

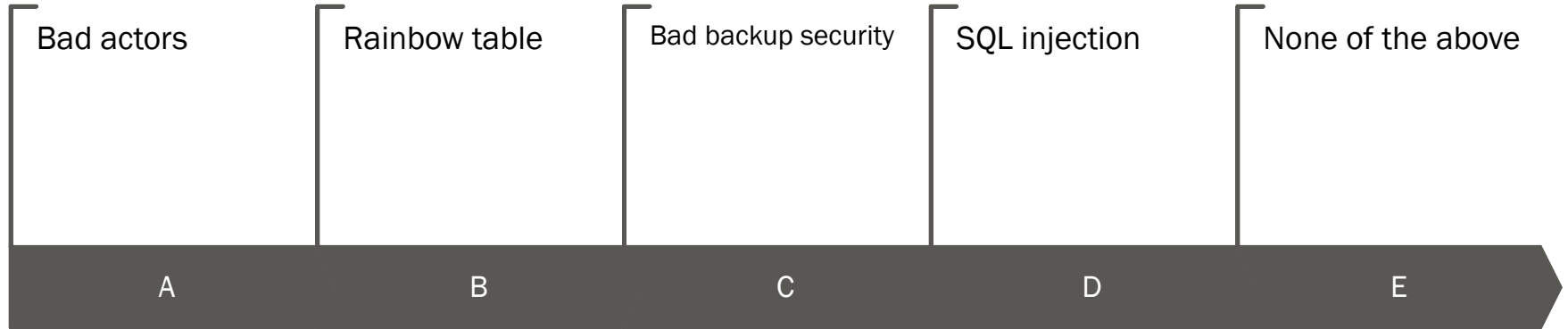


FULL-STACK NANODEGREE SESSION 9

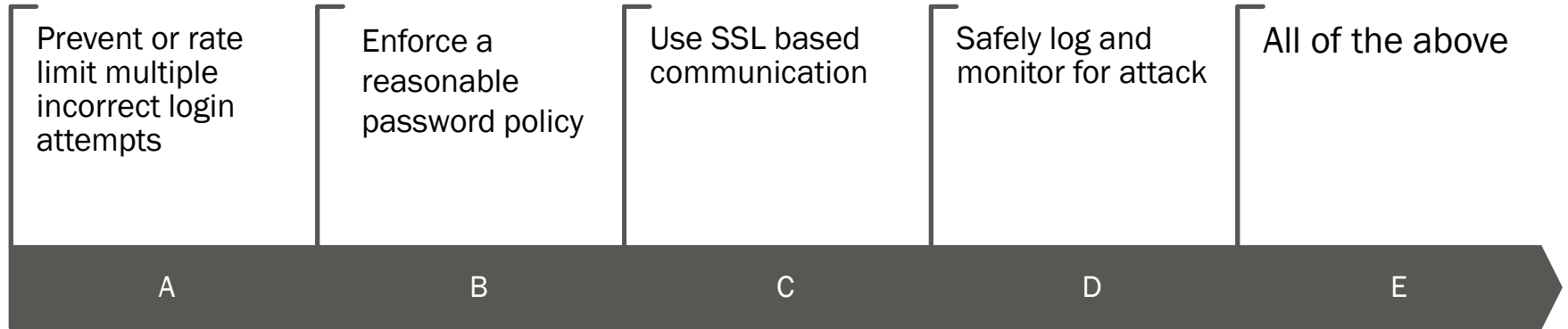
AJIROGHENE SUNDAY



1. WHICH OF THE FOLLOWING IS NOT A COMMON ATTACK ON PLAIN TEXT PASSWORDS?



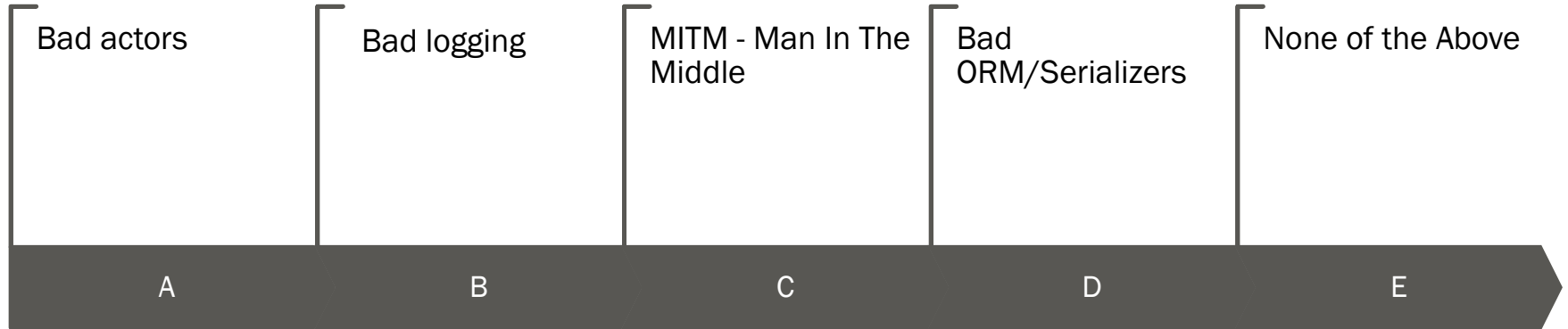
2. WHICH OF THE FOLLOWING IS NOT A WAY TO MITIGATE BRUTE FORCE ATTACK



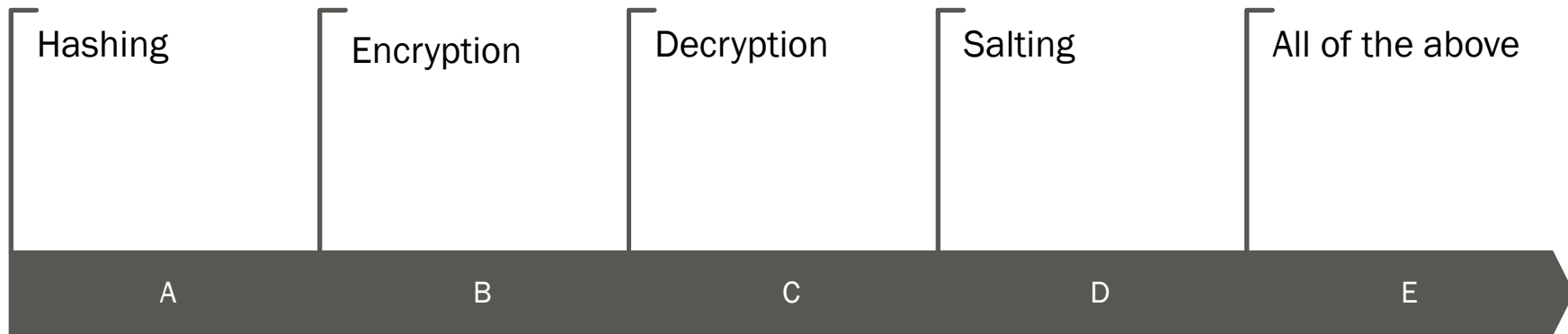
3 WHICH OF THE FOLLOWING IS A FORM OF ENCRYPTION USING 2 KEYS?

Hashing	Salting	Symmetrical encryption	Asymmetrical encryption	None of the above
A	B	C	D	E

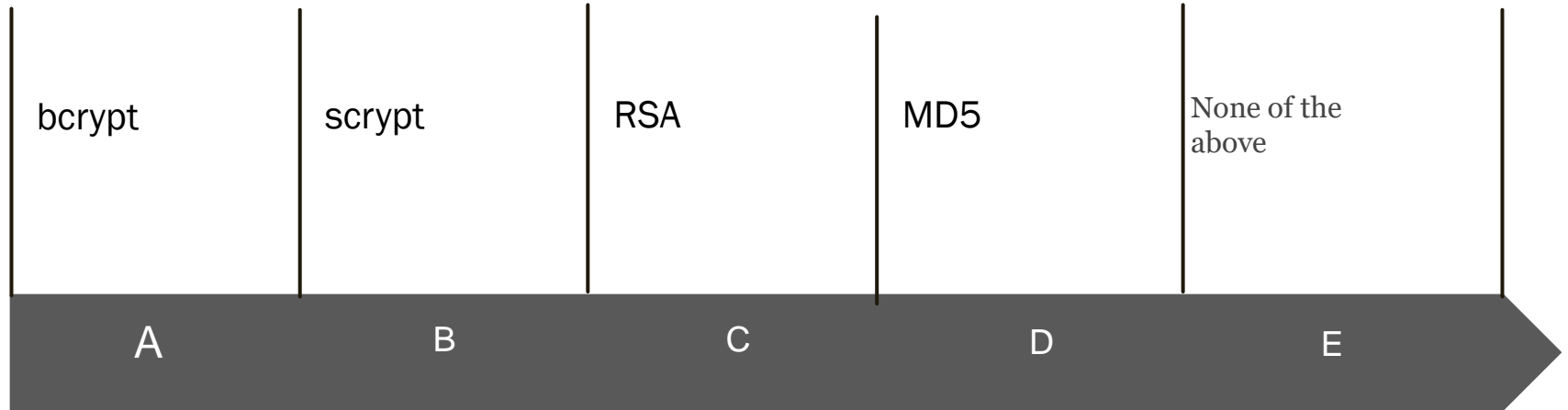
4 WHICH IS NOT AN EXAMPLE OF A RISK ASSOCIATED WITH ENCRYPTION KEYS?



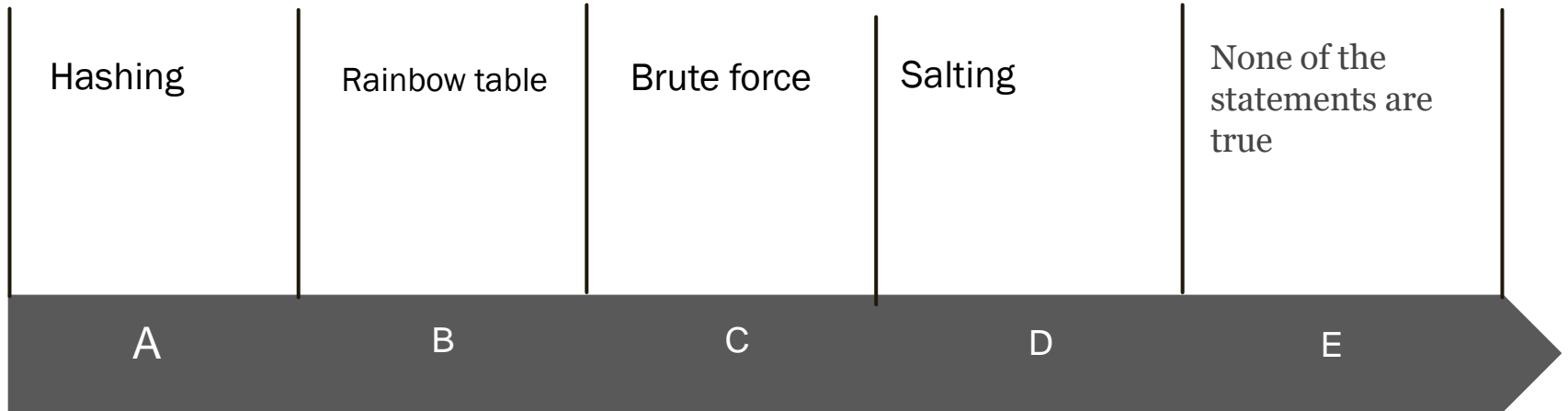
5 WHAT IS A ONE-WAY FUNCTION THAT JUMBLES TEXT IN ONLY ONE DIRECTION AND ELIMINATES THE KEY AND THE ABILITY TO GO BACKWARDS TO THE ORIGINAL STRING



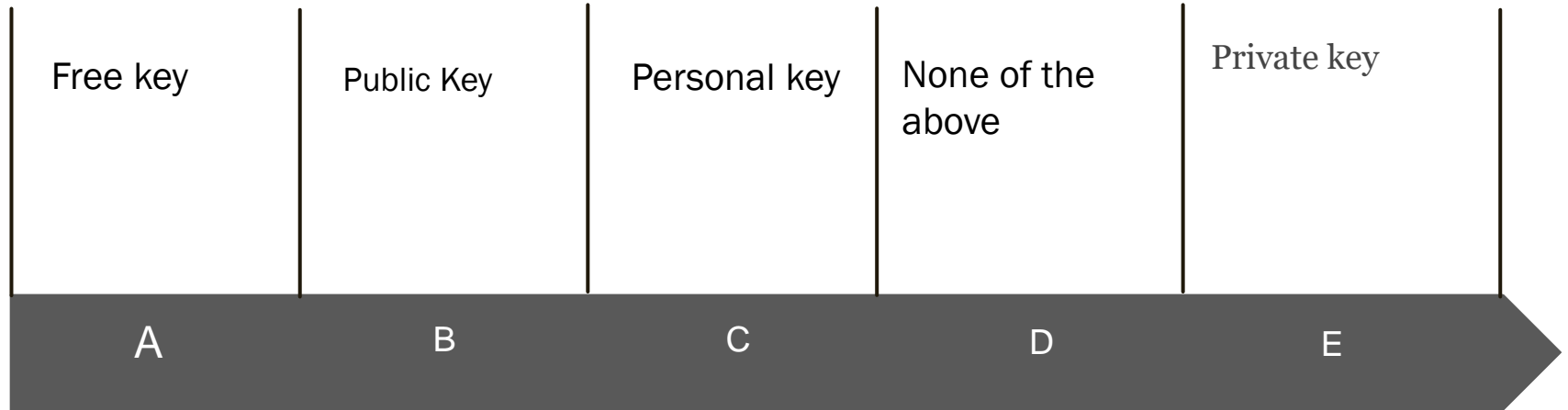
6 WHICH IS NOT A HASHING FUNCTION?



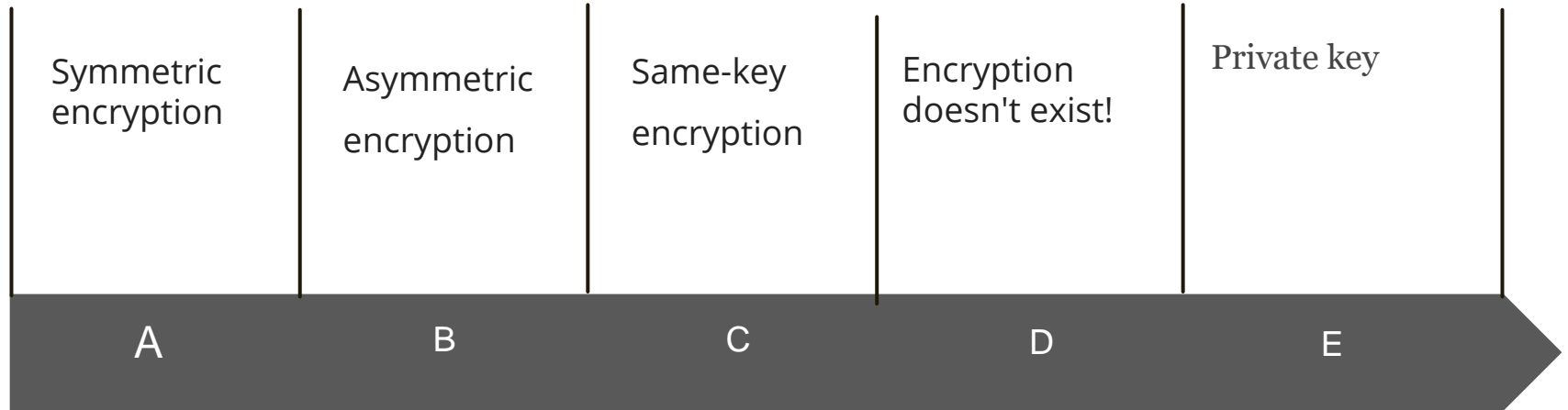
7 WHAT IS AN ATTACK THAT HASHES COMMONLY USED PASSWORDS AND GENERATES THE HASH-PASSWORD KEY-VALUE PAIR?



8 WHAT TYPE OF KEY IS NOT SHARED AND COMPUTERS CANNOT DECRYPT A MESSAGE WITHOUT IT??



9 IF THE SAME KEY IS USED TO ENCRYPT AND DECRYPT A MESSAGE, THIS IS KNOWN AS?



10 TAKING A SECRET MESSAGE AND REPRODUCING THE ORIGINAL PLAIN TEXT IS KNOWN AS?

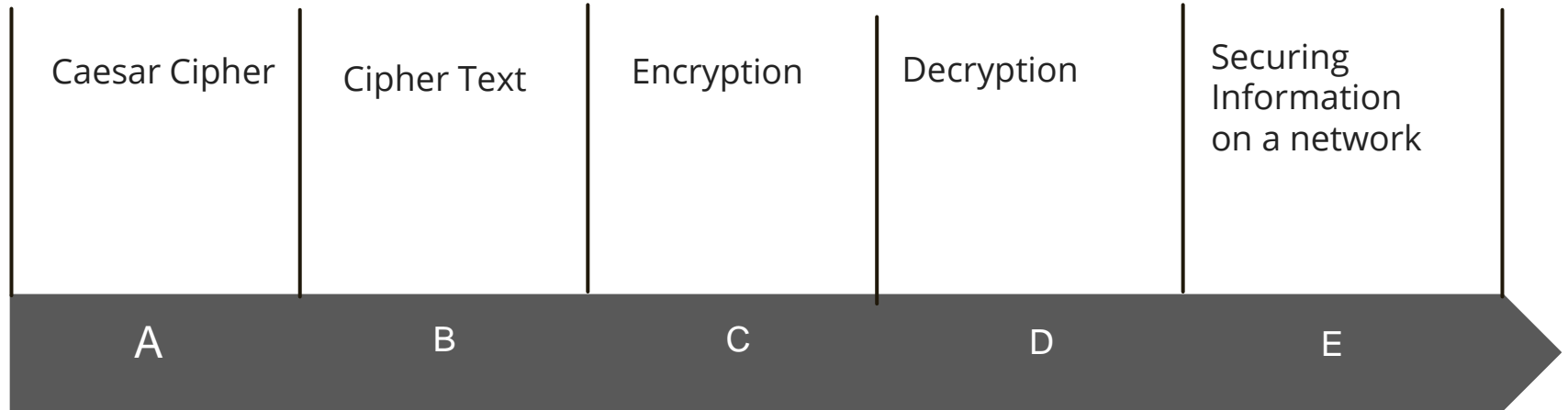


TABLE OF CONTENTS

01

RECAP

02

Authorization

05

What Next

03

ROLE-
PERMISSION
BASED ACCESS



04

ALTERNATIVE
ATTACK VECTORS





01

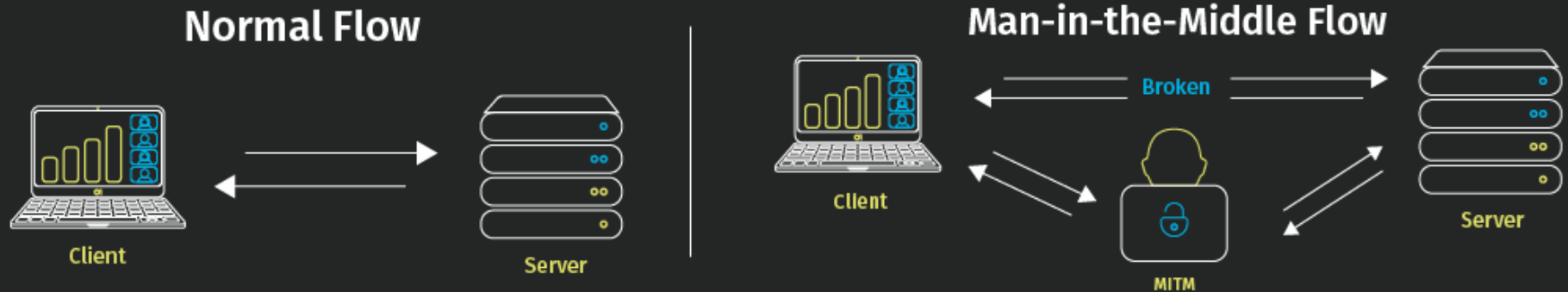
RECAP



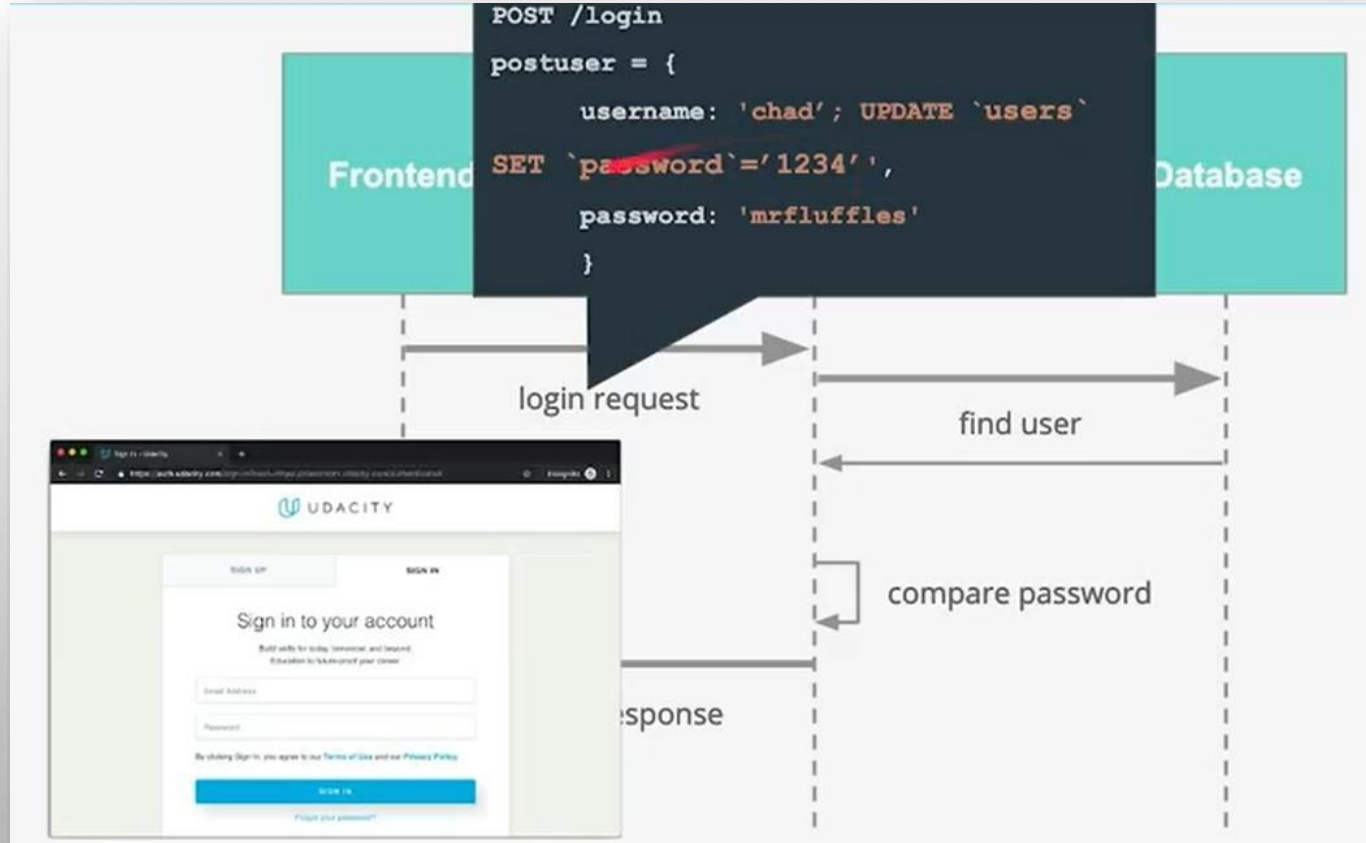
RISKS INVOLVED WITH PLAIN TEXT

- BAD ACTORS

- AUTHENTICATION DATA IS VULNERABLE TO MAN IN THE MIDDLE ATTACKS(MTIM).WHEN PLAIN TEXT PASSWORDS PASS THROUGH NON SSL BASED COMMUNICATION, THEY ARE VULNERABLE TO EAVESDROPPING VIA SNIFFING TOOLS LIKE [WIRESHARK](#).
- USE OF [SSL](#) EITHER DOES NOT GUARANTEE SAFETY TO SSL STRIPPING ATTACKS THAT CAN BE USED TO VIEW THE PLAIN TEXT PASSWORDS FLOWING THROUGH SSL ENCRYPTED MEDIUMS.



DEMO: SQL INJECTION



SQL MITIGATION

Mitigation of

SQL Injection

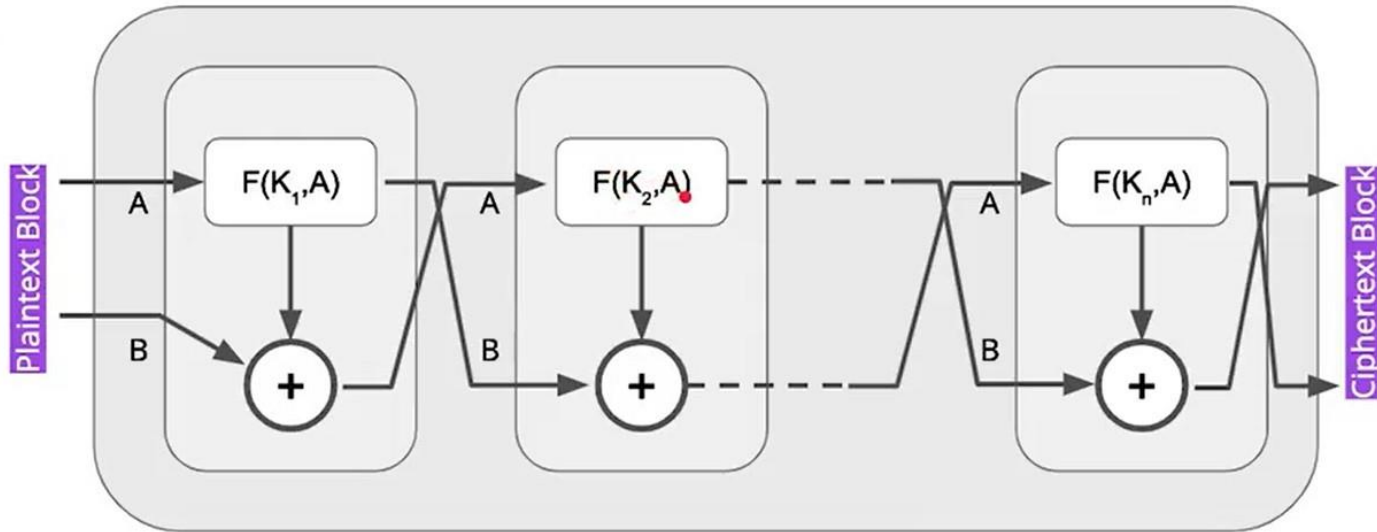
- Always VALIDATE Inputs
- Always SANITIZE Inputs
- Use Object Relational Maps (ORMs)
- Use prepared SQL statements

- `SELECT * FROM `users` WHERE `username` = %S`

ENCRYPTION & HASHING

Feistel Block Cipher

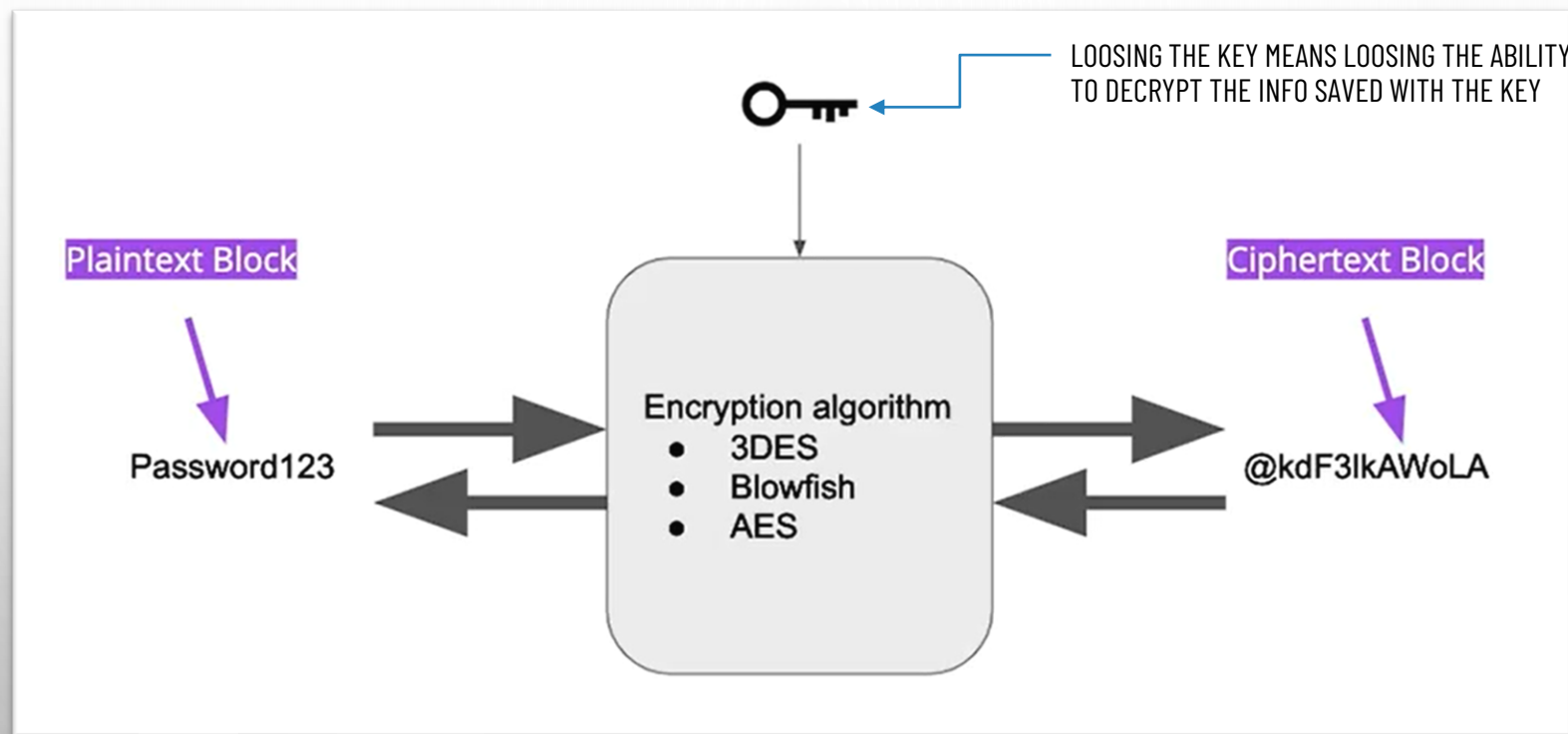
🔑 where k are round keys



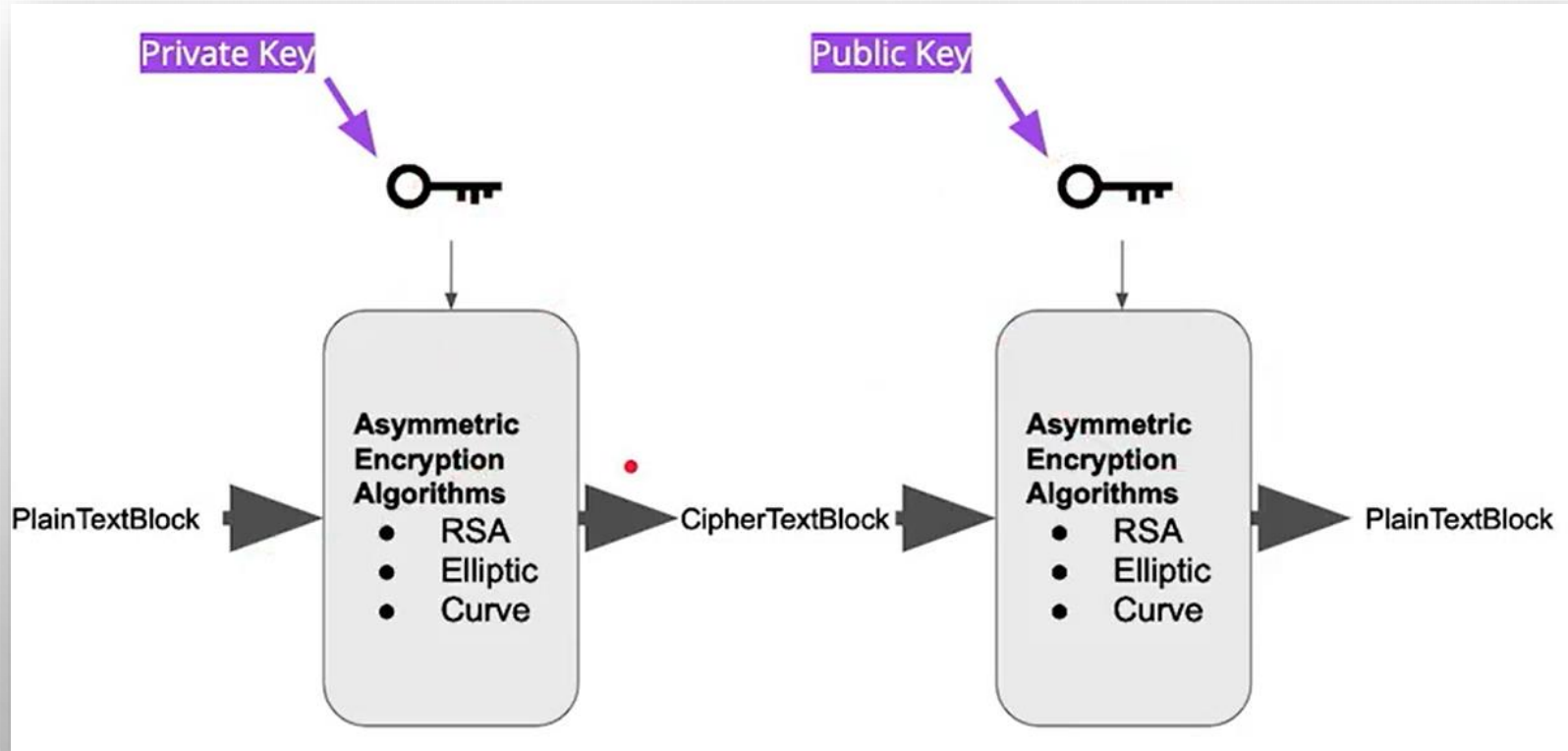
TYPES OF ENCRYPTION

- **SYMMETRIC ENCRYPTION** – USES THE SAME SECRET KEY TO ENCRYPT AND DECRYPT THE MESSAGE. THE SECRET KEY CAN BE A WORD, A NUMBER OR A STRING OF RANDOM LETTERS. BOTH THE SENDER AND THE RECEIVER SHOULD HAVE THE KEY. IT IS THE OLDEST TECHNIQUE OF ENCRYPTION.
- **ASYMMETRIC ENCRYPTION** – IT DEPLOYS TWO KEYS, A PUBLIC KEY KNOWN BY EVERYONE AND A PRIVATE KEY KNOWN ONLY BY THE RECEIVER. THE PUBLIC KEY IS USED TO ENCRYPT THE MESSAGE AND A PRIVATE KEY IS USED TO DECRYPT IT. ASYMMETRIC ENCRYPTION IS LITTLE SLOWER THAN SYMMETRIC ENCRYPTION AND CONSUMES MORE PROCESSING POWER WHEN ENCRYPTING DATA.
- **HYBRID ENCRYPTION** – IT IS A PROCESS OF ENCRYPTION THAT BLENDS BOTH SYMMETRIC AND ASYMMETRIC ENCRYPTION. IT TAKES ADVANTAGE OF THE STRENGTHS OF THE TWO ENCRYPTIONS AND MINIMIZES THEIR WEAKNESS.

ENCRYPTION METHOD

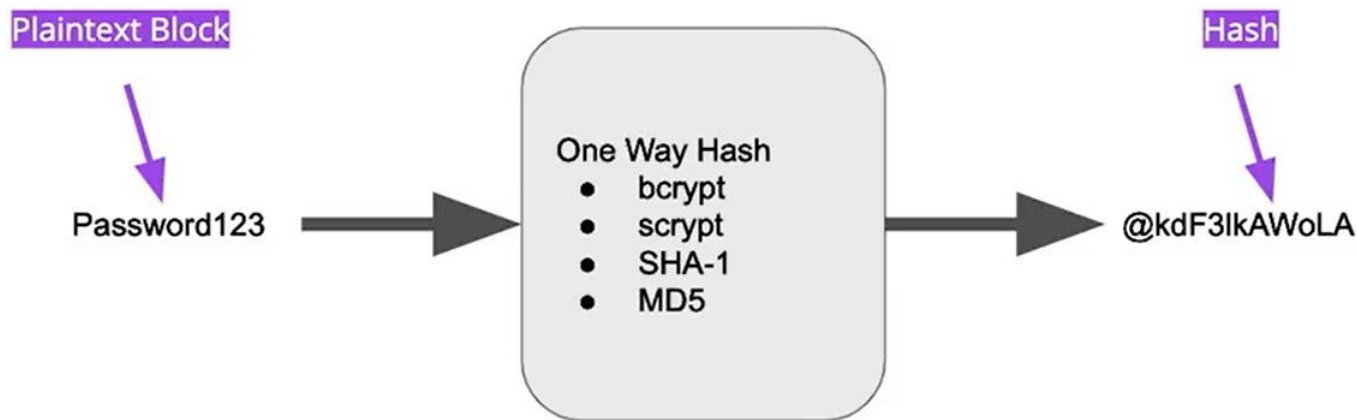


Asymmetric Block



One-Way Technique reduces the risk
associated with encryption

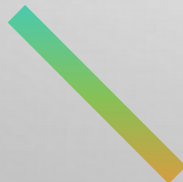
This is Called Hashing





02

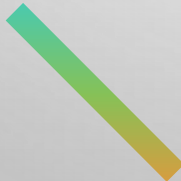
AUTHORIZATION





Authorization is the process of giving someone the ability to access a resource.

A good example is house ownership. The owner has full access rights to the property (the resource) but can grant other people the right to access it. You say that the owner **authorizes** people to access it. This simple example allows us to introduce a few concepts in the authorization context.



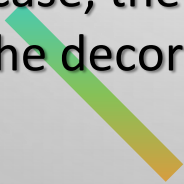
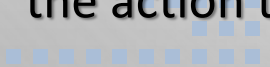

01

For instance, accessing the house is a **permission**, that is, an action that you can perform on a resource. Other permissions on the house may be furnishing it, cleaning it, repair it, etc.

02

A permission becomes a **privilege** (or right) when it is assigned to someone. So, if you assign permission to furnish your house to your interior decorator, you are granting them that privilege.

On the other hand, the decorator may ask you permission to furnish your house. In this case, the requested permission is a **scope**, that is, the action that the decorator would like to perform at your house

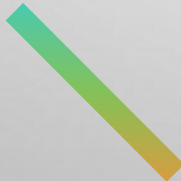


03



To define other concept around authorization, let us consider some more example...

Imagine the plane boarding process. You have your boarding pass that states you are **authorized** to fly with that plane. However, it is not enough for the gate agent to let you get on board. You also need your passport stating your **identity**. In this case, the gate agent compares the name on the passport with the name on the boarding pass and let you go through if they match.





In the authorization context, your name is an **attribute** of your identity. Other attributes are your age, your language, your credit card, and anything else relevant in a specific scenario.

04

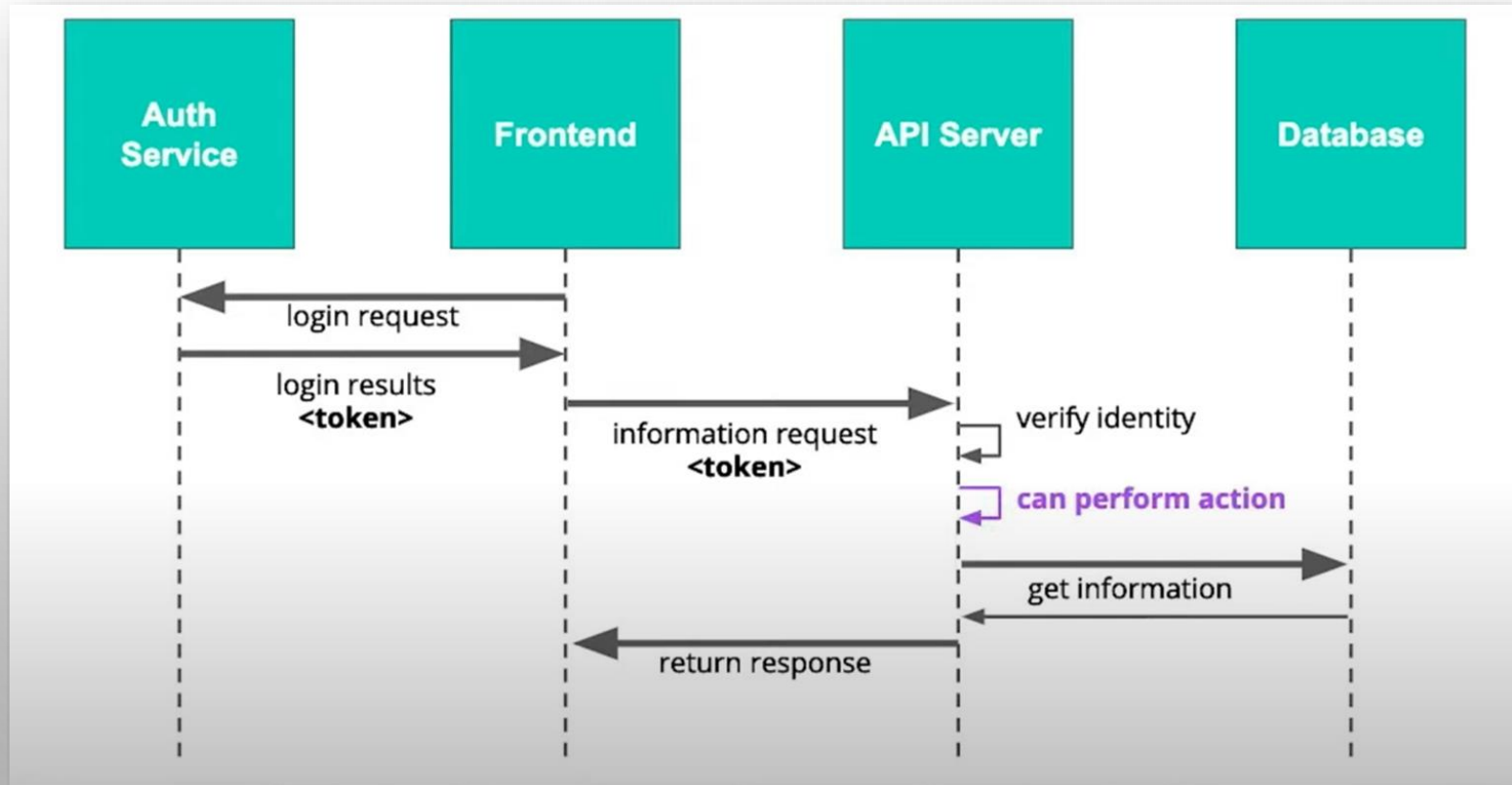


05

Your name written on the passport is a **claim**, that is, a declaration stating you've got that attribute. Someone reading your name on your passport can be sure of your name because they trust the government that issued your passport.



AUTHORIZATION PROCESS



QUESTION TO REMEMBER



REMEMBER, TO DEAL WITH THE THREATS TO WEB SECURITY, TWO PHILOSOPHIES ARE EMPLOYED:

- **AUTHENTICATION**

- ANSWERS THE QUESTION OF [WHO ARE YOU?]

- **AUTHORIZATION**

- ANSWERS THE QUESTION OF [WHAT ARE YOU ALLOWED TO DO?]

WE DON'T WANT TO GIVE SENSITIVE DATA TO EACH INDIVIDUAL PERSON AND SOME ACTIONS ARE ONLY LIMITED TO SPECIFIC PEOPLE.

AUTHORIZATION ANSWERS THE QUESTION OF WHAT CAN AN INDIVIDUAL DO NOW THAT WE KNOW WHO THEY ARE.

UNDERSTANDING THE OBVIOUS

While it is important to know WHO is interacting with our services. It is also important not to open the gates for anyone to do anything they want.



Attack Vector: Imagine if someone communicates with your API directly and supplies user ID as 1 when changing user settings.

01

02

Outcome: That user will be able to change settings related to any other user.

Result: We need to define roles and permissions for users to contain their actions.

03



03

ROLE-PERMISSION BASED ACCESS



WHAT IS RBAC?



ROLE-BASED ACCESS CONTROL (RBAC) TREATS AUTHORIZATION AS PERMISSIONS ASSOCIATED WITH ROLES AND NOT DIRECTLY WITH USERS.

LET US RE-DEFINE THE CONCEPT BASED ON CEONTEXT

- **Scope:** Defines where the role can work
- **Role:** It is a collection of permissions that you can apply to users. This special identity can be assumed by assigned users only.
- **Permissions:** What the users are allowed to do

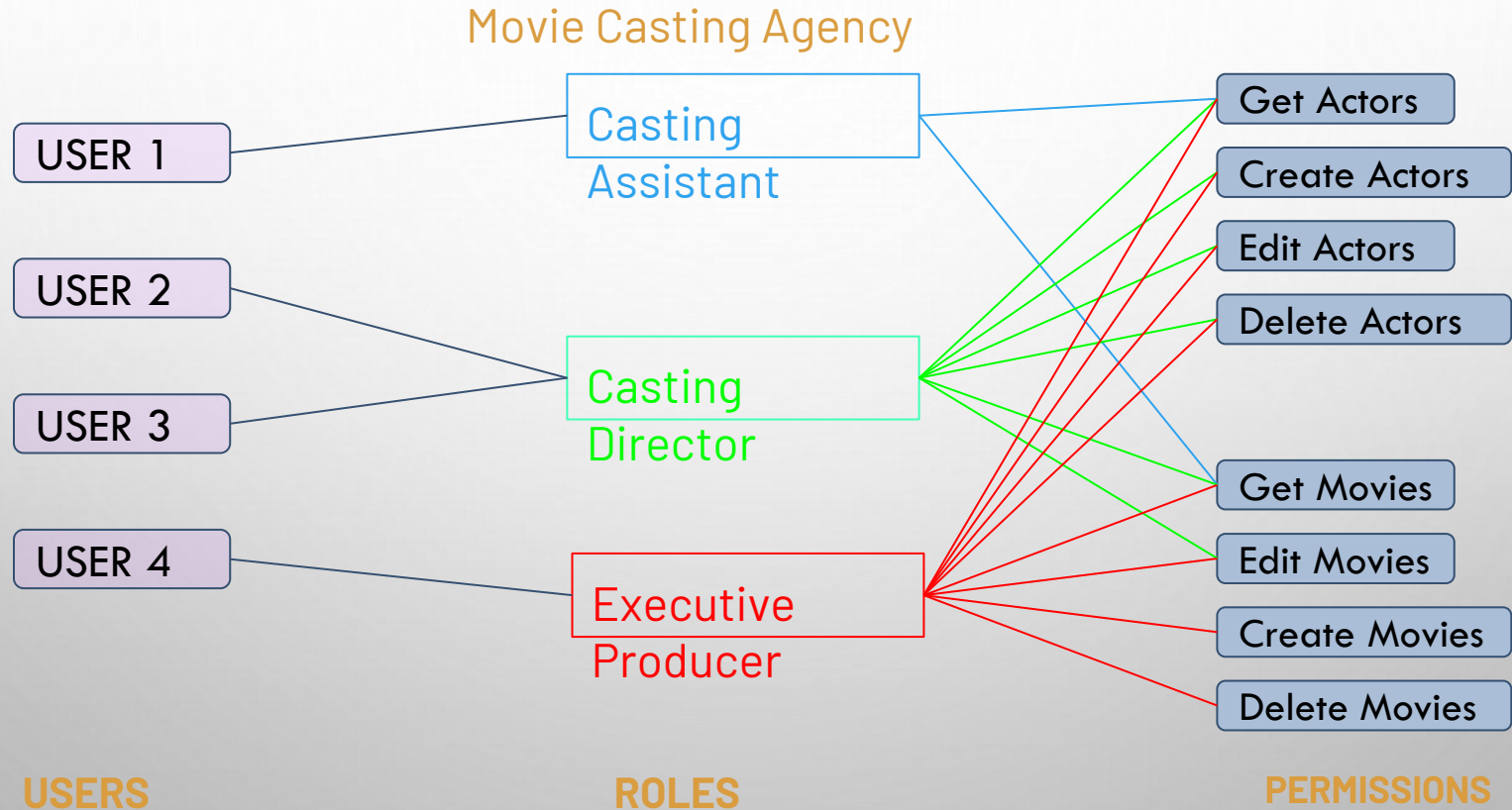
i.e get: drinks, post: drinks

With RBAC, you can enforce security policy at a more fine-grained level. RBAC uses the security principle of least privilege. Least privilege means that a user has precisely the amount of privilege that is necessary to perform a job.

BENEFITS OF RBAC

- CREATE SYSTEMATIC, REPEATABLE ASSIGNMENT OF PERMISSIONS
- EASILY AUDIT USER PRIVILEGES AND CORRECT IDENTIFIED ISSUES
- QUICKLY ADD AND CHANGE ROLES, AS WELL AS IMPLEMENT THEM ACROSS APIS
- CUT DOWN ON THE POTENTIAL FOR ERROR WHEN ASSIGNING USER PERMISSIONS
- INTEGRATE THIRD-PARTY USERS BY GIVING THEM PREDEFINED ROLES
- MORE EFFECTIVELY COMPLY WITH REGULATORY AND STATUTORY REQUIREMENTS FOR CONFIDENTIALITY AND PRIVACY

RBAC DEMO



RBAC DEMO IN FLASK

requires auth decorator

```
from functools import wraps

"""
Forces the endpoint it decorates to require authentication
@parameters:
    permission: string permission (i.e. 'post:actor')
"""
def requires_auth(permission=""):
    def requires_auth_decorator(f):
        @wraps(f)
        def wrapper(*args, **kwargs):
            token = get_token_auth_header()
            payload = verify_decode_jwt(token)
            check_permissions(permission, payload)
            return f(payload, *args, **kwargs)
        return wrapper
    return requires_auth_decorator
```

check permissions

```
"""
Checks if the required permission is present
@parameters
    permission: string permission (i.e. 'post:actor')
    payload: decoded jwt payload
"""
def check_permissions(permission, payload):
    if "permissions" not in payload:
        abort(403)

    if permission not in payload["permissions"]:
        abort(status=403, description="Permission Not found")
    return True
```

Note: You need to pass the permissions in the token as a payload which will be checked against the required permission on the endpoint.

RBAC DEMO IN FLASK

Decorate every endpoint with the `requires_auth` decorator function and pass the required permission for the given endpoint.

Actors Endpoints

```
@app.route('/actors', methods=['POST'])
@requires_auth('post:actors')
def create_actor():
    pass

@app.route('/actors', methods=['GET'])
@requires_auth('get:actors')
def get_actors():
    pass

@app.route('/actors/<int:actor_id>', methods=['GET'])
@requires_auth('get:actors')
def get_actor(actor_id):
    pass

@app.route('/actors/<int:actor_id>', methods=['PATCH'])
@requires_auth('patch:actors')
def edit_actor(actor_id):
    pass

@app.route('/actors/<int:actor_id>', methods=['DELETE'])
@requires_auth('delete:actors')
def delete_actor(actor_id):
    pass
```

Movies Endpoints

```
@app.route('/movies', methods=['POST'])
@requires_auth('post:movies')
def create_movie():
    pass

@app.route('/movies', methods=['GET'])
@requires_auth('get:movies')
def get_movies():
    pass

@app.route('/movies/<int:movie_id>', methods=['GET'])
@requires_auth('get:movies')
def get_movie(movie_id):
    pass

@app.route('/movies/<int:movie_id>', methods=['PATCH'])
@requires_auth('patch:movies')
def edit_movie(movie_id):
    pass

@app.route('/movies/<int:movie_id>', methods=['DELETE'])
@requires_auth('delete:movies')
def delete_movie(movie_id):
    pass
```

RBAC DEMO IN FLASK-RESTFUL

Decorate every endpoint with the `requires_auth` decorator function and pass the required permission for the given endpoint.

Note:

This Example uses Flask-Restful, a Flask extension that adds support for quickly building REST APIs.

```
Actors Restful Endpoints

class CreateListActorResource(Resource):

    @requires_auth("get:actors")
    def get(self, *args, **kwargs):
        pass

    @requires_auth("post:actors")
    def post(self, *args, **kwargs):
        pass

class RetrieveUpdateDestroyActorResource(Resource):

    @requires_auth("get:actors")
    def get(self, *args, **kwargs):
        pass

    @requires_auth("patch:actors")
    def patch(self, *args, **kwargs):
        pass

    @requires_auth("delete:actors")
    def delete(self, *args, **kwargs):
        pass
```

```
Movies Restful Endpoints

class CreateListMovieResource(Resource):

    @requires_auth("get:movies")
    def get(self, *args, **kwargs):
        pass

    @requires_auth("post:movies")
    def post(self, *args, **kwargs):
        pass

class RetrieveUpdateDestroyMovieResource(Resource):

    @requires_auth("get:movies")
    def get(self, *args, **kwargs):
        pass

    @requires_auth("patch:movies")
    def patch(self, *args, **kwargs):
        pass

    @requires_auth("delete:movies")
    def delete(self, *args, **kwargs):
        pass
```



04

ALTERNATIVE ATTACK VECTORS



WHAT IS ATTACK VECTOR?

An attack vector is a path or method that a hacker uses to gain unauthorized access to a network or computer in order to exploit system flaws. Hackers utilize a variety of attack vectors to launch assaults that exploit system flaws, compromise data, or steal login credentials. Malware and viruses, harmful email attachments and online links, pop-up windows, and instant messages are examples of such approaches, which entail the attacker duping an employee or individual user.

MOST COMMON ATTACK VECTOR

Intruders are always looking for new ways to attack. The following are the most popular attack vectors:

- **Software Vulnerabilities**

An attacker can employ a threat vector, such as malware, to gain unauthorized access if a network, operating system, computer system, or application has an unpatched security vulnerability.

- **Compromised User Credentials**

Users can reveal their user IDs and passwords knowingly or accidentally. This can be done orally, but cyber attackers can also employ a brute-force attack to acquire access to credentials by trying numerous combinations of user IDs and passwords until an authorized pair of credentials is discovered. These credentials are then used by the hacker to gain access to a network, system, or application.

- **Weak Passwords and Credentials**

Brute-force attacks concentrate cyber attackers' efforts on weak or readily guessed user IDs and passwords. Hackers can also steal credentials by monitoring public Wi-Fi networks for when users enter their login credentials. Hackers can also get access by persuading users to click unsolicited email attachments with malicious links to fake websites that persuade them to hand over personally identifiable information (PII).

MOST COMMON ATTACK VECTOR

- **Malicious Employees**

Employees who are malicious or unhappy can use their security clearances to hack into networks and systems and obtain sensitive information such as customer lists and [intellectual property \(IP\)](#), which they can either demand a ransom for or sell to others for nefarious purposes.

- **Poor or Missing Encryption**

Employees may forget to encrypt critical data kept on computers and smartphones when out in the field in some situations. In other circumstances, encryption algorithms have known design weaknesses or only encrypt and safeguard data with a restricted number of keys.

- **Ransomware**

Ransomware is a sort of malware that encrypts the data on the victim's computer and threatens to publish or prevent access to it unless the attacker is paid a ransom. Ransomware can encrypt a user's files and demand payment in exchange for the files' unlocking. The majority of ransomware is downloaded mistakenly onto a computer or network by a user.

MOST COMMON ATTACK VECTOR

- **Malicious Employees**

Employees who are malicious or unhappy can use their security clearances to hack into networks and systems and obtain sensitive information such as customer lists and [intellectual property \(IP\)](#), which they can either demand a ransom for or sell to others for nefarious purposes.

- **Poor or Missing Encryption**

Employees may forget to encrypt critical data kept on computers and smartphones when out in the field in some situations. In other circumstances, encryption algorithms have known design weaknesses or only encrypt and safeguard data with a restricted number of keys.

- **Ransomware**

Ransomware is a sort of malware that encrypts the data on the victim's computer and threatens to publish or prevent access to it unless the attacker is paid a ransom. Ransomware can encrypt a user's files and demand payment in exchange for the files' unlocking. The majority of ransomware is downloaded mistakenly onto a computer or network by a user.

SOCIAL ENGINEERING

Social engineering attacks use a combination of psychological manipulation and malware to gain access to systems or data or to gain elevated access to the system or its sensitive data.

Here's an interactive demo:

https://www.cssia.org/social_engineering/

PHISHING

Phishing is a kind of social engineering attack where a bad actor sends an email purporting to be from a legitimate source that tricks the user into providing sensitive information.

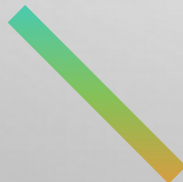
Here's an interactive demo:

<https://www.hacksplaining.com/exercises/email-spoofing>



05

WHAT NEXT

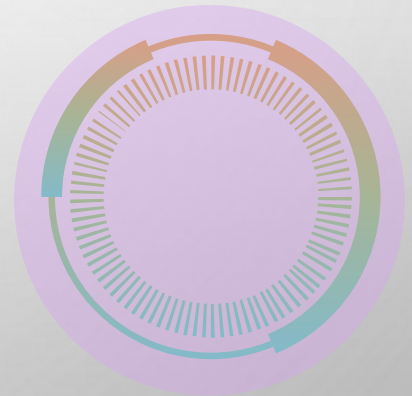
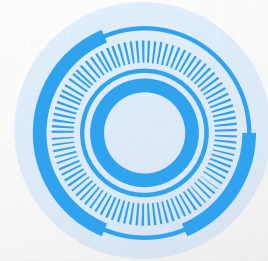


WHAT NEXT

STUDY THE FOLLOWING LESSONS FROM "SERVER DEPLOYMENT AND CONTAINERIZATION" PART:

- INTRODUCTION
- CONTAINERS

WORK ON SUBMITTING THE "COFFEESHOP FULLSTACK PROJECT"



QUESTIONS



THANK YOU