Project Summary Report – [**Secure Remote Health Monitoring System**]

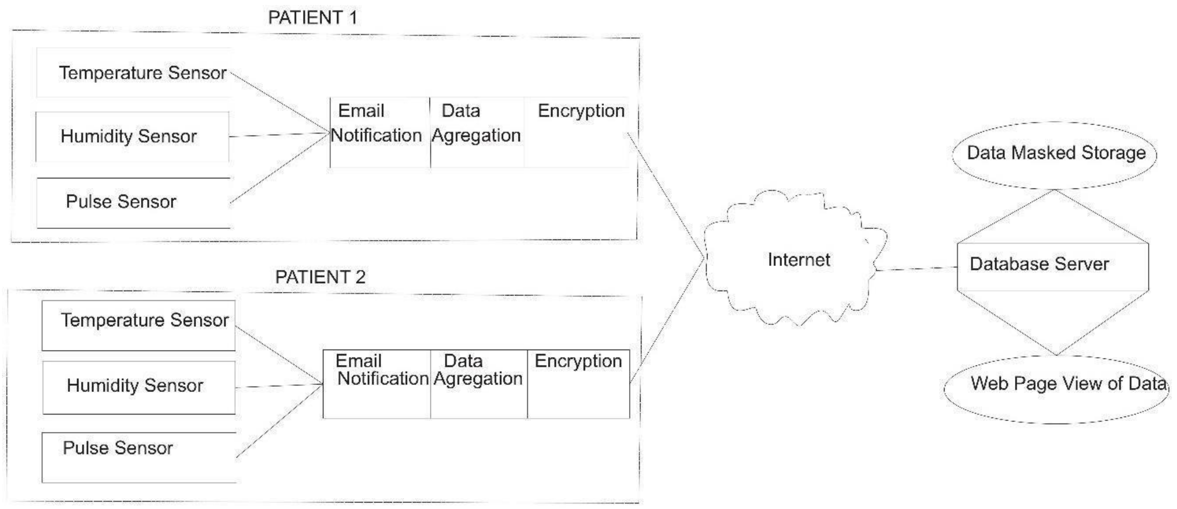
[Chibuikem Ezemaduka, ezemadukachibuikem@gmail.com], [James Onyejizu,

james.kel71@gmail.com]

1. Abstract

This work is about rendering medical care to patients without the need of the physical presence of a health worker. The Covid-19 pandemic brought to light the critical need of ensuring that basic and essential services are continuously made available to individuals even without the physical presence of the service provider. While these remote applications are being considered, it is also critical to assure the privacy and security of data being sent over the public and private network infrastructure. In view of this, this remote health monitoring system achieves the goal of collecting patients' health data without the presence of a health worker and enabling an authorized medical practitioner to access this data through the internet at a remote location. The system also ensures that the data is properly secured during transmission through the public internet and during its storage on the collection device. The design consists of several parts: Part A**:** Health data collection from a remote patient, Part B: Data aggregation of correlated data to reduce bandwidth utilization and energy consumption with immediate notification to a designated health worker if the measured data is above or below a particular threshold, and Part C: encryption of the data before being transmitted through the internet to a collection server where it can be viewed by an authorized party with data masking employed during data storage. The system uses temperature, humidity and pulse sensors connected to a raspberry pi to achieve the data collection, an efficient algorithm to achieve the data aggregation, and email notification. The system uses both asymmetric (built from scratch) and symmetric encryption to achieve security of the data transmission (TCP socket built from scratch) with a data masking algorithm for secure storage of the data on a web server where it can be unmasked and viewed by an appropriate party. The system achieves the performance of enabling secure remote health data collection and transmission, bandwidth and energy conservation through data aggregation, timely email notification of critical data readings, and secure storage and viewing of the data by an authorized party.

2. Methodology



A SCHEMATIC DIAGRAM OF THE SECURE REMOTE HEALTH MONITORING SYSTEM

The relationship between the system parts is shown in the above figure. The general working procedure is as follows: The system is modeled on two patients at different locations. The patients strap the temperature, humidity and pressure sensors to their bodies to measure the respective readings which are then sent to their Raspberry Pis. The Raspberry Pis collate the sensors' data and aggregate it based on correlation of successive measurements to save bandwidth and energy consumption. Sensor measurements that lie outside a particular set threshold are not aggregated but immediately sent through email to an authorized health practitioner. The aggregated data is then encrypted and transmitted by building a TCP socket to a database server hosted on an authorized PC. The server decrypts the received data and then masks it through an algorithm before storing it on a database. The server identifies and appropriately stores each patient's data through a unique patient ID that is also included in the payload. When prompted, the server accesses the database, unmasks the data and displays it to authorized personnel through a web browser. In the following sections,each part of the design is described in greater detail.

2.1 Part A

HARDWARE	SOFTWARE
DHT11 sensor	Python
Pulsesensor.com sensor	Thonny IDE
Raspberry Pi	Spidev Library
MCP-3008 I/P ADC	
Jumper Wires (Male to Female)	

The Raspbian OS is used on the Pi and the scripts are written in the python programming language. The Thonny IDE was used as programming environment on the RPi. Pictures of the hardware can be found in Appendix A.
The DHT11 sensor was chosen because of its capacity to measure both temperature and humidity on the same module hence less bulk to the system. Its output is also digital which corresponds to the digital GPIO pins of the Raspberry Pi. Data from the sensor is accessed using szazo dht_11 library. Instructions on implementing this can be found at this link: http://homepages.rpi.edu/~wangy52/PersonalWebsite/build/html/Course%20Content/IoT/Resources/Sensors/DHT11/DHT11.html

The Pulsesensor.com pulse sensor has an analog output and hence cannot be interfaced directly to the Raspberry Pi's digital pins. The system uses an Analog toDigital converter to read the data. The MCP-3008 1/P ADC was chosen. It is a low cost 8 channel 10-bit ADC. The pulse sensor and ADC wiring instructions can be accessed at the github page of the sensor manufacturers: https://github.com/WorldFamousElectronics/Raspberry_Pi/blob/master/PulseSensor_Processing_Pi/PulseSensor_Processing_Pi.md. The system uses the tutRPi pulse sensor library and the Spidev library to read the data from the sensor and ADC on the raspberry pi. The Spidev library is necessary to read data from the ADC whichconnects through the Raspberry Pi's SPI interface. The tutRPi libraries can be accessed at:https://github.com/tutRPi/Raspberry-Pi-Heartbeat-Pulse-Sensor and additional instructions can be found at:https://tutorials-raspberrypi.com/raspberry-pi-heartbeat-pulse-measuring/

2.1 Part B

In part b, the system implements the idea of data aggregation of highly correlated sensor measurements to reduce the bandwidth and energy consumption. Sensor readings can be significantly consistent over a period and hence can lead to transmission of the same data even when there is no difference between successive measurements. To avoid this, it uses a data aggregation algorithm that compresses sensor data based on their correlation to a moving average of previous successive readings and then transmits an average value in roughly per minute intervals instead of transmitting the instantaneous values. If an instantaneous value is significantly different from the moving average, the reading is transmitted immediately without being aggregated as this signifies that there could be an issue that requires attention and needs to be recorded. The algorithm for the data aggregation is given as:

```
If not (Moving average – 2) < Sensor reading < (Moving average + 2)
   Transmit data immediately due to low correlation
else
    If window is full
        Find average of all values and transmit
    else
        Delay transmission and take sensor measurements again
```

Email notification is also implemented in the algorithm to immediately notify authorized personnel if an instantaneous sensor reading is outside a defined threshold which could signify a health emergency that requires attention. This is implemented using the SmtpLib Library.

2.1 Part C

In part C, system security is achieved by encrypting the data before transmission to a server through the internet, decrypting the received data and then masking it before storage in a database and then accessing the stored data on a webpage. The Hardware involved are Windows PC and Home Router, while Software used are: SQLite, Flask module, Python, HTML, CSS, Pycharm IDE.
After being processed, the data is encrypted on the Raspberry Pi before being transmitted through a TCP socket that is built in the code.   TCP is used as transport protocol because of its connection-orientation and to ensure reliability in data delivery since we are sending aggregated data and cannot afford to be unaware of lost packets. Asymmetric and symmetric encryption is used to secure the data. RSA is used as the asymmetric key encryption and AES-128 as the symmetric encryption. RSA is used as the public key cryptography to share the symmetric keys used for the AES encryption which is used for the data transmission. RSA is not used for the full transmission because it is computationally expensive. The RSA algorithm is implemented from scratch but is assisted with the help of Yu Wang's Number_package library for some cryptographic calculations. The AES encryption is implemented using the python PyCrypto library. The AES-CFB mode of the library is utilized to ensure automatic padding of our plaintext to the appropriate size of 16 bytes.

Windows 10 operating system was used on the PC. The PC also has a connection to a home router that is connected to the internet. Port forwarding is configured on the router to allow the server to be reachable through the internet despite NAT. PyCharm IDE is used on the PC to write and run the server scripts. The server function is divided into backend and frontend scripts. The backend function is a python TCP socket server listening on ports 12000 and 13000. Two separate ports were used for the two patients. The received data is then decrypted using the symmetric key and then masked using an algorithm that is based on each patient's unique ID that is transmitted in the payload from the Raspberry Pi. The masked data is then stored in an SQLite database. SQLite was chosen because of its lightweight and easy integration with python. The data masking is done using this arithmetic formula:

$$C = D^{(e+2)} - I$$

Where D = original data, C = masked data, I = unique patient ID

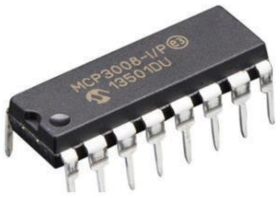
Because the data has been masked before storage, if the storage of the PC is accessed, anadversary would not know the true data values in the database. The system is modelled that the server scripts will be stored on a portable disk which serves as a security token. Without the disk, the adversary cannot understand the database content if he accesses the PC storage. The authorized doctor is to keep the portable disk with him/her safely. Access to the PC is also secured through the windows firewall, defender and Password login authentication. Anytime the doctor wants to view the data, he/she can connect the portable disk to the PC and run the server scripts.

For the server front end, flask server module is used. Whenever the doctor wants to view a patient's data, he/she goes to this url: 127.0.0.1:81/patient1 or 127.0.0.1:81/patient2 depending on the patient he/she is interested in. The server can only be accessed locallyon the PC and is blocked from external access for extra security. When this url is entered in the web browser, the Flask server which listens on port 81 accesses the SQLite database and unmasks the stored data using this formula:

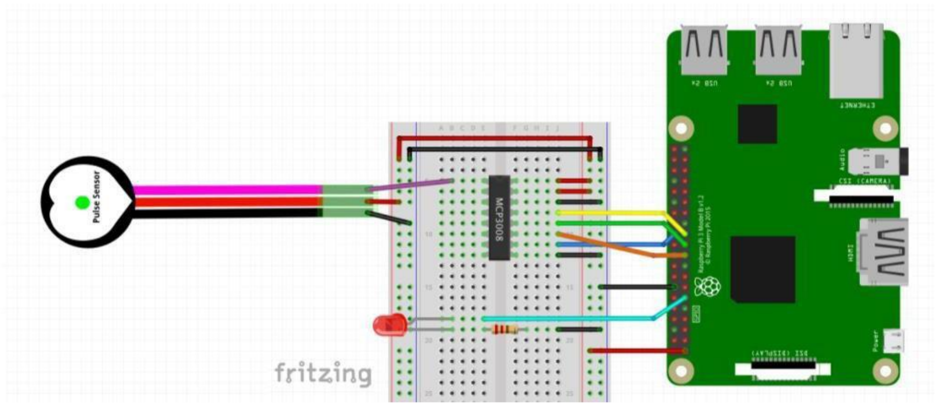
$$D = 10^{\frac{\log \frac{C+I}{1}}{e+2}}$$

The server then displays the data through HTML and CSS. The HTML and CSS scripts were written based on reference from this link: https://raw.githubusercontent.com/RuiSantosdotme/Random-Nerd-Tutorials/master/Projects/RPi-ESP-SQLite/templates/main.html

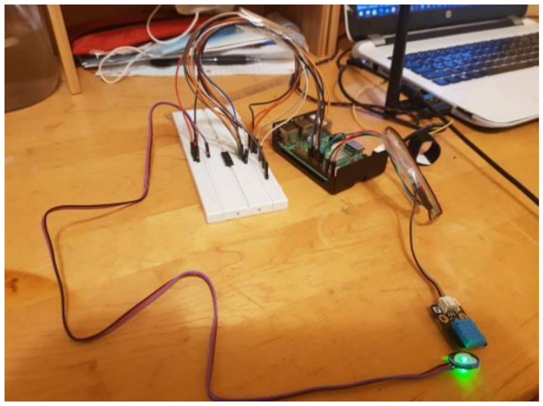
APPENDIX A



Pulse sensor and MCP-3008 I/P ADC

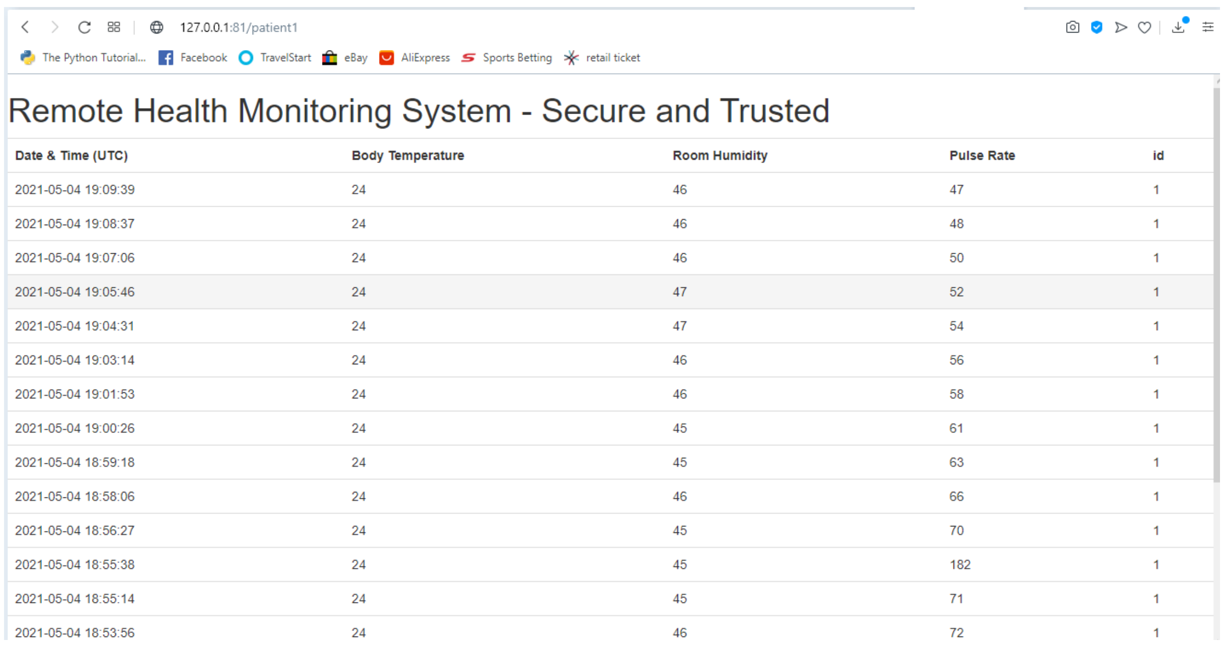


Pulse sensor schematic wiring to ADC on breadboard



Actual sensor connections to the Raspberry Pi

APPENDIX B



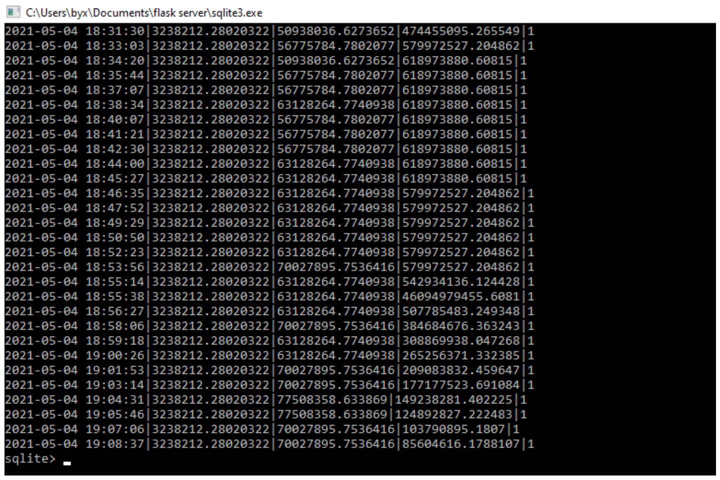
127.0.0.1:81/patient1

The Python Tutorial... Facebook TravelStart eBay AliExpress Sports Betting retail ticket

Remote Health Monitoring System - Secure and Trusted

Date & Time (UTC)	Body Temperature	Room Humidity	Pulse Rate	id
2021-05-04 19:09:39	24	46	47	1
2021-05-04 19:08:37	24	46	48	1
2021-05-04 19:07:06	24	46	50	1
2021-05-04 19:05:46	24	47	52	1
2021-05-04 19:04:31	24	47	54	1
2021-05-04 19:03:14	24	46	56	1
2021-05-04 19:01:53	24	46	58	1
2021-05-04 19:00:26	24	45	61	1
2021-05-04 18:59:18	24	45	63	1
2021-05-04 18:58:06	24	46	66	1
2021-05-04 18:56:27	24	45	70	1
2021-05-04 18:55:38	24	45	182	1
2021-05-04 18:55:14	24	45	71	1
2021-05-04 18:53:56	24	46	72	1

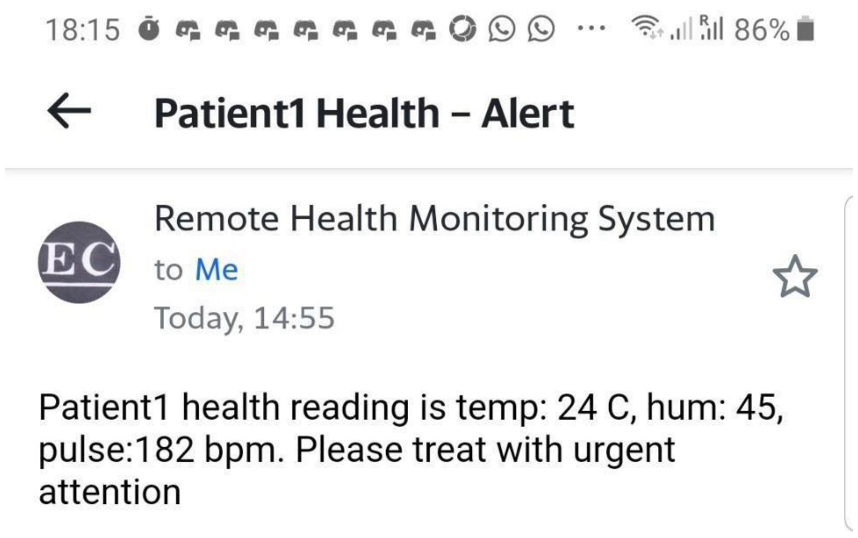
Web Page view of the collected system data



C:\Users\byx\Documents\flask server\sqlite3.exe

```
2021-05-04 18:31:30|3238212.28020322|50938036.6273652|474455095.265549|1
2021-05-04 18:33:03|3238212.28020322|56775784.7802077|579972527.204862|1
2021-05-04 18:34:20|3238212.28020322|50938036.6273652|618973880.60815|1
2021-05-04 18:35:44|3238212.28020322|56775784.7802077|618973880.60815|1
2021-05-04 18:37:07|3238212.28020322|56775784.7802077|618973880.60815|1
2021-05-04 18:38:34|3238212.28020322|63128264.7740938|618973880.60815|1
2021-05-04 18:40:07|3238212.28020322|56775784.7802077|618973880.60815|1
2021-05-04 18:41:21|3238212.28020322|56775784.7802077|618973880.60815|1
2021-05-04 18:42:30|3238212.28020322|56775784.7802077|618973880.60815|1
2021-05-04 18:44:00|3238212.28020322|63128264.7740938|618973880.60815|1
2021-05-04 18:45:27|3238212.28020322|63128264.7740938|618973880.60815|1
2021-05-04 18:46:35|3238212.28020322|63128264.7740938|579972527.204862|1
2021-05-04 18:47:52|3238212.28020322|63128264.7740938|579972527.204862|1
2021-05-04 18:49:29|3238212.28020322|63128264.7740938|579972527.204862|1
2021-05-04 18:50:50|3238212.28020322|63128264.7740938|579972527.204862|1
2021-05-04 18:52:23|3238212.28020322|63128264.7740938|579972527.204862|1
2021-05-04 18:53:56|3238212.28020322|70027895.7536416|579972527.204862|1
2021-05-04 18:55:14|3238212.28020322|63128264.7740938|542934136.124428|1
2021-05-04 18:55:38|3238212.28020322|63128264.7740938|46094979455.6081|1
2021-05-04 18:56:27|3238212.28020322|63128264.7740938|507785483.249348|1
2021-05-04 18:58:06|3238212.28020322|70027895.7536416|384684676.363243|1
2021-05-04 18:59:18|3238212.28020322|63128264.7740938|308869938.047268|1
2021-05-04 19:00:26|3238212.28020322|63128264.7740938|265256371.332385|1
2021-05-04 19:01:53|3238212.28020322|70027895.7536416|209083832.459647|1
2021-05-04 19:03:14|3238212.28020322|70027895.7536416|177177523.691084|1
2021-05-04 19:04:31|3238212.28020322|77508358.633869|149238281.402225|1
2021-05-04 19:05:46|3238212.28020322|77508358.633869|124892827.222483|1
2021-05-04 19:07:06|3238212.28020322|70027895.7536416|103790895.1807|1
2021-05-04 19:08:37|3238212.28020322|70027895.7536416|85604616.1788107|1
sqlite>
```

Database showing masked stored data



18:15 ··· 86%

Patient1 Health – Alert

Remote Health Monitoring System

to Me

Today, 14:55

Patient1 health reading is temp: 24 C, hum: 45, pulse:182 bpm. Please treat with urgent attention

Email Notification due to critical pulse rate reading of 182 bpm