRMATIONS" and "6.53444409 BTC".

A coinbase transaction is the first transaction in a block.

- Transaction created by a miner
- Used to collect the block reward for their work and transaction fees
- Current block reward: 6.25 BTC
- Next Bitcoin halving on April 19, 2024: 3.125 BTC

Schedule for today

- Key concepts from last class
 - Proof-of-Work (PoW)
 - Role of miners
 - Difficulty adjustment + The Longest Chain Rule
- Transactions and digital signatures
- Consensus for recording transactions
 - Balance checking
 - The Double Spending Problem
- **Selenium brief introduction & IDE demo**

What is Selenium?

Selenium is an open-source, automated testing tool used to test web applications across different browsers.

- Goal: **Automate testing process** in software development life cycle (SDLC)

Selenium test suite comprises of four tools:

- **Selenium IDE**: plugin for recording user interaction and playing back
- Selenium Remote Control (RC) (**deprecated**): server that allows users to write application tests
- **Selenium WebDriver**: remote control interface for writing application tests
- Selenium Grid (mostly for Continuous Integration): server for parallel execution of tests on different browsers and different operating systems

How can Selenium be useful?

- Essential tool for QA analyst
 - Regression Testing
 - Integration with Continuous Integration (CI) Pipelines
- Project proposal for web developer
 - Functional bugs in UI
 - Cross-Browser Testing
- Assistance for UI tester
 - Complex Use Case / Application Flows
 - UI/UX Testing
- (Legal) Web scraping tool for freelancer
 - Data mining and extraction

Selenium IDE: The Basics

A Simple Demo: **Assert text “Territorial Acknowledgement” element exists**

Step 1 - Launch your Chrome menu and open the Selenium IDE plugin.

Step 2 - Select “Record a test in a new project.” Provide the name for your test.

Step 3 - Provide a link to the UAlberta webpage. The IDE starts recording by navigating to this web page.

Step 4 - Once the website opens, click on “OUR STORY” in the header of the webpage.

Step 5 - Scroll to the bottom of the page, select “Territorial Acknowledgement”

Step 6 - Now, modify the recording and change “Command” to “assert text”, enter “Value” as “Territorial Acknowledgement”

Lecture 32

Selenium

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- **Introduction to Selenium**
 - History of Selenium
 - Why is automated testing important?
 - How can Selenium be useful?
- **Selenium basics**
 - Locating elements
 - XPath syntax
- **Demo: Cookie Clicker**
 - Navigating pages
 - Waits for elements

What is Selenium?

Selenium is an open-source, automated testing tool used to test web applications across different browsers.

- Goal: **Automate testing process** in software development life cycle (SDLC)

Selenium test suite comprises of four tools:

- **Selenium IDE**: plugin for recording user interaction and playing back
- Selenium Remote Control (RC) (**deprecated**): server that allows users to write application tests
- **Selenium WebDriver**: remote control interface for writing application tests
- Selenium Grid (mostly for Continuous Integration): server for parallel execution of tests on different browsers and different operating systems

History of Selenium

- 2005: **Selenium IDE**
 - Chrome and Firefox plugin that records user interactions on the browser and plays them back as automated tests
- 2007: Selenium Remote Control (RC) (**deprecated**)
 - Selenium Server + Client libraries
 - Selenium Server: launches and kills browsers
 - Client libraries: manages user interactions with the browser
- 2008: Selenium WebDriver 2.0
- 2016: Selenium WebDriver 3
- 2021: **Selenium WebDriver 4**
 - Programming interface that instructs the behavior of web browsers

How can Selenium be useful?

- Essential tool for QA analyst
 - Regression Testing
 - Integration with Continuous Integration (CI) Pipelines
- Project proposal for web developer
 - Functional bugs in UI
 - Cross-Browser Testing
- Assistance for UI tester
 - Complex Use Case / Application Flows / Acceptance Tests
 - UI/UX Testing
- (Legal) Web scraping tool for freelancer
 - Data mining and extraction

Advantages of Selenium

Selenium offers a competitive edge over others tool:

- Open source: [available on GitHub](#)
 - Transparency + Flexibility + Security
- Multiple browsers, languages and platforms supports
 - Languages: Python, Java, C#, JavaScript, Ruby, Rust
 - Browsers: Chrome, Firefox, Safari, Internet Explorer, Microsoft Edge
 - Platforms: Windows, MacOS, Ubuntu
- Framework supports
 - TestNG and JUnit
 - Behat + Mink
- Parallel test execution
 - Optimize Continuous Integration and Delivery (DevOps)

Schedule for today

- Introduction to Selenium
 - History of Selenium
 - **Why is automated testing important?**
 - How can Selenium be useful?
- Selenium basics
 - Locating elements
 - XPath syntax
- Demo: Cookie Clicker
 - Navigating pages
 - Waits for elements

Why is automated testing important?

Before automated testing was introduced, manual testing was the norm. However, it had several drawbacks:

- Expensive: Require a full time Quality Assurance (QA) team
- Error-prone: Human-driven manual process for review and validation
- Slow: New features on-hold until the QA team finished testing
- Redundant and tedious: Manually execution of use cases every time a new update was pushed to production

Most modern **Agile** and **DevOps** development requires continuous testing.

Test automation is becoming an industrial relevant topic in software education community: Software Testing Education workshop ([TestEd 2024](#) as part of [ICST](#))

Schedule for today

- Introduction to Selenium
 - History of Selenium
 - Why is automated testing important?
 - How can Selenium be useful?
- **Selenium basics**
 - Locating elements
 - XPath syntax
- Demo: Cookie Clicker
 - Navigating pages
 - Waits for elements

Selenium basics

Writing an automated test in Selenium typically involves three steps:

- **Locating elements**
 - Obtaining element references to work with
 - Selenium uses locator strategies to uniquely identify each HTML element
- **Navigating pages**
 - Interacting with web elements
 - Only 5 basic commands on each element: click, send keys, clear, submit, select
- **Waits for elements**
 - Avoid race condition in browser (delay exists in loading elements)
 - Example: Clicking on a button before the button element is present in the DOM

Locating elements

Selenium provides built-in methods to locate web elements in a page:

- `find_element()` for locate a single element
- `find_elements()` for locating multiple elements

These methods takes two parameters as inputs → `find_element(Locator, Value)`:

- **Locator**: identifies the locator used for finding the element
- **Value**: gives the value of the locator

Example: Find element with locator `By.CLASS_NAME`

```
<a aria-label="University of Alberta"  
    class="navbar-brand en-logo"  
    href="https://www.ualberta.ca/index.html">  
</a>
```

```
find_element(By.CLASS_NAME, "en-logo")
```

or

```
find_element(By.CLASS_NAME, "navbar-brand")
```

Available attributes for Locator

Locator (used in `find_element`) has eight available attributes:

Locator attribute	Sample element	Example of code
By.ID	<code><h1 id="header">Header</h1></code>	<code>find_element(By.ID, "header")</code>
By.NAME	<code><h1 name="header">Header</h1></code>	<code>find_element(By.NAME, "header")</code>
By.CLASS_NAME	<code><h1 class="header">Header</h1></code>	<code>find_element(By.CLASS, "header")</code>
By.LINK_TEXT	<code>Home</code>	<code>find_element(By.LINK_TEXT, "Home")</code>
By.PARTIAL_LINK_TEXT	<code>Home</code>	<code>find_element(By.LINK_TEXT, "ome")</code>
By.TAG_NAME	<code><h1>Header</h1></code>	<code>find_element(By.TAG_NAME, "h1")</code>
By.XPATH	<code><h1>Header</h1></code>	<code>find_element(By.XPATH, "//h1")</code>
By.CSS_SELECTOR	<code><h1 class="header">Header</h1></code>	<code>find_element(By.CSS_SELECTOR, "h1.header")</code>

Locating by XPath

XPath (XML Path Language) is a path expression language designed to select nodes or node-sets in a XML (or HTML) document.

- Nodes are selected by following a path

One of the main reasons for using XPath is for its reliability:

- Id and name attributes may not exist on all elements
 - By.ID, By.NAME
- Link attributes are associated with links only
 - By.LINK_TEXT, By.PARTIAL_LINK_TEXT
- Tag and class attributes may not always be unique
 - By.TAG_NAME, By.CSS_SELECTOR, By.CLASS_NAME

XPath syntax

Overall idea: Tagname[some identifiers] + Path to the tag

XPath = // tagname [@ Attribute = 'Value']

XPath syntax:

- `//`: Select particular nodes in the HTML tree (Set search space)
 - Single forward slash (`/`) selects only the immediate child elements
 - Double forward slash (`//`) selects all descendants of the current node, regardless of their level
- `Tagname`: Set tagname of the particular node
- `@`: Select attribute
- `Attribute`: Set attribute name of the node
- `Value`: Set value of the attribute

Absolute and relative XPath

There are two types of XPath:

- **Absolute**: Find element through its absolute path, starting from the root element
 - By using the single forward slash (/)
- **Relative**: Find elements anywhere on the page
 - By using the double forward slash (//)

Example of XPath: Look for the UAlberta logo ([Try it yourself!](#))

- **Absolute**: /html/body/header/div/div/div/div/a[@aria-label='University of Alberta']
 - Starting from the root element <html>, go through this path and find element
- **Relative**: //header//a[@aria-label='University of Alberta']
 - Anywhere on the page: Find the first <header>
 - Anywhere inside this header: Find the first <a> with aria-label attribute equals to University of Alberta

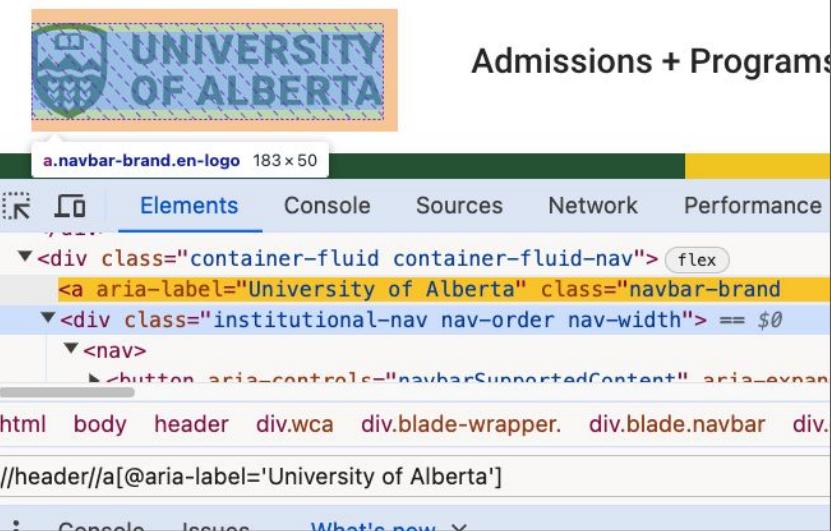
Try it yourself!

Use the following XPath to find the first UoA logo in the <header> of [UAlberta.ca](#):

- /html/body/header/div/div/div/a[@aria-label='University of Alberta']
- //header//a[@aria-label='University of Alberta']

Instructions: On [UAlberta website](#)

- Inspect the main webpage
- In the Elements view, press Ctrl + F
(or command + F)
- Search //a[@aria-label='University of Alberta'] in the source code
- All UoA should now be highlighted



Schedule for today

- Introduction to Selenium
 - History of Selenium
 - Why is automated testing important?
 - How can Selenium be useful?
- Selenium basics
 - Locating elements
 - XPath syntax
- Demo: Cookie Clicker
 - Navigating pages
 - Waits for elements

Demo: Cookie Clicker



[Cookie Clicker](#) is a 2013 web game designed to have users click on a big cookie on the screen.

- [Steam version](#) released in 2021, reaching top 15 of Steam games at the time

Build an automation script for clicking the cookie 100 times

- Input: Cookie Clicker URL
- Learning Objectives: Click + Wait + Scrap + Assert
- Output: Assert that the cookie has been clicked 100 times
- Source code available on GitHub: [automation script](#)

Demo guide (Python)

Step 1: Installation

- [Python language bindings for selenium package](#)

Step 2: Python script setup

- [Getting started with WebDriver](#)

Step 3: Navigate and click

- [Interacting with the page](#)
- [Explicit Waits](#)

Step 4: Assertion

- [Test case with assertions](#)

Step 1: Installation

Step 1: To run Selenium, we first need to install a WebDriver.

- **WebDriver** is responsible for managing the content interactions, without requiring browser-specific code

Every browser provides WebDriver supports:

- Chrome: [ChromeDriver](#)
- Firefox: [GeckoDriver](#)
- Edge: [Microsoft Edge WebDriver](#)
- Safari: [SafariDriver](#)

If you are using Chrome version 115 or newer, check the [Chrome for Testing availability dashboard](#)

Note that this demo uses: chromedriver for mac-x64 ([click here to download](#))

Step 2: Python script setup

[main.py](#) available

Step 2: create a Python script (e.g., main.py) in the same folder as the WebDriver

- **main.py** is the automation script we use for clicking the cookie

Set up the script by importing the WebDriver, then try to launch the browser:

- Import the **WebDriver** and **By** class (for locating elements)

```
from selenium import webdriver  
from selenium.webdriver.common.by import By
```

- Create an instance of **Chrome WebDriver**

```
driver = webdriver.Chrome()
```

- Use **driver.get** method to navigate to a given URL

```
driver.get("https://orteil.dashnet.org/cookieclicker/")
```

Step 2: Python script setup

- Launch browser in incognito mode (reproductivity, without cookies)

```
chrome_options = webdriver.ChromeOptions()  
chrome_options.add_argument("--incognito")
```

- Add a "pause" time to make the browser visible

```
import time  
time.sleep(10)
```

- Close the browser once done

```
driver.quit()
```

Step 3: Navigate and click

Step 3: Select a language and click on the cookie

- Step 3.1: Select the English element from language popup
 - o Appear on first visit (incognito for test case rerun)
 - o Take time to load (wait for the element to appear)
- Step 3.2: Click on the cookie
 - o Take time to load (wait for the element to appear)
 - o Click 100 times (add loop)



Navigating pages

Navigating pages = interacting with HTML elements (e.g., links) within a page

- Locating + simulating a click

Example: Click on a web element

- Locating elements with XPath

```
logo_xpath = "//*[@aria-label='University of Alberta']"  
logo_element = driver.find_element(By.XPATH, logo_xpath)
```

- Simulate a click on elements

```
logo_element.click()
```

Waits for elements

When a page is loaded by the web browser, the elements within that page are often loaded at different time intervals.

- This makes locating element difficult
- Elements that we are locating may not be present in the DOM yet

Solution: Assign **explicit waits** for elements

- Wait for a condition to occur before proceeding further in the code

Example: Wait until the element appears for 10 seconds

```
wait = WebDriverWait(driver, 10)
wait.until(EC.presence_of_element_located((By.XPATH, element_xpath)))
```

Step 3: Navigate and click

[main.py available](#)

Step 3.1: Select the English element from language popup

- Inspect the page and write (or copy) the XPath for "English" web element

The screenshot shows a game interface for 'Pirate Potato's Bakery'. On the left, there's a large cookie icon with the text '0 cookies per second: 0'. In the center, there's a language selection dropdown menu with the following options: English (highlighted in blue), Deutsch, Nederlands, Čeština, Polski, Italiano, Español, Português, 日本語, and 中文. Below the dropdown is some UI text: 'Option' (partially visible), 'Staff', 'Info', and 'Legacy'. At the bottom of the screen is a browser's developer tools Elements tab, which displays the following code snippet:

```
<div class="langSelectButton title" style="padding:4px;" id="langSelect-EN">English</div>
<div class="langSelectButton title" style="padding:4px;" id="langSelect-FR">Français</div>
```

A context menu is open over the 'English' button in the dropdown. The menu has the following items:

- Add attribute
- Edit attribute
- Edit as HTML
- Duplicate element
- Delete element
- Cut
- Copy** (selected)
- Paste
- Hide element
- Force state
- Break on
- Expand recursively
- Collapse children
- Capture node screenshot
- Scroll into view

The 'Copy XPath' option is highlighted in blue.

Step 3: Navigate and click

Step 3.1: Select the English element from language popup

- Wait for the English element to appear

```
language_xpath = "//*[@id='langSelect-EN']"  
wait = WebDriverWait(driver, 10)  
language_element = wait.until(  
    EC.presence_of_element_located((By.XPATH, language_xpath))  
)
```

- Click on English

```
language_element.click()
```

Step 3: Navigate and click

[main.py](#) available

Step 3.2: Click on the cookie

- Wait for the cookie element to appear

```
cookie_xpath = "//*[@id='bigCookie']"  
wait = WebDriverWait(driver, 10)  
cookie_element = wait.until(  
    EC.presence_of_element_located((By.XPATH, cookie_xpath))  
)
```

- Click on cookie

```
cookie_element.click()
```

Step 3: Navigate and click

Step 3.2: Click on the cookie

- Find cookie counter element and get integer

```
count = 0
count_xpath = "//*[@id='cookies']"
count = driver.find_element(By.XPATH, count_xpath).text.split(' ')[0]
```

- Click 100 times (string to int)

- Find count and cookie elements after each click (avoid stale elements)

```
count = 0
count_xpath = "//*[@id='cookies']"
while int(count) < 100:
    cookie_element = driver.find_element(By.XPATH, cookie_xpath)
    cookie_element.click()
    count = driver.find_element(By.XPATH, count_xpath).text.split(' ')[0]
```

Step 4: Assertion

[main.py](#) available

Step 4: Assert that the cookie is less than 100, after an upgrade

- Upgrade for cookie auto-generation

```
grandma_xpath = "//*[@id='product1']"  
grandma_element = driver.find_element(By.XPATH, grandma_xpath)  
grandma_element.click()
```

- Capture count element again and assert that it is less than 100

- If the assertion fails, then the script throws an AssertionError

```
count = driver.find_element(By.XPATH, count_xpath).text.split(' ')[0]  
assert int(count) < 100
```

Lecture 33

Selenium - Part II

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Demo: Cookie Clicker
 - Step 4: Assertion
- What more can we do with Selenium?
- Surveys, feedbacks and ideas
 - SPOT Survey
 - ECE 422 course website
- Final exam
 - Format and type of questions
 - Review sessions

Demo: Cookie Clicker



[Cookie Clicker](#) is a 2013 web game designed to have users click on a big cookie on the screen.

- [Steam version](#) released in 2021, reaching top 15 of Steam games at the time

Build an automation script for clicking the cookie 100 times

- Input: Cookie Clicker URL
- Learning Objectives: Click + Wait + Scrap + Assert
- Output: Assert that the cookie has been clicked 100 times
- Source code available on GitHub: [automation script](#)

Demo guide (Python)

Step 1: Installation

- [Python language bindings for selenium package](#)

Step 2: Python script setup

- [Getting started with WebDriver](#)

Step 3: Navigate and click

- [Interacting with the page](#)
- [Explicit Waits](#)

Step 4: Assertion

- [Test case with assertions](#)

Step 1: Installation

Step 1: To run Selenium, we first need to install a WebDriver.

- **WebDriver** is responsible for managing the content interactions, without requiring browser-specific code

Every browser provides WebDriver supports:

- Chrome: [ChromeDriver](#)
- Firefox: [GeckoDriver](#)
- Edge: [Microsoft Edge WebDriver](#)
- Safari: [SafariDriver](#)

If you are using Chrome version 115 or newer, check the [Chrome for Testing availability dashboard](#)

Note that this demo uses: chromedriver for mac-x64 ([click here to download](#))

Step 2: Python script setup

[main.py](#) available

Step 2: create a Python script (e.g., main.py) in the same folder as the WebDriver

- **main.py** is the automation script we use for clicking the cookie

Set up the script by importing the WebDriver, then try to launch the browser:

- Import the **WebDriver** and **By** class (for locating elements)

```
from selenium import webdriver  
from selenium.webdriver.common.by import By
```

- Create an instance of **Chrome WebDriver**

```
driver = webdriver.Chrome()
```

- Use **driver.get** method to navigate to a given URL

```
driver.get("https://orteil.dashnet.org/cookieclicker/")
```

Step 3: Navigate and click

Step 3: Select a language and click on the cookie

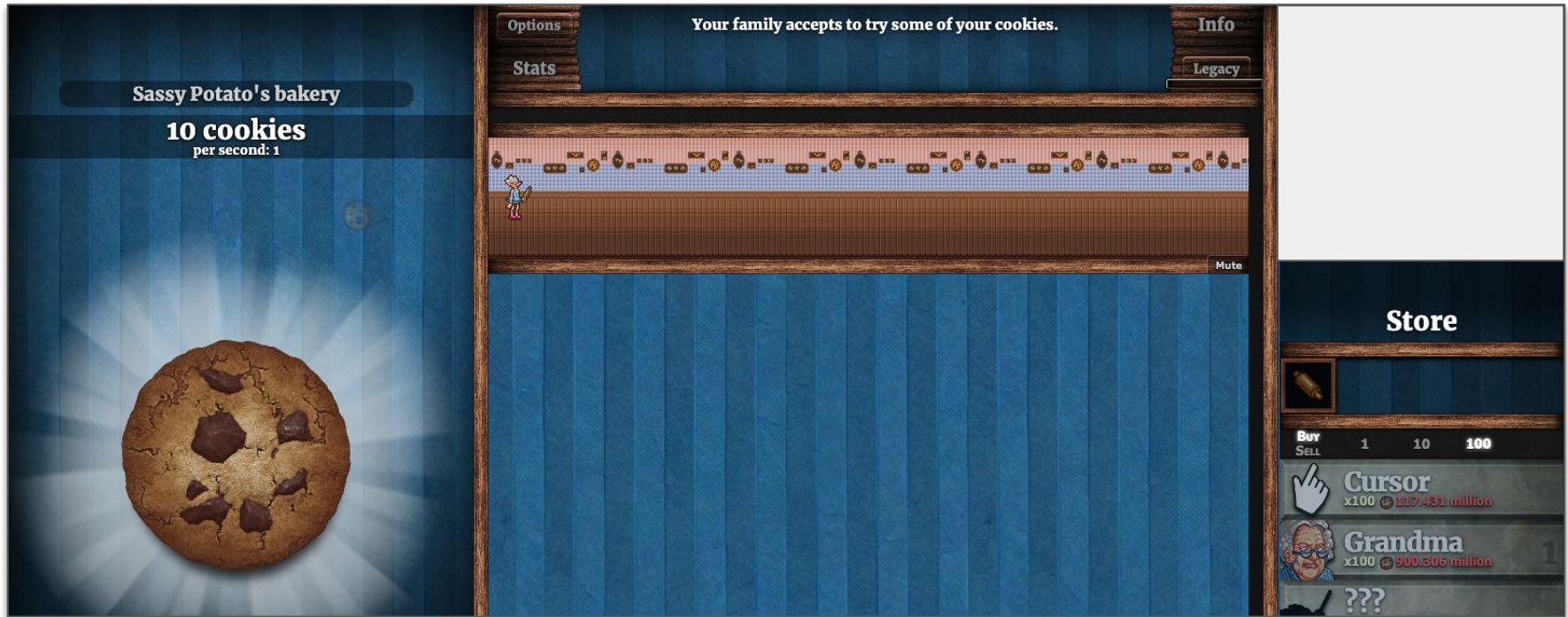
- Step 3.1: Select the English element from language popup
 - o Appear on first visit (incognito for test case rerun)
 - o Take time to load (wait for the element to appear)
- Step 3.2: Click on the cookie
 - o Take time to load (wait for the element to appear)
 - o Click 100 times (add loop)



Step 4: Assertion

[main.py](#) available

Step 4: Assert that the cookie is less than 100, after an upgrade



Step 4: Assertion

[main.py available](#)

Step 4: Assert that the cookie is less than 100, after an upgrade

- Upgrade for cookie auto-generation

```
grandma_xpath = "//*[@id='product1']"  
grandma_element = driver.find_element(By.XPATH, grandma_xpath)  
grandma_element.click()
```

- Capture count element again and assert that it is less than 100
 - To avoid elements become stale, it is essential to refresh the reference
 - If the assertion fails, then the script throws an AssertionError

```
count = driver.find_element(By.XPATH, count_xpath).text.split(' ')[0]  
assert int(count) < 100
```

What more can we do with Selenium?

- Scrape data from websites
 - Python, Selenium WebDriver + [pandas](#)
- Automated testing
 - Java, Selenium WebDriver + [TestNG](#)
 - Three-steps process:
 - Set up the test data `@BeforeTest`
 - Perform a set of actions `@Test`
 - Evaluate the results `@AfterTest`
 - Tests are often about the "happy path"
 - General works (no corner cases)
 - [Encouraged behaviors for Selenium tests](#)

```
public class LoginTest {  
    @BeforeTest  
    public void setUp(){ ... }  
  
    @Test  
    public void login(){ ... }  
  
    @AfterTest  
    public void tearDown(){ ... }  
}
```

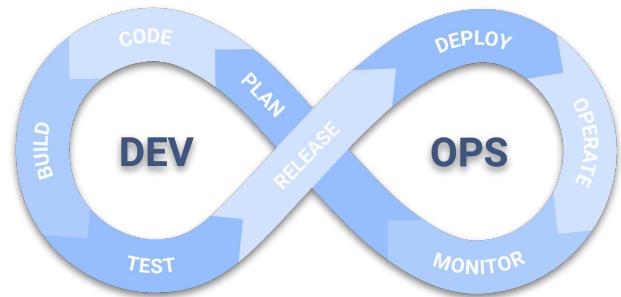
Sample automated test: Login

```
@Test  
public void login(){  
    driver.get("https://example.com/login");  
    WebElement username = driver.findElement(By.id("user_email"));  
    WebElement password = driver.findElement(By.id("user_pass"));  
    WebElement login = driver.findElement(By.name("submit"));  
    username.sendKeys("bob@example.com");  
    password.sendKeys("bob_password");  
    login.click();  
    WebElement welcome = driver.findElement(By.xpath("//span[@name='welcome_text']"));  
    Assert.assertEquals(welcome.getText(), "Welcome Bob")  
}
```

Selenium in DevOps

Selenium can be integrated into **DevOps workflow** to encourage collaboration:

- For Dev (**developers**)
 - Continuous Integration (CI)
 - Automatically run tests when code changes occur
 - Continuous Deployment (CD)
 - Validate functionality before pushing changes to production
- For Ops (**operators**)
 - Parallel Test Execution
 - Speed up the test execution time with cross-browser and cross-platform testing
 - Monitoring and Alerting
 - Continuous testing in production and staging environment to monitor the application



CV Template

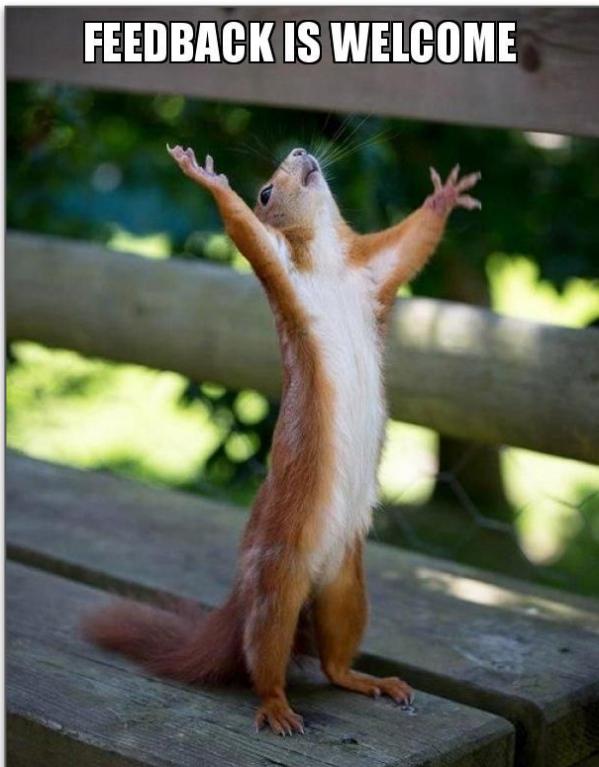
- [CV template available](#) on Overleaf (written in [LaTeX](#))
- Highlights of Achievements and Qualifications
 - Be specific and concise in your experiences, achievements and qualifications
 - E.g., Bachelor degree in Software Engineering with 5 years of Object-Oriented Programming experience
- Interview tips (Undergrad.)
 - Competence based on experience
 - Character as a coworker
 - Practice speaking points (bullet points)
 - Discuss career direction
 - Hint on no relocation
 - Hint on dream job



Schedule for today

- Demo: Cookie Clicker
 - Step 4: Assertion
- What more can we do with Selenium?
- Surveys, feedbacks and ideas
 - SPOT Survey
 - ECE 422 course website
- Final exam
 - Format and type of questions
 - Review sessions

SPOT Survey



SPOT Survey: [Click here](#)

- 15 minutes
- Course number: 15754
- Survey close: Apr 14, 11:59 PM

“Beginning with winter 2024 courses, instructors are now required to provide 15 minutes of class time to complete the survey. Please note that instructors will not be present during this class time.”

ECE 422 course website

Why? Course archive for past students (you) + Preview for future students:

- Syllabus + Lecture slides
 - Module 1: DevOps + SREs
 - Module 2: Reliable and Fault-Tolerant Design
 - Module 3: Security Principles
 - Module 4: Emerging topics - Bitcoin and Selenium
- Demos + Course projects
 - Web Security: XSS, CSP, SOP
 - Selenium: Cookie Clicker
- Midterm and final exam

ECE 422 - Reliable and Secure Systems Design

★ 3 (fi 8)(EITHER, 3-0-0)

[Faculty of Engineering](#)

Causes and consequences of computer system failure. Structure of fault-tolerant computer systems. Methods for protecting software and data against computer failure. Quantification of system reliability. Introduction to formal methods for safety-critical systems. Computer and computer network security. Prerequisite: CMPUT 301. Corequisite: ECE 487. Credit may be obtained in only one of CMPE 420 or ECE 422.

ECE 422 course website

How can you help? Any feedbacks are welcome!

- Email me (anran6@ualberta.ca)

Example of feedbacks and ideas:

- Available content for both past or future students
 - E.g., Logs/blogs for new content updates
- Website designs
 - E.g., Hidden content when you know how to reverse-engineering a hash function
- Interactive demos
 - E.g., Hosting a server to have students try XSS

Schedule for today

- Demo: Cookie Clicker
 - Step 4: Assertion
- What more can we do with Selenium?
- Surveys, feedbacks and ideas
 - SPOT Survey
 - ECE 422 course website
- Final exam
 - Format and type of questions
 - Review sessions

Format of exam

- Classroom: The Centennial Centre for Interdisciplinary Science (CCIS) 1-160
- Date: Wednesday, Apr 24th, 2024
- Duration: 2 hours
- Closed book
- Materials: all materials posted in the lecture slides
 - => 60% on new materials, < 40% on old materials
- The final counts for 30% of the overall course grade

Email me if you need to know your grade early for work permit or graduation
(estimated **early grade release date**: ~ May 1).

Type of question

- True/False
- Multiple choice
- Short answer
- **Essay questions**
 - Explain and give examples
- Computational questions
 - Bring your own calculator

Course registration system

User story: As a security admin, I want to ensure the integrity of the data

- **Lecture 13:** The CIA Triad
- **Lecture 14:** Hash function and Digital Signature
 - Hash collision on integrity checks

Key concepts:

- Digital Signature: Authenticity + Integrity + Non-repudiation
- Difference between hashing and encryption: irreversibility

Course registration system

User story: As a security admin, I want to ensure the confidentiality of the system.

- **Lecture 15:** Authentication
 - Multi-factor authentication
- **Lecture 16:** Access Control
 - Models of access control
- **Lecture 17:** Encryption
 - Symmetric and asymmetric encryption

Key concepts:

- MFA: knowledge, possession, biologic, location and time
- DAC, RBAC, MAC, ABAC
- Asymmetric: encryption = sender's private key; decryption = public key

Course registration system

User story: As a web admin, I want to prevent race condition for course registration.

- **Lecture 20:** The Dining Philosophers Problem
 - Race condition
 - Atomic locking

Key concepts:

- Locks to prevent multiple users from registering the courses at the same time
- Atomic locks for course reservation

Midterm review

ECE 422: Reliable and Secure Systems Design



UNIVERSITY OF
ALBERTA

Instructor: An Ran Chen
Term: 2024 Winter

What is the format of the exam?

- Classroom: ETLC E2-002
- Duration: 45 minutes
 - 30 minutes should be enough
 - Please arrive 10 minutes before
- Closed book
- Materials: all the materials posted in the lecture slides
 - There will be a question on the project
- The midterm counts for 25% of the overall course grade

What types of question to expect?

- True/False
- Multiple choice
- Short answer
- Computational questions
 - Please bring a calculator

What to expect on the midterm?

- Concepts: things to know
 - Understand the definition
 - True/false or multiple choice questions
- Explanation: things to explain
 - Be able to explain the concepts, and give examples
 - Multiple choice questions or short answer questions
- Problem: things to solve
 - Understand the mechanism behind the concepts
 - Computational questions

List of materials to prepare you

- Consider the review slides a study guide and a sample midterm
 - Computational questions available
- Go through the concepts in the review slides
- Go through the questions in the lecture slides
- Textbooks: [Google Drive link](#)
 - [Building Secure and Reliable Systems](#)
- Materials from the past years: [Google Drive link](#)
 - Software Redundancy -> Lecture 4 Fault-Tolerant Design Winter 2024
 - Sample midterm available only for the format of the exam

Software development methodologies

- Lecture 1 and 2: DevOps
 - Agile methodology
 - Workflow
 - User stories
 - Planning poker
 - Continuous integration, continuous delivery, continuous deployment
 - Docker container

Software development methodologies

- Lecture 3: Software Reliability Engineering
 - Availability, reliability [Explanation]
 - Mean time between failures, mean time to repair
 - Service level agreement, objective, indicator (SLA, SLO, SLI)
 - Error budget

Availability



MTBF (Mean Time Between Failure) is the average time between two consecutive failures.

$$\text{MTBF} = \frac{\text{Operational Time}}{\text{Number of Failures}}$$

MTTR (Mean Time to Repair) is the average time it takes to restore the system after a failure

$$\text{MTTR} = \frac{\text{Total Repair Time}}{\text{Number of Failures}}$$

Availability is the percentage of time that a system is operational

$$\text{Availability} = \frac{\text{MTBF}}{\text{MTBF} + \text{MTTR}}$$

Reliability



Reliability is the likelihood that a system will perform its function without failure.

- Calculated by MTBF (Mean Time Between Failure)
- The higher MTBF, the more reliable and available the system becomes

How to define reliability standards?

- Risks
 - Safety-critical systems vs personal websites
- Customer expectations
 - Translating “it should never fail” into “99.99% chance of successful operation”
 - Function, environment, and probability of failure
 - E.g., Sales transaction should work 99.99% of the time during Black Friday sales
- SLAs (Service level agreements)

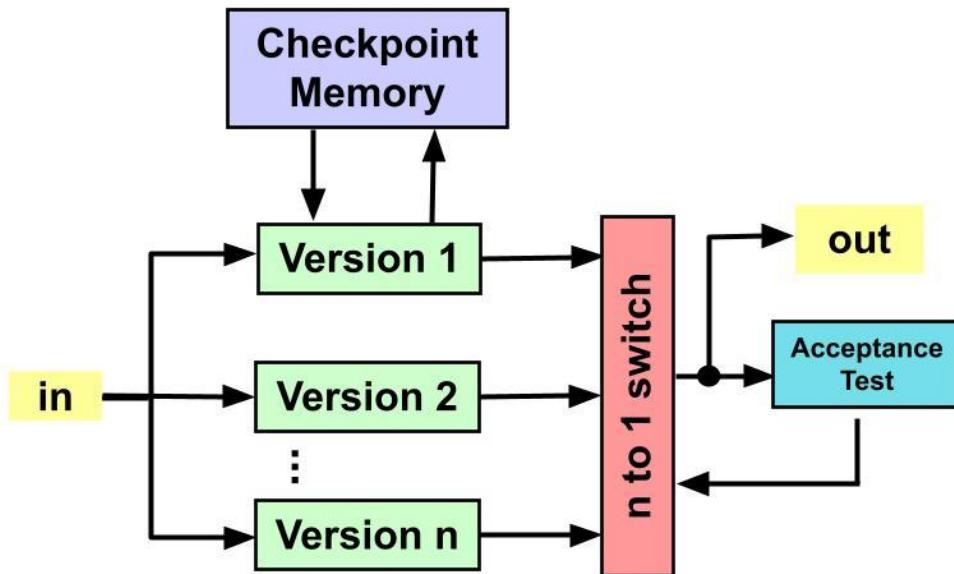
Reliability designs

- Lecture 4: Fault recovery
 - Backward error recovery
 - Forward error recovery
- Lecture 4: Fault tolerance techniques
 - Exception handling
 - Recovery blocks [Explanation]
 - N-version programming [Explanation]
 - N self-checking programming [Explanation]

Recovery blocks



Example

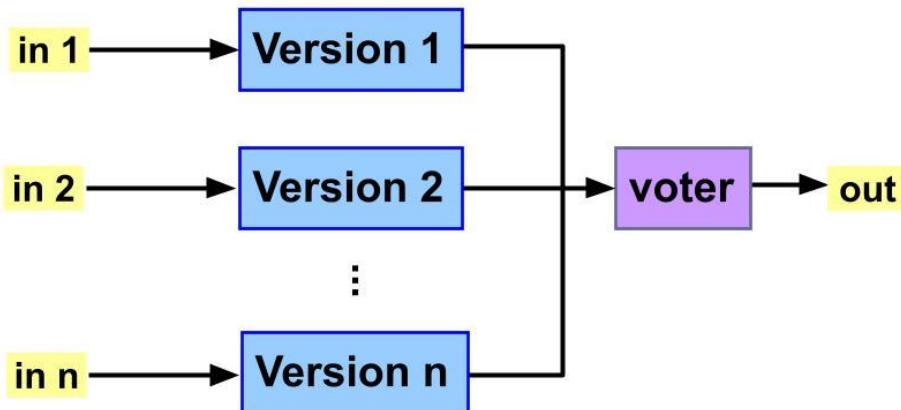


1. Run the primary version for the acceptance test
2. If the primary version fails, roll back the state of the system before the execution
3. Run the next version for the same acceptance test until there is an acceptable output
4. If none produce acceptable outputs, the system fails

N-version programming



Example

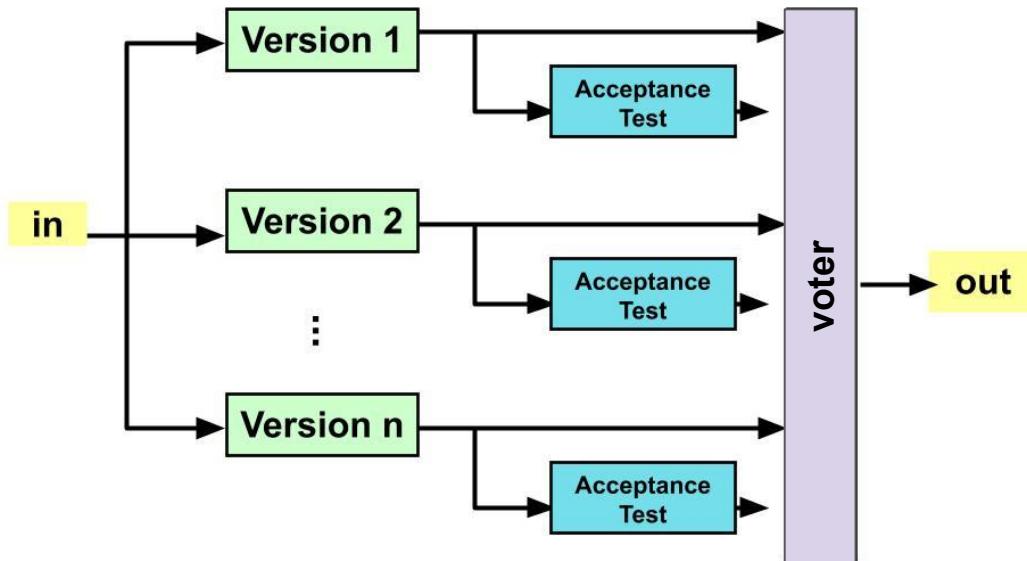


1. Run all versions concurrently
2. Run the decision mechanism based on the voter (e.g., majority win)
3. Return the most common output from the individual versions

N self-checking programming



Example: N self-checking by acceptance tests



1. Run all versions concurrently or sequentially
2. Run the corresponding acceptance test for each version
3. Run the voter
4. Return the majority agreement as output (only applied to versions that have passed their acceptance test)

Reliability designs

- Lecture 5: Fault removal
 - Functional testing
 - Unit test, integration test, acceptance test, regression test
 - Structural testing
 - Mutation test, data flow test, control flow test
 - Code coverage [\[Explanation\]](#)
 - Statement, branch, path coverage
- Lecture 5: Dependability
 - Impairments, measures, means

Code coverage

```
x = 0;  
if (condition)  
    x = x + 1;  
y = 10/x;
```

Test 1 where condition = true

- 100% statement coverage
- No error found in the code

Take-home 1: there
is an insensitivity of
statement coverage
to control structures.

Test 2 where condition = false

- 75% statement coverage
- Error found in the code

Take-home 2: 100%
statement coverage
does not mean there
is no bug in the
code.



Code coverage



```
if (condition1)  
    x = 0;  
else  
    x = 2;  
if (condition2)  
    y = 10*x;  
else  
    y = 10/x;
```

Test 1 where condition1 = true,
and condition2 = true,

Test 2 where condition1 = false,
condition2 = false,

- 100% branch coverage
- No error found in the code

Test 3 where condition1 = true,
and condition2 = false,

- 50% branch coverage
- Error found in the code

Take-home 1: 100%
branch coverage
does not mean there
is not bug in the
code.

Branch coverage =
(Number of Decisions
Outcomes tested / Total
Number of Decision
Outcomes) x 100 %

Reliability designs

- Lecture 6: Fault localization
 - Rubber duck debugging
 - Spectrum-based fault localization
 - 4-step process [Explanation]
 - Suspiciousness score [Problem]
 - Ochiai formula given
 - Information retrieval-based fault localization
 - 3-step process [Explanation]
 - Suspiciousness score [Problem]
 - Cosine similarity formula given

Spectrum-based fault localization



Question: Based on the following execution profile, find the most suspicious statement and compute its suspiciousness score.

	T ₁	T ₂	T ₃
S ₁	✓		
S ₂	✓		
S ₃			✓
S ₄	✓	✓	
Result	P	F	F

$$Ochiai(element) = \frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}}$$

e_f Number of failed tests that execute the program element.

e_p Number of passed tests that execute the program element.

n_f Number of failed tests that do not execute the program element.

n_p Number of passed tests that do not execute the program element.

Spectrum-based fault localization



Question: Based on the following execution profile, find the most suspicious statement and compute its suspiciousness score.

	T ₁	T ₂	T ₃
S ₁	✓		
S ₂	✓		
S ₃			✓
S ₄	✓	✓	
Result	P	F	F

Solution: S₃ is the most suspicious with a score of 0.71

- S₁ = 0, no failed test
- S₂ = 0, no failed test
- S₃ = 0.71, e_f = 1, n_f = 1
- S₄ = 0.50, e_f = 1, e_p = 1, n_f = 1

$$Ochiai(element) = \frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}}$$

e_f Number of failed tests that execute the program element.

e_p Number of passed tests that execute the program element.

n_f Number of failed tests that do not execute the program element.

n_p Number of passed tests that do not execute the program element.

Information retrieval-based fault localization

Question: Based on the following bug report and source file, what is the suspiciousness score of the file?

- Bug report = “a problem with the classNotFound exception.”
- Source file = “get classNotFound exception return exception”

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2} \cdot \sqrt{\sum_{i=1}^n B_i^2}}$$

Information retrieval-based fault localization



Step 1: create a vector representation of the query and document.

- Bug report/query = “a problem with the classNotFound exception.”
- Source file/document = “ get classNotFound exception return exception”

	a	problem	with	the	classNotFound	exception	get	return
A	1	1	1	1	1	1	0	0
B	0	0	0	0	1	2	1	1

Vector representation:

- A = [1, 1, 1, 1, 1, 1, 0, 0]
- B = [0, 0, 0, 0, 1, 2, 1, 1]

Information retrieval-based fault localization



Step 2: calculate the dot product and magnitude of these vectors

Vector representation:

- $A = [1, 1, 1, 1, 1, 1, 0, 0]$
- $B = [0, 0, 0, 0, 1, 2, 1, 1]$

Dot product of the vectors:

$$A * B = 1 \times 0 + 1 \times 0 + 1 \times 0 + 1 \times 0 + 1 \times 1 + 1 \times 2 + 0 \times 1 + 0 \times 1 = 3$$

Magnitude of the vectors:

$$\| A \| = \sqrt{(1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 1^2 + 0 + 0)} = \sqrt{6}$$

$$\| B \| = \sqrt{(0^2 + 0^2 + 0^2 + 0^2 + 1^2 + 2^2 + 1^2 + 1^2)} = \sqrt{5}$$

Information retrieval-based fault localization



Step 3: calculate the cosine similarity

$$\text{similarity}(A, B) = \frac{A * B}{\|A\| \|B\|} = \frac{3}{\sqrt{6} * \sqrt{5}} = 0.5477$$

The bug report and source code file could be said to be 55% similar.

Information redundancy

- Lecture 7: Error detecting and correcting code
 - Code, codeword, word, codespace
 - Encoding and decoding
 - Hamming distance
 - Code distance in detection and correction [\[Explanation\]](#)
 - Information rate, formula given (in Lecture 8)
 - Repetition codes
 - Parity codes

Code distance



Question: Give the appropriate (n, k, d) designation for a $(7, 4)$ Hamming code. Also give the number parity bits and the information rate.

Hint 1: n is number of bits in each codeword, k is the number of data bits, and d is the minimum Hamming distance between codewords.

Hint 2: a $(7, 4)$ Hamming code can correct and detect any single-bit error.

Information rate = k/n

Code distance



Question: Give the appropriate (n, k, d) designation for a $(7, 4)$ Hamming code. Also give the number parity bits and the information rate.

Solution: $(7, 4)$ Hamming code uses 7 bits to encode 4 bits of data

- $n = 7$, codeword bits
- $k = 4$, data bits
- $d = 3$, minimal Hamming distance
 - To correct at least one (1) bit error, the code distance must be larger or equal to $2(1)+1$
- parity bits = $7 - 4 = 3$
- Information rate = $4/7$

Information rate = k/n

Information redundancy



- Lecture 8: Hamming codes
 - Encoding and decoding [Problem]
 - Page 16 and 21 on (15, 11) Hamming code
 - Error detection and correction
 - Extended Hamming codes [Problem]

Error detection



-			0
	1	1	1
	0	1	0
0	1	0	0

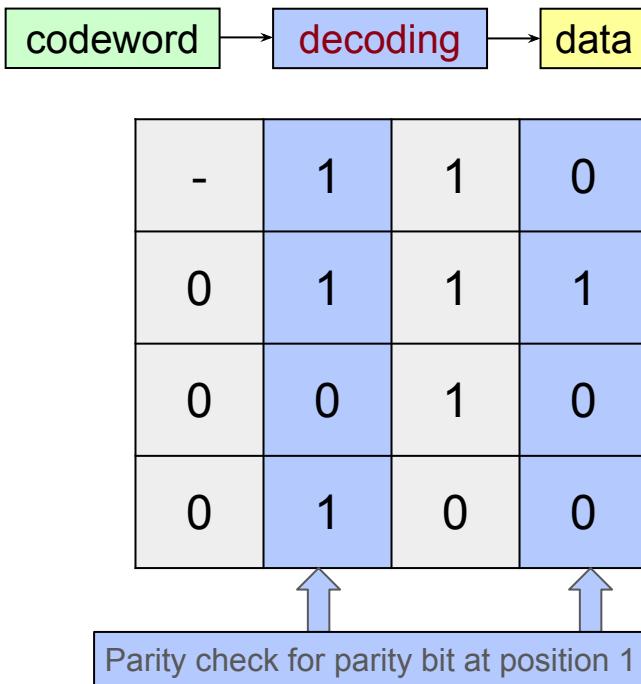
Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Step 1: calculate parity bit #1 at position 1

- We check the parity of the bits in the second and last columns.
- The bits {0 1 1 0 0 1 0} contain three “1”s.
- That is an odd number of “1”s.
- Therefore, we set the **first parity bit to 1** to make up for an even parity.

Error detection



Example: using even parity

Data: {0 1 1 1 0 1 0 0 1 0 0}

Scenario 1: No error

The receiver repeat the same process.

Parity bit at position 1: {1 0 1 1 0 0 1 0}, even parity, no error

Parity bit at position 2: {1 0 1 1 1 0 0 0}, even parity, no error

Parity bit at position 4: {0 1 1 1 0 1 0 0}, even parity, no error

Parity bit at position 8: {0 0 1 0 0 1 0 0}, even parity, no error

Information redundancy

- Lecture 9 and 10: Cyclic codes
 - Linear and cyclic properties [Explanation]
 - Encoding and decoding [Problem]
 - Polynomial multiplication and division
 - Error detection

Encoding



Suppose $g(x) = (1+x+x^3)$ for a (7,4) cyclic code

Question: Find the codeword polynomial for the data 1010. (show your steps)

Encoding



Suppose $g(x) = (1+x+x^3)$ for a (7,4) cyclic code

Question: Find the codeword polynomial for the data 1010. (show your steps)

Thought process: We have the values $g(x)$ and $d(x)$, asked to calculate $c(x)$

- Solve $c(x) = d(x)g(x)$ using the cyclic property
 - Step 1: write down the data as a 7-bit codeword ($n = 7$)
 - Step 2: apply the cyclic property to calculate polynomial multiplication
 - Step 3: convert the codeword into a codeword polynomial

Encoding



Suppose $g(x) = (1+x+x^3)$ for a (7,4) cyclic code

Question: Find the codeword polynomial for the data 1010. (show your steps)

Solution: Solve $c(x) = d(x)g(x)$ using the cyclic property

Step 1: write down the data as a 7-bit codeword

- data = {1010000}

Step 2: apply the cyclic property: shift the codeword based on the power of x

- $d(x) \cdot g(x) = 1010000 + 0101000 + 0001010 = 1110010$

Step 3: convert the codeword into a codeword polynomial

- $1110010 = 1 + x + x^2 + x^5$

Decoding



Suppose $g(x) = (1101)$ for a (7,4) cyclic code

Question: Bob receives a codeword 0110111 from Alice. Is there an error? If yes, justify why. (show your steps)

Decoding



Suppose $g(x) = (1101)$ for a (7,4) cyclic code

Question: Bob receives a codeword 0110111 from Alice. Is there an error? If yes, justify why. (show your steps)

Thought process: We have the values $g(x)$ and $c(x)$, asked whether there is a remainder in $c(x)/g(x)$. If yes, then there is an error.

- Solve $d(x) = c(x)/g(x)$
 - Step 1: convert $c(x)$ and $g(x)$ into polynomials
 - Step 2: calculate polynomial division $c(x)/g(x)$, check for remainder

Decoding



Suppose $g(x) = (1101)$ for a (7,4) cyclic code

Question: Bob receives a codeword 0110111 from Alice. Is there an error? If yes, justify why. (show your steps)

Solution: Solve $d(x) = c(x)/g(x)$

Step 1: convert $c(x)$ and $g(x)$ into polynomials

- $g(x) = 1 + x + x^3$
- $c(x) = x + x^2 + x^4 + x^5 + x^6$

Decoding



Solution (cont.):

Step 2: calculate polynomial division $c(x)/g(x)$, check for remainder

- $(x + x^2 + x^4 + x^5 + x^6)/(1 + x + x^3) = x^2 + x^3$ remainder x

$$\begin{array}{r} x^6 + x^5 + x^4 + x^2 + x \\ \underline{-} x^6 + x^4 + x^3 \\ \hline x^5 + x^3 + x^2 + x \\ \underline{-} x^5 + x^3 + x^2 \\ \hline x \end{array} \quad \left| \begin{array}{r} x^3 + x + 1 \\ \hline x^3 + x^2 \end{array} \right.$$

There is a remainder, so
the error is detected.

Byzantine fault tolerance

- Lecture 11 and 12: Byzantine fault tolerance
 - Timing failure, omission failure, crash failure, Byzantine failure
 - Fail-stop, fail-noisy, fail-silent, fail-safe, fail-arbitrary
 - The Two Generals Problem
 - The Byzantine Generals Problem [Explanation]
 - E.g., $m = 1, n = 3$, is it solvable? Why?

System security

- Lecture 13 and 14: CIA triad and digital signature
 - Confidentiality, integrity and availability
 - Hash function
 - Hash collision [\[Explanation\]](#)
 - Digital signature [\[Explanation\]](#)
 - Collision attack
 - SHA256

System security

- Lecture 15: Authentication
 - Authentication methods
 - Password-based, magic links, SMS-based, authenticator apps, biometric authentication
 - Time-based one-time password [\[Explanation\]](#)
 - HMAC-based one-time password [\[Explanation\]](#)
 - Multi-factor authentication

System security

- Lecture 16: Access control
 - Threats, vulnerabilities, attacks
 - Access control lists
 - Models of access controls [Explanation]
 - Discretionary access control (DAC)
 - E.g., what, who, benefits and problems
 - Role-based access control (RBAC)
 - ...
 - Mandatory access control (MAC)
 - ...
 - Attribute-based access control (ABAC)
 - ...

System security

- Lecture 17: Encryption
 - Symmetric encryption [Explanation]
 - Man-in-the-middle attack [Explanation]
 - Asymmetric encryption
 - RSA algorithm [Problem]

RSA algorithm



Alice wants to send a message ($m = 3$) to Bob through the RSA algorithm. Assume that the two prime numbers used to generate the keys are $p = 13$, $q = 7$, and Alice must choose a value $e < 10$.

Question: What is the ciphertext? (show your calculations and assumptions)

Encryption key (e, n)

- $m^e \bmod(n) = c$

RSA algorithm



Alice wants to send a message ($m = 3$) to Bob through the RSA algorithm. Assume that the two prime numbers used to generate the keys are $p = 13$, $q = 7$, and Alice must choose a value $e < 10$.

Question: What is the ciphertext? (show your calculations and assumptions)

Thought process: We have the values m , p , q , asked to calculate c

- We need to calculate the public key (e, n) to find c
 - Step 1: calculate n
 - Step 2: calculate $\varphi(n)$
 - Step 3: find e that satisfies the conditions
 - Step 4: calculate c with (e, n)

RSA algorithm



Solution: We need to calculate the public key (e, n) to find c

Step 1: calculate n

- If $p = 13, q = 7$, then $n = pq = 91$

Step 2: calculate $\phi(n)$

- $\phi(n) = (p-1)(q-1) = 12 \times 6 = 72$

RSA algorithm



Solution (cont.):

Step 3: find e that satisfies the conditions

- Condition: $1 < e < \varphi(n)$, and given that $e < 10$
 - $1 < e < 10$
- Condition: e and $\varphi(n)$ must be coprime
 - e and 72 must be coprime
 - e cannot be a divisor of 72 including 2, 3, 4, 6, 8, 9
- e can either be 5 or 7
- Public key: (5, 91) or (7, 91)

RSA algorithm



Solution (cont.):

Step 4: calculate c with (e, n)

If e = 5

- $m^e \bmod(n) = c$
- $3^5 \bmod 91 = 243 \bmod 91 = 61$

If e = 7

- $3^7 \bmod 91 = 2187 \bmod 91 = 3$

Encryption key (e, n)

- $m^e \bmod(n) = c$

RSA algorithm



Part I: Bob's public and private key setup

- Chooses two prime numbers, p and q
- Calculate the product $n = pq$
- Solve $\varphi(n) = (p-1)(q-1)$
- Choose numbers e and d so that ed has a remainder of 1 when divided by $\varphi(n)$
 - $1 < e < \varphi(n)$, where e must be an integer
 - e and $\varphi(n)$ must be coprime
 - $e^*d \pmod{\varphi(n)} = 1$

Example

- $p = 11, q = 3$
- $n = pq = 33$
- $\varphi(n) = 10 \times 2 = 20$
- Pick e and d so that $ed = 20+1$
 - e.g.,: $e = 3, d = 7$
 - $1 < 3 < 20$
 - 3 and 7 are coprime

RSA algorithm



Assume that the values of p and q are larger than 5, and the value of n is less than 100.

Question: Alice sends a ciphertext ($c = 3$) using the public key (5, 91). How can Eve break the ciphertext? (show your calculations and assumptions)

Decryption key (d, n)

- $c^d \text{ mod}(n) = m$

This question also shows why we should use larger prime numbers.

RSA algorithm



Assume that the values of p and q are larger than 5, and the value of n is less than 100.

Question: Alice sends a ciphertext ($c = 3$) using the public key $(5, 91)$. How can Eve break the ciphertext? (show your calculations and assumptions)

Thought process: We have the values c , e , n , asked to calculate m

- We need to calculate the private key (d, n) to find m
 - Step 1: find p and q
 - Step 2: calculate $\varphi(n)$
 - Step 3: find d that satisfies the condition
 - Step 4: calculate m with (d, n)

RSA algorithm



Solution:

Step 1: find p and q

- $p > 5$ and $q > 5$
- If $n = 91$, then $p = 7$, $q = 13$
 - Trial and error, also the only two prime numbers whose product is 91

Step 2: calculate $\varphi(n)$

- $\varphi(n) = (p-1)(q-1) = 6 \times 12 = 72$

RSA algorithm



Solution (cont.):

Step 3: find d that satisfies the conditions

- Condition: $e \cdot d \pmod{\varphi(n)} = 1$
 - Find a value of d so that $5d \pmod{72} = 1$ holds
 - Trial and error is enough to solve this
 - $(72+1) / 5$ not an integer
 - $(144+1) / 5 = 29$
 - $29 < 100$
- $d = 29$
- Private key: $(29, 91)$

RSA algorithm



Solution (cont.):

Step 4: calculate m with (d, n)

- $c^d \text{ mod}(n) = m$
- $61^{29} \text{ mod } 91 = 3$
 - If your calculator does not support the modulus operator
 - Step 1) Break down the exponent, e.g., $29 = 5 + 5 + 5 + 5 + 5 + 4$
 - Step 2) Solve for $(61^5)^5 \text{ (mod } 91) * 61^4 \text{ (mod } 91)$
 - $61^5 \text{ mod } 91 = 3$
 - $61^4 \text{ mod } 91 = 9$
 - Step 3) $3^5 * 9 \text{ (mod } 91) = 3$

Decryption key (d, n)

- $c^d \text{ mod}(n) = m$

ECE 422 – Midterm exam
Winter 2024
Monday, February 26, 2024
Duration: 45 minutes

An Ran Chen
anran6@ualberta.ca

University of Alberta
Department of Electrical and Computer Engineering

Instructions

- Answer all questions on these sheets in the space provided.
- No books, notes or extra paper.
- No cell phones, laptops or any electronic devices except approved non-programmable calculators.
- This exam is 9 pages long, including the cover page. It has 22 questions labeled from question 1 to question 22. Check that your copy is complete.
- This exam is graded on 25 points.
- This exam counts for 25% of your final grade.

Consistent with the university regulations concerning cheating and plagiarism
I will not cheat during this examination:

Student ID: _____

First Name / Last Name: _____

Signature: _____

Version A

I. True/False questions [3 points]

Question 1 [0.5 pts]

SRE is an implementation of the practices that DevOps describes.

- True
- False

Question 2 [0.5 pts]

SLI is an agreement within an SLA about a specific metric like uptime or response time.

- True
- False, SLI is the actual performance metric.

Question 3 [0.5 pts]

An example of crash failure describes the situation where the server has no response to incoming requests.

- True
- False, crash failure implies when the server has valid responses until it crashes.

Question 4 [0.5 pts]

Fail-noisy failures describe halting failures that are eventually and reliably detected.

- True
- False

Question 5 [0.5 pts]

N-version programming runs individual acceptance tests for each version of the system, and returns the majority agreement as output.

- True
- False, N-version programming executes N versions in parallel, and returns the majority agreement as output.

Question 6 [0.5 pts]

In an HMAC-based one-time password implementation, the server synchronizes the counter variable with the application before each passcode generation.

- True
- False, the counter variable synchronization happens after the passcode has been verified.

II. Multiple Choice Questions [4 points + 0.5 bonus point]

Please encircle the right choice for the following questions.

Question 7 [0.5 point]

What does the CMD instruction do in Dockerfile?

- a. Execute build commands
- b. **Specify default commands**
- c. Create a new stage from a base image
- d. Change working directory

Question 8 [0.5 point]

Which process **prioritizes** on delivering changes through rigorous automated testing in the staging phase?

- a. Continuous deployment
- b. Continuous integration
- c. **Continuous delivery**
- d. Continuous testing

Question 9 [0.5 point]

The course instructor creates hashes of student project files in order to monitor whether information has been tampered with the ones submitted through GitHub. The above scenario verifies what property of the project?

- a. Availability
- b. Confidentiality
- c. **Integrity**
- d. Irreversibility
- e. Redundancy

Question 10 [0.5 point]

On the trade-offs between reliability and security, a redundant solution makes the system fail _____. It provides alternative solutions in case of failure. However, it increases the attack surface.

- a. **safely**
- b. redundantly
- c. securely
- d. reliably

Version A

Question 11 [0.5 point]

Which one of the following helps upgrade the security of a current authenticator apps authentication system to a multi-factor authentication system? (select all possible choices)

- a. Access card - possession
- b. PIN code - knowledge
- c. Fingerprint - biological
- d. IP address - location
- e. All of the above

Question 12 [0.5 point]

Which of the following statements is the most likely to be suspicious based on Spectrum-based Fault Localization?

	T ₁	T ₂	T ₃
S ₁	✓		
S ₂		✓	✓
S ₃	✓		
S ₄	✓	✓	✓
Result	P	F	F

$$Ochiai(element) = \frac{e_f}{\sqrt{(e_f + n_f) \cdot (e_f + e_p)}}$$

- a. S₁
- b. S₂
- c. S₃
- d. S₄

Version A

Question 13 [0.5 point]

Given that 0001011 is a codeword in (7, 4) Hamming code, which of the following cannot be the valid codeword in the codespace? (Hint: a (7, 4) Hamming code can correct and detect any single-bit error.)

- a. 0011101
- b. 0101100
- c. 0011010, a code distance = 2
- d. 1110100

Question 14 [0.5 point]

Consider the following statements, which ones are true?

- 1. 100% statement coverage guarantees 100% branch coverage.
 - 2. 100% branch coverage guarantees 100% statement coverage.
 - 3. 100% path coverage guarantees 100% statement coverage.
 - 4. 100% path coverage guarantees 100% branch coverage.
 - 5. 100% statement coverage guarantees 100% path coverage.
-
- a. Statement 2
 - b. Statements 1 and 5
 - c. Statements 2 and 3
 - d. Statements 2, 3, and 4, 100% branch coverage implies 100% statement coverage. 100% path coverage implies 100% branch and statement coverage. Therefore, statements 2, 3, and 4 are true.

Question 15: Bonus question [0.5 point]

What is the probability that at least two people have the same birthday in our classroom (Hint: we are a class of 78 people)?

- a. 12%
- b. 50%
- c. 97%
- d. 99%

Version A

III. Short Answer Questions [8 points]

Question 16 [2 points]

a) Given a (255, 247) Hamming code, how many parity bits does it have?

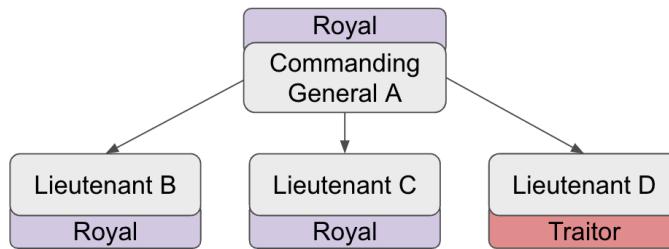
Answer: 8 parity bits

b) What is its information rate? (Hint: information rate = k/n)

Answer: 97% information rate

Question 17 [3 points]

Given the following Byzantine general problem where the Commanding General A sends an “attack” message to others.



a) Will Lieutenant B and C attack? Show their decision making process (e.g., $V(D) = (R, R, A)$ where A denotes attack, R denotes retreat).

Answer: Both general B and C will attack at the same time.

$V(B) = (A, A, R)$

$V(C) = (A, A, R)$

b) Is consistency satisfied? If yes, why? If not, why not?

Answer: Yes, B and C executed the same order.

c) Is validity satisfied? If yes, why? If not, why not?

Answer: Yes, B and C obeyed the order from A.

Version A

Question 18 [3 points]

a) Name the most secure access control model.

Answer: Mandatory access control (MAC)

b) Who controls the resources?

Answer: Administrator

c) Give a brief description of how the resources are protected (i.e., how users gain access).

Answer: Users can only access the resource when their security labels entitles the access

III. Computational Questions [10 points]

Question 19 [2 points]

Alice wants to send a message ($m = 5$) to Bob through the RSA algorithm.

Assume that the two prime numbers used to generate the keys are $p = 5$, $q = 11$, and Alice must choose a value $e < 6$. What is the ciphertext? (show your calculations and assumptions)

Hint:

- $m^e \text{ mod}(n) = c$
- $c^d \text{ mod}(n) = m$

Answer: The ciphertext is 15.

With RSA, we perform the following calculations:

$$n = pq = 55 \quad (0.5 \text{ pts})$$

$$\phi(n) = (p - 1)(q - 1) = 4 \times 10 = 40 \quad (0.5 \text{ pts})$$

Condition: $1 < e < \phi(n)$, and given that $e < 6$. Therefore, $1 < e < 6$

Condition: e and $\phi(n)$ must be coprime. Therefore, e cannot be a divisor of 40 including 2, 4, 5

Since e must be less than 6, it must be 3.

$$\begin{aligned} c &= m^e \text{ mod } (n) \\ &= 5^3 \text{ mod } (55) \\ &= 15 \end{aligned} \quad (1 \text{ pt})$$

Version A

Question 20 [3 points]

In Question 19, Alice used the RSA algorithm to send the message ($m = 5$) to Bob so that others could not read the message. Suppose that you have been listening to their communication channel:

- a) Do you know Alice's public key? If yes, what is it? (1~2 sentences) (1 pt)

Answer: No, the public and private key needed for encryption belong to Bob.
Nothing is known about Alice's public key.

- b) Do you know Bob's public key? If yes, what is it? (1~2 sentences) (1 pt)

Answer: Yes, Bob's public key is {3, 55}.

- c) If Bob uses a digital signature, then what is its purpose? (i.e., what does the digital signature do?) (1~2 sentences) (1 pt)

Answer: The digital signature certifies that the public key belongs to Bob.

Version A

Question 21 [2 points]

Suppose $g(x) = 1101$ for a $(7, 4)$ cyclic code. Bob receives a codeword 0010111 from Alice. Is there an error? If yes, justify why. If not, what is the original data? (show your steps)

Answer: No error, because remainder is null. The original data is 0011 (or 0011000).

$$c(x) = x^2 + x^4 + x^5 + x^6 \quad (0.5 \text{ pts})$$

$$g(x) = 1 + x + x^3 \quad (0.5 \text{ pts})$$

$$d(x) = c(x)/g(x) = x^2 + x^3 \quad (1 \text{ pt})$$

Question 22 [3 points]

Calculate the final code after encoding the code 11110100100 into 16-bit even parity extended Hamming code. (show your steps, e.g., $P_1: \{0000001\}$, odd parity, $P_1 = 1$) (Hint: $1\ 111\ 010\ 0100$)

Answer: $0001\ 0111\ 0010\ 0100$

$$P_1: \{1110010\}, \text{ even parity, } P_1 = 0 \quad (0.6 \text{ pts})$$

$$P_2: \{1111000\}, \text{ even parity, } P_2 = 0 \quad (0.6 \text{ pts})$$

$$P_3: \{1110100\}, \text{ even parity, } P_3 = 0 \quad (0.6 \text{ pts})$$

$$P_4: \{0100100\}, \text{ even parity, } P_4 = 0 \quad (0.6 \text{ pts})$$

$$P_5: \{001011100100100\}, \text{ even parity, } P_5 = 0 \quad (0.6 \text{ pts})$$

Review Session I

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- Review session I
 - Lecture 17 Encryption - Part I
 - Lecture 19 Encryption - Part II
 - Lecture 20 The Dining Philosophers Problem
 - Lecture 21 Deadlocks
 - Lecture 23 Cookies and Sessions
- SPOT Survey reminder

Format of exam

- Classroom: The Centennial Centre for Interdisciplinary Science (CCIS) 1-160
- Date: Wednesday, Apr 24th, 2024
- Duration: 2 hours
- Closed book
- Materials: all materials posted in the lecture slides
 - => 60% on new materials, < 40% on old materials
- The final counts for 30% of the overall course grade

Email me if you need to know your grade early for work permit or graduation
(estimated **early grade release date**: ~ May 1).

Type of question

- True/False
- Multiple choice
- Short answer
- **Essay questions**
 - Explain and give examples
- Computational questions
 - Bring your own calculator

List of materials to prepare you

- Go through the concepts in the (midterm + final) review slides
- Go through the questions in the lecture slides

Additional information:

- Materials from the past years: [Google Drive link](#)
 - [ECE422 W2023 Security in Computing 3 Programs and Programming](#)
 - [ECE422 W2023 Security in Computing 4 The Web-User Side](#)
 - [ECE422 W2023 Security in Computing 6 Networks](#)
 - [ECE422 W2023 Security in Computing 7 Database](#)

Course registration system



User story: As a security admin, I want to ensure the integrity of the data.

- **Lecture 13:** The CIA Triad
- **Lecture 14:** Hash function and Digital Signature
 - Hash collision on integrity checks

Key concepts:

- Digital Signature: Authenticity + Integrity + Non-repudiation
- Difference between hashing and encryption: irreversibility

Course registration system



User story: As a web admin, I want to prevent race condition for course registration.

- **Lecture 20:** The Dining Philosophers Problem
 - Race condition
 - Atomic locking

Key concepts:

- Locks to prevent multiple users from registering the courses at the same time
- Atomic locks for course reservation

Course registration system



User story: As a security admin, I want to ensure the confidentiality of the system.

- **Lecture 15:** Authentication
 - Multi-factor authentication
- **Lecture 16:** Access Control
 - Models of access control
- **Lecture 17:** Encryption
 - Symmetric and asymmetric encryption

Key concepts:

- MFA: knowledge, possession, biologic, location and time
- DAC, RBAC, MAC, ABAC
- Asymmetric: encryption = sender's private key; decryption = public key

Essay question



Given the following user story written by a portfolio manager or business analyst:

- As a security admin, I want to ensure the confidentiality of the system.

Question: As software developers, how can we implement such a user story?

- Explain confidentiality
- Name an example of technique to ensure confidentiality
- Describe how the technique works
- Explain how it protects the confidentiality

Essay question



Given the following user story written by a portfolio manager or business analyst:

- As a security admin, I want to ensure the confidentiality of the system.

Potential answer:

- **Confidentiality** is about making sure that only the authorized user has access to particular resources.
- An example of such technique is **access control**.
- We restrict user access to the system through different **models of access control**.
- For example, **discretionary access control** is an identity-based access control model that enforces a security policy based on the identity of the user.

Schedule for today

- Review session I
 - Lecture 17 Encryption - Part I
 - Lecture 19 Encryption - Part II
 - Lecture 20 The Dining Philosophers Problem
 - Lecture 21 Deadlocks
 - Lecture 23 Cookies and Sessions
- SPOT Survey reminder

Security principles

- Lecture 17 Encryption - Part I
 - Rivest–Shamir–Adleman (RSA) algorithm [\[Problem\]](#)
- Lecture 19 Encryption - Part II
 - Diffie-Hellman Key Exchange [\[Problem\]](#)
 - Primitive roots [\[Problem\]](#)
 - Discrete logarithm problem [\[Explanation\]](#)

Question on DF algorithm



Suppose that Alice and Bob want to agree on a shared "secret key" with $p = 11$ and $g = 2$.

Question: If Alice chooses 9 and Bob chooses 4 as their respective secret integers, what are their public integers and what is their secret key? (show the calculations for both Alice and Bob)

- $A = g^a \text{ mod}(p)$
- $B = g^b \text{ mod}(p)$
- $K = g^{ab} \text{ mod}(p) = A^b \text{ mod}(p)$
- $K = g^{ba} \text{ mod}(p) = B^a \text{ mod}(p)$

Question on DF algorithm

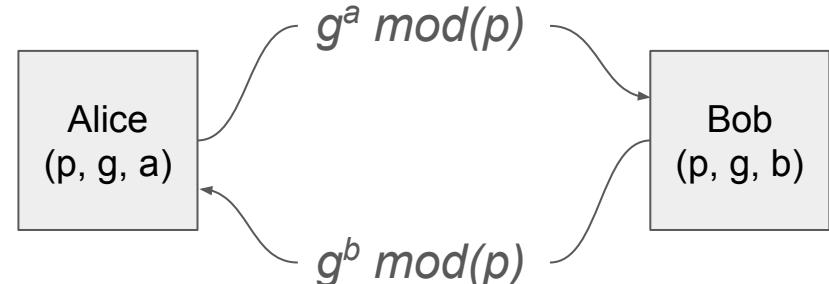


Suppose that Alice and Bob want to agree on a shared "secret key" with $p = 11$ and $g = 2$.

Question: If Alice chooses 9 and Bob chooses 4 as their respective secret integers, what are their public integers and what is their secret key? (show the calculations for both Alice and Bob)

Thought process: We have the values p , g , a and b , asked to calculate A , B , K

- We need to calculate A , B to find K
 - Step 1: calculate the public integers
 - Step 2: calculate the secret key
 - Bob uses $K = A^b \text{ mod}(p)$
 - Alice uses $K = B^a \text{ mod}(p)$



Question on DF algorithm



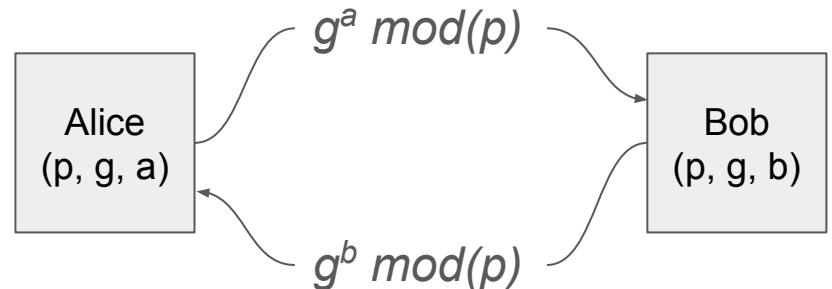
Solution: We have the values p, g, a and b, asked to calculate A, B, K

Step 1: calculate the public integers

- $A = g^a \text{ mod}(p) = 2^9 \text{ mod } (11) = 6$
- $B = g^b \text{ mod}(p) = 2^4 \text{ mod } (11) = 5$

Step 2: calculate the secret key

- For Bob, $K = A^b \text{ mod}(p) = 6^4 \text{ mod } (11) = 9$
- For Alice, $K = B^a \text{ mod}(p) = 5^9 \text{ mod } (11) = 9$



Question on DF algorithm



Suppose that Alice and Bob want to agree on a shared "secret key" with $p = 11$ and $g = 2$.

Question: If Alice chooses 9 and Bob chooses 4 as their respective secret integers, what are their public integers and what is their secret key? (show the calculations for both Alice and Bob)

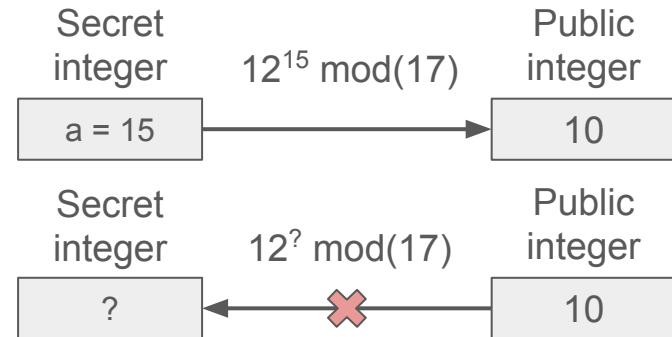
In real-life scenario, much larger values of a , b and p are required. An eavesdropper cannot discover the shared secret key even if she knows p and g and can obtain the public integers.

Discrete Logarithm Problem (DLP)



For Eve to find the secret integer, she needs to solve the discrete logarithm problem, however

- It is easy to calculate the public integer
- Hard to find the secret integer (many solutions)



The idea is similar to the shared secret color solution:

- Easy to mix a secret color with the starting one to get a mixture
- Hard to find the secret color from the mixed color (many color combinations)

Primitive roots



Question: Is 3 a primitive root of prime number 7?

Answer: Yes, 3 is a primitive root of 7.

- For every integer coprime to 7, there is a power of 3 that is congruent.
- Integers that are coprimes to 7: 1, 2, 3, 4, 5, 6
- $3^1 \text{ mod}(7) = 3$
- $3^2 \text{ mod}(7) = 2$
- $3^3 \text{ mod}(7) = 6$
- $3^4 \text{ mod}(7) = 4$
- $3^5 \text{ mod}(7) = 5$
- $3^6 \text{ mod}(7) = 1$
- Verified: For 1, 2, 3, 4, 5, 6, there is a power of 3 that is congruent

Reliable design

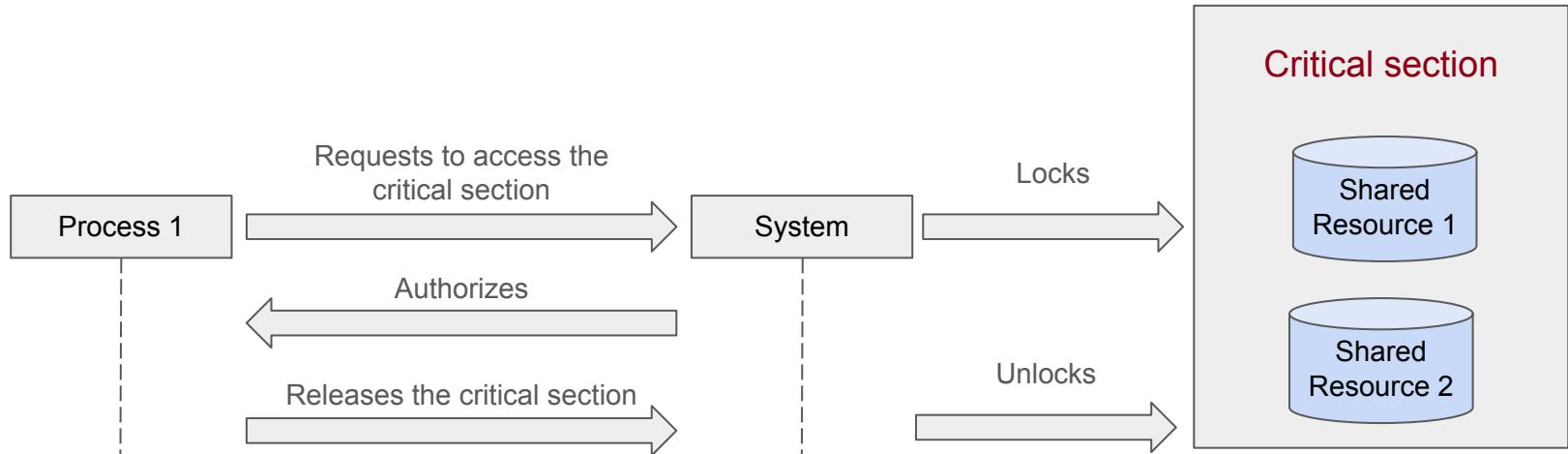
- Lecture 20 The Dining Philosophers Problem
 - Deadlocks vs starvation
 - Deadlock prevention (mutual exclusion, hold and wait, no preemption, circular wait)
 - Prevent circular wait: atomic locks on critical section [\[Explanation\]](#)
 - Locking mechanism: Race condition
 - Atomic property: Deadlocks
- Lecture 21 Deadlocks
 - Deadlock avoidance: Resource Allocation Graph (RAG) [\[Problem\]](#)
 - Banker's Algorithm = Resource Allocation Table

Solution: atomic locking



Introducing atomic locks to the resources:

- Ensure that the resource is accessed by only one process at a time
- If the lock is free, the system will allow the process to access the resources and lock the critical section

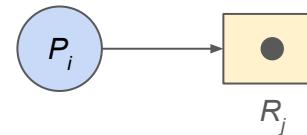


Resource Allocation Graph (RAG)

- P_i , Process



- P_i requests instance of R_j



- R_j , Resource with instances

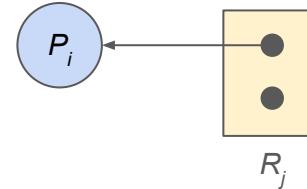
- Resource with a single instance



- Resource with 4 instances



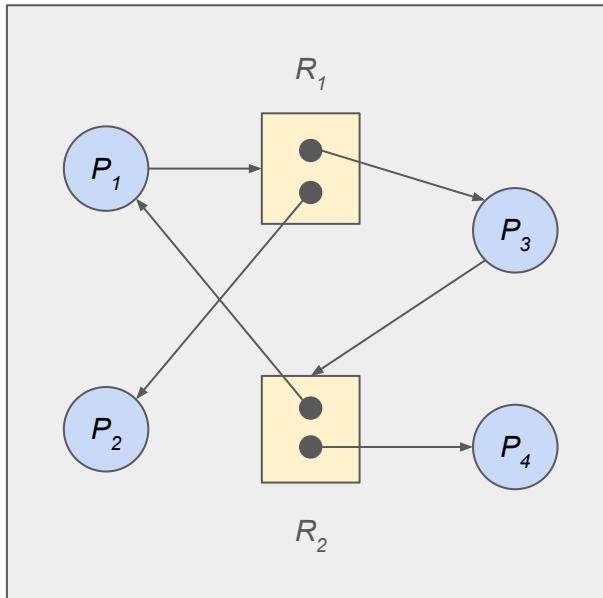
- P_i is holding an instance of R_j



Question on RAG



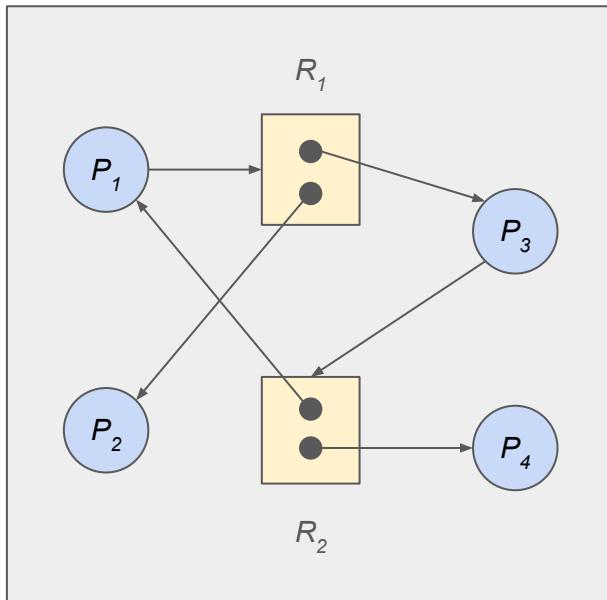
Question: Based on the following RAG, is there a deadlock?



Question on RAG



Question: Based on the following RAG, is there a deadlock?



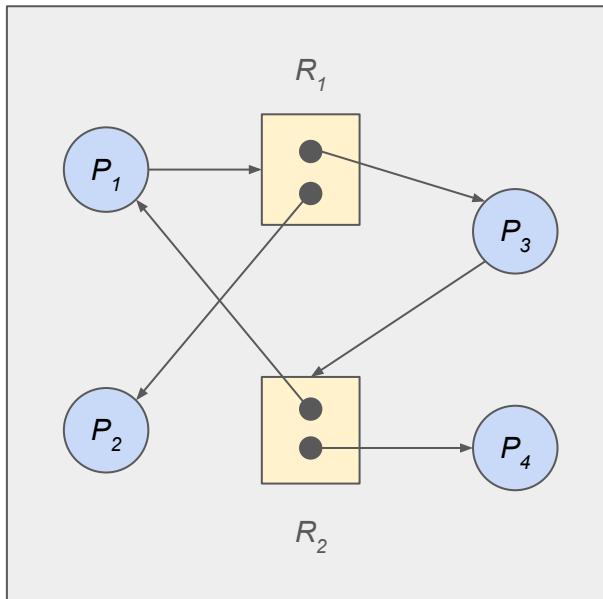
Thought process: Perform two checks. If neither of them indicates there is a deadlock, then solve the resource allocation table.

- Cycle check
 - If none, then no deadlock
- Single instance check
 - If only one instance per resource, then deadlock
- Solve the resource allocation table

Question on RAG



Question: Based on the following RAG, is there a deadlock?



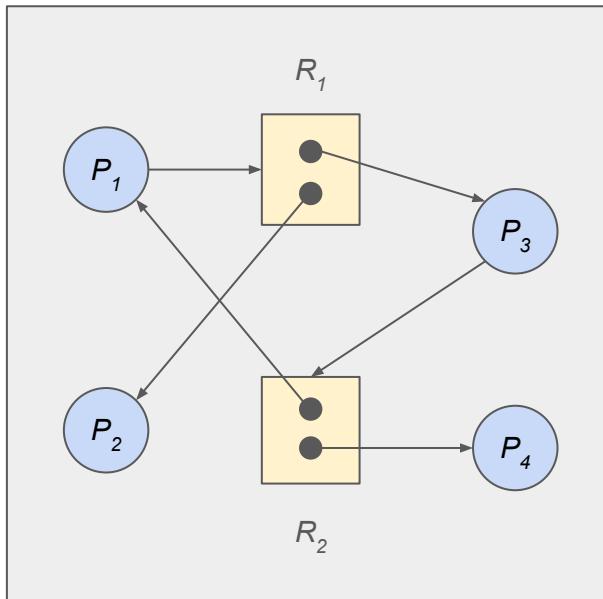
Step 1: Perform two checks

- Cycle check
 - Yes, $P_1 \rightarrow R_1 \rightarrow P_3 \rightarrow R_2 \rightarrow P_1$
- Single instance check
 - R_1 and R_2 : more than one instance
- Solve the resource allocation table

Question on RAG



Question: Based on the following RAG, is there a deadlock?



Step 2: Solve the resource allocation table

- Fill in the allocated/requested resources

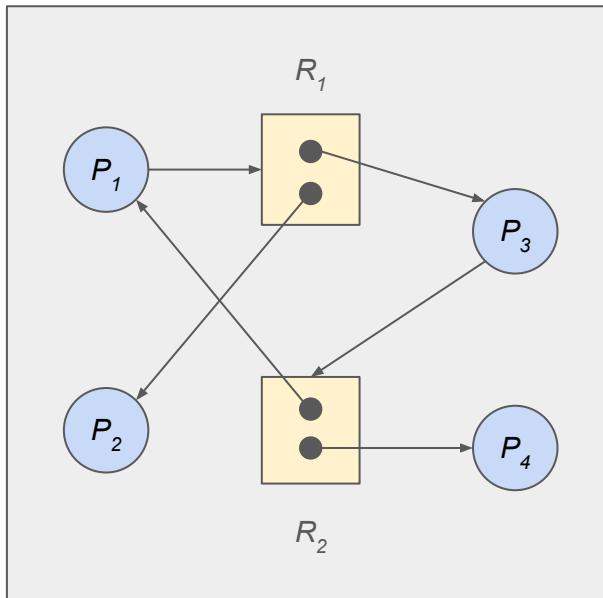
	Allocated		Requested	
	R1	R2	R1	R2
P1				
P2				
P3				
P4				

Hint: allocated = $R \rightarrow P$; requested = $P \rightarrow R$

Question on RAG



Question: Based on the following RAG, is there a deadlock?



Step 2: Solve the resource allocation table

- Fill in the allocated/requested resources

	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	1	0
P2	1	0	0	0
P3	1	0	0	1
P4	0	1	0	0

Hint: allocated = R → P; requested = P → R

Question on RAG



Question: Based on the following RAG, is there a deadlock?

	Allocated		Requested	
	R1	R2	R1	R2
P1	0	1	1	0
P2	1	0	0	0
P3	1	0	0	1
P4	0	1	0	0

Step 3: Free the requested resources

- Availability = $(R1, R2) = (0, 0)$
- Availability = $(1, 0)$ after P2
- Availability = $(1, 1)$ after P4
- **Availability = $(1, 1)$ after P3**

Cycle with no deadlock

Question on RAG



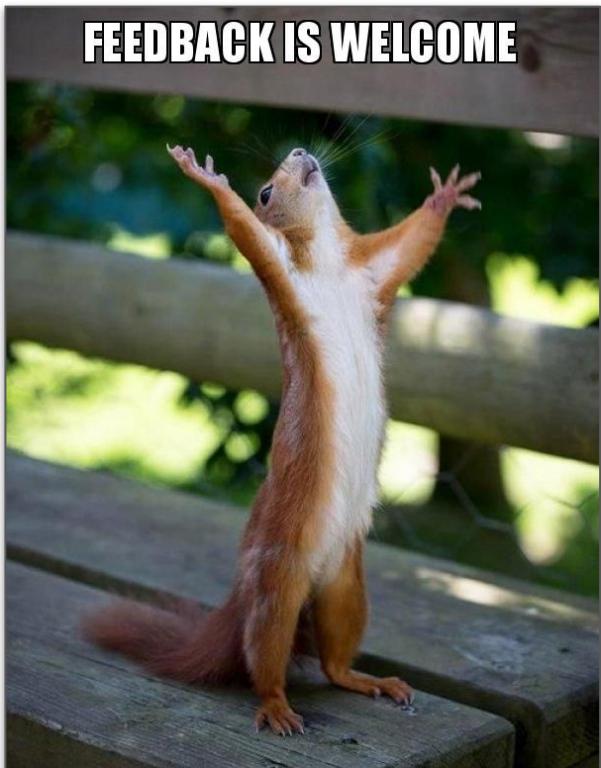
Question: Based on the following resource allocation, is there a deadlock (draw RAG)?

- Suppose that there are two resources (R1 and R2), each with two instances
- P1 allocates an instance of R2, and requests an instance of R1
- P2 allocates an instance of R1
- P3 allocates an instance of R1 and requests an instance of R2
- P4 allocates an instance of R2

Security principles

- Lecture 23 Cookies and Sessions
 - Ambient authority (cookies, IP checking, certificates, basic authentication)
 - Session cookies
 - Cookie protections
 - Signing cookies [Explanation]
 - Session fixation attack
 - Cookie attributes [Explanation]

SPOT Survey



SPOT Survey: [Click here](#)

- 15 minutes
- Course number: 15754
- Survey close: Apr 14, 11:59 PM

If we can just get 5 more students completing the survey, I will have the final exam graded within a week (May 1st) and [...]

Review Session II

ECE 422: Reliable and Secure Systems Design



Instructor: An Ran Chen
Term: 2024 Winter

Schedule for today

- **Review session II**
 - Lecture 24 Cross Site Scripting
 - Lecture 25 Cross Site Scripting Prevention
 - Lecture 26 Content Security Policy
 - Lecture 27 CSP nonce and strict-dynamic
 - Lecture 28 Phishing and Denial-of-Service
 - Lecture 29 Blocks and blockchain
 - Lecture 30 Mining Principles
 - Lecture 31 Digital Signature & Double Spending Problem
- Google CTF: Pasteurize Web

Security principles

- Lecture 23 Cookies and Sessions
 - Ambient authority (cookies, IP checking, certificates, basic authentication)
 - Session cookies
 - Cookie protections
 - Signing cookies [Explanation]
 - Session fixation attack
 - Cookie attributes [Explanation]

Question on cookie attributes



Suppose that we build our course website on <https://ualberta.ca/course/ECE422>. We implement our authentication system by sending a response with a Set-Cookie HTTP header to set a **sessionId** cookie in the user's browser (e.g., "Set-Cookie: sessionId=1234;").

Question: Alice proposes to specify the Path attribute. Explain how it protects the website (e.g., "Set-Cookie: sessionId=1234; Path=/ECE422")?

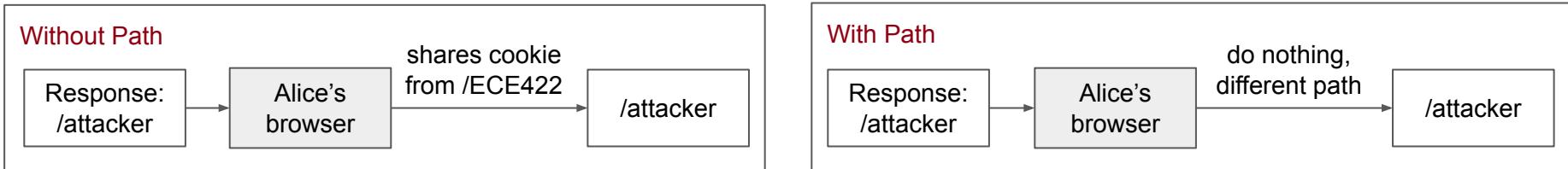
Question on cookie attributes



Question: Alice proposes to specify the Path attribute. Explain how it protects the website (e.g., "Set-Cookie: sessionId=1234; Path=/ECE422")?

Answer: The cookie is **scoped to the path prefix /ECE422**.

- This implies that the cookie will be sent when the user visits <https://ualberta.ca/course/ECE422> or <https://ualberta.ca/course/ECE422/lectures>
- But not when they visit <https://ualberta.ca/course/attacker>
- **Preventing cookie sharing (session fixation attack)**



Security principles

- Lecture 24 Cross Site Scripting
 - Cross-Site Request Forgery (CSRF) [\[Explanation\]](#)
 - Same Origin Policy (SOP) [\[Explanation\]](#)
 - Origin matching
 - Reflected vs Stored Cross-site scripting (XSS) [\[Explanation\]](#)

Question on web security



Question: "Mixing program control and user data" happens where an application accidentally treats user input as code and executes it. Which of the following attacks exploit this vulnerability? (Select all that apply)

- A. CSRF
- B. Stored XSS
- C. Reflected XSS
- D. IDN homograph attack
- E. All of the above

Question on web security



Question: "Mixing program control and user data" happens where an application accidentally treats user input as code and executes it. Which of the following attacks exploit this vulnerability? (Select all that apply)

- A. CSRF, exploit trusts from authenticated users
- B. Stored XSS, replace data with code, stored in database
- C. Reflected XSS, replace data with code, reflected as part of the response
- D. IDN homograph attack, phishing with a website that looks alike
- E. All of the above

Security principles

- Lecture 25 Cross Site Scripting Prevention
 - HttpOnly flag [Explanation]
 - HTML Escaping [Explanation]
- Lecture 26 Content Security Policy
 - CSP vs SOP: Whitelisting vs blocking data interaction
 - CSP directives [Problem]

Question on CSP



Suppose that an attacker injects a XSS payload into a HTML page sent by our web server:

```
<script>alert(document.cookie)</script>
```

Question: Given the following CSP, would the XSS attack succeed? Justify your answer.

Content-Security-Policy: default-src 'self';

Question on CSP



Suppose that an attacker injects a XSS payload into a HTML page sent by our web server:

```
<script>alert(document.cookie)</script>
```

Question: Given the following CSP, would the XSS attack succeed? Justify your answer.

```
Content-Security-Policy: default-src 'self';
```

Answer: No, because CSP checks the origin of the script. The default-src 'self' directive **only allows the script loaded from the same origin**. The above script is an **inline script** which is blocked.

Question on CSP



Given a CSP:

Content-Security-Policy: default-src 'self'; script-src 'self'; img-src 'self' https://img.example.com;

Question: Which of the following resources will be **blocked** (Select all that apply and read code carefully)?

- A. Resource A (link)
- B. Resource B (script)
- C. Resource C (img)

```
<!doctype html>
<html lang='en'>
  <head>
    <link rel='stylesheet' href='/style.css' />
  </head>
  <body>
    <script>alert('Welcome to ECE 422!')</script>
    <h1>Honorable mention:</h1>
    <img src='https://img.example.org/cat.jpg'>
  </body>
</html>
```

(A) (B) (C)

Question on CSP



Given a CSP:

Content-Security-Policy: default-src 'self'; script-src 'self'; img-src 'self' https://img.example.com;

Question: Which of the following resources will be **blocked** (Select all that apply and read code carefully)?

- A. Resource A (link)
- B. Resource B (script)
- C. Resource C (img)

```
<!doctype html>
<html lang='en'>
  <head>
    <link rel='stylesheet' href='/style.css' />
  </head>
  <body>
    <script>alert('Welcome to ECE 422!')</script> Inline script (B)
    <h1>Honorable mention:</h1>
    <img src='https://img.example.org/cat.jpg'> Different origin (C)
  </body>
</html>
```

Common CSP directives and values



Definition of CSP: Content-Security-Policy: **directives 'value'**:

Directives	Example	Description
default-src	default-src ' self '	Default policy to allow any resources (JavaScript, Fonts, CSS, etc) from the site's own origin
img-src	img-src *	Allow images from anywhere (* as wildcard)
script-src	script-src ' self ' example.com	Allow scripts from its own origin and example.com

However, the **script-src** directive (when specified) **blocks inline scripts and event handling attributes**:

- Even if '**self**' is defined
- To allow this, use '**unsafe-inline**' in script-src, but it is bad
 - Same as not having CSP

Security principles

- Lecture 27 CSP nonce and strict-dynamic
 - Challenge: Nested scripts
 - CSP nonces [\[Explanation\]](#)
 - CSP strict-dynamic [\[Explanation\]](#)
 - Read-only mode

Question on CSP with nonce



Suppose that an attacker injects a XSS payload into a HTML page sent by our web server:

```
<script>alert(document.cookie)</script>
```

Question: Given the following CSP, would the XSS attack succeed? Justify your answer.

Content-Security-Policy: script-src 'self' 'nonce-6301901420';

Question on CSP with nonce



Suppose that an attacker injects a XSS payload into a HTML page sent by our web server:

```
<script>alert(document.cookie)</script>
```

Question: Given the following CSP, would the XSS attack succeed? Justify your answer.

Content-Security-Policy: script-src 'self' 'nonce-6301901420';

Answer: No, because CSP checks the origin of the script. The script-src 'self' directive **only allows the script loaded from the same origin, or any script with the specified nonce as attribute** (e.g., `<script nonce='6301901420'>`). The above script is an inline script without nonce which is blocked.

Another example of CSP nonces



External script without CSP nonce

Content-Security-Policy:

```
script-src: 'self' https://*.google-analytics.com
```

```
<script  
src='https://www.google-analytics.com/  
analytics.js'></script>
```

External script with CSP nonce

Content-Security-Policy:

```
script-src: 'self' 'nonce-rAnd0m'
```

```
<script  
src='https://www.google-analytics.com/  
analytics.js' nonce='rAnd0m'></script>
```

- Same idea applies to external scripts
- Adding the nonce attribute to the script tag provides another solution to add <https://www.google-analytics.com/analytics.js> to CSP

Security principles

- Lecture 28 Phishing and Denial-of-Service
 - Phishing
 - Typosquatting
 - IDN Homograph attack [\[Explanation\]](#)
 - Pharming
 - Denial-of-Service attack
 - Goal of UI attacks: Override browser defaults, scareware, and annoy the user
 - Why some of the “features” of the Annoying Site work? [\[Explanation\]](#)
 - Link to CSP and SOP

Security principles

- Lecture 29 Blocks and blockchain [Explanation]
 - Blocks and blockchain
 - Block rewards and Bitcoin halving
- Lecture 30 Mining Principles
 - Proof-of-Work (PoW) [Explanation]
 - Difficulty adjustment [Problem]
 - The Longest Chain Rule
 - The 51% Attack
 - Finding the special number
- Lecture 31 Digital Signature & Double Spending Problem

Question on difficulty adjustment



Suppose that there are 5,000 Bitcoin miners on the Bitcoin network and their total computational power can achieve 6×10^{18} hash checks per hour.

Question: What should be the value of n in difficulty adjustment?

- Hint: $2^{59} = 5.8 \times 10^{17}$

Question on difficulty adjustment



Suppose that there are 5,000 Bitcoin miners on the Bitcoin network and their total computational power can achieve 6×10^{18} hash checks per hour.

Question: What should be the value of n in difficulty adjustment?

Thought process: A block is created every 10 minutes.

- Look for the number of hash checks in 10 minutes
- Find the difficulty based on the closest number of hash checks

Question on difficulty adjustment



Suppose that there are 5,000 Bitcoin miners on the Bitcoin network and their total computational power can achieve 6×10^{18} hash checks per hour.

Question: What should be the value of n in difficulty adjustment?

Step 1: Look for the number of hash checks in 10 minutes

- 6×10^{18} hash checks per hour $\rightarrow 10^{18}$ hash checks in 10 minutes

Step 2: Find the difficulty based on the closest number of hash checks

- Probability ($n=59$): 1 out of $(2^{59}) = 1$ out of 5.8×10^{17}
- Probability ($n=60$): 1 out of $(2^{60}) = 1$ out of 1.1×10^{18}
- **Difficulty should be adjusted to $n = 60$**

Why Proof-of-Work works?

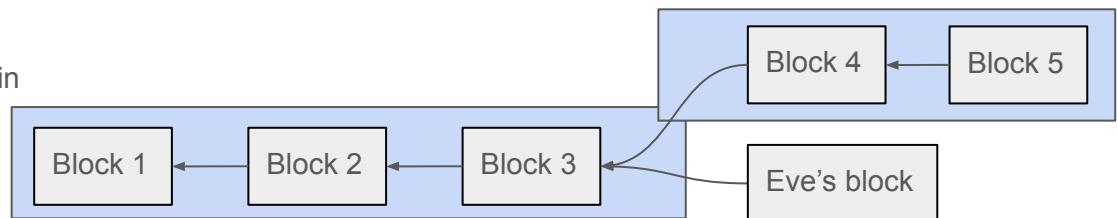


Suppose Eve tries to send a block with fraudulent transactions:

- Eve first needs to find the special number based on the fraudulent transactions before everyone else, and broadcasts the blockchain
- Bob verifies the blockchain and copies it over
- **However**, Bob continues to listen to the broadcast
 - Any longer blockchain will replace the current one
 - For Bob to keep Eve's blockchain, Eve needs to keep extending the blockchain



Holds and waits for longer blockchain

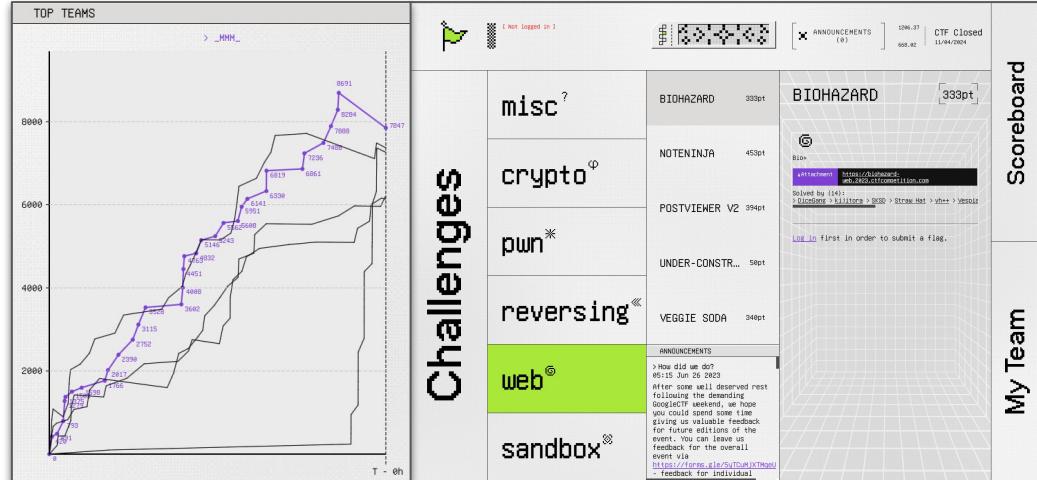


Schedule for today

- Review session II
 - Lecture 24: Cross Site Scripting
 - Lecture 25 Cross Site Scripting Prevention
 - Lecture 26 Content Security Policy
 - Lecture 27 CSP nonce and strict-dynamic
 - Lecture 28 Phishing and Denial-of-Service
 - Lecture 29 Blocks and blockchain
 - Lecture 30 Mining Principles
 - Lecture 31 Digital Signature & Double Spending Problem
- Google CTF: Pasteurize Web

Google CTF

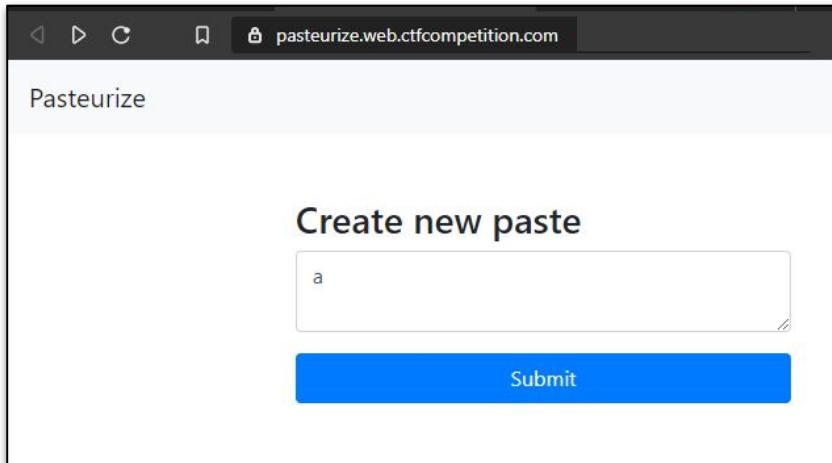
- Google CTF 2024: 21 June, 18:00 UTC — 23 June 2024, 18:00 UTC
 - Winners = greatest number of points earned
 - Points based on the number of teams that solved it
 - Vs Hackathon: time to sleep and eat
- Google CTF 2023
 - Beginners quest
- Past Google CTF challenges
 - From 2017 to 2023
 - Challenges and solutions



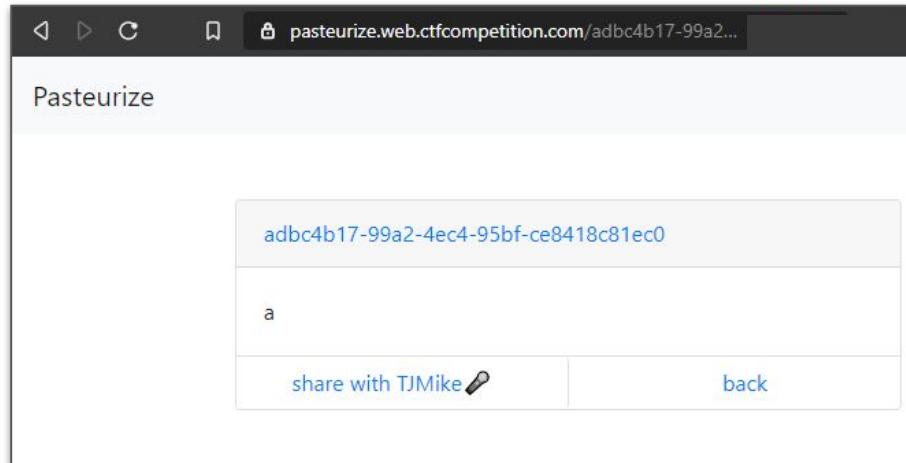
Google CTF: Pasteurize Web

XSS challenge in 2020: [Pasteurize web challenge + solution](#)

- Website (no longer available): create a new “Paste” and share it with a friend
- Difficulty: easy, 50 points



A screenshot of a web browser showing a simple form for creating a new paste. The URL in the address bar is `pasteurize.web.ctfcompetition.com`. The page title is "Pasteurize". The main heading is "Create new paste". Below it is a text input field containing the letter "a". At the bottom is a large blue "Submit" button.

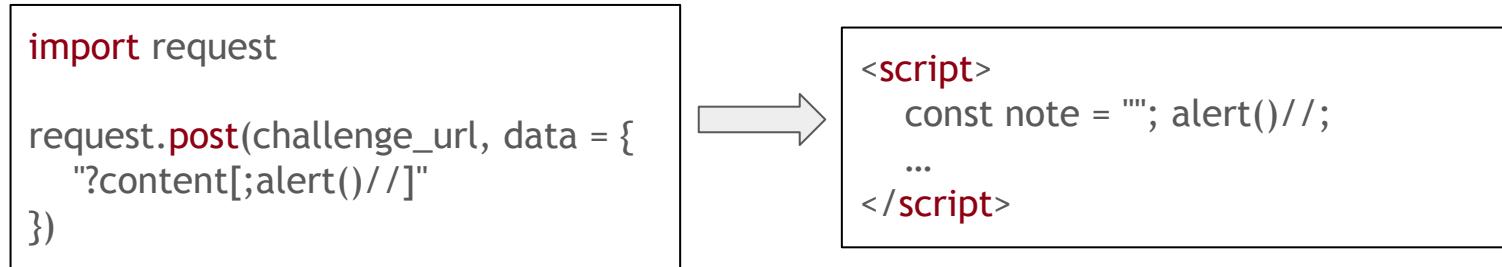


A screenshot of a web browser showing the result of pasting the character "a". The URL in the address bar is `pasteurize.web.ctfcompetition.com/adbc4b17-99a2-4ec4-95bf-ce8418c81ec0`. The page title is "Pasteurize". The pasted content is displayed in a text area. Below the text area are two buttons: "share with TJMike" with a link icon and "back".

Google CTF: Pasteurize Web

Solution: Bypass the DOM purification

- Send a POST request with "?content[;alert()//]" as a query parameter

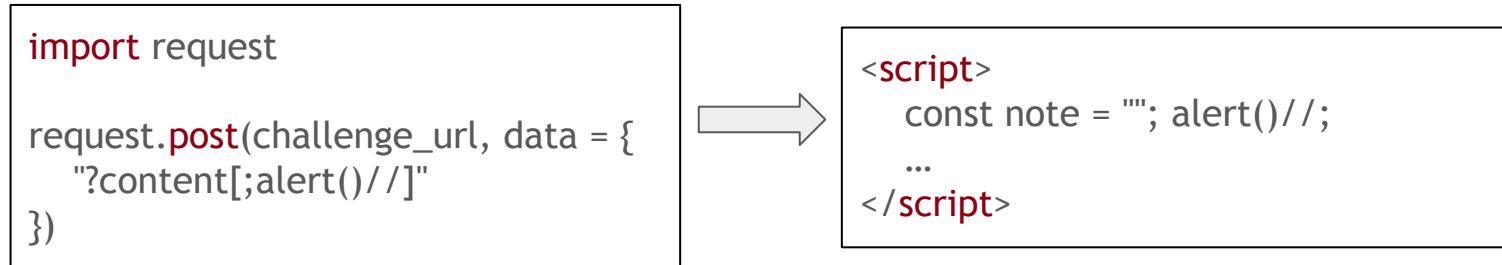


- Upgrade 1:** Use automated script to send the POST request with a list of known vulnerabilities (basic loop)
 - Example: Github repository [XSS payload list](#)
 - ~ 2,600 XSS payloads

Google CTF: Pasteurize Web

Solution: Bypass the DOM purification

- Send a POST request with "?content[;alert()//]" as a query parameter



- Upgrade 2:** Use Selenium to take a screenshot and compare HTML

- Challenge: in real-world websites, XSS often leads to some hints rather than the actual vulnerability itself (e.g., malformed HTML rather than the successful injection)
- In python, `driver.get_screenshot_as_file('site_xss_' + id + '.png')`

Be aware!



If you intend to participate in an upcoming CTF, remember to:

- Be prepared
 - Build a toolkit: automated scripts, static analysis scripts, list of vulnerabilities
 - Check the format: Jeopardy (challenges) vs Attack-Defense (defending vs attacking a host)
- Try to team up with people with different backgrounds
- Sleep well and eat!