This is a school project instead of deep research

Raspberry Pi Security Camera

Name: Wenqi Guo

Course: INFOTC-3530 UNIX OPERATING SYSTEM

Professor: Ronny Bazan Antequera, Ph.D.

School: University of Missouri – Columbia

E-mail: wg25r@umsystem.edu

Keywords – Raspberry Pi, Security Camera, Internet on Things, Low Power Computing

I. Topic

A security camera is an important device to keep the property and life safe. This article shows the steps to implement a security camera using Raspberry Pi. Raspberry Pi is a low-power single-board computer. Its low power consumption makes it suitable for devices like security cameras to run continuously.

II. Research and Application

A. Setting Up the Raspberry Pi

I used the following materials to set up for this project, most of which can be alternated with similar items.

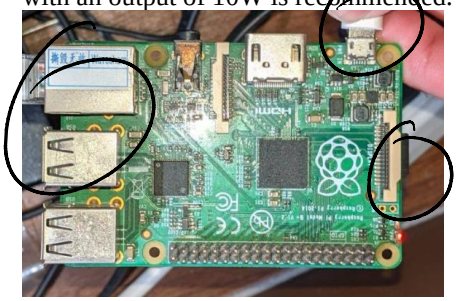
a. Raspberry Pi 1 B+
b. TF Card and TF-SD adapter
c. Passive Speaker
d. Waveshare 4.2 Inch 400x300 two-color E-Ink Display
e. PlayStation 3 Eye Camera and microphone array
f. Depending on the size of the TF card, an external HDD or SSD might be needed
g. A USB Drive, which is not shown here

I used the PlayStation 3 Eye Camera that is completable with Raspberry Pi and a microphone array. [1]

Then we need to burn the image to the SD card. Since here I am using one of the oldest Raspberry Pis, I will use a lightweight image called DietPi instead of the official Raspberry Pi OS. This image can be downloaded from https://dietpi.com/ by selecting Raspberry Pi. Because I am using Raspberry Pi 1 B+ here, the only working image is the *ARMv6 32-bit image*; if another version of the Raspberry Pi is used here, please select the matched image. [2]

Once downloaded, it can be burned to the TF card using software called balenaEtcher, which can be downloaded from https://www.balena.io/etcher/. [2] Then the burning process should be easy by just following the wizard. Keep in mind that the burning will wipe all the data on the TF card. (Since my TF card already has DietPi installed, I cannot demonstrate this step by screenshots.)

Next, we can insert the TF card to the Raspberry Pi and connect the power and Ethernet cable. Note that the Raspberry Pi will be unstable if the power source cannot provide enough current, especially if we connect a camera later. Thus, a USB power adaptor with an output of 10W is recommended.



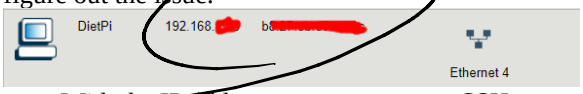
Then we need to know the IP Address of the Pi. The easiest way to do so is to log into the router admin page, the address and password of which are usually on the router's label.

Every router has different admin pages, but most of them have a page that lists all the devices. On this

The repository of this project is on https://github.com/weathon/IT3530-Project

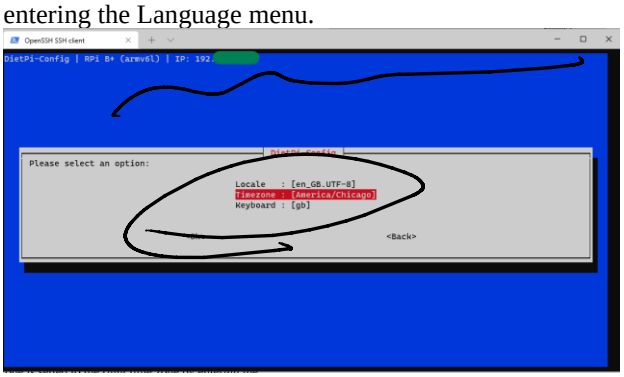
Many Coding Processes are NOT shown in the paper

page, look for a device named “DietPi” and copy its IP address. Make sure the IP address is the one of the Pi, I used the wrong address, and it took me hours to figure out the issue.

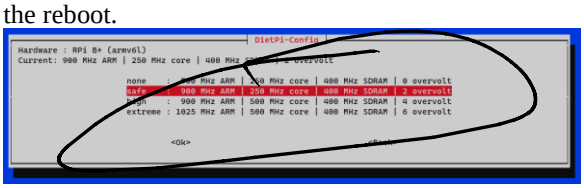


With the IP address, we can now use SSH to connect to the Pi. OpenSSH should be installed by default on Windows. If you are using SSH in a Linux environment using a virtual machine, make sure the virtual machine is connected to the network with the bridge mode. The default login username with DirtPi is “root,” and the password is “dietpi.” [3]

If this is the first login, the setup screen should pop up. The command `dietpi-config` can also bring up this screen if it is not popped up or the setting needs to be changed later. We only need to make sure the time zone is set to the correct time zone by entering the Language menu.



Since I am using the Raspberry Pi B+, which is slow for many applications in 2021, I overclocked my CPU using Performance Options. A reboot might be required after saving the settings. The Pi is set up after the reboot.

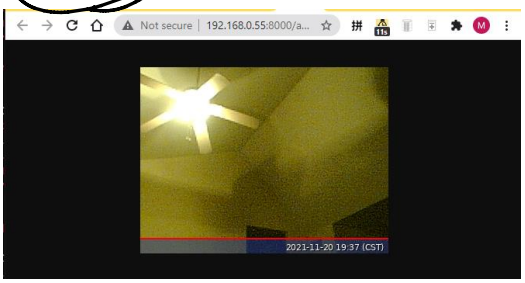


B. *Implementing the video recording function*

Because of time limitations, this security camera can only record the videos but not stream them. The PlayStation 3 Eye Camera (PS Eye) is driver-free on Raspberry Pi. Almost any camera software could be used to take photos. Here I am going to use fswebcam to test the camera, and it can be installed by the following commands. [4]

```
sudo apt update
sudo apt install fswebcam
```

After installing, we can run `fswebcam /tmp/a.jpg` [5] to take a picture and save it as /tmp/a.jpg. We can then copy this image to local and confirm it works.



Since the `fswebcam` can only take a single frame and it is for testing only. We need another tool for video capturing. After trying a few tools, I find out FFmpeg is the easiest tool for this. However, my Raspberry Pi is not fast enough to encode the video in real-time, even with the one fps setting. Thus, a method of recording audio and taking image serials is used here. The FFmpeg is used here to record the audio, and fswebcam will take a photo every 2 seconds. In order to record video and audio at the same time, we need to have two scripts. The continuous capturing script can be further optimized: because it has repeated initialization.

```
audio_script = """ ffmpeg -f alsa  '# * Use alsa [6]' \
    -channels 4  '# Channel numbers are 4 for ps eyes' \
    -i hw:1,0 a.wav # * Audio input [6]' \
    -t 65 '# * Not 60 because it might be within a min, and the -t should be placed before filename'\
    audio/%d/%d/%d/%d.wav """
import os, datetime
if 1:
    time = datetime.datetime.now()
    os.system("mkdir audio/%d/%d/%d -p" % (time.month, time.day, time.hour))
    os.system("mkdir audio/%d/%d/%d -p" % (time.month, time.day, time.hour + 1)) # In case it is right on the time between hours
    os.system(audio_script % (time.month, time.day, time.hour, time.minute))
```

```
import os, datetime
from time import sleep
script = "fswebcam images/%d/%d/%d/%d/%d.jpg"
while 1:
    time = datetime.datetime.now()
    os.system("mkdir images/%d/%d/%d/%d -p" % (time.month, time.day, time.hour, time.minute))
    os.system("mkdir images/%d/%d/%d/%d -p" % (time.month, time.day, time.hour, time.minute + 1))
    os.system(script % (time.month, time.day, time.hour, time.minute, time.second))
    sleep(1)
```

To check the videos remotely, a mobile application is needed. Here I am using Python Fast API as the back end and Ionic Framework as the front end. Because of the low performance of the Pi, we can build these on another machine then deploy them to the Pi.

Since our videos are stored in directories, we can simply build a file browsing server. And it will show all the images of the same minutes on one page. And these are the APIs we need:

- /getMonths_images
- /getDays_images
- /getHours_images
- /getMins_images
- /getImages
- /getImage
- /getMonths_audio
- /getDays_audio
- /getHours_audio
- /getMins_audio
- /getAudio

Then we can implement them in Python using the FastAPI. We first install the FastAPI using two commands: [8]

```
pip install fastapi
pip install "uvicorn[standard]"
```

I put these two commands in the bash script server/install.sh in the GitHub repository. Then we can implement these APIs. I put the code and its explanation in the GitHub repository

The repository of this project is on https://github.com/weathon/IT3530-Project

Many Coding Processes are NOT shown in the paper

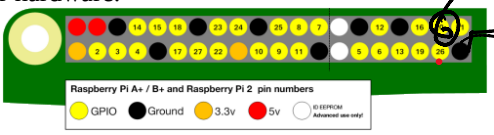
https://github.com/weathon/IT3530-Project. The front end will take these APIs and show the data to the user.

C. *Implementing additional functions*

It would be boring if we only made the raspberry pi to record videos, so I will add some additional functions: a remote-controlled beeper and an E-Ink text sign.

1. The Beeper

The Raspberry Pi provides the GPIO (General-purpose input/output) pins to control other hardware. [9]



These pins can output PWM signals which can power the speaker we have. Then we can connect the speaker positive to pin 20 and negative to ground. (Pins 19, 23, 24, 22, 11, and 18 will be used by the E-Ink, [10] and pins 27 and 28 are reserved. [11])

To control these GPIO pins, we can use a python library called RPi.GPIO, which can be installed by the command `pip install RPi.GPIO`. [12] This library has a built-in PWM control, and the following code could use that to make the beeper make noise. [13] [14] If the volume is not loud enough, an activity could be used here.

```
from RPi import GPIO
import time

GPIO.setmode(GPIO.BCM) # [13]
GPIO.setup(20, GPIO.OUT)

pin = GPIO.PWM(20, 1500)
pin.start(10) # Duty cycle [14]
time.sleep(1)
pin.stop()
```

Now we can add this to our server python program. The initialization code should be added initially, and the API function should be added at the end. After that, the beeper should work when /beep is loaded.

```
from RPi import GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setup(20, GPIO.OUT)
pin = GPIO.PWM(20, 1500)

@app.get("/beep")
def beep():
    pin.start(10) # Duty cycle [14]
    time.sleep(5)
    pin.stop()
```

2. The E-Ink Display

E-Ink is a type of display that can hold the content without power. Thus, it is ideal for text signs. Here I am using the Waveshare 4.2 Inch 400x300 two-color E-Ink display, whose documentation can be found on [10]. The use case of the text sign is that when the security camera faces the door, this could be used as a sign for delivery people when you leave your house.

The wires of the E-Ink are connected to the Raspberry Pi, according to the document. [10]

3v3 Power	1	2	5v Power	
GPIO 2 (I2C1 SDA)	3	4	5v Power	
GPIO 3 (I2C1 SCL)	5	6	Ground	
GPIO 4 (GPCLK0)	7	8	GPIO 14 (UART TX)	
Ground	9	10	GPIO 15 (UART RX)	
GPIO 17	11	12	GPIO 18 (PCM CLK)	
GPIO 27	13	14	Ground	
GPIO 22	15	16	GPIO 23	
3v3 Power	17	18	GPIO 24	
GPIO 10 (SPI0 MOSI)	19	20	Ground	
GPIO 9 (SPI0 MISO)	21	22	GPIO 25	
GPIO 11 (SPI0 SCLK)	23	24	GPIO 8 (SPI0 CE0)	
Ground	25	26	GPIO 7 (SPI0 CE1)	
GPIO 0 (EEPROM SDA)	27	28	GPIO 1 (EEPROM SCL)	
GPIO 5	29	30	Ground	
GPIO 6	31	32	GPIO 12 (PWM0)	
GPIO 13 (PWM1)	33	34	Ground	
GPIO 19 (PCM FS)	35	36	GPIO 16	
GPIO 26	37	38	GPIO 20 (PCM DIN)	
Ground	39	40	GPIO 21 (PCM DOUT)	

[12]

We then need to turn on the SPI interface using the Advance Options by command `dietpi-config`. This step is different from that on the original Raspberry Pi OS, on which this setting is under Interfacing Options by command `sudo raspi-config`. [10] A reboot is required after this step. Then we should follow [10] to install the libraries. Referring to the examples, we wrote our Hello World and tested it. The code is on GitHub with the filename of /server/eink.py

Then we can make it as a model and link it to the API. Code is not shown here, but it is on GitHub.

D. *Connect the security camera to the Internet*

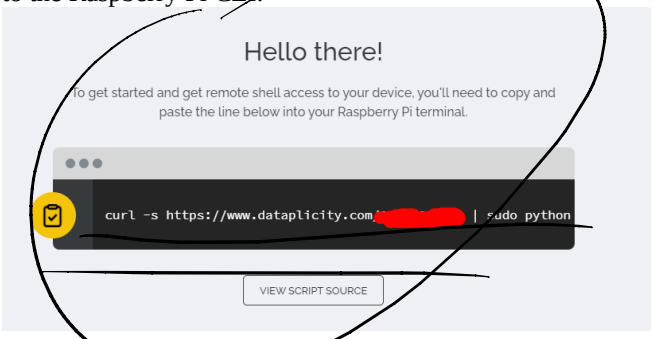
Since our Raspberry Pi is running under our home local area network, we cannot access it outside the home. This makes our security useless: the time we need to check the security camera is when we are not at home. However, it is easy to make it accessible from the

The repository of this project is on https://github.com/weathon/IT3530-Project

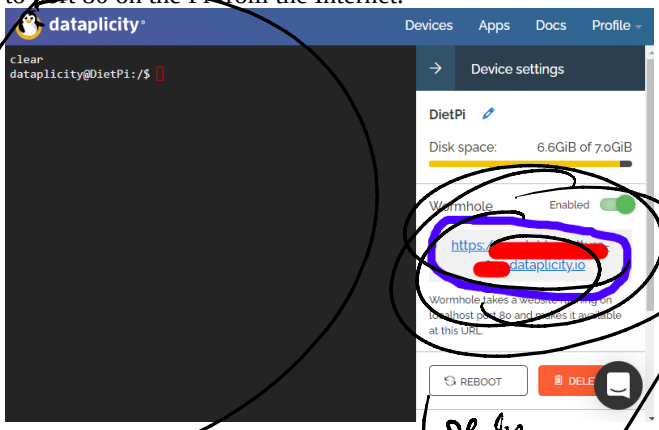
Many Coding Processes are NOT shown in the paper

Internet; we can use some thirty-party tools like PI Tunnel or Data Plicity. They are similar enough, so I am using Data Plicity as an example here.

We first need to sign up for an account on https://dataplicity.com/. Then we can copy the command to the Raspberry Pi CLI.



After installing, the Raspberry Pi should be shown on the devices page on dataplicity.com. (If nothing shows up, a reboot might be required.) We then should enable the Wormhole feature, which allows us to connect to port 80 on the Pi from the Internet.



E. Adding Security Features

Our security camera is on the Internet; everyone knows the address can connect to it and get complete control of the camera. Thus, some security features should be added. For our situation, what we need to consider is connection security, authentication, and CSRF. However, since the Wormhole already provides an HTTPS connection to the Internet and the local connection doesn't leave the Pi, there is no additional work to be done here.

The authentication in this project prevents unauthorized individual views recorded videos and control the beeper and the screen. Since we only have one user here, we do not need a database. We will use a customized version of JWT (JSON Web Token). After the user input the password, the server will hash it and check its validity. If the password passed the checking, the server would return a token containing two parts: the first part is a random number and the expiry time, and the second part is the RSA-encrypted message of the first part using a private key. This two-part token will be stored in the cookie. Before every action, the server will decrypt the second part of the token using the public key and check if it is the same as the first part.[20]

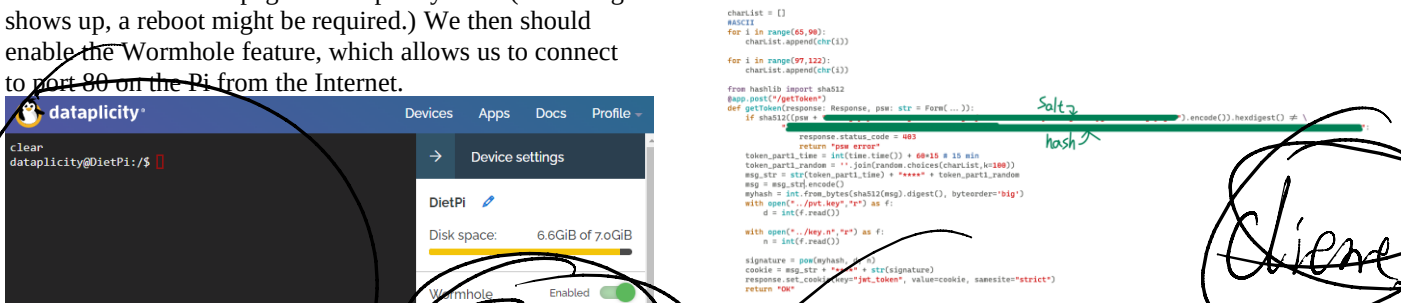
We have two options to solve the CSRF issue: using a traditional CSRF token or the SameSite flag. [21] The second method is much easier, so we will use it.

We first need to generate our public key and private key. [18]

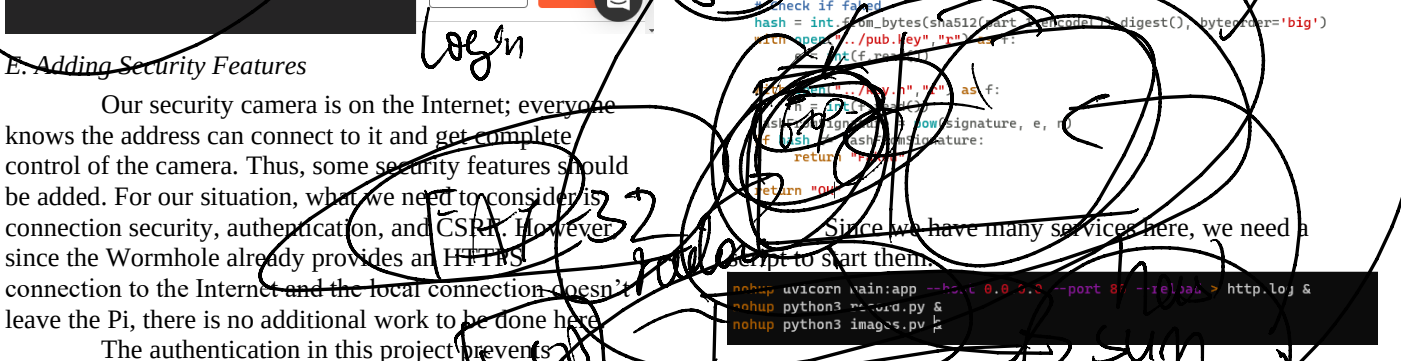
```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import binascii

def main():
    from Crypto.PublicKey import RSA
    keyPair = RSA.generate(bits=1024)
    print(f"Public key:  (n={hex(keyPair.n)}, e={hex(keyPair.e)})")
    print(f"Private key: (n={hex(keyPair.n)}, d={hex(keyPair.d)})")
    with open('pub.key','w') as f:
        f.write(str(keyPair.e))
    with open('pvt.key','w') as f:
        f.write(str(keyPair.d))
    with open('key.n','w') as f:
        f.write(str(keyPair.n))
if __name__ == '__main__':
    main()
```

The back-end login function is shown here: [19] [22] [23] [24]



This is the checking function and should be placed before every API. [23]



Since we have many services here, we need a script to start them.

```
nohup uvicorn main:app --host 0.0.0.0 --port 80 --reload > http.log &
nohup python3 record.py &
nohup python3 images.py &
```

F. Using a USB Drive

Using a USB Drive here can extend the storage, provide faster-writing speed, resulting in better reliability, and allow the data to be read from a non-Linux machine (Using FAT-32 filesystem). Most of the modern Linux can mount the drive when it is plugged in. We just need to move the whole directory to the USB drive and run the code.

The repository of this project is on https://github.com/weathon/IT3530-Project

Many Coding Processes are NOT shown in the paper

References

[1] Asad, B. S. (n.d.). *Best Raspberry Pi Cameras*. Linux Hint. Retrieved November 20, 2021, from https://linuxhint.com/best_raspberry_pi_cameras/

[2] DietPi.com Team. (n.d.). *DietPi.com Docs - How to install DietPi*. DietPi Documentation. Retrieved November 20, 2021, from https://dietpi.com/docs/install/

[3] T. (2017, July 1). *Default users on dietpi*. DietPi. Retrieved November 20, 2021, from https://dietpi.com/phpbb/viewtopic.php?t=2026qqq

[4] rɑːdʒɑ. (2012, February 23). *Take a picture from terminal*. Ask Ubuntu. Retrieved November 20, 2021, from https://askubuntu.com/questions/106770/take-a-picture-from-terminal/276232#276232

[5] *Ubuntu Manpage: Fswebcam - Small and Simple Webcam for *nix.* http://manpages.ubuntu.com/manpages/bionic/man1/fswebcam.1.html. Accessed 21 Nov. 2021.

[6] *Capture/ALSA – FFmpeg*. https://trac.ffmpeg.org/wiki/Capture/ALSA. Accessed 21 Nov. 2021.

[7] *Icecat: Open Feed with Product Information, Data-Sheets for Ecommerce.* https://icecat.biz/p/sony/9473459/webcams-eye-camera-+ps3-1269549.html. Accessed 21 Nov. 2021.

[8] *FastAPI*. https://fastapi.tiangolo.com/. Accessed 24 Nov. 2021.

[9] *Physical Computing with Python - GPIO Pins*. https://projects.raspberrypi.org/en/projects/physical-computing/1. Accessed 25 Nov. 2021.

[10] *4.2inch e-Paper Module - Waveshare Wiki*. https://www.waveshare.com/wiki/4.2inch_e-Paper_Module. Accessed 25 Nov. 2021.

[11] "Raspberry Pi 4 Pins - Complete Practical Guide." *The Robotics Back-End*, 6 May 2019, https://roboticsbackend.com/raspberry-pi-3-pins/.

[12] Croston, Ben. *RPi.GPIO: A Module to Control Raspberry Pi GPIO Channels*. *PyPI*, http://sourceforge.net/projects/raspberry-gpio-python/. Accessed 25 Nov. 2021.

[13] "Simple Guide to the Raspberry Pi GPIO Header." *Raspberry Pi Spy*, 9 June 2012, https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins/.

[14] *Raspberry-Gpio-Python / Wiki / PWM*. https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/. Accessed 25 Nov. 2021.

[15] *Using HTTP Cookies - HTTP | MDN*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies. Accessed 25 Nov. 2021.

[16] *Differences Between Fast Hashes and Slow Hashes [Openwall Community Wiki]*. https://openwall.info/wiki/john/essays/fast-and-slow-hashes#:~:text=Slow%20hashes%2C%20on%20the%20other,are%20bcrypt%2C%20PBKDF2%20and%20scrypt. Accessed 25 Nov. 2021.

[17] *Raspberry Pi GPIO Pinout*. https://pinout.xyz/. Accessed 25 Nov. 2021.

[18] *RSA Signatures*. https://cryptobook.nakov.com/digital-signatures/rsa-signatures. Accessed 26 Nov. 2021.

[19] Developers, The Python Cryptographic Authority. *Bcrypt: Modern Password Hashing for Your Software and Your Servers*. *PyPI*, https://github.com/pyca/bcrypt/. Accessed 26 Nov. 2021.

[20] Ionic. "Open-Source UI Toolkit to Create Your Own Mobile or Desktop Apps." *Ionic Docs*, https://ionicframework.com/docs/undefined. Accessed 26 Nov. 2021.

[21] auth0.com. *JWT.IO - JSON Web Tokens Introduction*. http://jwt.io/. Accessed 26 Nov. 2021.

[22] *SameSite Cookies - HTTP | MDN*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite. Accessed 26 Nov. 2021.

[23] *XMLHttpRequest.Send() - Web APIs | MDN*. https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/send. Accessed 26 Nov. 2021.

[24] Welcome to Python-RSA's Documentation! — Python-RSA 4.8 Documentation. https://stuvel.eu/python-rsa-doc/. Accessed 26 Nov. 2021.

The repository of this project is on https://github.com/weathon/IT3530-Project

Many Coding Processes are NOT shown in the paper