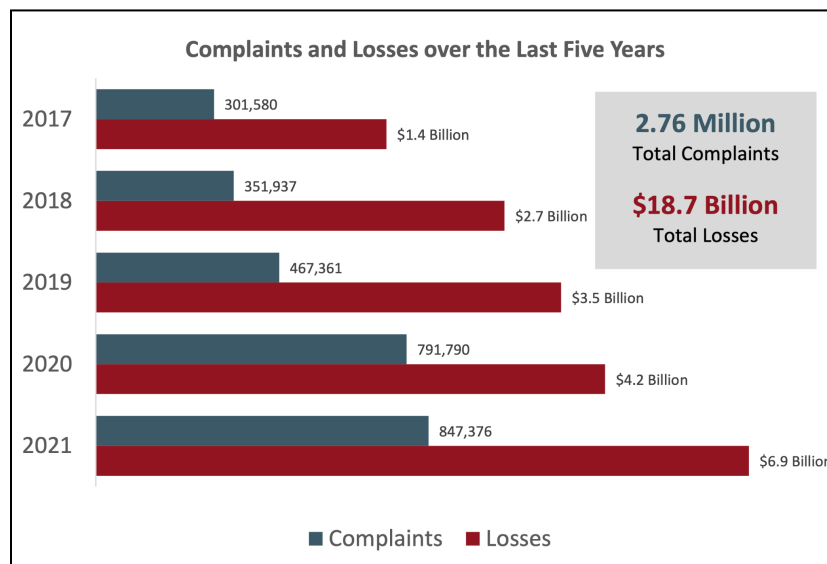


## Secure Smart Parking Payment System

### I) Introduction:

As of 2023, it has been reported that 76% of people in the United States shop online at least once a month [1]. While online shopping can be fun, there are many other transactions that come out of e-commerce. With online transactions becoming more prevalent, there comes the issue of how our sensitive information should be kept secure. The FBI's Internet Crime Report of 2021 states that over the past five years, there have been 2.76 million complaints that have led to a total loss of \$18.7 billion, as shown in the figure below [2].



**Figure 1:** FBI Internet Crime Report

For this project, a user interface will be created to simulate a payment system tied to the smart parking system implemented. The payment system will primarily be used to simulate how users would pay for their parking space. The two main security aspects that this project will incorporate will be hashing and encryption using the AES algorithm. Hashing involves the conversion of data into a fixed length string of ciphertext [3]. Hash functions are irreversible, meaning that it is impossible to retrieve a message from the hash value. Hash functions are typically used for password storage. However, it is not the actual password that is stored, it is the hash of the password that is stored on a file. Hashing is also used for data integrity to detect any changes made to a file. As for the AES algorithm, it replaced DES as the encryption standard in 2002 [4]. The AES algorithm is computed on Galois Field of  $2^8$  and has an input/output of 128 bits (16 bytes). Each round of AES consists of four layers: byte substitution, shift row, mixed column, and key addition. There can be key sizes of 128, 192, and 256 bits with 10, 12, and 14

rounds, respectively. AES has been the encryption standard over the past two decades for its speed and reliability in encrypting/decrypting data.

Through this project, hashing will be incorporated into a login system and payment details will be encrypted using AES in the CCM mode. Through completion of the project, one should be able to see the security algorithms and methods that go behind a secure login system. While this implementation is not entirely secure, the purpose is to identify and apply methods in cybersecurity to create a more secure system in which users can have greater peace of mind.

## **II) Tools Required:**

- Raspberry Pi
- Python (to code the raspberry pi)
- Tkinter and Cryptodome Libraries
- LCD Display (to display results from each GUI window)

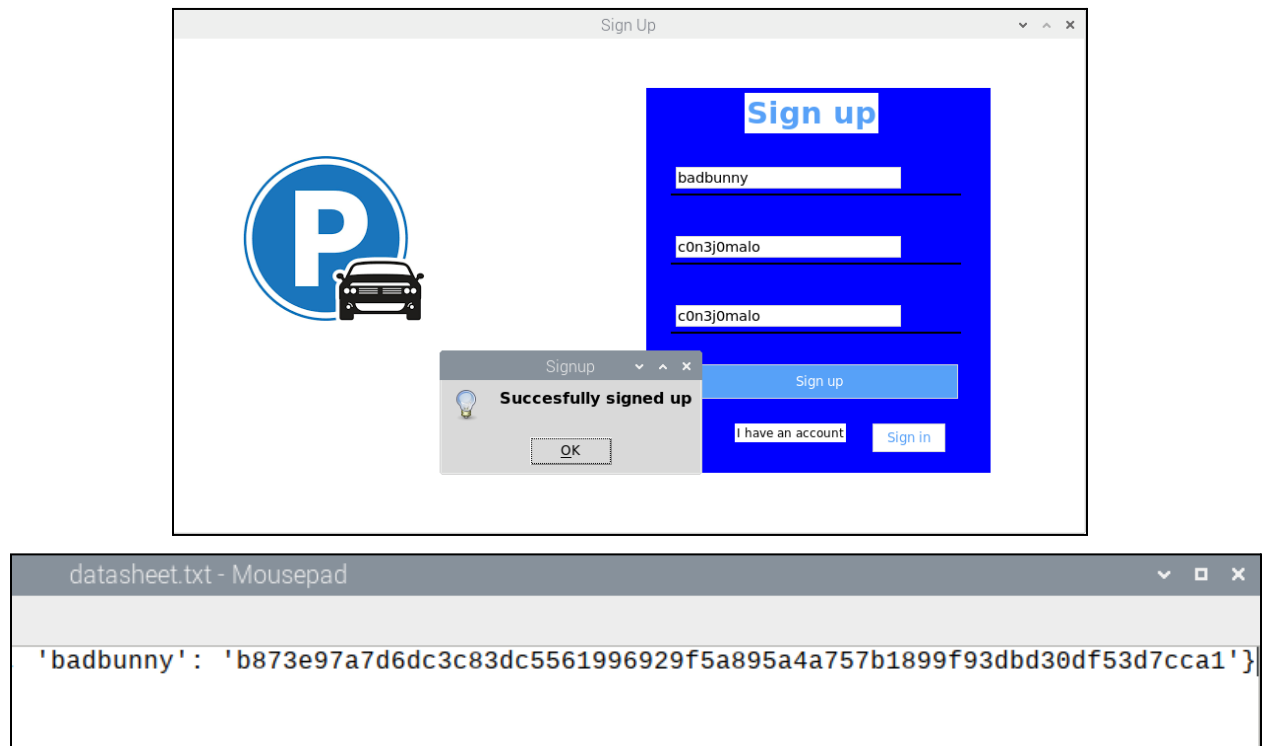
## **III) Implementation Details with Code:**

The first portion of the project was creating a GUI for users to sign up for an account in the payment system. Users are first required to enter a username and password. Users are prompted to enter the password field a second time in order to confirm before submission to the database. It is not ideal to save users' passwords into a database, so their data must be encrypted using hashing methods. Once users have created an account, they can login using their credentials and the system checks the datasheet to make sure the username and hashed password are a match. Once users have signed in, a payment window prompts them to enter their credit card information, which is then encrypted using AES.

### **A) Setup Details:**

The GUI was implemented using the Tkinter library part of Python. The print function displays results on the python window, while the LCD display was set up to display messages such as "Sign Up Successful," "Invalid," and "Payment Successful." The series of code below displays how each window was created.

The first window created was the sign up window. Users are prompted to create a username and password. Users are also prompted to confirm their password. If the original password and confirm password fields do not match, an error message is displayed to tell the reader that "Both passwords must match." If the fields match, a window will display "Successfully Signed Up." The figure below displays the implementation of the GUI for the signup window:



**Figure 2:** Sign Up Window and Datasheet with Username/Hashed Password

It was discussed in class that databases do not save the actual password in order to provide security. Instead, it is the hash of the password that is saved. For this implementation, the SHA-256 algorithm was used. It can be seen from the figure how the original password was “c0n3j0malo” (Bad Bunny), but the hash transforms the password into a unique string of ciphertext. The code below displays how the sign up window was implemented:

### Code 1:

```

window = Toplevel(root)
#####This is to obtain the data from what the user enters from the fields
username = user.get()
password = code.get()
confirm_password = confirm_code.get()

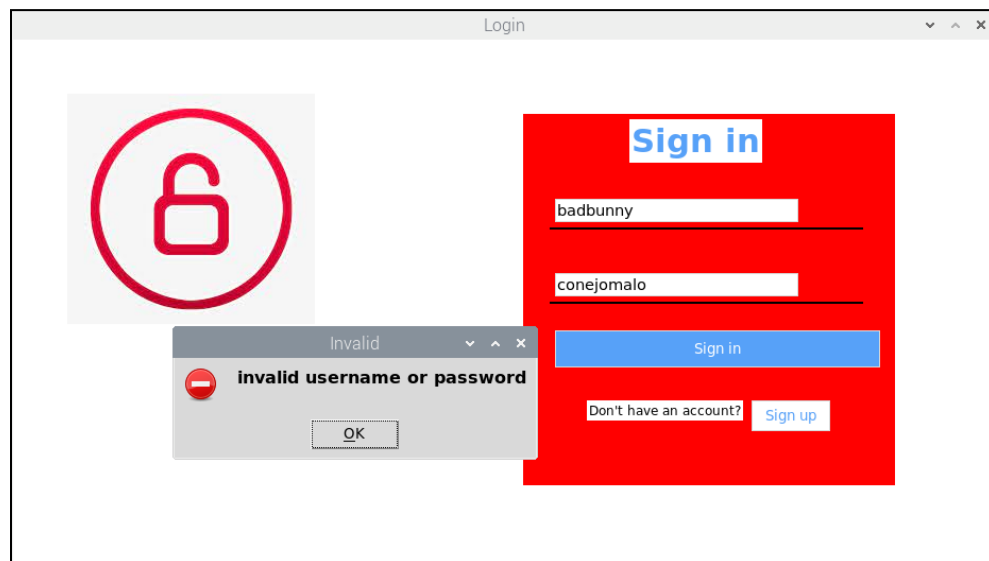
## Used to take password and hash it
h = hashlib.new("SHA256")
h.update(password.encode())
password_hash = h.hexdigest()
## checks if password and confirm password hashes are the same
if password_hash == confirm_password_hash:
    try:
        ### writes the username and hashed password to a file called "datasheet.txt" if a match
        file = open('datasheet.txt', 'r+')
    
```

```
d = file.read()
r = ast.literal_eval

dict2={username:password_hash}
r.update(dict2)
file.truncate(0)
file.close()
file=open('datasheet.txt', 'w')
w=file.write(str(r))

messagebox.showinfo('Signup', 'Successfully signed up')
except:
    ##This is when no one has signed up yet and the "Username:Password" are written to datasheet
    file = open('datasheet.txt', 'w')
    pp = str({'Username': 'password'})
    file.write(pp)
    file.close()
else:
    ##This is the condition that the password and confirm password field are not a match.
    messagebox.showerror('Invalid', "Both passwords should match")
```

The next step was to create the login window that checks through the database and sees if the username and hashed password are recognized. If the username or password are incorrect from what was entered under the sign up window, an error message reads "Invalid Username or Password." This case can be seen in the figure below.



**Figure 3:** Login Window

If the username and hashed password match with what is found in the database, then users are allowed access into the payment system. The code below displays how the login window checks for the username and password.

**Code 2:**

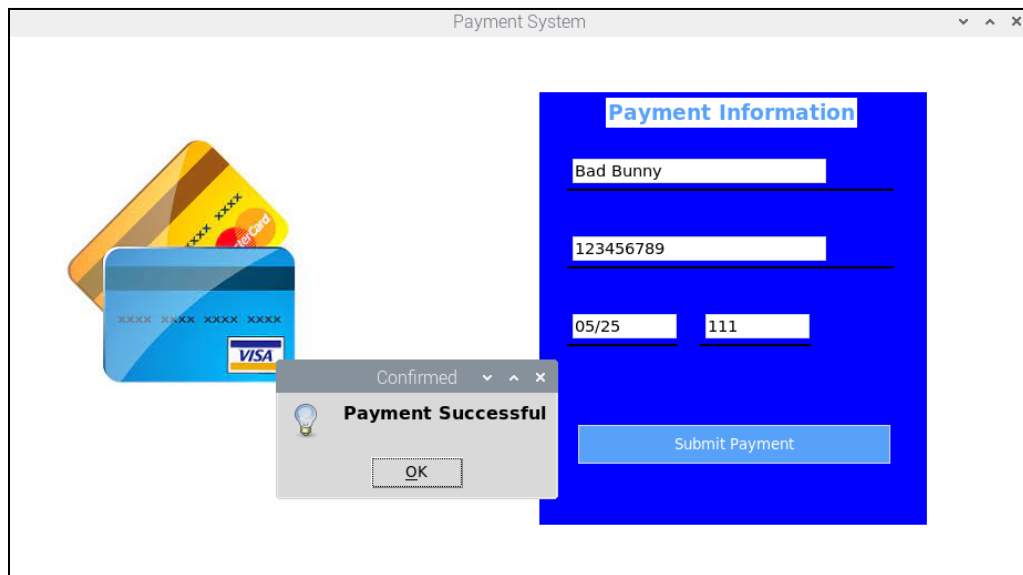
```
root = Tk()
####Used to get what the user enters for username and password fields
username = user.get()
password = code.get()

##Transforms password that was entered into hash
h = hashlib.new("SHA256")
h.update(password.encode())
password_hash = h.hexdigest()

##Used to read what has been saved from datasheet
file = open('datasheet.txt', 'r')
d=file.read()
r=ast.literal_eval(d)
file.close()

###Checks if the entered username and hashed password match with what is in the datasheet
if username in r.keys() and password_hash == r[username]:
    Open window for payment system
else:
    ##If there is not a match, display Invalid username or password error to new window/LCD Display
    messagebox.showerror('Invalid', 'Invalid username or password')
    lcd.write_string('Invalid')
    time.sleep(3)
    lcd.clear()
```

The next step in the GUI was to create the payment system window for users to enter their credit card and payment information. The window prompts users to enter their full name, credit card number, expiration date, and cvv code. Once users have submitted their payment, a window is displayed to say "Payment Successful." To show how the payment fields are encrypted, the shell of Python displays the plaintext and ciphertext of each field. The figure below displays the implemented payment window, along with the results of AES encryption in CCM mode.



```

Shell
~/Desktop/login_secure
>>> %Run login_whash.py
Plaintext: b'123456789' Ciphertext: b'\xc1\xda\x0b\xf0\t\x9a\x9c\xaa6'
|
Expiration: 05/25 CVV: 111
Python 3.7.3

```

**Figure 4:** Payment System and AES Encrypted Results

What should be noted is how the b' in the printed results signifies how each field needed to be converted into bytes since the AES algorithm is a block cipher that is computed in the  $2^8$  Galois Field. The code below displays how the payment window was implemented.

### Code 3:

```
screen=Toplevel(root)
```

```
##Used to obtain full_name, credit card number, expiration date, and cvv code entered by user
```

```
full_name = name.get()
```

```
cc_num = cc.get()
```

```
expire=exp.get()
```

```
code=cvv.get()
```

```
##### Used to transform credit card number into bytes, generate keys for AES encryption
```

```
data = bytes(cc_num, 'UTF-8')
```

```
key = get_random_bytes(16)
```

```
#Set cipher to be in AES CCM mode, generate ciphertext from credit card number entered
```

```
cipher = AES.new(key, AES.MODE_CCM)
```

```
ciphertext, tag = cipher.encrypt_and_digest(data)
```

*##This prints the entered credit card number, displaying the plaintext and ciphertext. Also prints other encrypted fields*

```
print('Plaintext: %s Ciphertext: %s\n' % (data,ciphertext))
```

```
print('Expiration: %s CVV: %s\n' % (expire,code))
```

```
try:
```

```
    ###Writes full name and credit card number to a file called "record.txt"
```

```
    file=open('record.txt','r+')
```

```
    d=file.read()
```

```
    r=ast.literal_eval(d)
```

```
    dict2={full_name:cc_num}
```

```
    r.update(dict2)
```

```
    file.truncate(0)
```

```
    file.close()
```

```
    file=open('record.txt','w')
```

```
    w=file.write(str(r))
```

```
#####Displays "Payment Successful" to Python shell and LCD Display
```

```
messagebox.showinfo('Confirmed','Payment Successful')
```

```
screen.destroy()
```

```
lcd.write_string('Payment Successful')
```

```
time.sleep(3)
```

```
lcd.clear()
```

```
Except:
```

```
    ###Writes "Name:CCNum" if there is no data saved yet.
```

```
    file=open('record.txt','w')
```

```
    pp=str({'Name':'CCNum'})
```

```
    file.write(pp)
```

```
    file.close()
```

The last portion of the project was to have all parts of the GUI flowing together. The code below displays how each portion of the project was integrated.

### **Integrated Code:**

To make the windows seamless and without having to run three different sets of code, the windows were integrated using the following approach:

```
root=Tk()
```

```
    Login window():
```

```
        if(Sign Up):
```

```
            Sign Up window():
```

```
                get.username()
```

```
                get.password()
```

```
                Hashing function to transform password(pwd)
```

```
                Write username and password to datasheet.txt
```

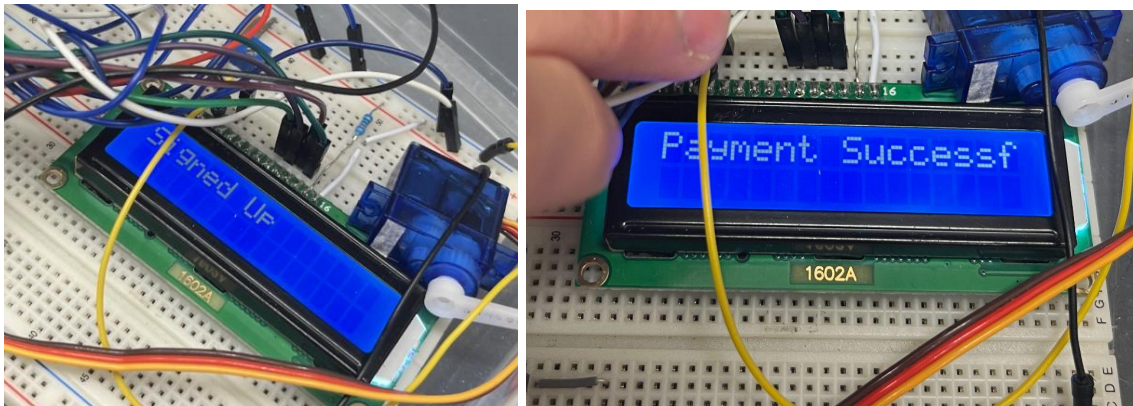
```
        else:
```

```
    Sign in window():  
        check username and password hash from datasheet()  
        if(match):  
            Payment window():  
                get.name()  
                get.creditcardnumber()  
                get.expiration()  
                get.cvv()  
                AES(name,ccnumber,expiration, cvv)  
                print("Payment Successful")  
        else:  
            print("Invalid Username or Password")  
  
root.mainloop()
```

### Integrated Code for GUI

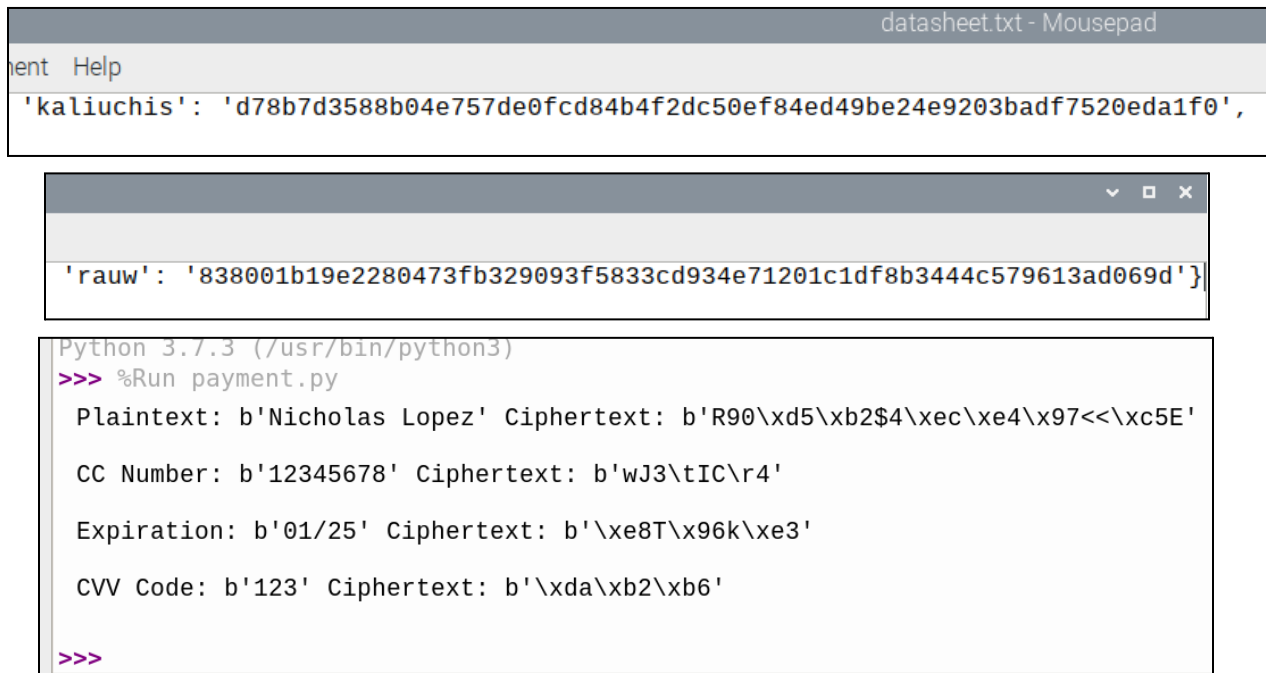
The above code first brings up the login page for users. If the Sign Up button is pressed, users are directed to the sign in window where they can enter their username or password. The hashing function transforms the password and saves the username and hash to a datasheet. Back in the login page, users can enter their username and password. A function checks the datasheet to ensure that the username and password are a match. If there is a match, users are directed to the payment window. If not, an error message displays "Invalid Username or Password." Under the payment system window, users are prompted to enter their name, credit card number, expiration date, and cvv code. Each field is encrypted using AES from the Cryptodome library. Once the user hits submit, a window pops up to display "Payment Successful."

The figure below displays some of the messages output from the LCD display, along with more examples of encrypted data under the final implementation:



**Figure 5:** LCD Display





The image shows a terminal window with a title bar 'datasheet.txt - Mousepad'. The first line of the terminal output is 'kaliuchis': 'd78b7d3588b04e757de0fcd84b4f2dc50ef84ed49be24e9203badf7520eda1f0','. Below this is a second line of output: 'rau': '838001b19e2280473fb329093f5833cd934e71201c1df8b3444c579613ad069d'}'. The bottom portion of the terminal shows the output of a Python script 'payment.py' run in a Python 3.7.3 environment. The script outputs four lines of encrypted data: 'Plaintext: b'Nicholas Lopez' Ciphertext: b'R90\xd5\xb2\$4\xec\xe4\x97<<\xc5E', 'CC Number: b'12345678' Ciphertext: b'wJ3\tIC\r4', 'Expiration: b'01/25' Ciphertext: b'\xe8T\x96k\xe3', and 'CVV Code: b'123' Ciphertext: b'\xda\xb2\xb6'.

```

datasheet.txt - Mousepad
kaliuchis': 'd78b7d3588b04e757de0fcd84b4f2dc50ef84ed49be24e9203badf7520eda1f0',
'rau': '838001b19e2280473fb329093f5833cd934e71201c1df8b3444c579613ad069d'}

Python 3.7.3 (/usr/bin/python3)
>>> %Run payment.py
Plaintext: b'Nicholas Lopez' Ciphertext: b'R90\xd5\xb2$4\xec\xe4\x97<<\xc5E'
CC Number: b'12345678' Ciphertext: b'wJ3\tIC\r4'
Expiration: b'01/25' Ciphertext: b'\xe8T\x96k\xe3'
CVV Code: b'123' Ciphertext: b'\xda\xb2\xb6'
>>>

```

**Figure 6:** More encrypted Data (Hashing and AES-CCM Mode)

It can be seen from Figure 5 how the LCD displays messages such as “Signed Up” and “Payment Successful” when users have clicked the corresponding buttons on the GUI to submit their responses. Figure 6 shows the first two more sets of username and hashed password. The bottom portion of Figure 6 displays the full results of all the encrypted fields on the payment form. It can be seen how the fields for name, credit card number, expiration, and cvv code are all encrypted using AES. The full demo can be found in the zip folder of the final submission.

#### IV) Project Progress Details:

The Calendar below displays a rough outline of how the project was completed.

\*Week 1 (April 3rd-8th): Submit project description and get proximity sensors/RFID module started.

Week 2 (April 9th-15th): Setup LCD display, test each individual module, and get started with creating GUI.

Week 3 (April 16th-22nd): Finish setting up login GUI/credit card page, start with encryption of GUI, and submit first project interim report.

\*Week 4 (April 23rd-29th): Integrate RFID module with servo motor, integrate count with proximity sensor, and submit second project interim report.

Week 5 (April 30th-May 6th): Integrate all modules, add comments to code.

Week 6 (May 7th-10th): Verify that hardware works as expected and submit the final report.

#### Project Timeline

\*Indicate weeks where the IoT end of project was the main focus, working on the sensor/protocols involved with the overall functionality of the project.

During the Week 2 (April 9th-15th), the LCD display and creating the sign up GUI were the main focus. During this week, it took some getting used to creating the overall GUI using the Tkinter library from Python. During Week 3 (April 16th-22nd), the other windows were created to include the login and payment windows. The final aspects of the project were encrypting the data and making a database during Week 3. In the final weeks, most of the work was in getting the LCD to function and to display properly.

### V) Pitfalls and Challenges:

Some of the main challenges in this project at first were trying to find the right libraries for the AES encryption. Furthermore, there were some issues in transforming the string fields into bits. After looking at some sources, I was able to find out that PyCrypto was no longer supported on Python 3.0. The correct library was then downloaded and able to run on my Raspberry Pi. As for the issue of transforming the fields into bits, it was done using the bytes(variable, 'UTF-8'). Prior to this project, I had been exposed to many different login systems for various applications. However, it was interesting to implement my own system that ensures user data is encrypted. Security is a top priority, and that is what I had in mind when creating a payment system for the smart parking garage. It would be an even greater advantage to use homomorphic encryption, but that is something to further develop upon in the project. I enjoyed working on this project because it allowed me to view security in a different manner and implement what I learned in class. Thank you for the opportunity, and I definitely think it would be a great idea to incorporate projects into the course.

### References:

1. Fox, Sapphire. "Online Shopping Statistics & Trends in 2023." Cloudwards. 09 September 2022. [Online Shopping Statistics, Facts & Trends in 2023 \(cloudwards.net\)](https://cloudwards.net/online-shopping-statistics-facts-trends-2023)
2. Federal Bureau of Investigation (FBI). "Internet Crime Report 2021." Internet Crime Complaint Center. 2021. [2021\\_IC3Report.pdf](https://www.fbi.gov/ice3/2021-ic3-report)
3. Codecademy Team. "What is Hashing, and How Does It Work?" Codecademy. 28 April 2023. [What Is Hashing, and How Does It Work? - Codecademy Blog](https://www.codecademy.com/blog/what-is-hashing-and-how-does-it-work)
4. Rimkiene, Ruta. "What is AES encryption and how does it work?" Cybernews. 29 August 2022. [What is AES Encryption and How Does It Work? | Cybernews](https://www.cybernews.eu/what-is-aes-encryption-and-how-does-it-work)
5. \*Parvat Computer Technology. "How To Create Login Form for Apps Using Python | Tkinter Project." March 26, 2022. [https://youtu.be/X9reTI\\_Mckk](https://youtu.be/X9reTI_Mckk)
6. \*Jadhav, Rahul. "Car parking System using Raspberry Pi." January 23, 2022. <https://youtu.be/azCb5Zj76UQ>
7. <https://pycryptodome.readthedocs.io/en/latest/src/cipher/modern.html>
8. <https://youtu.be/i-h0CtKde6w>

9. \*<https://onboardbase.com/blog/aes-encryption-decryption/>

10. \*<https://youtu.be/YWQZULbQ9z4>

\*Sources used for help with implementation of code