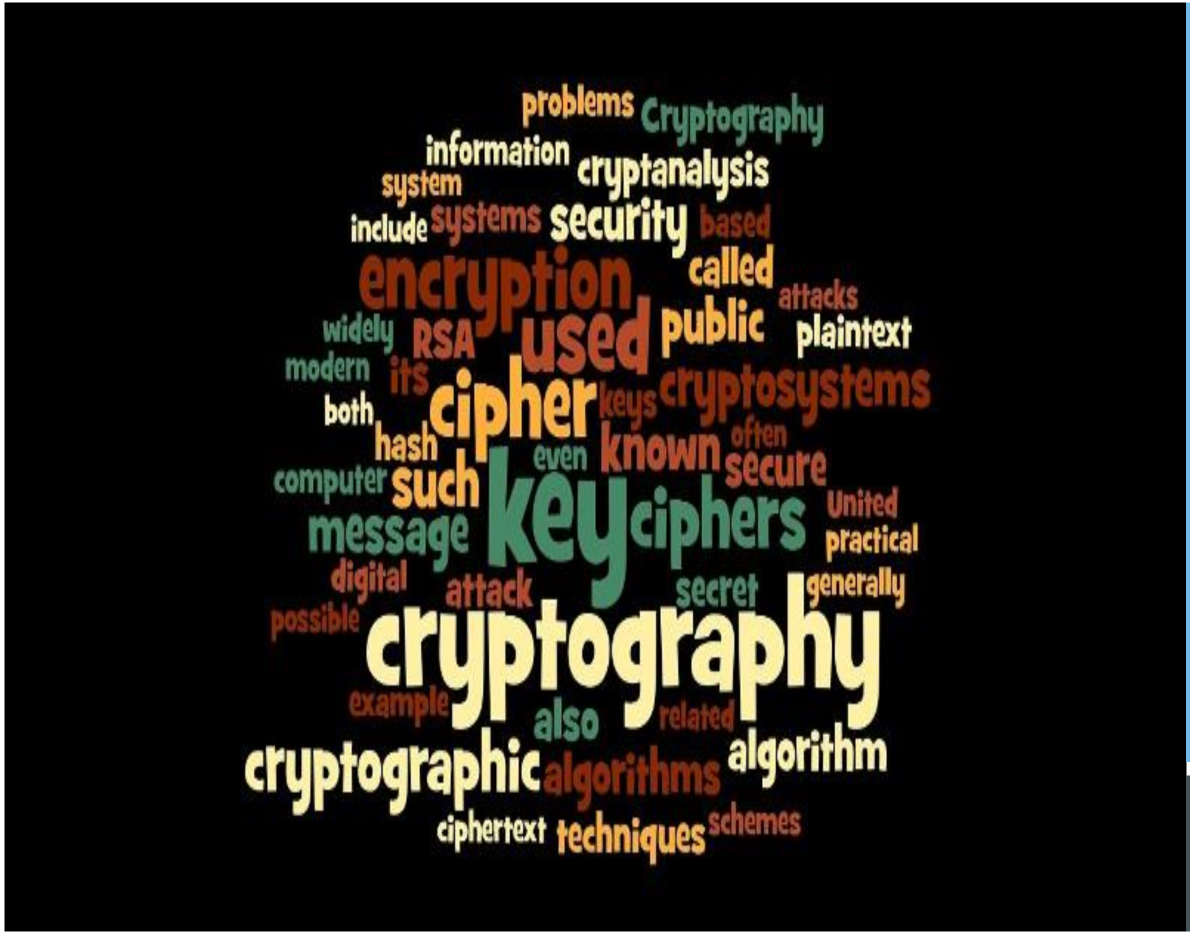
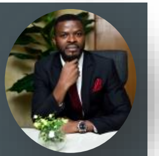


FULL-STACK NANODEGREE SESSION 8

AJIROGHENE SUNDAY



Ajiroghene Sunday | LinkedIn: in/sunday-ajiroghene

1. WHAT IS THE PROCESS OF GIVING AN INDIVIDUAL ACCESS TO A SYSTEM OR RESOURCE?

A. Encryption

B. Authorization

C. Auditing

D. Authentication

E. None of the above

2. JWT IS MADE UP OF THE FOLLOWING PART EXCEPT __?

Header	Payload	Signature	Password	Encryption
A	B	C	D	E

3 WHAT TYPE OF ARGUMENTS ARE USED TO CREATE GENERAL-PURPOSE DECORATORS?

A	B	C	D	E
Positional arguments	Default arguments	Keyword arguments	Arbitrary arguments	None of the above

4 WHAT SYMBOL IS USED TO DECORATE A FUNCTION?

A	B	C	D	E
@	!	+	/	None of the Above

5 DECORATORS DO NOT MODIFY A FUNCTION ______.

A	B	C	D	E
Permanently	Temporarily	None of the above	Can not be determined	All of the above

6 WHICH OF THE FOLLOWING STATEMENTS IS TRUE?

A function Should be Intuitive and Organize by resource

A function can not be passed as an argument

A function is not an object

A function can return another function

A function can not be assigned to a variable

A B C D E

7 WHICH OF THE FOLLOWING STATEMENT(S) IS/ARE TRUE?

STATEMENT 1: NESTED FUNCTIONS ARE FUNCTIONS THAT ARE DEFINED INSIDE ANOTHER FUNCTION

STATEMENT 2: NESTED FUNCTIONS CAN ACCESS VARIABLES OF THE ENCLOSING SCOPE

A	B	C	D	E
Statement 2 is true	Both statements are true	All the above	Statement 1 is true	None of the statements are true

8 IDENTIFY THE DECORATOR FUNCTION IN THE BELOW CODE.

CODE

```
def func1(func):
  def func2():
     return func()
  return func2
def func3():
  return “hello world!”

func3 = func1(func3)
print(func3())
```

func1	func2	func3	All of the above	None of the statements are true
A	B	C	D	E

9 IDENTIFY THE FUNCTION WHICH CAN BE USED AS A GENERAL-PURPOSE DECORATOR IN THE BELOW CODE.

code

```
def func1(func):
  def func2(*args, **kwargs):
     print("running the function with following arguments")
     print(args, kwargs)
     return func(*args, **kwargs)
  return func2
```

```
def func3(func):
  def func4():
     return func
  func4()
  return func4
```

A func1

B func2

C func3

D All of the above

E func4

10 AN AUTHENTICATION PROTOCOL IS DEFINED AS _____.

A	B	C	D	E
A computer system protocol that allows users to move authentic files from one computer to another	A data transfer protocol used by network routers on the internet to authenticate data packets	A protocol used to authenticate trading securities between two parties	A set of rules between computers, designed to securely transfer authenticated data between parties	None of the statements are true

TABLE OF CONTENTS

01

RECAP

PYTHON DECORATORS

- A DECORATOR IS A DESIGN PATTERN IN PYTHON THAT ALLOWS A USER TO ADD NEW FUNCTIONALITY TO AN EXISTING OBJECT OR FUNCTION WITHOUT MODIFYING ITS STRUCTURE.

- FUNCTIONS ARE OBJECTS.

- FUNCTIONS WHICH TAKE OTHER FUNCTIONS AS PARAMETERS OR PERFORM OPERATIONS ON OTHER FUNCTIONS ARE CALLED **HIGHER ORDER FUNCTIONS**

- **PYTHON DECORATORS** ARE FUNCTIONS OR CLASSES IN PYTHON THAT TAKES ANOTHER FUNCTION AS A PARAMETER OR RETURN A FUNCTION

PYTHON DECORATORS

This is also called **metaprogramming** because a part of the program tries to modify another part of the program at compile time.

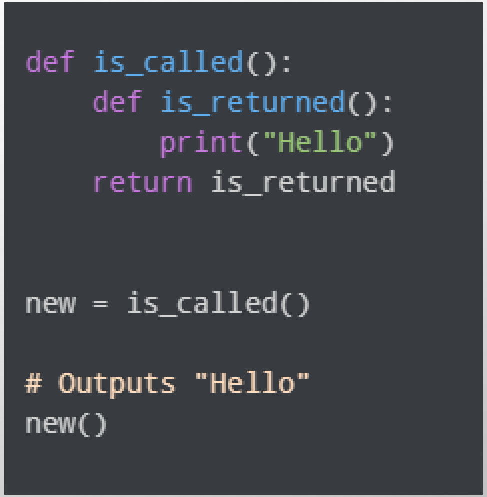
EXAMPLE

```
def first(msg):
    print(msg)


first("hello")

second = first
second("hello")

OUTPUT

hello
hello
```

Functions can be passed as arguments to another function. Such functions that take other functions as arguments are also called **higher order functions**. Here is an example of such a function.

```
def inc(x):
    return x + 1

def dec(x):
    return x - 1



def operate(func, x):
     result = func(x)
     return result

Invoking the Function
>>> operate(inc, 3)
    4
>>> operate(dec, 3)
    2
```

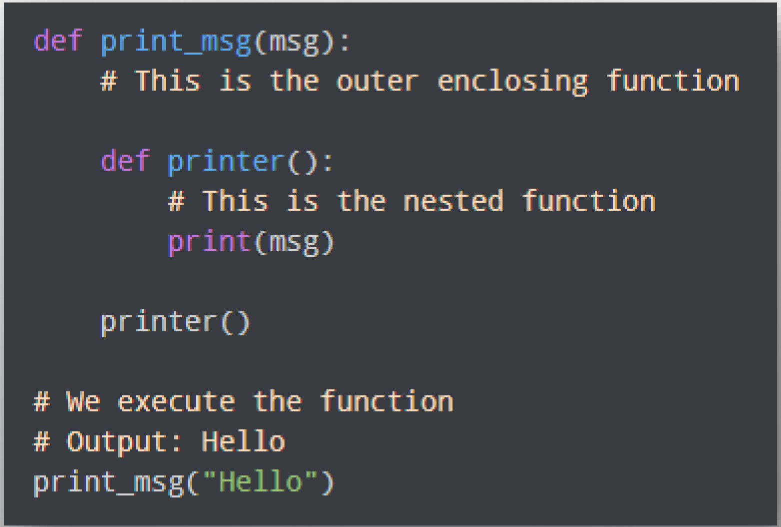
Furthermore, a function can return another function. Below ***is_returned()*** is a nested function which is defined and returned each time we call ***is_called()***



```
def is_called():
    def is_returned():
        print("Hello")
    return is_returned


new = is_called()

# Outputs "Hello"
new()
```

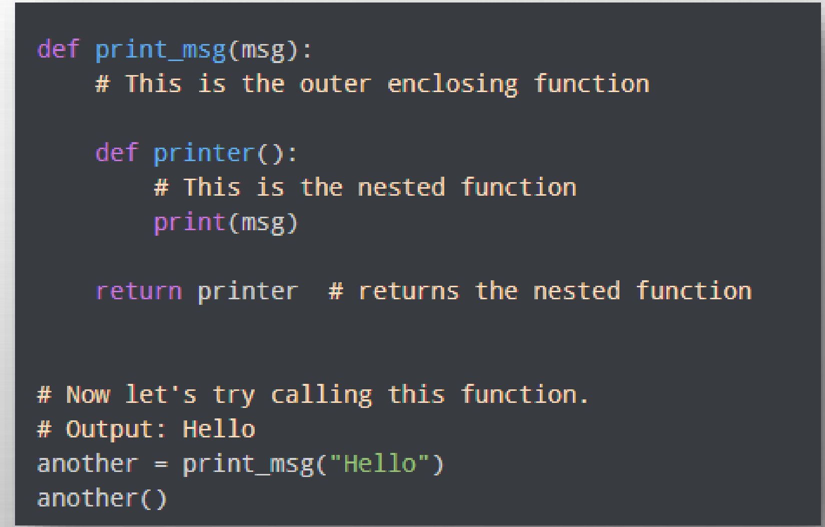
LET’S TALK ABOUT NON-LOCAL VARIABLE & CLOSURE

Nested functions can access variables of the enclosing scope. In Python such non-local variables are read-only by default, however in order to modify them, we declare them explicitly as non-local (using nonlocal keyword). Following is an example of a nested function accessing a non-local variable.



```python
def print_msg(msg):
    # This is the outer enclosing function

    def printer():
        # This is the nested function
        print(msg)

    printer()

# We execute the function
# Output: Hello
print_msg("Hello")
```

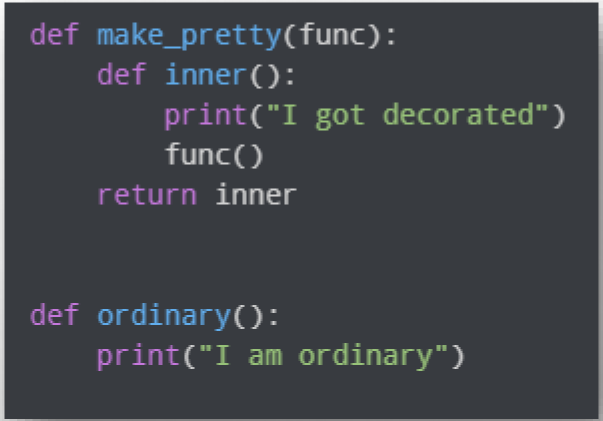
What would happen if the last line of the function print_msg() returned the printer() function instead of calling it? This means the function was defined as follows:



```python
def print_msg(msg):
    # This is the outer enclosing function

    def printer():
        # This is the nested function
        print(msg)

    return printer  # returns the nested function


# Now let's try calling this function.
# Output: Hello
another = print_msg("Hello")
another()
```

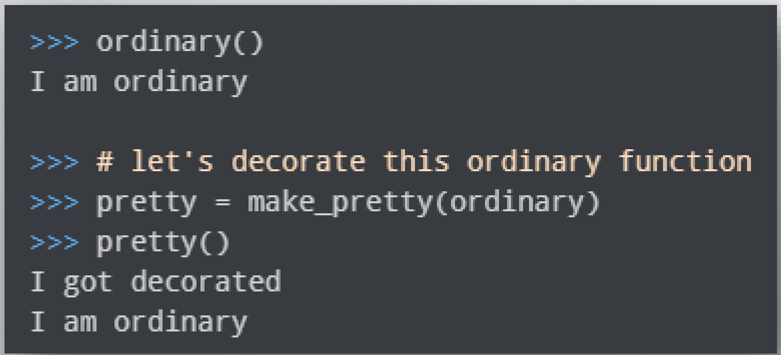
DECORATOR

Basically, a decorator takes in a function, adds some functionality and returns it.



```python
def make_pretty(func):
    def inner():
        print("I got decorated")
        func()
    return inner


def ordinary():
    print("I am ordinary")
```

When you run the following codes in shell,



```
>>> ordinary()
I am ordinary

>>> # let's decorate this ordinary function
>>> pretty = make_pretty(ordinary)
>>> pretty()
I got decorated
I am ordinary
```

The function `ordinary()` got decorated and the returned function was given the name *pretty*.

We can see that the decorator function added some new functionality to the original function. This is similar to packing a gift.

The decorator acts as a wrapper. The nature of the object that got decorated (actual gift inside) does not alter. But now, it looks pretty (since it got decorated).

Generally, we decorate a function and reassign it.

WEB SECURITY

PLAYING WITH FIRE

- DEALING WITH THE WEB MEANS THAT YOUR DATA IS EXPOSED TO VIRTUALLY BILLIONS OF PEOPLE
- WE CAN'T "ASSUME" THAT NONE OF THEM ARE "BAD ACTORS" MEANING TO HARM OUR DATA

TO DEAL WITH THAT THREAT, TWO PHILOSOPHIES WERE BORN

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

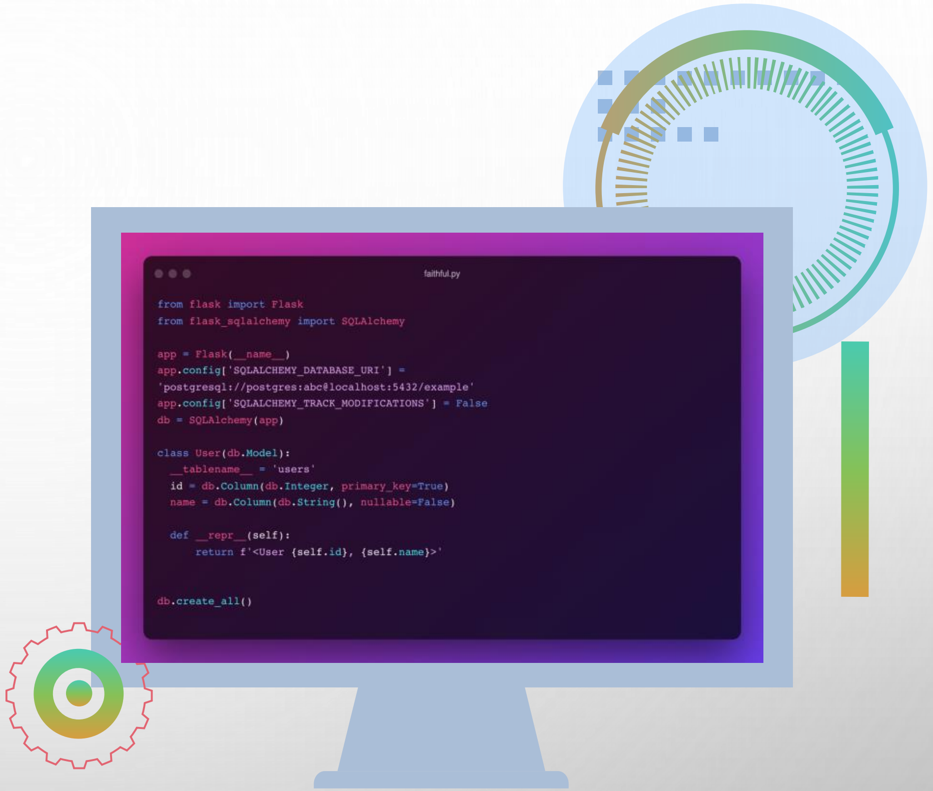
BORN FROM NECESSITY

TO DEAL WITH THAT THREAT, TWO PHILOSOPHIES WERE BORN:

- **AUTHENTICATION**
  - ANSWERS THE QUESTION OF [WHO ARE YOU?]
  - VERIFIES THE IDENTITY OF THE USER
  - WORKS THROUGH AUTHENTICATION METHODS
- **AUTHORIZATION**
  - ANSWERS THE QUESTION OF [WHAT ARE YOU ALLOWED TO DO?]
  - DETERMINES THEIR ACCESS RIGHTS
  - WORKS THROUGH SETTINGS THAT ARE IMPLEMENTED AND MAINTAINED BY THE ORGANIZATION

PRACTICAL

Express yourself



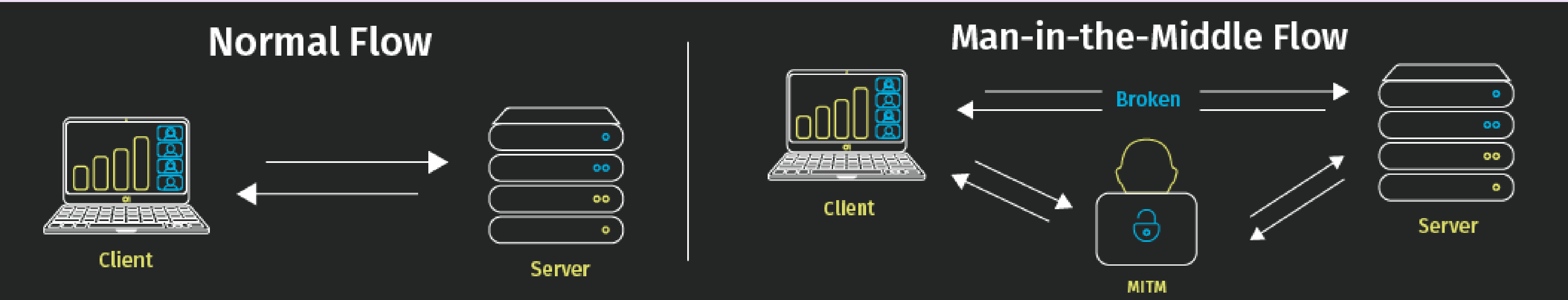
```python
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'postgresql://postgres:abc@localhost:5432/example'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

class User(db.Model):
  __tablename__ = 'users'
  id = db.Column(db.Integer, primary_key=True)
  name = db.Column(db.String(), nullable=False)

  def __repr__(self):
      return f'<User {self.id}, {self.name}>'


db.create_all()
```

02

PASSWORDS

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.



A MAN IN THE MIDDLE (MITM)

This attack is a general term for when a perpetrator positions himself in a conversation between a user and an application – either to eavesdrop or to impersonate one of the parties, making it appear as if a normal exchange of information is underway.

- *The goal of an attack is to steal personal information, such as login credentials, account details and credit card numbers. Targets are typically the users of financial applications, saas businesses, e-commerce sites and other websites where logging in is required.*

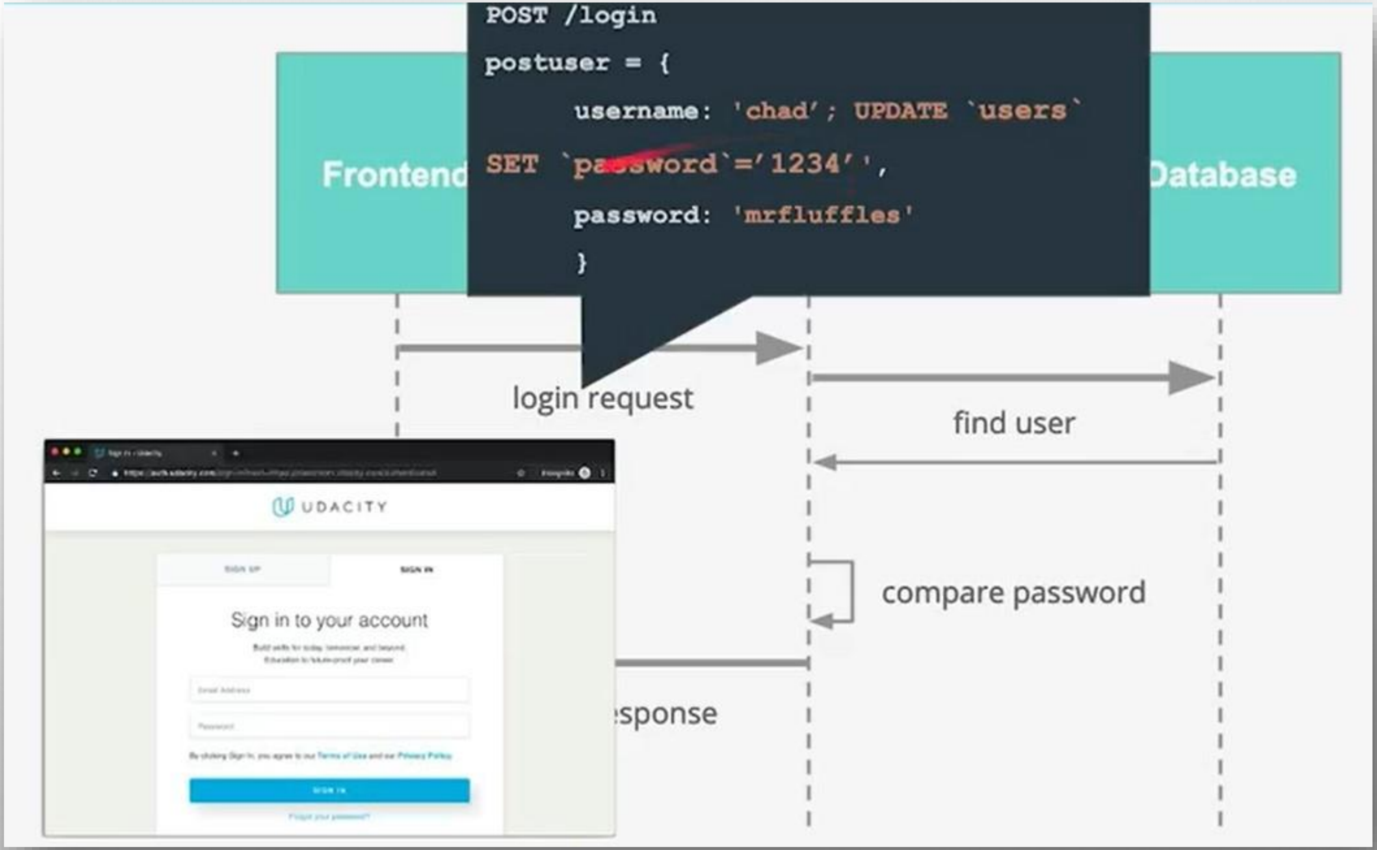
SSL

Stands for secure sockets layer and, in short, it's the standard technology for keeping an internet connection secure and safeguarding any sensitive data that is being sent between two systems, preventing criminals from reading and modifying any information transferred, including potential personal details.

RISKS INVOLVED WITH PLAIN TEXT

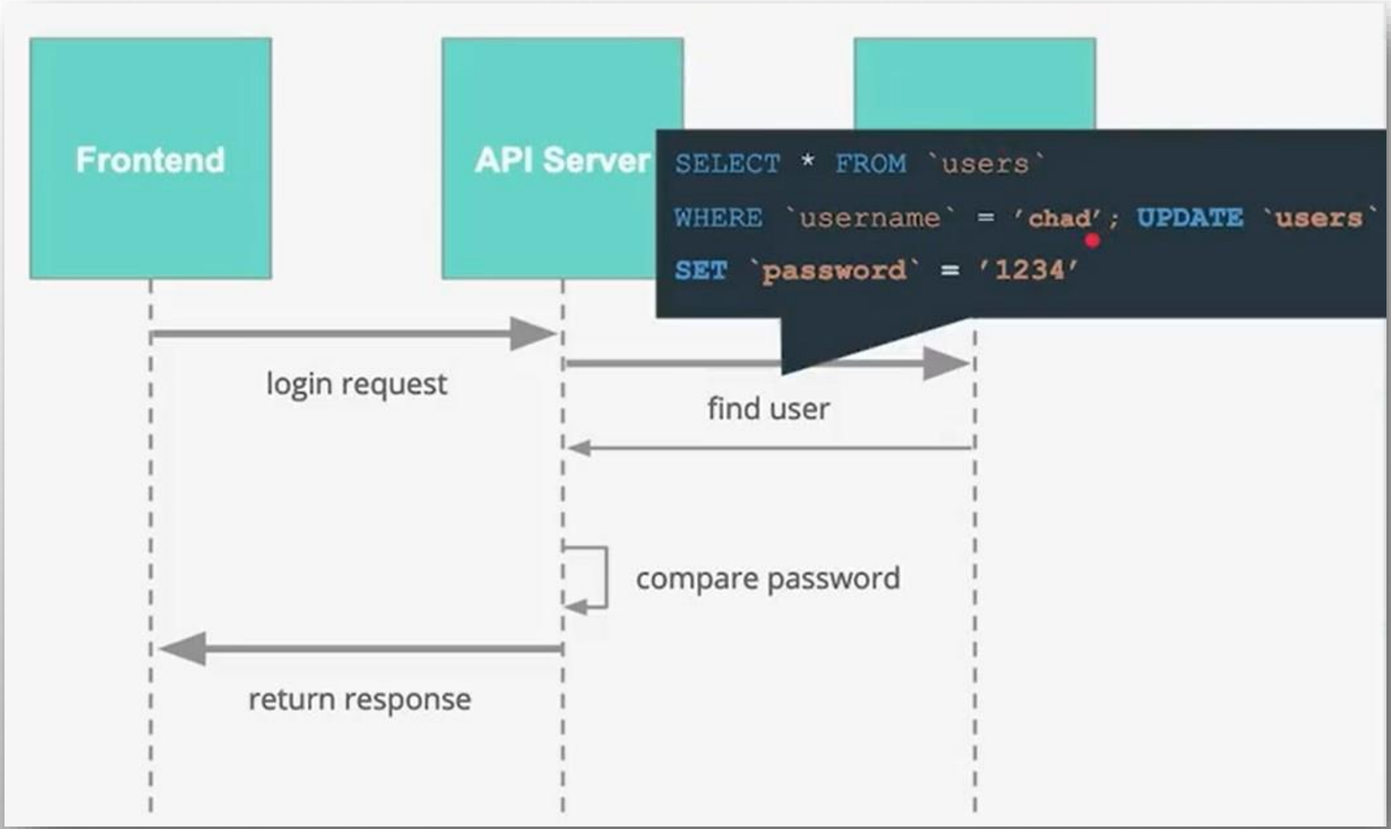
- BAD DATABASE PASSWORDS
  - AUTHENTICATION DATA STORED IN A DATABASE SERVER ARE VULNERABLE TO INSIDE JOBS. THE STOLEN PLAIN TEXT PASSWORDS CAN BE SOLD ON THE BLACK MARKET BY DISGRUNTLED COMPANY EMPLOYEES.
  - ONCE THE DATABASE IS COMPROMISED, PEOPLE WHO USE THE SAME PASSWORD ACROSS SITES ARE IN JEOPARDY OF HAVING ACCOUNTS DRAINED OR THEIR IDENTITIES STOLEN.
- BAD BACKUP SECURITY.
  - A BAD ACTOR CAN ATTACK A BACKUP SERVER AND GET ACCESS TO PASSWORDS. WHY SIMPLIFY THE JOB FOR HIM?

DEMO: SQL INJECTION



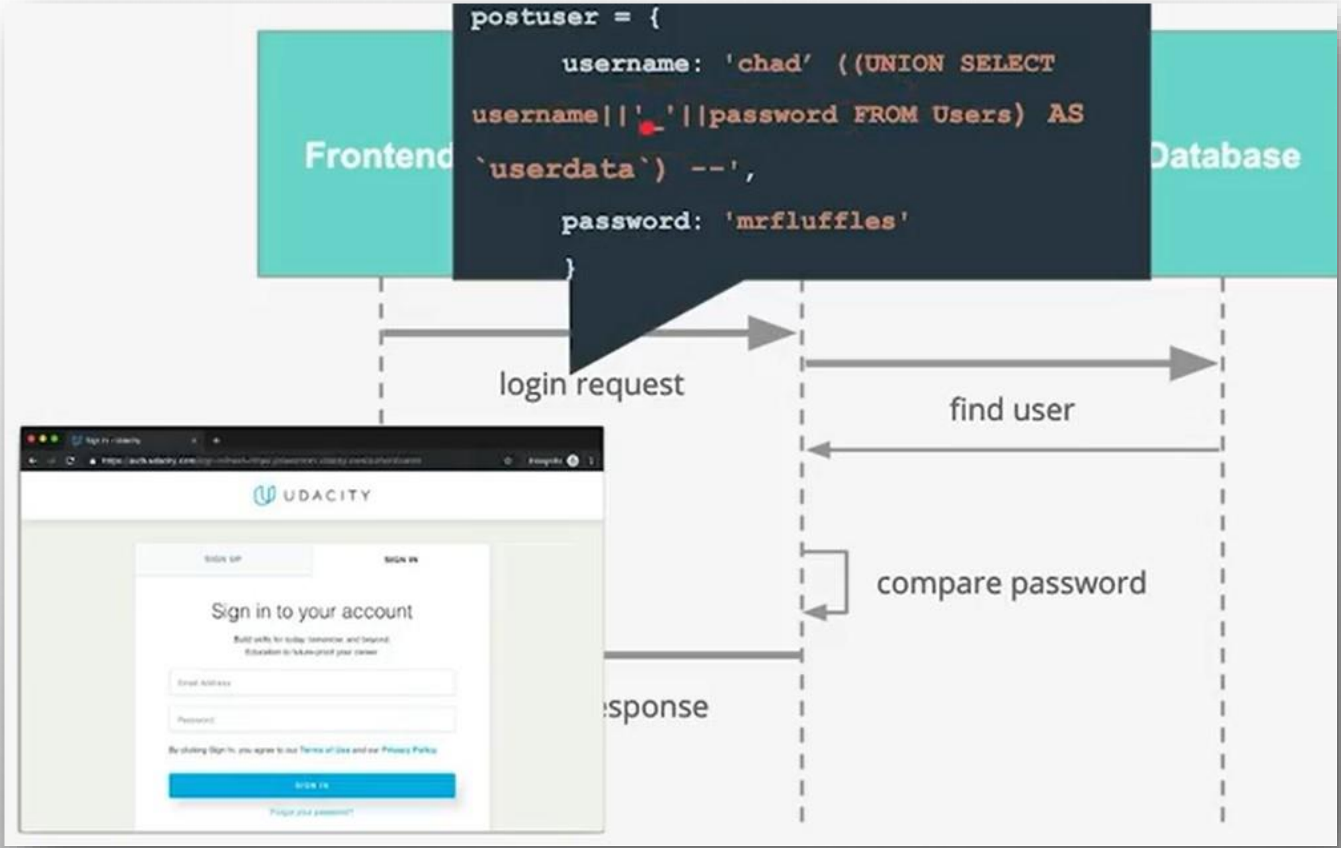
```
POST /login
postuser = {
      username: 'chad'; UPDATE `users`
SET `password`='1234'',
      password: 'mrfluffles'
      }
```

DEMO: SQL INJECTION



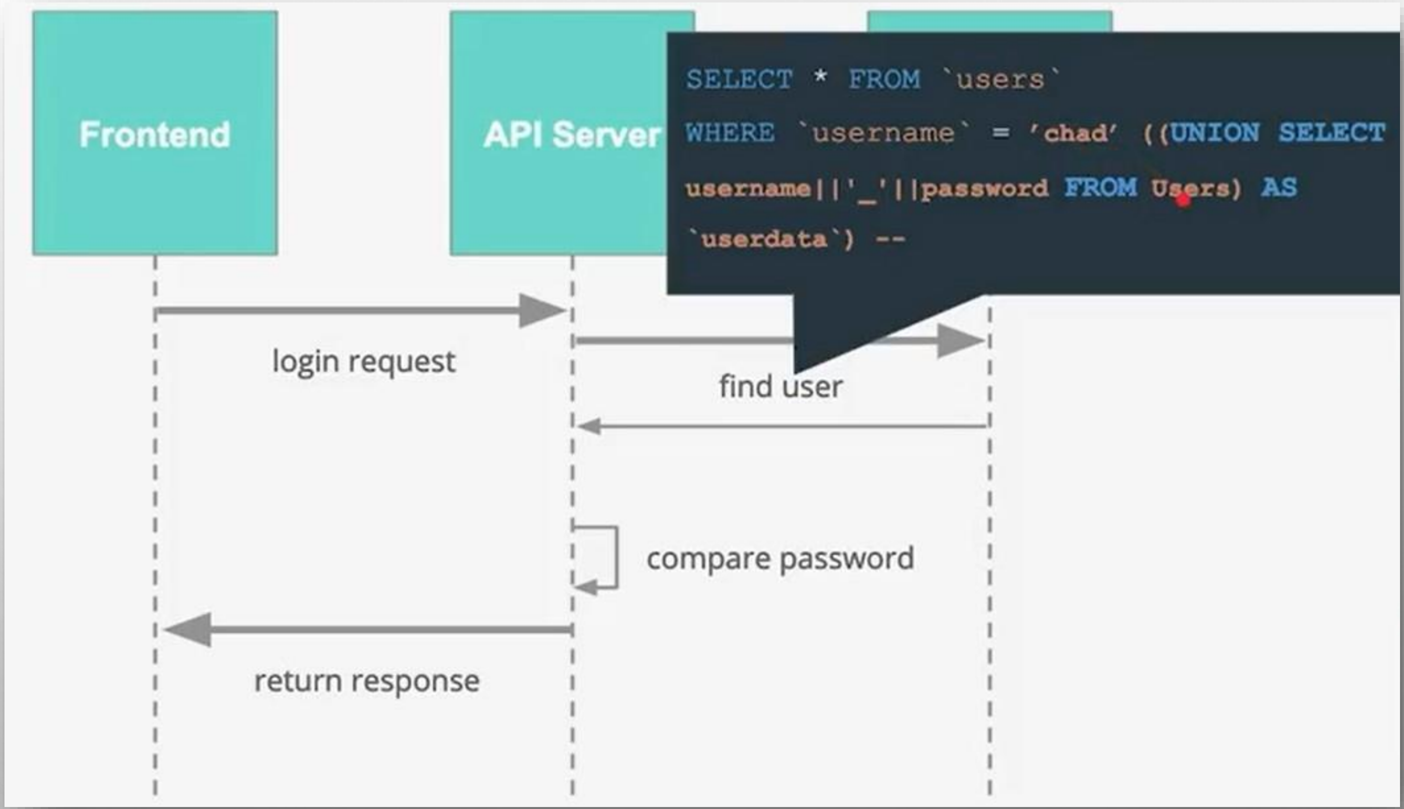
```
SELECT * FROM `users`
WHERE `username` = 'chad'; UPDATE `users`
SET `password` = '1234'
```

DEMO: SQL INJECTION



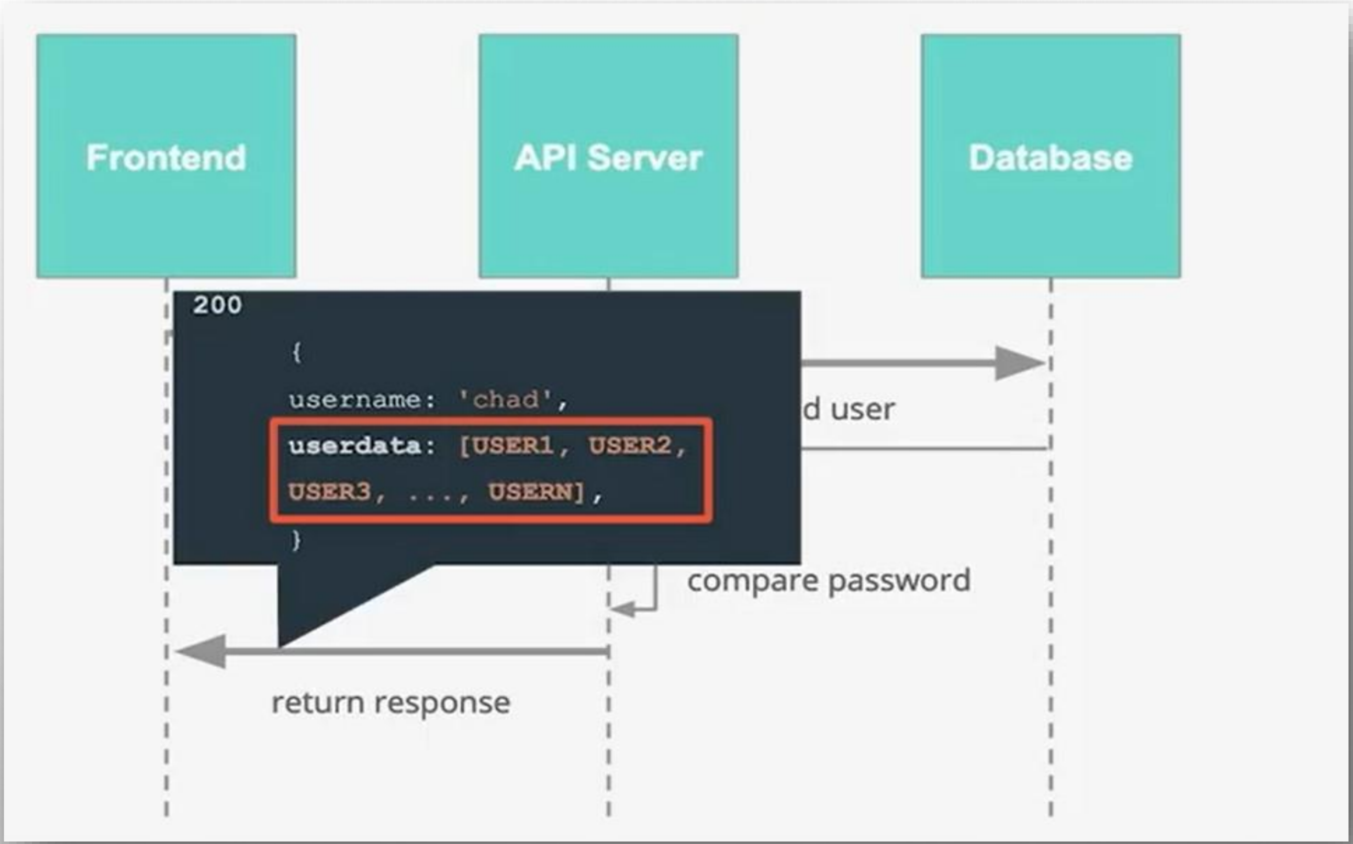
```
postuser = {
      username: 'chad' ((UNION SELECT
username||'_'||password FROM Users) AS
`userdata`) --',
      password: 'mrfluffles'
    }
```

DEMO: SQL INJECTION



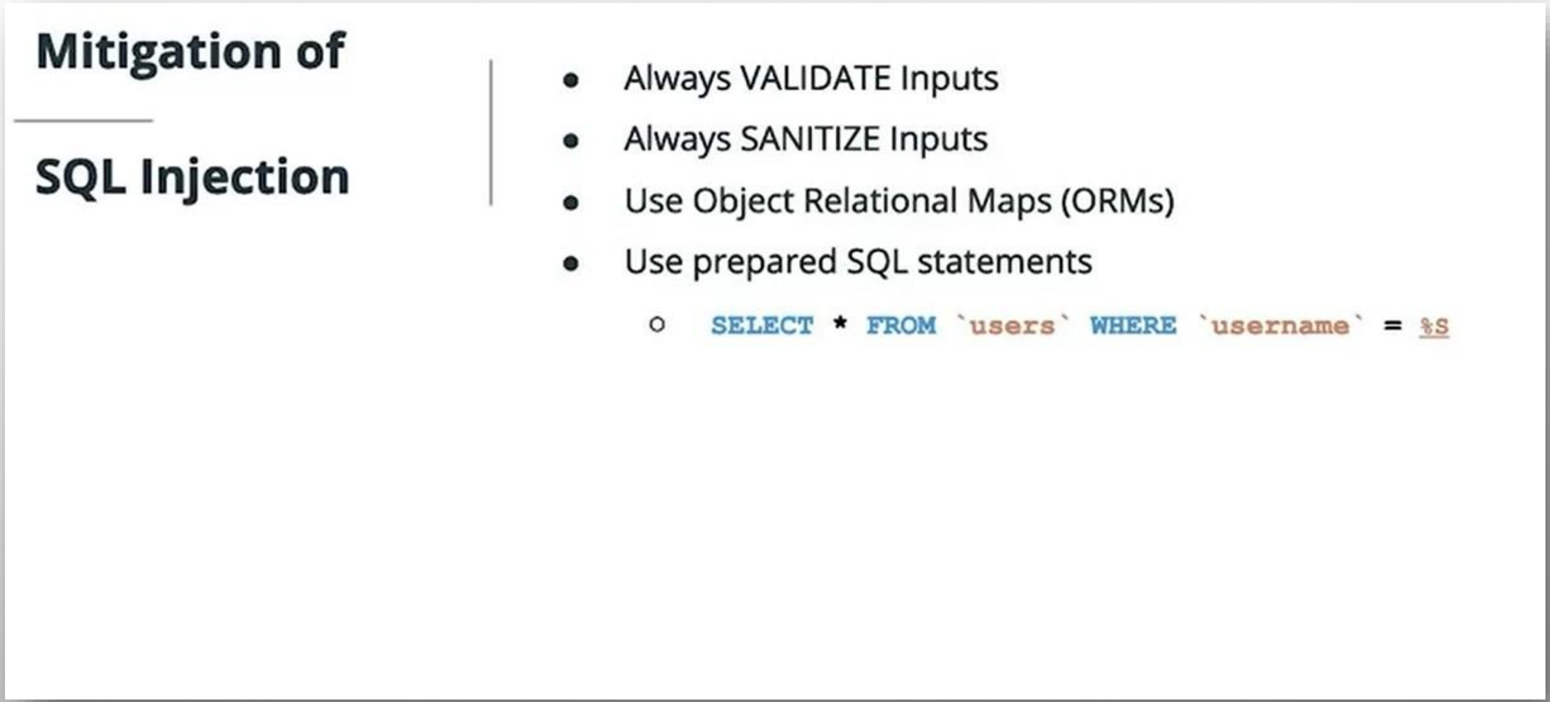
```
SELECT * FROM `users`
WHERE `username` = 'chad' ((UNION SELECT
username||'_'||password FROM Users) AS
`userdata`) --
```

DEMO: SQL INJECTION



```
200
        {
        username: 'chad',
        userdata: [USER1, USER2,
        USER3, ..., USERN],
        }
```

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 ``

RISKS INVOLVED WITH PLAIN TEXT

- **SQL INJECTION**
  - **SQL INJECTION IS THE PLACEMENT OF MALICIOUS CODE IN SQL STATEMENTS, VIA WEB PAGE INPUT.**
  - **HACKERS CAN USE SQL INJECTION TO GAIN UNAUTHORIZED ACCESS TO SENSITIVE DATA OR CONFIDENTIAL INFORMATION.**
  - **VULNERABLE APPLICATION TO DEMO SQL INJECTION DEMO**

03

ENCRYPTION & HASHING

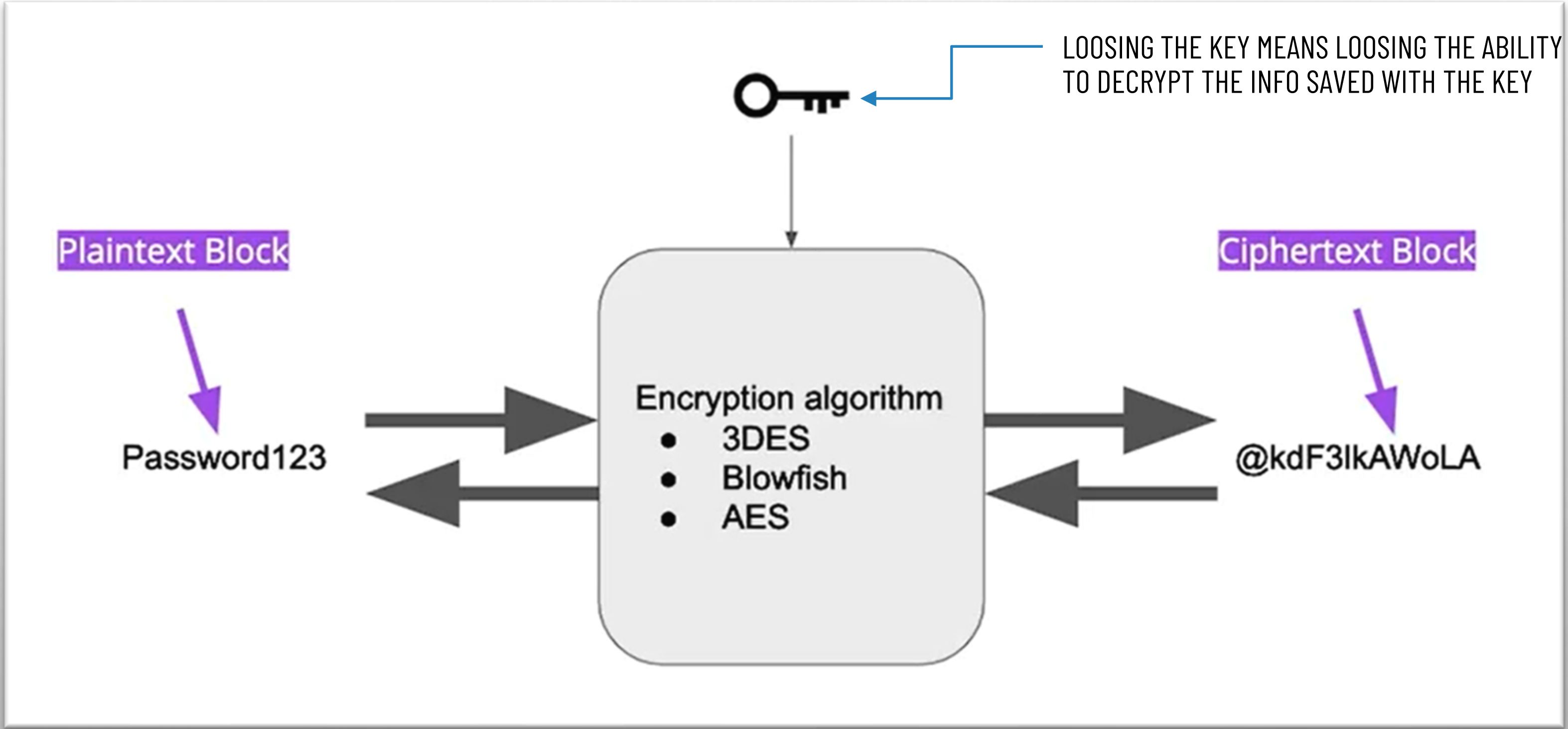
ENCRYPTION

- ENCRYPTION IS THE PROCESS OF ENCODING SIMPLE TEXT AND OTHER INFORMATION THAT CAN BE ACCESSED BY THE SOLE AUTHORIZED ENTITY IF IT HAS A DECRYPTION KEY.

- IT WILL PROTECT YOUR SENSITIVE DATA FROM BEING ACCESSED BY CYBERCRIMINALS. IT IS THE MOST EFFECTIVE WAY OF ACHIEVING DATA SECURITY IN MODERN COMMUNICATION SYSTEMS.

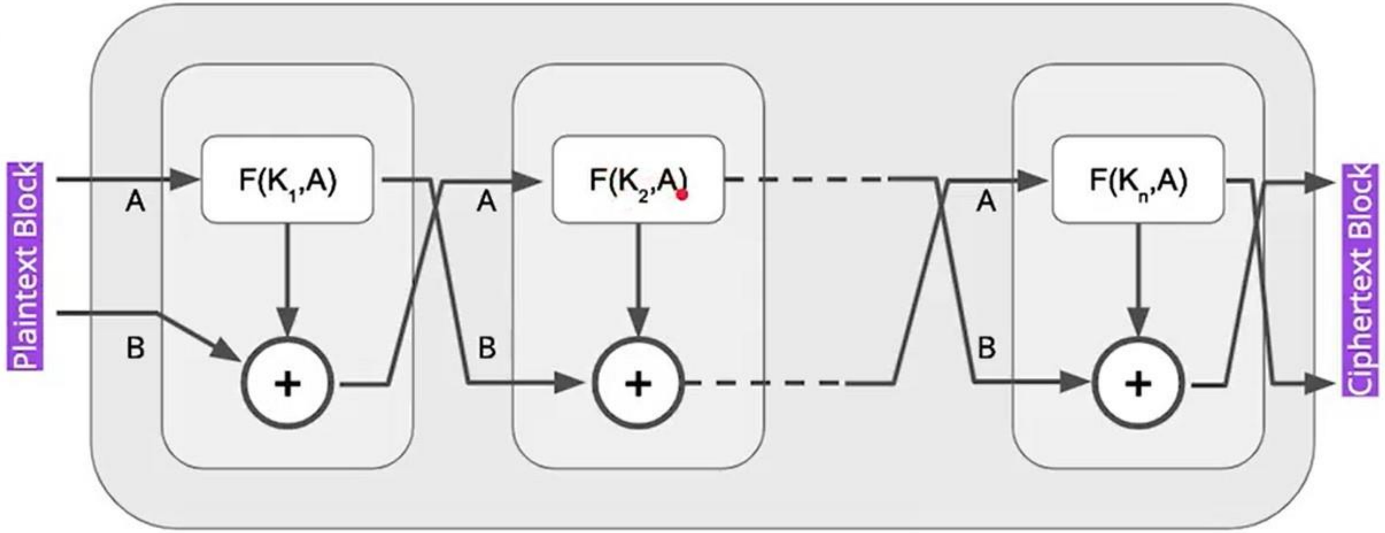
- IN ORDER FOR THE RECEIVER TO READ AN ENCRYPTED MESSAGE, HE/SHE SHOULD HAVE A PASSWORD OR A SECURITY KEY THAT IS USED IN DECRYPTION. DATA THAT HAS NOT BEEN ENCRYPTED IS KNOWN AS PLAIN TEXT WHILE ENCRYPTING DATA IS KNOWN AS A CIPHER TEXT.

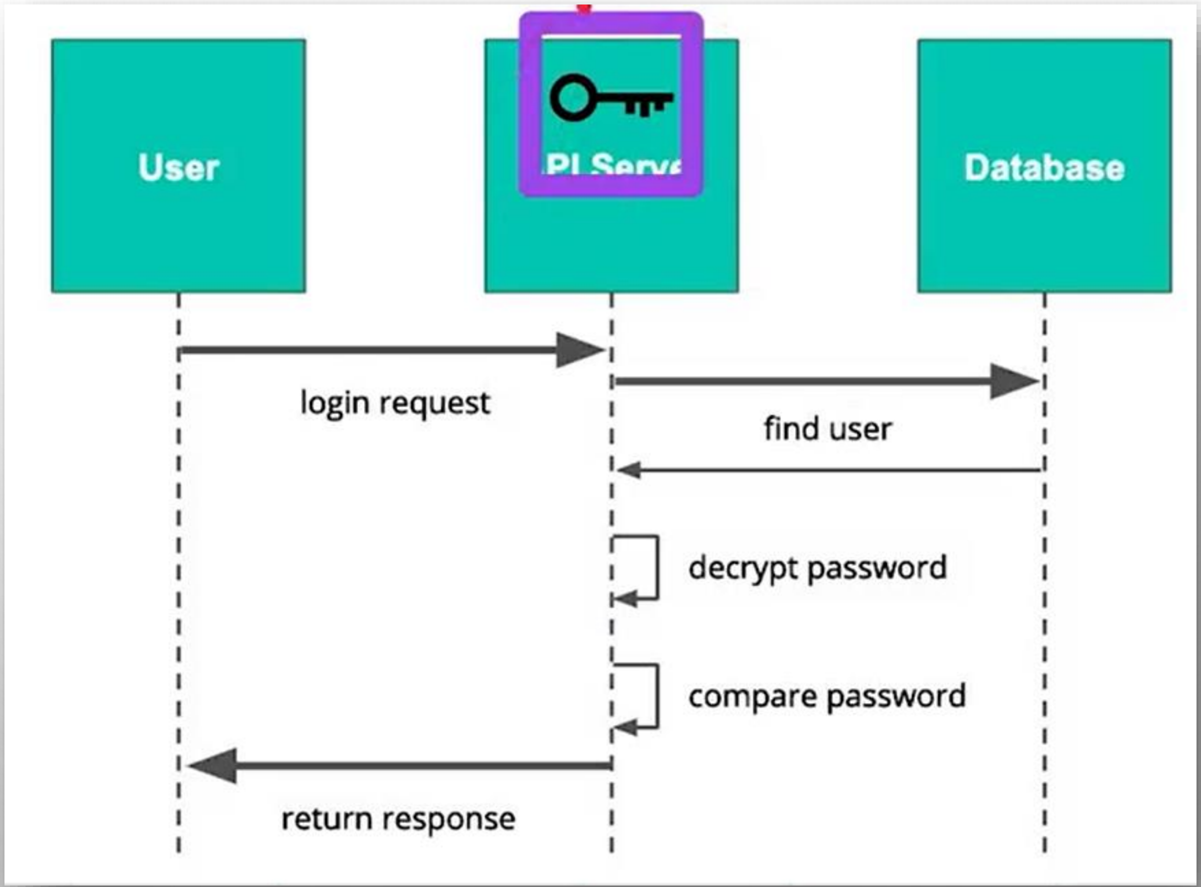
ENCRYPTION METHOD



Feistel Block Cipher

where k are round keys




User
PI Serve
Database
login request
find user
decrypt password
compare password
return response

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.

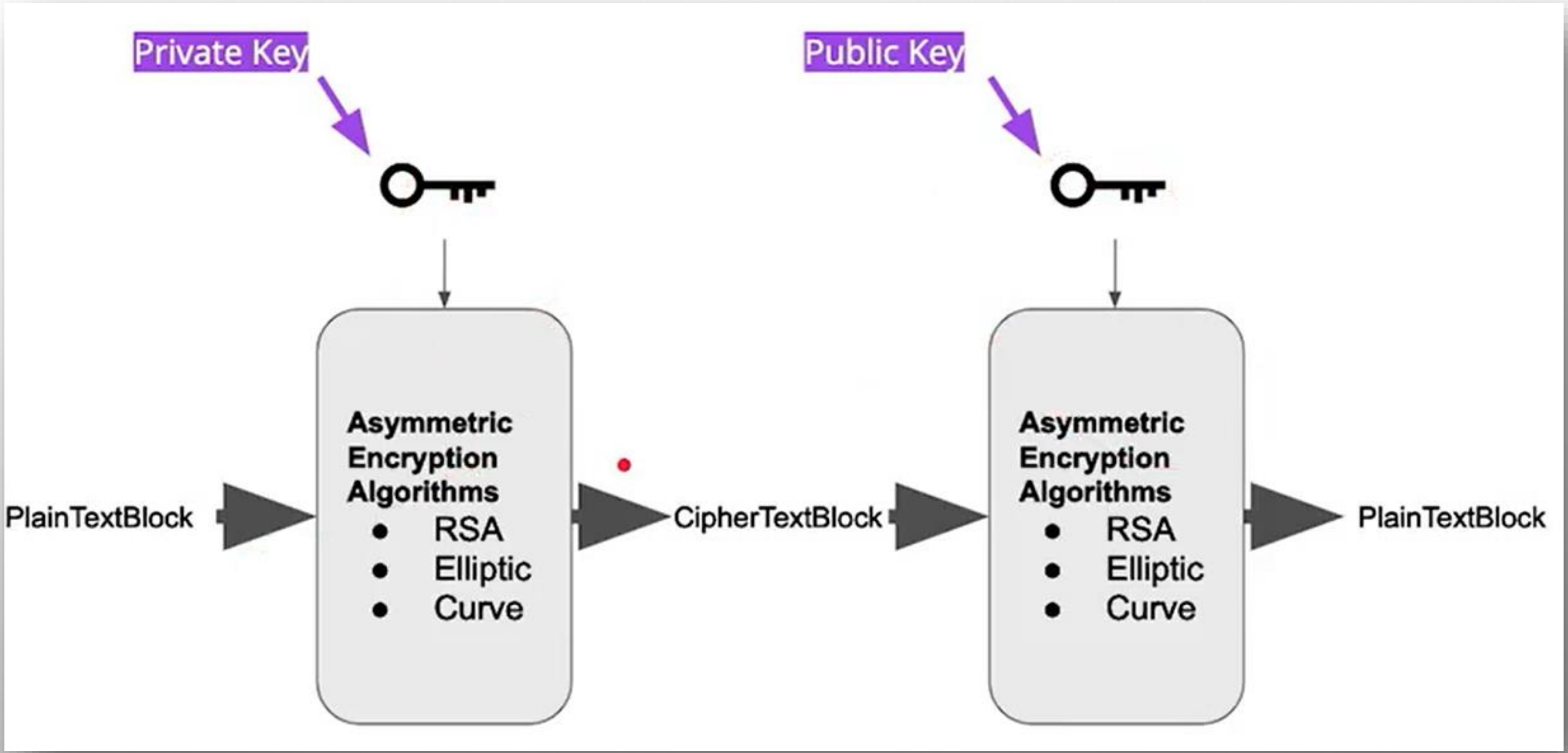
- **ASYMMETRIC ENCRYPTION** – IT DEPLOYS TWO KEYS, A PUBLIC KEY KNOWN BY EVERYONE AND A PRIVATE KEY KNOWN ONLY BY THE RECEIVER. THE PUBLIC KEY IS USED TO ENCRYPT THE MESSAGE AND A PRIVATE KEY IS USED TO DECRYPT IT. ASYMMETRIC ENCRYPTION IS LITTLE SLOWER THAN SYMMETRIC ENCRYPTION AND CONSUMES MORE PROCESSING 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.

KEY DIFFERENCES BTW OF CRYPTOGRAPHIC ALGORITHMS

- SYMMETRIC ENCRYPTION:
  - uSES A SINGLE KEY FOR BOTH ENCRYPTION AND DECRYPTION.
  - USED FOR PRIVACY AND CONFIDENTIALITY. (TO ENCRYPT DATA)
- ASYMMETRIC ENCRYPTION:
  - uSES ONE KEY FOR ENCRYPTION AND ANOTHER FOR DECRYPTION.
  - USED FOR AUTHENTICATION, KEY EXCHANGE (TO ENCRYPT SYMMETRIC KEYS DURING EXCHANGES) …

Asymmetric Block

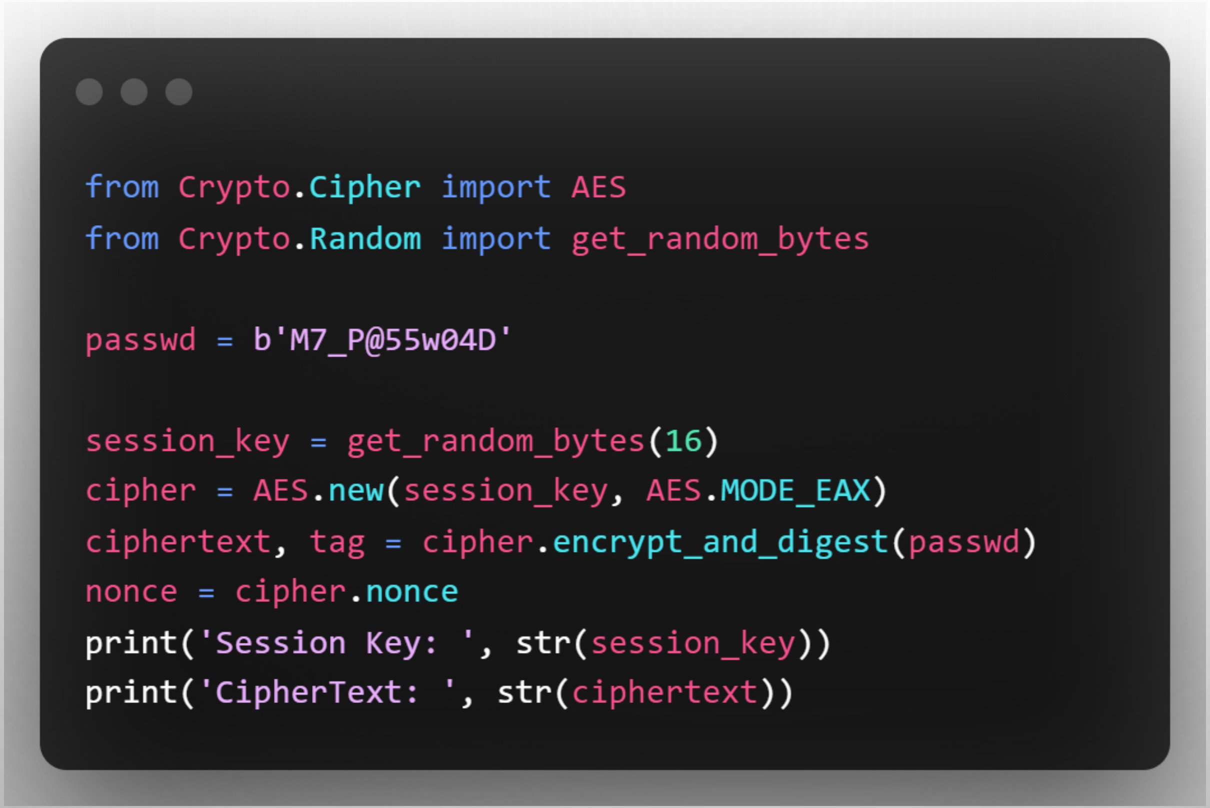


PURPOSE OF ENCRYPTION

THE MAIN IDEA OF ENCRYPTION IS TO PROTECT DATA FROM AN UNAUTHORIZED PERSON WHO WANTS TO READ OR GET INFORMATION FROM A MESSAGE THAT WAS NOT INTENDED FOR THEM. ENCRYPTION ENHANCES SECURITY WHEN SENDING MESSAGES THROUGH THE INTERNET OR THROUGH ANY GIVEN NETWORK. THE FOLLOWING ARE KEY ELEMENTS OF SECURITY THAT ENCRYPTION HELPS TO ENHANCE.

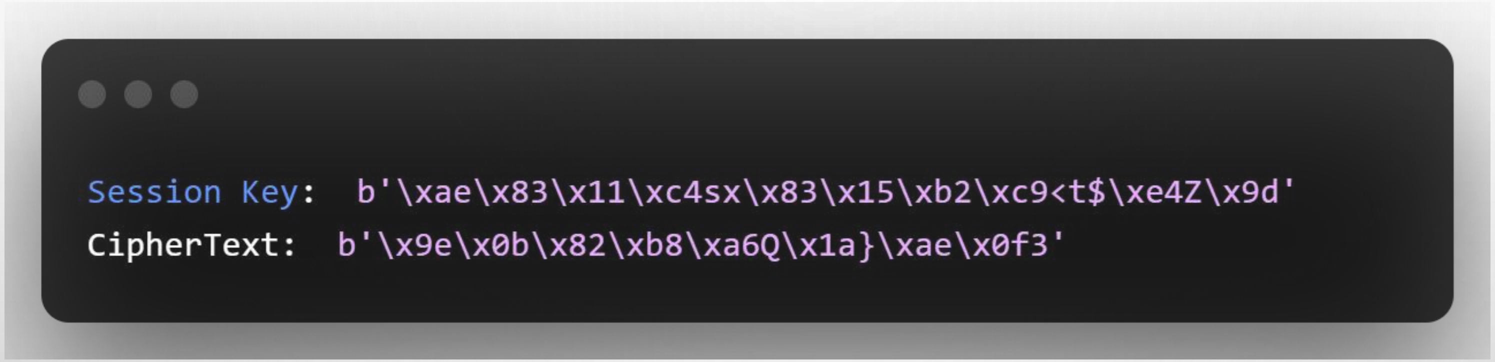
- **CONFIDENTIALITY** – ENCRYPTED MESSAGE CANNOT BE READ OR CHANGED BY ANOTHER PERSON.
- **ENCRYPT** – IT TRANSFORMS DATA IN SUCH A WAY THAT ONLY SPECIFIC INDIVIDUALS CAN TRANSFORM THE MESSAGE.
- **GRANULAR ACCESS CONTROL** – USERS ARE LIMITED TO WHAT THEY CAN SEE AND DO.
- IT MAKES AUDITING FOR ACCOUNTABILITY EASY. IN THE CASE OF MESSAGE LEAKED, IT IS EASY TO TRACE WHO DID THAT AND WHEN THUS SECURITY BREACHES CAN BE SORTED OUT EFFICIENTLY.
- **AUTHENTICATION** – THE ORIGIN OF THE MESSAGE RECEIVED CAN BE TRACED THUS FACILITATING AUTHENTICATION. SOME OF THE MOST POPULAR ENCRYPTION ALGORITHMS ARE **AES** AND **PGP.**

DEMO: SYMMETRIC ENCRYPTION (AES)



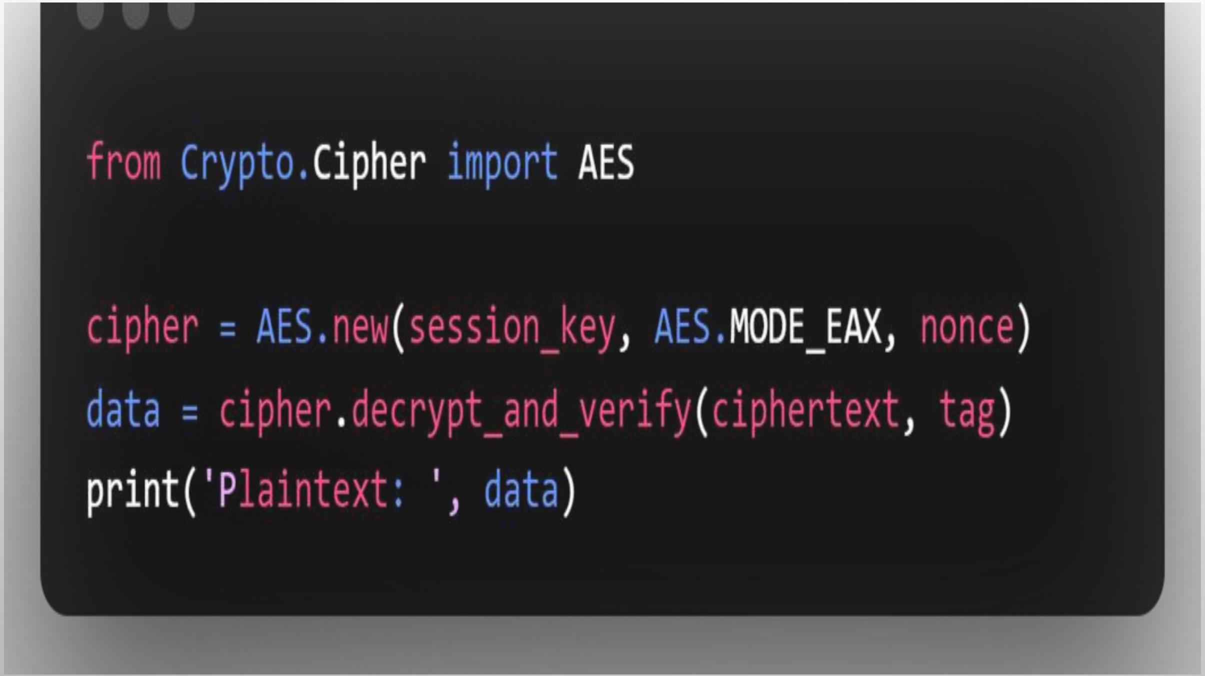
```python
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes

passwd = b'M7_P@55w04D'

session_key = get_random_bytes(16)
cipher = AES.new(session_key, AES.MODE_EAX)
ciphertext, tag = cipher.encrypt_and_digest(passwd)
nonce = cipher.nonce
print('Session Key: ', str(session_key))
print('CipherText: ', str(ciphertext))
```

OUTPUT: SYMMETRIC ENCRYPTION



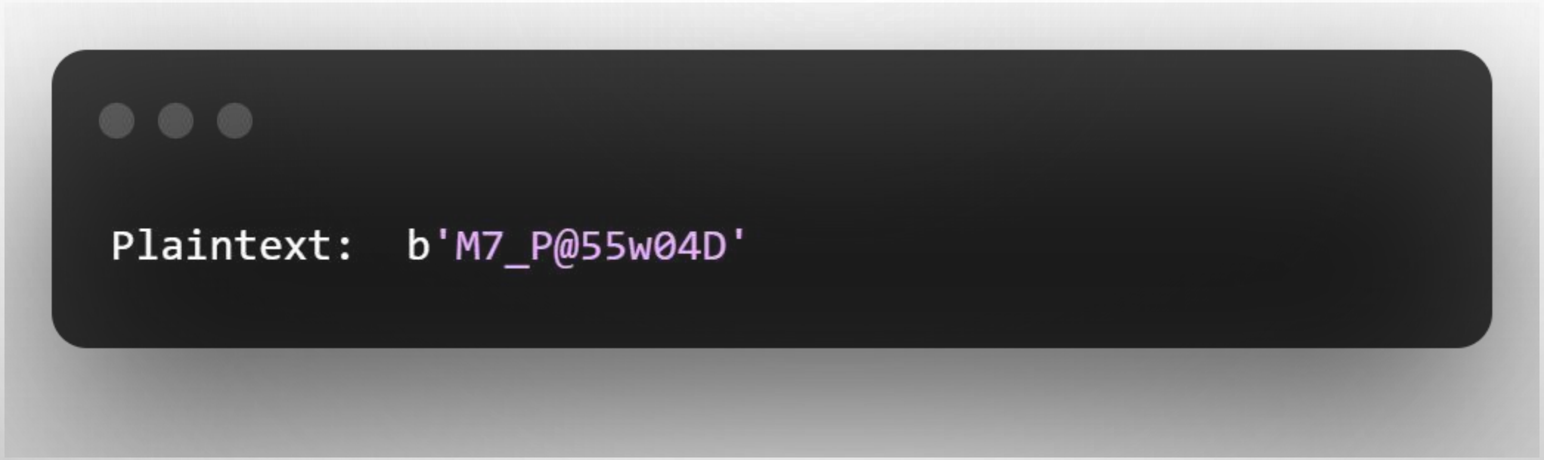
```
Session Key:  b'\xae\x83\x11\xc4sx\x83\x15\xb2\xc9<t$\xe4Z\x9d'
CipherText:  b'\x9e\x0b\x82\xb8\xa6Q\x1a}\xae\x0f3'
```

DEMO: SYMMETRIC DECRYPTION (AES)



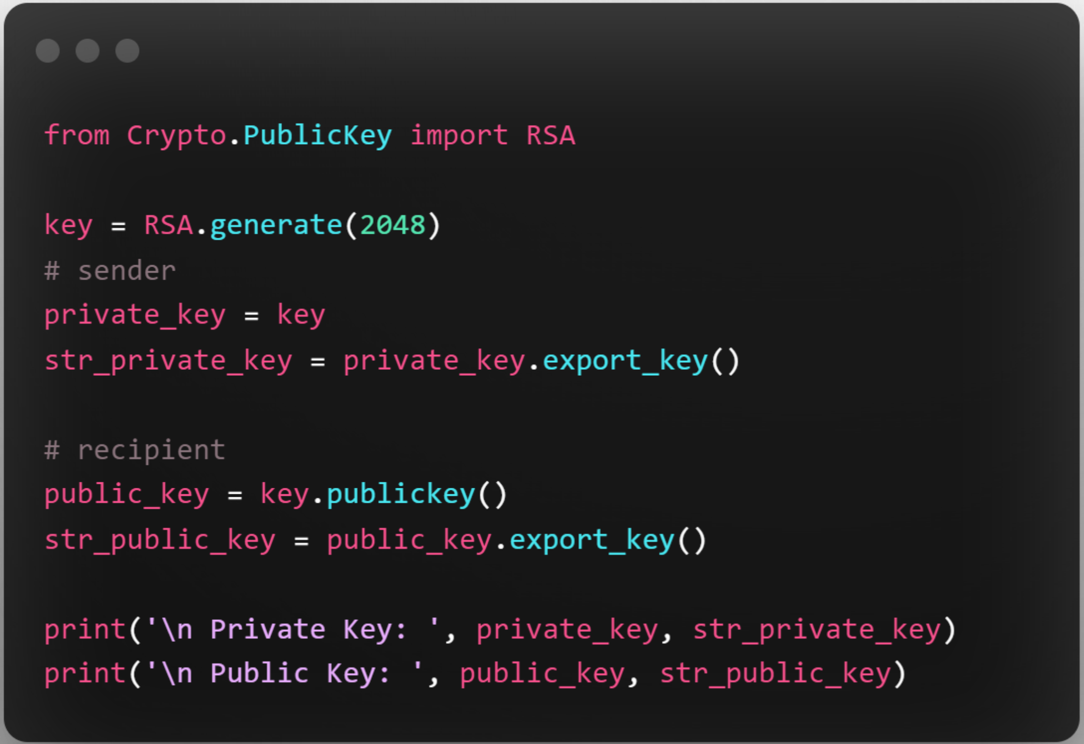
```python
from Crypto.Cipher import AES


cipher = AES.new(session_key, AES.MODE_EAX, nonce)
data = cipher.decrypt_and_verify(ciphertext, tag)
print('Plaintext: ', data)
```

OUTPUT: SYMMETRIC DECRYPTION



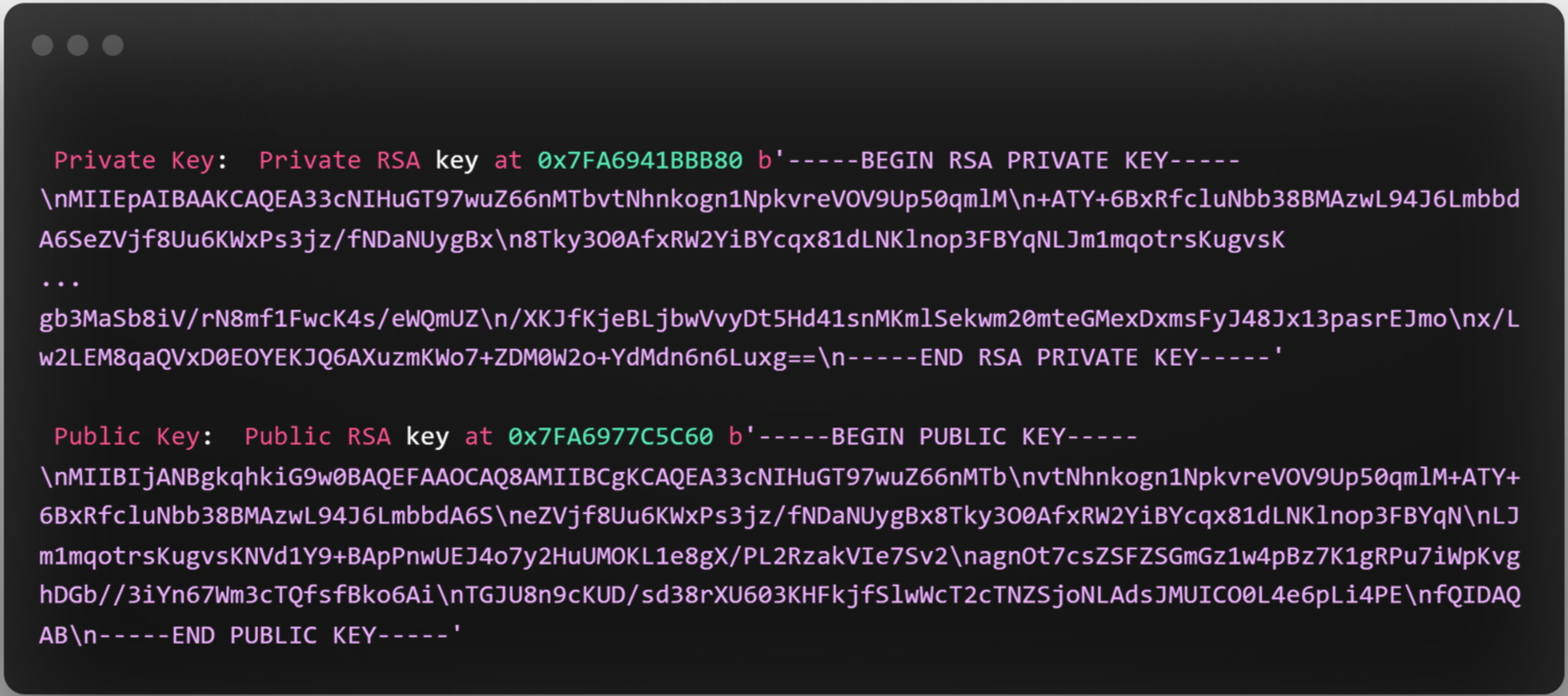
```
Plaintext:  b'M7_P@55w04D'
```

DEMO: GENERATING RSA KEYS



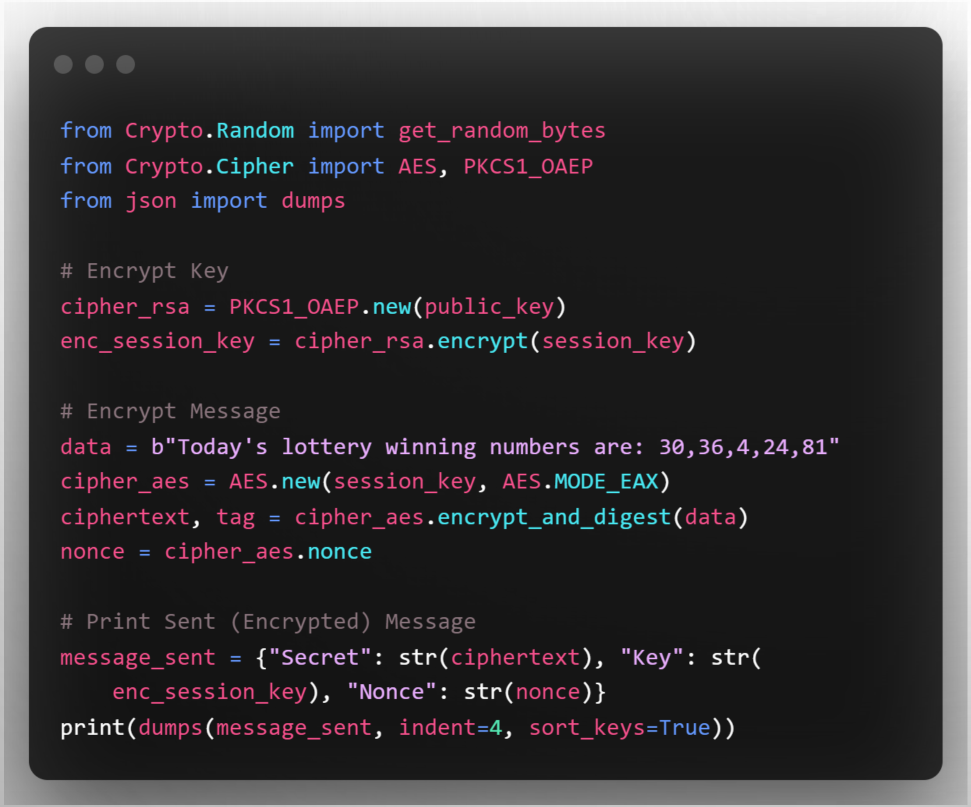
```
from Crypto.PublicKey import RSA

key = RSA.generate(2048)
# sender
private_key = key
str_private_key = private_key.export_key()

# recipient
public_key = key.publickey()
str_public_key = public_key.export_key()

print('\n Private Key: ', private_key, str_private_key)
print('\n Public Key: ', public_key, str_public_key)
```

OUTPUT: GENERATING RSA KEYS



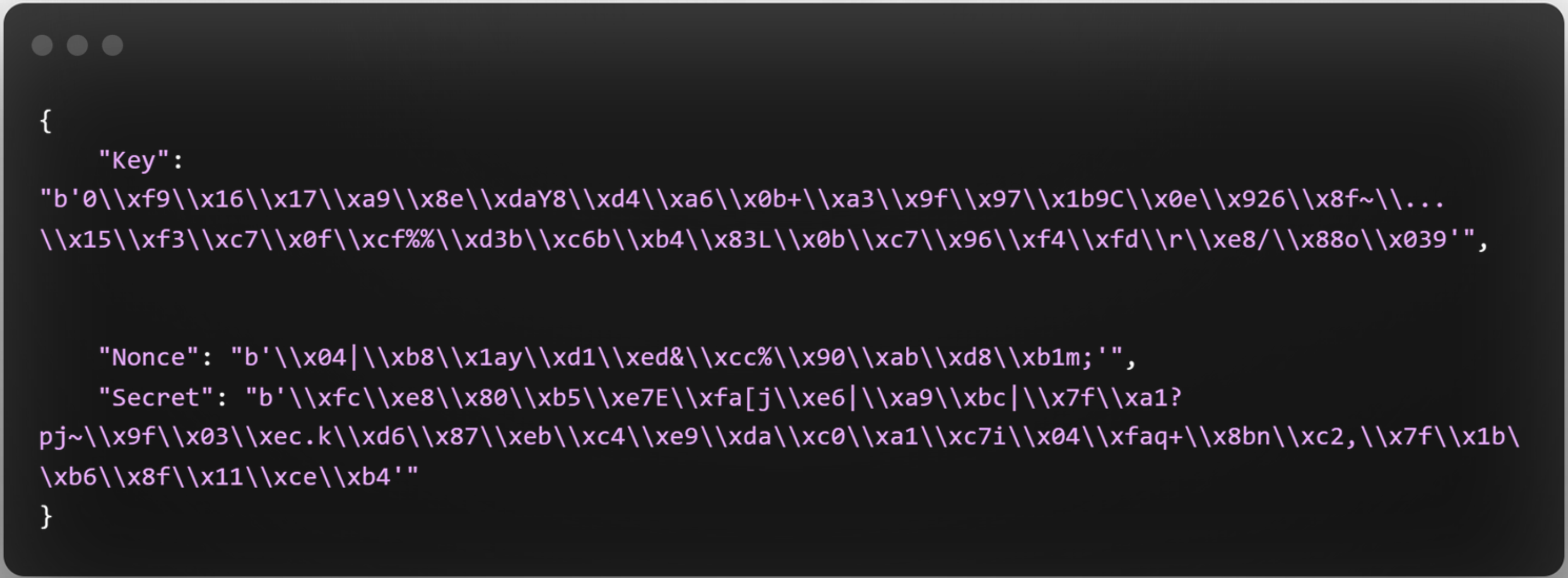
```
 Private Key:  Private RSA key at 0x7FA6941BBB80 b'-----BEGIN RSA PRIVATE KEY-----
\nMIIEpAIBAAKCAQEA33cNIHuGT97wuZ66nMTbvtNhnkogn1NpkvreVOV9Up50qmlM\n+ATY+6BxRfcluNbb38BMAzwL94J6Lmbbd
A6SeZVjf8Uu6KWxPs3jz/fNDaNUygBx\n8Tky3O0AfxRW2YiBYcqx81dLNKlnop3FBYqNLJm1mqotrsKugvsK
...
gb3MaSb8iV/rN8mf1FwcK4s/eWQmUZ\n/XKJfKjeBLjbwVvyDt5Hd41snMKmlSekwm20mteGMexDxmsFyJ48Jx13pasrEJmo\nx/L
w2LEM8qaQVxD0EOYEKJQ6AXuzmKWo7+ZDM0W2o+YdMdn6n6Luxg==\n-----END RSA PRIVATE KEY-----'

 Public Key:  Public RSA key at 0x7FA6977C5C60 b'-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA33cNIHuGT97wuZ66nMTb\nvtNhnkogn1NpkvreVOV9Up50qmlM+ATY+
6BxRfcluNbb38BMAzwL94J6LmbbdA6S\neZVjf8Uu6KWxPs3jz/fNDaNUygBx8Tky3O0AfxRW2YiBYcqx81dLNKlnop3FBYqN\nLJ
m1mqotrsKugvsKNVd1Y9+BApPnwUEJ4o7y2HuUMOKL1e8gX/PL2RzakVIe7Sv2\nagnOt7csZSFZSGmGz1w4pBz7K1gRPu7iWpKvg
hDGb//3iYn67Wm3cTQfsfBko6Ai\nTGJU8n9cKUD/sd38rXU603KHFkjfSlwWcT2cTNZSjoNLAdsJMUICO0L4e6pLi4PE\nfQIDAQ
AB\n-----END PUBLIC KEY-----'
```

DEMO: SENDING KEY & MESSAGE



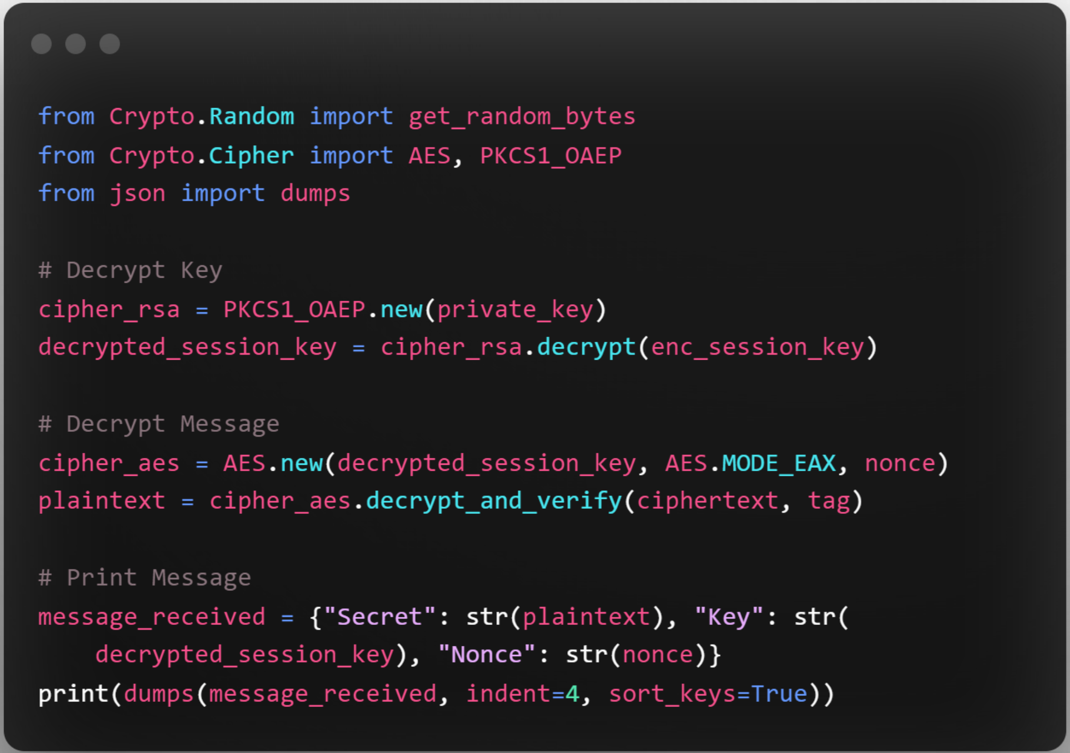
```python
from Crypto.Random import get_random_bytes
from Crypto.Cipher import AES, PKCS1_OAEP
from json import dumps

# Encrypt Key
cipher_rsa = PKCS1_OAEP.new(public_key)
enc_session_key = cipher_rsa.encrypt(session_key)

# Encrypt Message
data = b"Today's lottery winning numbers are: 30,36,4,24,81"
cipher_aes = AES.new(session_key, AES.MODE_EAX)
ciphertext, tag = cipher_aes.encrypt_and_digest(data)
nonce = cipher_aes.nonce

# Print Sent (Encrypted) Message
message_sent = {"Secret": str(ciphertext), "Key": str(
    enc_session_key), "Nonce": str(nonce)}
print(dumps(message_sent, indent=4, sort_keys=True))
```

OUTPUT: SENDING KEY & MESSAGE



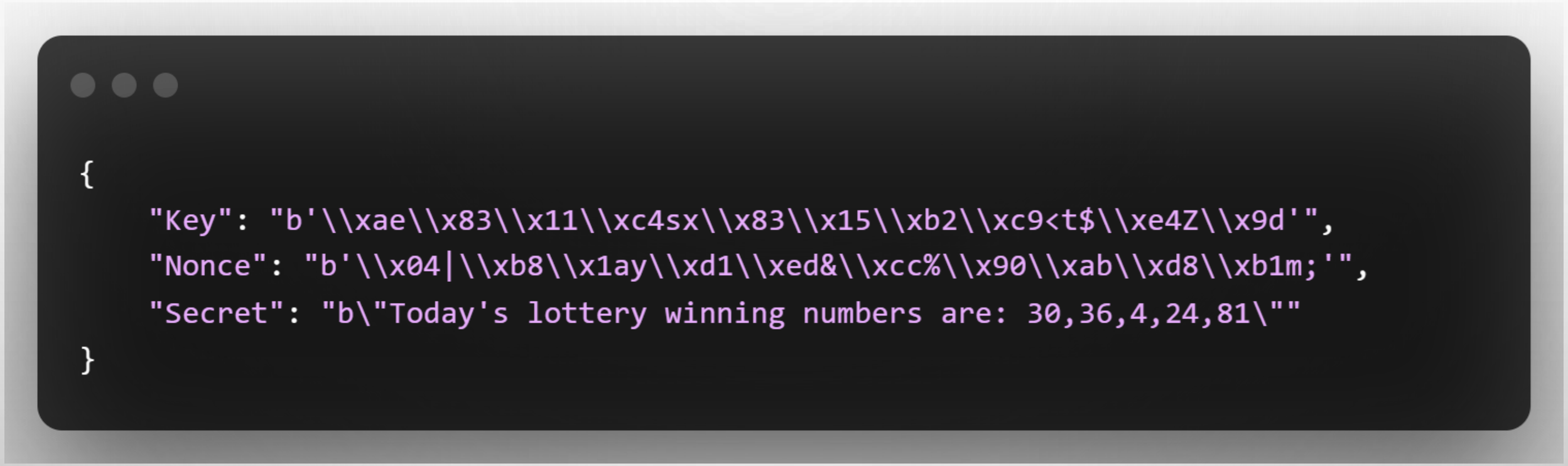
```
{
    "Key":
"b'0\\xf9\\x16\\x17\\xa9\\x8e\\xdaY8\\xd4\\xa6\\x0b+\\xa3\\x9f\\x97\\x1b9C\\x0e\\x926\\x8f~\\...
\\x15\\xf3\\xc7\\x0f\\xcf%%\\xd3b\\xc6b\\xb4\\x83L\\x0b\\xc7\\x96\\xf4\\xfd\\r\\xe8/\\x88o\\x039'",


    "Nonce": "b'\\x04|\\xb8\\x1ay\\xd1\\xed&\\xcc%\\x90\\xab\\xd8\\xb1m;'",
    "Secret": "b'\\xfc\\xe8\\x80\\xb5\\xe7E\\xfa[j\\xe6|\\xa9\\xbc|\\x7f\\xa1?
pj~\\x9f\\x03\\xec.k\\xd6\\x87\\xeb\\xc4\\xe9\\xda\\xc0\\xa1\\xc7i\\x04\\xfaq+\\x8bn\\xc2,\\x7f\\x1b\
\xb6\\x8f\\x11\\xce\\xb4'"
}
```

DEMO: RECEIVING KEY & MESSAGE



```python
from Crypto.Random import get_random_bytes
from Crypto.Cipher import AES, PKCS1_OAEP
from json import dumps

# Decrypt Key
cipher_rsa = PKCS1_OAEP.new(private_key)
decrypted_session_key = cipher_rsa.decrypt(enc_session_key)

# Decrypt Message
cipher_aes = AES.new(decrypted_session_key, AES.MODE_EAX, nonce)
plaintext = cipher_aes.decrypt_and_verify(ciphertext, tag)

# Print Message
message_received = {"Secret": str(plaintext), "Key": str(
    decrypted_session_key), "Nonce": str(nonce)}
print(dumps(message_received, indent=4, sort_keys=True))
```

OUTPUT: RECEIVING KEY & MESSAGE

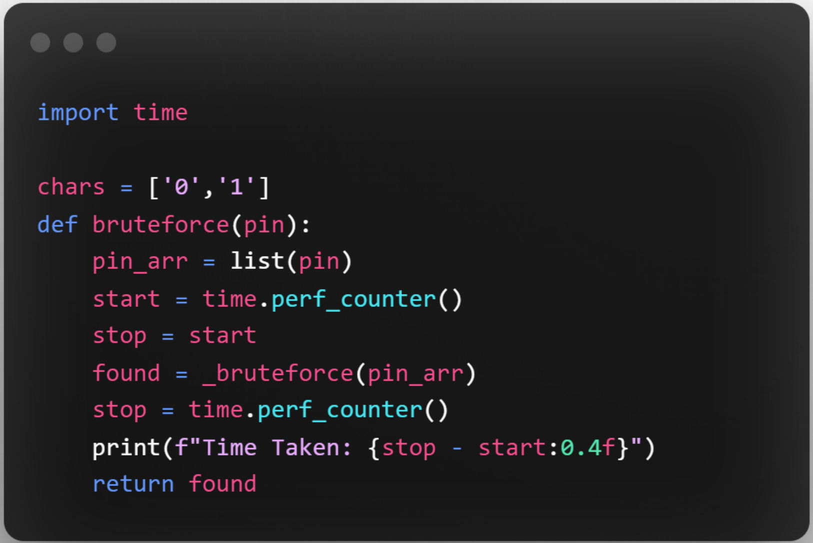


```
{
    "Key": "b'\\xae\\x83\\x11\\xc4sx\\x83\\x15\\xb2\\xc9<t$\\xe4Z\\x9d'",
    "Nonce": "b'\\x04|\\xb8\\x1ay\\xd1\\xed&\\xcc%\\x90\\xab\\xd8\\xb1m;'",
    "Secret": "b\"Today's lottery winning numbers are: 30,36,4,24,81\""
}
```

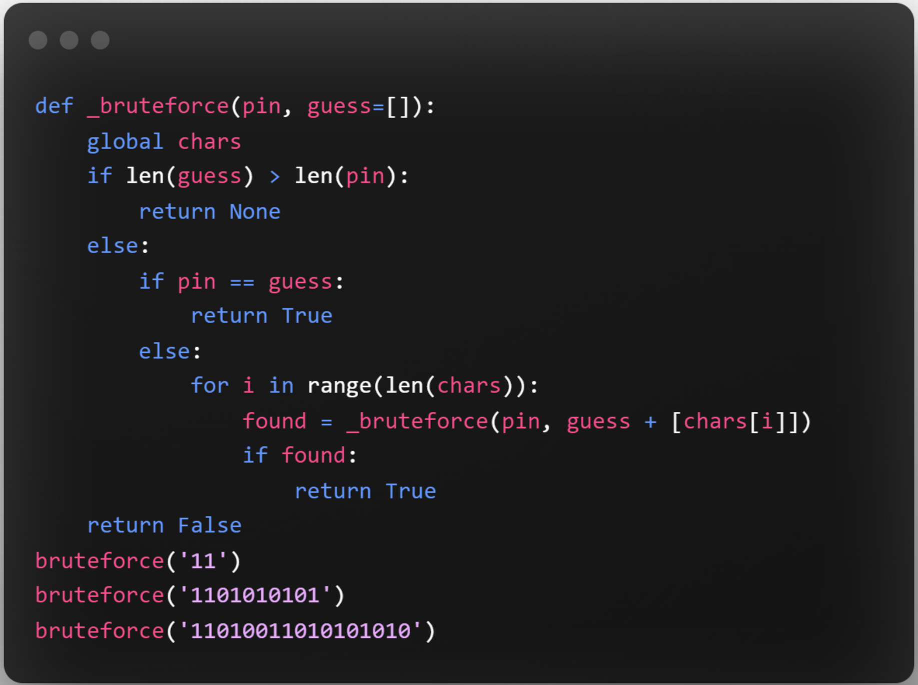
PROBLEM: BRUTE FORCE ATTACK

- A BRUTE FORCE ATTACK USES TRIAL-AND-ERROR TO GUESS LOGIN INFO.
- IT WORKS BY CYCLING THROUGH ALL POSSIBLE COMBINATIONS OF LETTERS TO TRY AND GUESS THE CORRECT PASSWORD.
- A BAD ACTOR COULD INTERCEPT AN ENCRYPTED PASSWORD AND TRY TO GUESS THE PLAINTEXT BY TRYING TO ENCRYPT NUMEROUS GUESSES AND USING THE ONE THAT PRODUCES THE SAME CIPHERTEXT AS THE INTERCEPTED PASSWORD.
- ALTERNATIVELY, THEY COULD TRY TO LOGIN TO AN APPLICATION USING NUMEROUS GUESSED PASSWORDS.
- THE AMOUNT OF TIME IT WILL TAKE TO FIGURE OUT THE CORRECT PASSWORD DEPENDS ON THE LENGTH AND COMPLEXITY OF THE PASSWORD.
    - PASSWORDS WITH MORE CHARACTERS TAKE LONGER TO FIGURE OUT.
    - PASSWORDS WITH A WIDER VARIETY OF CHARACTERS (ALPHABET, NUMERALS, SYMBOLS …) TAKE LONGER TO FIGURE OUT.

DEMO: BRUTE FORCE ATTACK

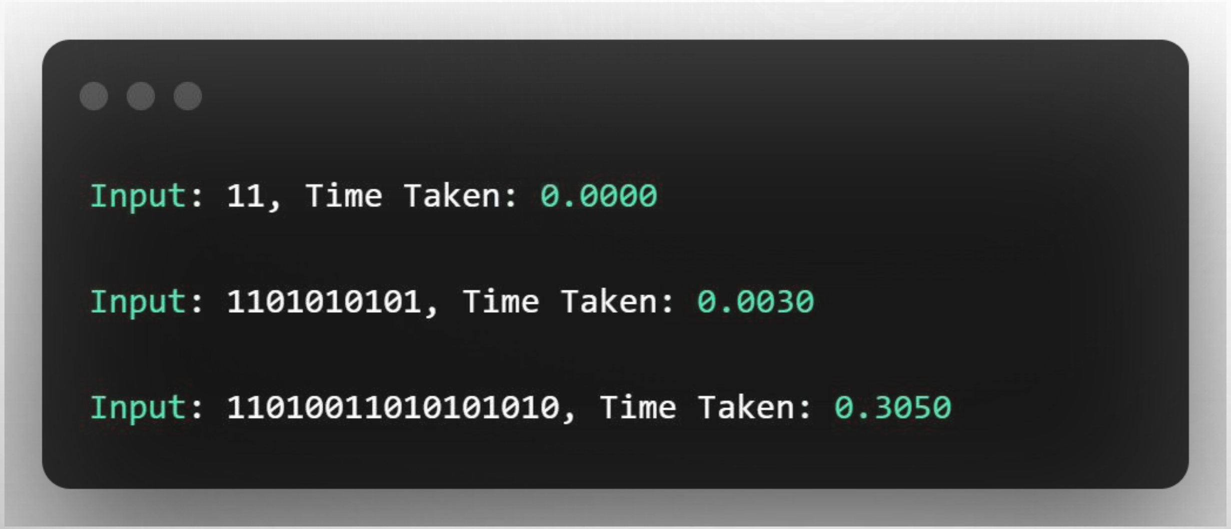


```python
import time

chars = ['0','1']
def bruteforce(pin):
    pin_arr = list(pin)
    start = time.perf_counter()
    stop = start
    found = _bruteforce(pin_arr)
    stop = time.perf_counter()
    print(f"Time Taken: {stop - start:0.4f}")
    return found
```



```python
def _bruteforce(pin, guess=[]):
    global chars
    if len(guess) > len(pin):
        return None
    else:
        if pin == guess:
            return True
        else:
            for i in range(len(chars)):
                found = _bruteforce(pin, guess + [chars[i]])
                if found:
                    return True
    return False
bruteforce('11')
bruteforce('1101010101')
bruteforce('110100110101010l0')
```

BRUTE FORCE ATTACK OUTPUT



```
Input: 11, Time Taken: 0.0000

Input: 1101010101, Time Taken: 0.0030

Input: 11010011010101010, Time Taken: 0.3050
```

REMEDY FOR BRUTE FORCE ATTACKS

- RATE LIMIT LOGIN ATTEMPTS
- DO NOT ALLOW SIMPLE PASSWORDS
- IMPLEMENT CAPTCHAS
- LOG AND MONITOR ATTACKS
- DO NOT ENCRYPT AND STORE PASSWORDS, THIS LEAVES THE POSSIBILITY OF BAD ACTORS DECRYPTING THE PASSWORDS IF THEY ACQUIRE THE DATABASE.
    - INSTEAD, HASH THE PASSWORDS BECAUSE HASH ALGORITHMS ARE NOT REVERSIBLE.

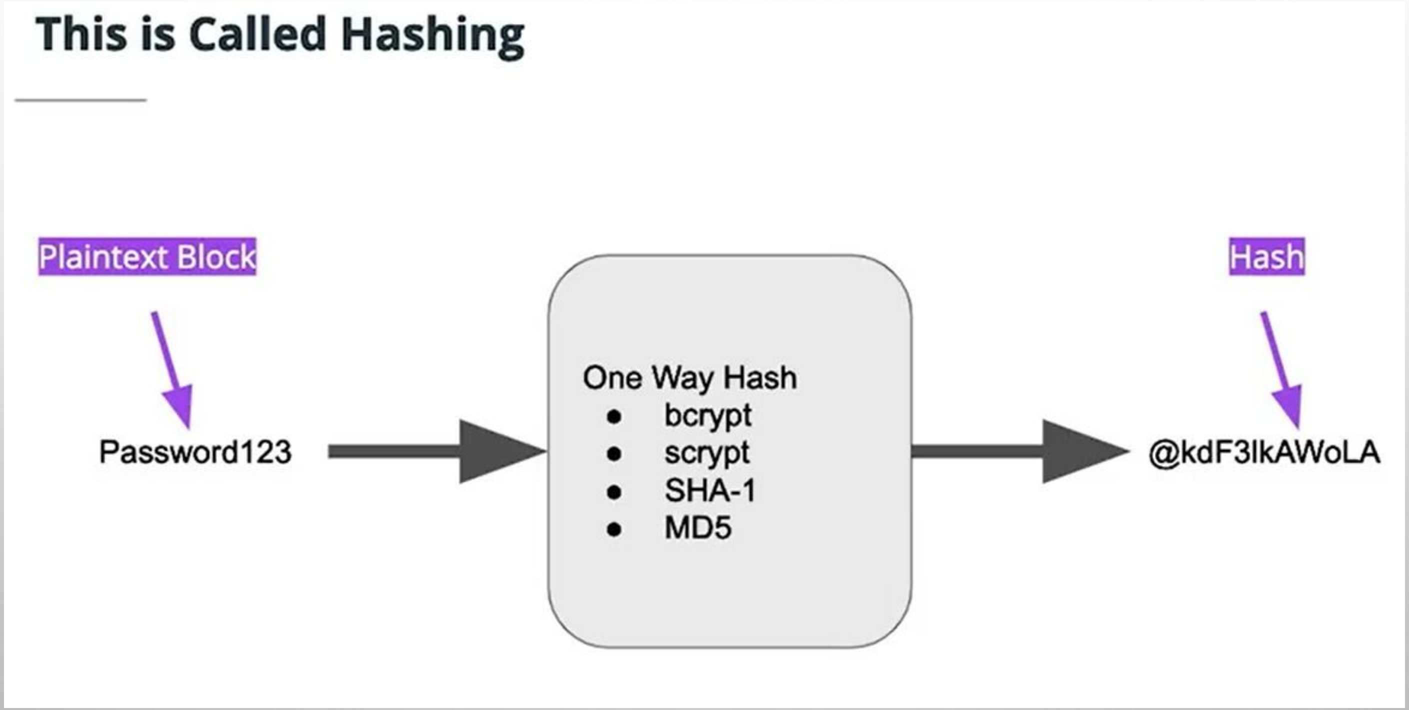
HASHING

A HASH CAN SIMPLY BE DEFINED AS A NUMBER GENERATED FROM A STRING OF TEXT. ALSO KNOWN AS MESSAGE DIGEST.

IT IS GENERATED IN A WAY THAT A SIMILAR HASH WITH THE SAME VALUE CANNOT BE PRODUCED BY ANOTHER TEXT

HASHING IS THE PROCESS OF PRODUCING HASH VALUES FOR THE PURPOSE OF ACCESSING DATA AND FOR SECURITY REASONS IN COMMUNICATION SYSTEMS.

One-Way Technique reduces the risk associated with encryption



RULE OF THUMB, HASHING WILL HAVE THE FOLLOWING ATTRIBUTES:

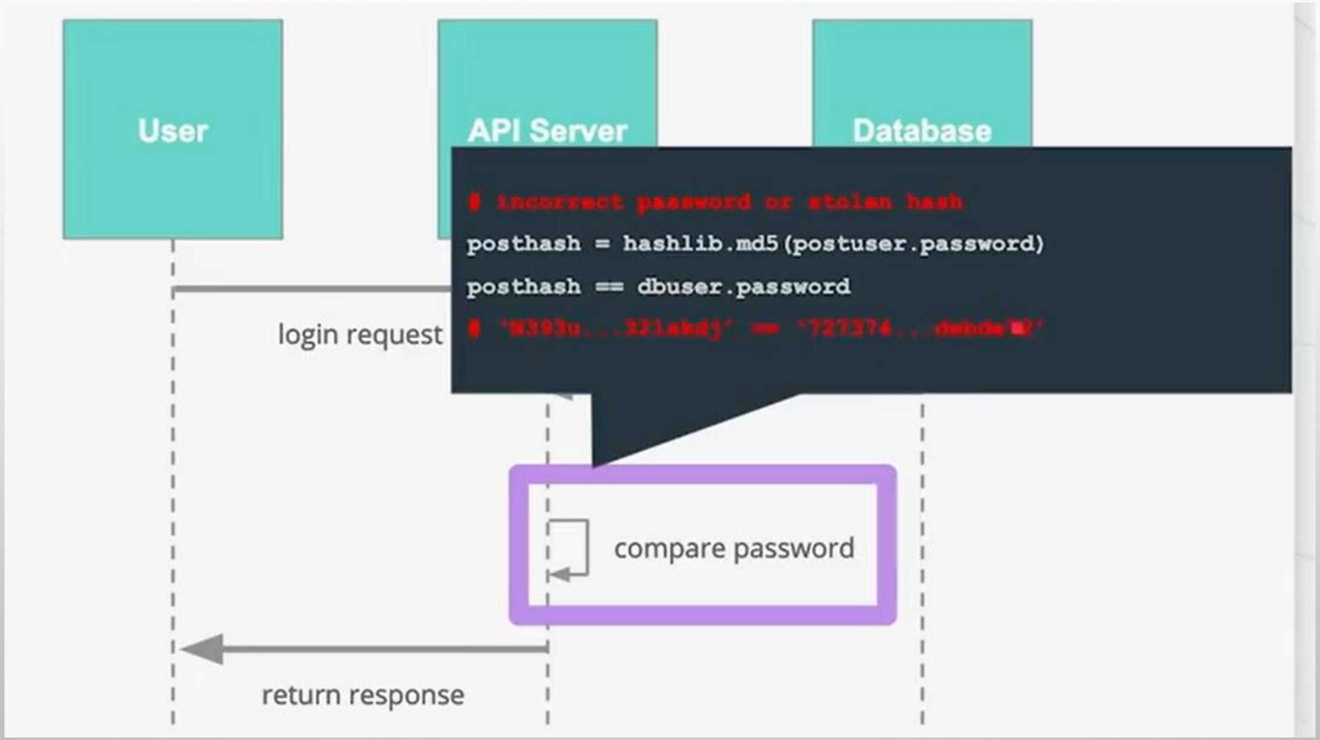
- ❖ A GIVEN KNOWN INPUT MUST ALWAYS PRODUCE ONE KNOWN OUTPUT.
- ❖ ONCE HASHING HAS BEEN DONE, IT SHOULD BE IMPOSSIBLE TO GO FROM THE OUTPUT TO THE INPUT.
- ❖ DIFFERENT MULTIPLE INPUTS SHOULD GIVE A DIFFERENT OUTPUT.
- ❖ MODIFYING AN INPUT SHOULD MEAN A CHANGE IN THE HASH.

HASH ALGORITHM

A HASH ALGORITHM IS A FUNCTION THAT CAN BE USED TO MAP OUT DATA OF RANDOM SIZE TO DATA OF FIXED SIZE.

- HASH VALUES,
- HASH CODES AND
- HASH SUMS ARE RETURNED BY FUNCTIONS DURING HASHING.

Hash Demo



```
# incorrect password or stolen hash
posthash = hashlib.md5(postuser.password)
posthash == dbuser.password
# 'N393u...321skdj' == '727374...dsbde82'
```

PURPOSE

THE MOST POPULAR USE FOR HASHING IS THE IMPLEMENTATION OF HASH TABLES.

- A HASH TABLE STORES KEY AND VALUE PAIRS IN A LIST THAT IS ACCESSIBLE THROUGH ITS INDEX.
- BECAUSE KEY AND VALUE PAIRS ARE UNLIMITED, THE HASH FUNCTION WILL MAP THE KEYS TO THE TABLE SIZE.
- A HASH VALUE THEN BECOMES THE INDEX FOR A SPECIFIC ELEMENT.

These characteristics mean that hash can be used to store passwords. This way, it becomes difficult for someone who has the raw data to reverse them.

APPLICATION

- ✓ HASHING CAN BE USED TO COMPARE A LARGE AMOUNT OF DATA. HASH VALUES CAN BE CREATED FOR DIFFERENT DATA, MEANING THAT IT IS EASIER COMPARING HASHES THAN THE DATA ITSELF.
- ✓ IT IS EASY TO FIND A RECORD WHEN THE DATA IS HASHED.
- ✓ HASHING ALGORITHMS ARE USED IN CRYPTOGRAPHIC APPLICATIONS LIKE A DIGITAL SIGNATURE.
- ✓ HASHING IS USED TO GENERATE RANDOM STRINGS TO AVOID DUPLICATION OF DATA STORED IN DATABASES.
- ✓ GEOMETRIC HASHING – WIDELY USED IN COMPUTER GRAPHICS TO FIND CLOSET PAIRS AND PROXIMITY PROBLEMS IN PLANES. IT IS ALSO CALLED GRID METHOD AND IT HAS ALSO BEEN ADOPTED IN TELECOMMUNICATIONS.

PROBLEM: RAINBOW TABLES

A RAINBOW TABLE IS A PRECOMPUTED TABLE FOR CACHING THE OUTPUT OF CRYPTOGRAPHIC HASH FUNCTIONS, USUALLY FOR CRACKING PASSWORD HASHES. TABLES ARE USUALLY USED IN RECOVERING A KEY DERIVATION FUNCTION (OR CREDIT CARD NUMBERS, ETC.) UP TO A CERTAIN LENGTH CONSISTING OF A LIMITED SET OF CHARACTERS.

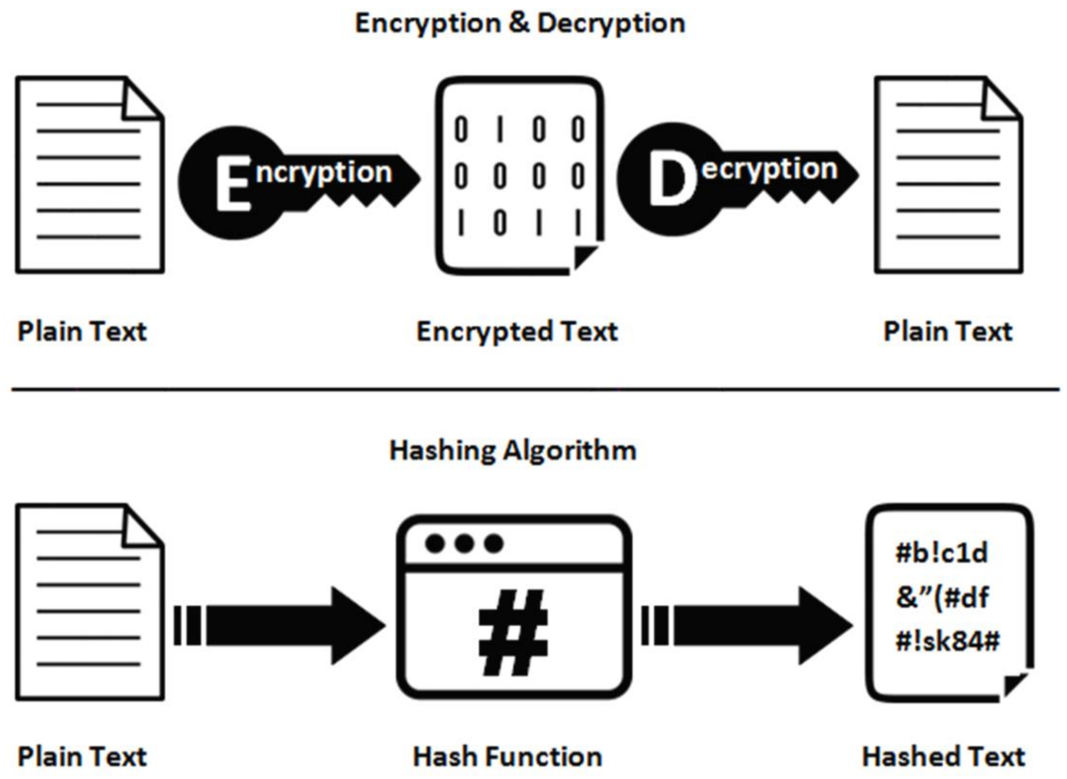
USE OF A KEY DERIVATION THAT EMPLOYS A SALT MAKES THIS ATTACK INFEASIBLE.

REMEDY: RAINBOW TABLES

USE OF A KEY DERIVATION THAT EMPLOYS A SALT MAKES THIS ATTACK INFEASIBLE.

- SALTING OUR PASSWORDS USING A RANDOMLY GENERATED STRING
- IT GETS ADDED TO THE INPUT WE WANT TO HASH
- SO NOW INSTEAD OF PASS123 => PASS123XYZWTY
- RESULTING IN A MISMATCH IN THE RAINBOW TABLE

The difference between hashing and encryption



PRACTICAL

Brute force –Rainbow table

Salt

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config['SQLALCHEMY_DATABASE_URI'] =
'postgresql://postgres:abc@localhost:5432/example'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

class User(db.Model):
  __tablename__ = 'users'
  id = db.Column(db.Integer, primary_key=True)
  name = db.Column(db.String(), nullable=False)

  def __repr__(self):
      return f'<User {self.id}, {self.name}>'


db.create_all()
```

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

ADDITIONAL RESOURCES

- WHAT IS MITM (MAN IN THE MIDDLE) ATTACK?
- 3 MINUTE YOUTUBE VIDEO ON WHAT IS A MAN IN THE MIDDLE ATTACK
- 2 MINUTE YOUTUBE VIDEO ON WHAT IS SQL INJECTION?
- DATABASES AND SECURITY ISSUES
- 22 CYBERSECURITY MYTHS ORGANIZATIONS NEED TO STOP BELIEVING IN 2022
- THE LIMITATIONS AND SECURITY OF LOCALSTORAGE IN JAVASCRIPT