Securing HTTP Communication for Homemade IoT Projects

Jose R. De la Vega López
Master in Computer Science
Advisor: Dr. Jeffrey Duffany
Electrical and Computer Engineering & Computer Science Department
Polytechnic University of Puerto Rico

Abstract *— The Internet of Things (IoT) is a topic in the Computer Science and Engineering field that has rapidly grown in the last couple of years. The popularity of the topic has made different companies to make proprietary IoT devices which can be purchased with ease. Although these devices are very accessible there are many students and professionals in the STEM area that like to create their own IoT devices using tools such as Raspberry Pi [1] and Arduino. Working with these devices in such projects will give students a boost in networking knowledge, but many miss the security part of the project. Using AES encryption to share messages and RSA public key encryption to encrypt the AES key can greatly improve the security of the HTTP communication for these projects as well as improving the student's knowledge in network security.*

Key Terms *— DIY, IoT, Network Security, Raspberry Pi.*

INTRODUCTION

The Internet of Things (IoT) is a trending topic in the Computer Science and Engineering area. According to IBM IoT is the concept of connecting any electronic device to the internet and to other devices [2]. It is not a surprise why this topic is so popular nowadays living in a world where convenience and information are a priority. People often have home devices connected to the internet such as the fridge, thermostats, humidifiers, artificial intelligence assistants, and even cleaning robots. These devices can bring a lot of helpful and useful information to the owner of said devices, leading to a more convenient life. Having devices connected to the internet imminently establishes a security risk. Security measures must be taken in order to protect the information stored and transferred by these devices. Although security risks are true for any IoT device, it is even more risky to create a homemade IoT device.

Creating one's own IoT device using tools such as the Raspberry Pi can be very educational and inexpensive for students and STEM professionals. The Raspberry Pi offers great resources for people to develop all sorts of projects, but people mainly focus on the functionality of the device they are building and not the security. The goal of this research is to provide a guide for people interested in making their homemade IoT devices on how to apply security to the system.

For this project a simple circuit was build. It consists of three LEDs of different colors, which the user could turn on and off by using a web client from anywhere in the world. Using the web interface the actual status of the different LEDs can be seen whether they are on or off and their color. For this project to work, a client and a server to communicate with each other were develop, and the circuit was connected to the server so that the server could turn on and off the LEDs. Figure 1 shows how the project is connected and how it should work.

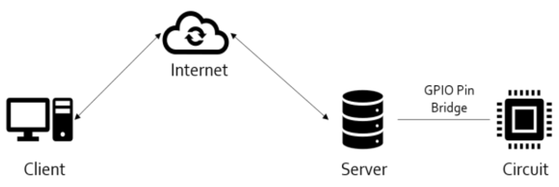


Figure 1
Client-Server Communication Diagram

The client and the server will communicate with each other over the internet, encrypting the communication using AES and RSA public key encryption. The server will be connected to a breadboard via the GPIO pins by using a bridge adapter. This way the server will be able to send voltage to the breadboard and turn on and off the LEDs of the circuit.

MATERIALS AND TOOLS

The total cost of the materials and tools used in this project is under $100 USD which is considered affordable and most of the tools can be used for various IoT projects. The list of materials is:

- **Raspberry Pi 3 Model B Breadboard:** This tool will be the base of the circuit that will be used to represent the IoT device.
- **GPIO Pin Bridge to Breadboard:** This is not a requirement for the project as a solder or female to male jumper wires can be used and it will serve the same purpose. This tool was added to the list because it made the usage of GPIO Pins easier.
- **220-ohm Resistors:** The resistors are used in the circuit so that it does not burn the LEDs.
- **LED (red, blue, green):** The LEDs are used in the circuit as well. The user will interact with the LEDs by using the client interface.
- **Jumper Wires:** The jumper wires are used to connect the whole circuit together.
- **Raspbian:** The Operating System used in this project is Raspbian, which is a distribution of Debian (Linux) optimized for the Raspberry Pi.
- **Python and PyCrypto [3]:** The code running in the server is built with Python and the Library used for encryption is PyCrypto. The PyCrypto library is mainly used for its AES and RSA encryption and decryption methods as well as RSA key generation.
- **React-JS and Crypto-JS [4]:** The client is built using React-JS which is a JavaScript Framework. The AES and RSA encryption functions are done with Crypto- JS.
- **Wireshark [5]:** The analysis of the network traffic for this project is done using Wireshark. Wireshark is an open source packet sniffing tool mainly used for network traffic analysis.

THE CIRCUIT

The circuit built for this project is a very simple one. Its purpose is to turn on and off 3 different LEDs of different colors, red, blue and green. This simple circuit is only used as a representation of any homemade IoT device. This circuit could have been one used to unlock doors, measure room temperature or any other functionality. Every time a user interacts with one of the LEDs via the web client, information about the specific LEDs are sent via HTTP protocol. The information each LED holds is the color of the LED, the GPIO pin that sends voltage to that LED and the status of the LED (if it is on or off). In a real-world scenario, a person could have more information regarding the LEDs, like for example the name of the room where the LED is in.

The idea behind creating this circuit was to be able to control something at home from far away over a TCP/IP network and verifying how secure was the data being transmitted. It was decided to use the LEDs because it was very simple to make, and all the materials for the circuit were available. More details about how the circuit was built can be found in the next section under the Setting Up the Raspberry Pi subsection.

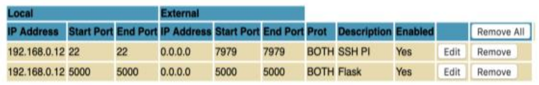
METHODOLOGY

The project can be divided into 4 parts: setting up the Raspberry Pi, building the code for the client, building the code for the server, and applying the security measures for the client-server communication.

Setting up the Raspberry Pi

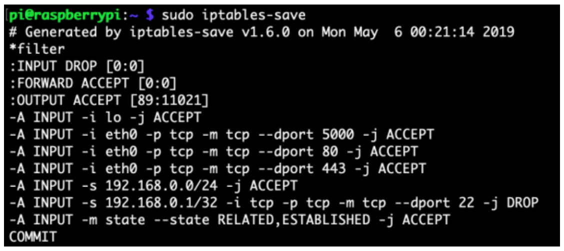
The first step is to set up the Raspberry Pi and the circuit. To make the Raspberry Pi available to external networks the first thing needed to do is to give the device a static IP address. This is because it will be forwarding traffic from external networks to that static IP address if the traffic is intended to reach the server of the IoT device. This can be done by accessing the /etc/dhcp.conf path in the operating system and manually typing the static IP address desired. Now that the device has a static IP address, the next step would be to add some rules into the router for port forwarding. In this case a forwarding rule stating that all traffic coming into the router intended for port 7979 will be sent to the Pi's address using port 22 (SSH). Instead of using port 22, 7979 was used to make it less obvious that the SSH port is open. The second forwarding rule, which can be seen in Figure 2, will be for all traffic intended for port 5000; any traffic for port 5000 will be sent to the Pi as well. This port will be used by the server to listen for commands.

At this point there is a device that is opened to external networks which can be accessed by using ports 5000 for the application, and 7979 (routed to 22) for SSH. Since the ports that wanted to be able to listen for communications in the Raspberry Pi are known, Iptables for the system can be created. Iptables are a software firewall for Linux systems, which can be used to drop or allow network traffic to the device. The Iptables will be used to create rules that make the Pi ignore all traffic unless it is intended for ports 22 or 5000. In Figure 3 the specific rules made for this project can be found. The most important rules are to drop everything by default and accept every output the device makes. Now that no traffic is received and anything can be sent, it is possible to create rules to allow the traffic specifically for ports 5000, 80, 443, and 22.



Local			External							
IP Address	Start Port	End Port	IP Address	Start Port	End Port	Prot	Description	Enabled		Remove All
192.168.0.12	22	22	0.0.0.0	7979	7979	BOTH	SSH PI	Yes	Edit	Remove
192.168.0.12	5000	5000	0.0.0.0	5000	5000	BOTH	Flask	Yes	Edit	Remove

Figure 2
Rules for Port Forwarding on the Router



```
pi@raspberrypi:~ $ sudo iptables-save
# Generated by iptables-save v1.6.0 on Mon May  6 00:21:14 2019
*filter
:INPUT DROP [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [89:11021]
-A INPUT -i lo -j ACCEPT
-A INPUT -i eth0 -p tcp -m tcp --dport 5000 -j ACCEPT
-A INPUT -i eth0 -p tcp -m tcp --dport 80 -j ACCEPT
-A INPUT -i eth0 -p tcp -m tcp --dport 443 -j ACCEPT
-A INPUT -s 192.168.0.0/24 -j ACCEPT
-A INPUT -s 192.168.0.1/32 -i tcp -p tcp -m tcp --dport 22 -j DROP
-A INPUT -m state --state RELATED,ESTABLISHED -j ACCEPT
COMMIT
```

Figure 3
Iptables Rules That allow Access to Ports 5000, 443, and 22

Now the Raspberry pi can be accessed from anywhere in the world through ports 5000 and 22. The next and last step of the first part of the project, setting up the Raspberry Pi, is to build the circuit. Figure 4 shows exactly how the circuit is built. The first thing to do is attach the GPIO Pin bridge from the Raspberry Pi to the Breadboard. This will allow to use voltage as desired from the Raspberry Pi to the breadboard using specific GPIO Pins. The pins used in this project are pin 17 for the red LED, pin 27 for the blue LED and pin 22 for the green LED. There is a jumper cable from each pin to the positive leg of the corresponding LED and a leg of a 220-ohm resistor in the negative leg of each LED, while the other end of each resistors go to the ground side of the board. At this point the Raspberry Pi es ready to receive network traffic and has a built-in circuit that can turn on/off three different LEDs.

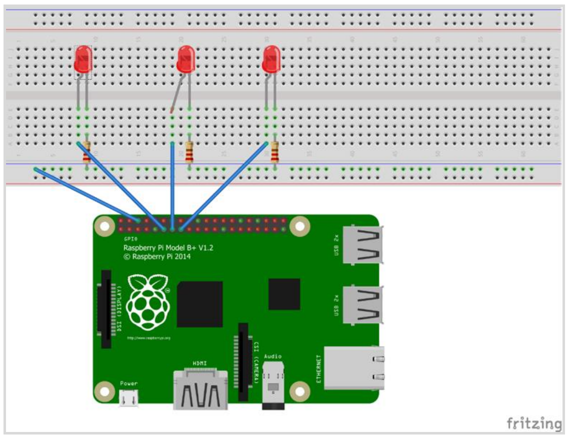


Figure 4
Layout of the Circuit Built in the Bread Board Designed Using Fritzing [6]

Building the Code for the Client and Server

The second and third parts of the project will be done simultaneously so they can be considered a single part. Building the client code and building the server code, could fuse into a single part because the functions were developed for the client and the server simultaneously to test what was being build. The client was developed using React-JS so that the user interface could update in real time as the user pressed the different buttons on the interface. The interface consists of three different buttons, one for each LED in the circuit. There are three main functions in the client code: request the status of the LEDs, change a LED status, and update the interface. To understand the client better, there will be a discussion on how the server works. The server developed using Flask [7] and Python has a dictionary in which each entry has information for the GPIO Pin number, the color of the LED and the status of the LED (on/off), as well as their respective values. When the client starts running, the first thing it does is make a GET request to the server asking for the dictionary. The dictionary is the sent as a JSON by the server. When the client receives this JSON, it updates the buttons on the interface with the information provided by the server as seen in Figure 5. The user has the option to change the value of the LEDs, which, when done, will send a PUT request to the server. This PUT request will contain a JSON with the GPIO Pin number and the new status. Once the server receives the PUT request it will update the dictionary of the LEDs and then it will turn on or off the LED as requested. The LEDs are turned on/off by the Raspberry Pi using the GPIO pins on the circuit.

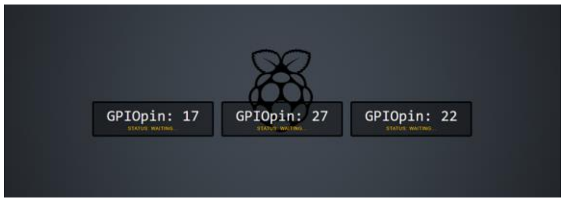
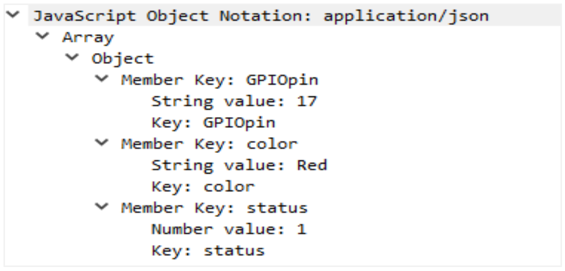


Figure 5
React-JS Client Interface

Once the functions are implemented, it can be said that a client and a server that can communicate with each other via REST API over HTTP was effectively built. Anyone in the world who has the client code can turn on and off the LEDs of the new homemade IoT device. Wireshark, an open source packet sniffer that would help analyze the traffic sent to and by the IoT device, was used now to test the security of the HTTP communication. To analyze the network traffic, a person just has to run Wireshark and select the network interface of the server so that they could sniff everything going in and out of the Pi. Once the response to the initial GET request that a client does (this response has the status of all the LEDs) is found, the information sent over HTTP can be found by digging deeper. If this is done, the whole JSON in plain text can be seen as shown in Figure 6. This is a huge security risk since anyone that is sniffing a network will be able to see the status of the LEDs in the house. In this case only simple LEDs are used to represent the circuit, but this could be another homemade IoT device that controls things in your home such as temperatures or even the lock of your door.



```
JavaScript Object Notation: application/json
  Array
    Object
      Member Key: GPIOpin
          String value: 17
          Key: GPIOpin
      Member Key: color
          String value: Red
          Key: color
      Member Key: status
          Number value: 1
          Key: status
```

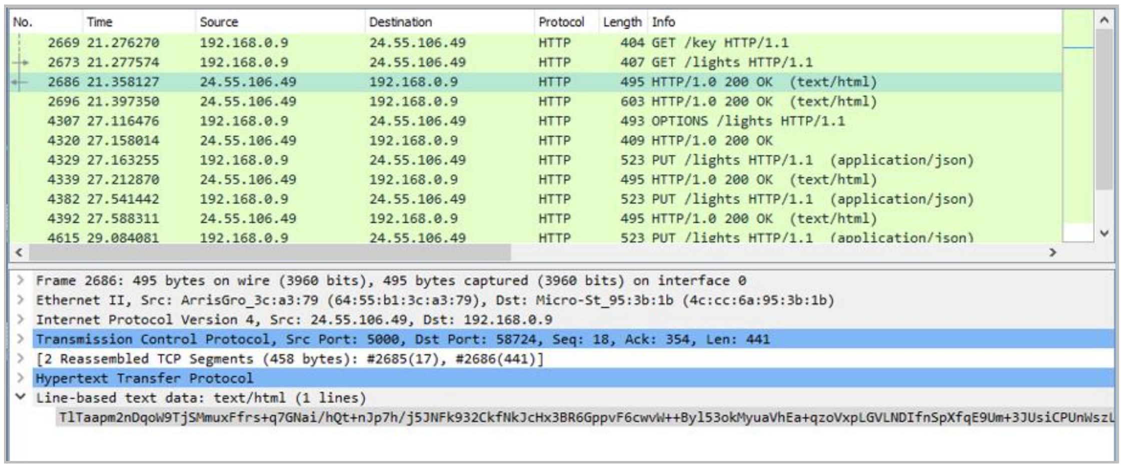
Figure 6
Plain Text of the JSON Response from the Server to the Client as Seen in Wireshark

Applying Security Measurements to the System

Now that there is a functioning client-server communication that is not secure, some communication security was implemented by using AES encryption. AES stands for Advanced Encryption Standard, and AES is a symmetric block cipher. In symmetric encryption the key needs to be shared between the sender and the receiver of the messages. This means that the key must be sent through a secure channel, otherwise the secured communication can be compromised. To make the AES encryption, some changes had to be done in both the client and the server. The first change was that now the server would have a 16-byte key that both the server and the client will use for the AES encryption and decryption. The second change is that now the client’s first request will be a GET request for the AES key instead of the JSON. Once the client makes the GET request, the server will send the key to the client and then the client will make the GET request to the server asking for the JSON. When the request for the JSON gets to the server, the server will AES encrypt the JSON using the 16-byte key and it will send it to the client. The client will use the key to decrypt the AES JSON and then it will do the same it did originally. Another change in the client is that now when the user clicks a button, it will encrypt the JSON before sending it to the server. Basically, both the server and the client now use AES encryption for the communication. After the changes are made, it is time to analyze the network traffic again. Using Wireshark, the server response was captured to the first two GET requests of the client, which are for the key and for the JSON. After finding the responses it can be seen (Figure 7 and Figure 8) that the JSON is now AES encrypted, but the key is sent in plain text.

Although having a secure HTTP communication is close, it is still needed to find a way to secure the AES key. To solve this problem, RSA public key encryption is going to be used. The advantage of public key encryption is that it is asymmetric. An asymmetric encryption method means that each unit will have two keys: a private and a public key. The public key of each unit is shared between them, while the private is kept save and secret. The idea behind this type of encryption is that if Bob and Alice want to communicate with each other, Bob will encrypt the message using Alice's public key, which he has. The only key that can decrypt the message encrypted with Alice's public key is Alice's private key, which she has. This eliminates having to share keys using a secure channel. To implement this, the server and the client's RSA public and private keys was generated using python and the key distribution was made. The server has both of its keys and the clients public key and the client has both its keys and the server public key. Now the changes to the code would be to add, in both the client and server, functions to properly encrypt and decrypt using RSA public key encryption. It is important to note that this encryption is slower than AES, thus it will only be used once in the client-server communication to encrypt and decrypt the AES key. All the code will remain the same, but when the client makes the GET request of the key to the server, the latter will RSA encrypt the AES key using the public key of the client. In the other hand, the client will receive the RSA encrypted key and it will decrypt it using its own private key. Once the key is decrypted, the whole code will run the same as before using AES encryption between communications.

Now that both RSA and AES encryption were implemented, another network traffic analysis using Wireshark must be run. This time the responses to the two initials GET requests done by the client will be captured, which are supposed to contain the AES key and the JSON. As expected, Figure 9 and Figure 10 show that the response to the GET request for the JSON is still AES encrypted, just as before, but now the AES key in plain text can no longer be seen. Now the key RSA are getting encrypted.

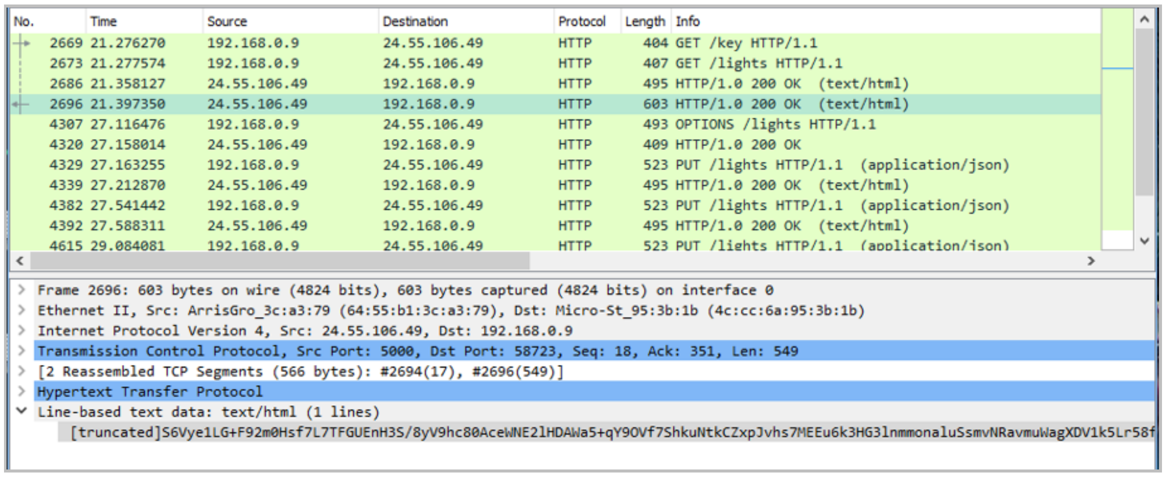


```
No.    Time         Source          Destination     Protocol Length Info
  2669 21.276270    192.168.0.9     24.55.106.49    HTTP      404 GET /key HTTP/1.1
  2673 21.277574    192.168.0.9     24.55.106.49    HTTP      407 GET /lights HTTP/1.1
  2686 21.358127    24.55.106.49    192.168.0.9     HTTP      495 HTTP/1.0 200 OK  (text/html)
  2696 21.397350    24.55.106.49    192.168.0.9     HTTP      603 HTTP/1.0 200 OK  (text/html)
  4307 27.116476    192.168.0.9     24.55.106.49    HTTP      493 OPTIONS /lights HTTP/1.1
  4320 27.158014    24.55.106.49    192.168.0.9     HTTP      409 HTTP/1.0 200 OK
  4329 27.163255    192.168.0.9     24.55.106.49    HTTP      523 PUT /lights HTTP/1.1  (application/json)
  4339 27.212870    24.55.106.49    192.168.0.9     HTTP      495 HTTP/1.0 200 OK  (text/html)
  4382 27.541442    192.168.0.9     24.55.106.49    HTTP      523 PUT /lights HTTP/1.1  (application/json)
  4392 27.588311    24.55.106.49    192.168.0.9     HTTP      495 HTTP/1.0 200 OK  (text/html)
  4615 29.084081    192.168.0.9     24.55.106.49    HTTP      523 PUT /lights HTTP/1.1  (application/json)

> Frame 2686: 495 bytes on wire (3960 bits), 495 bytes captured (3960 bits) on interface 0
> Ethernet II, Src: ArrisGro_3c:a3:79 (64:55:b1:3c:a3:79), Dst: Micro-St_95:3b:1b (4c:cc:6a:95:3b:1b)
> Internet Protocol Version 4, Src: 24.55.106.49, Dst: 192.168.0.9
> Transmission Control Protocol, Src Port: 5000, Dst Port: 58724, Seq: 18, Ack: 354, Len: 441
> [2 Reassembled TCP Segments (458 bytes): #2685(17), #2686(441)]
> Hypertext Transfer Protocol
v Line-based text data: text/html (1 lines)
    TlTaapm2nDqoW9TjSMmuxFfrs+q7GNai/hQt+nJp7h/j5JNFk932CkfNkJcHx3BR6GppvF6cwvW++Byl53okMyuaVhEa+qzoVxpLGVLNDIfnSpXfqE9Um+3JUsiCPUnWszL
```

Figure 9

JSON Message Encrypted Using AES as Seen in Wireshark



```
No.   Time        Source          Destination     Protocol  Length Info
2669 21.276270   192.168.0.9     24.55.106.49    HTTP       404 GET /key HTTP/1.1
2673 21.277574   192.168.0.9     24.55.106.49    HTTP       407 GET /lights HTTP/1.1
2686 21.358127   24.55.106.49    192.168.0.9     HTTP       495 HTTP/1.0 200 OK  (text/html)
2696 21.397350   24.55.106.49    192.168.0.9     HTTP       603 HTTP/1.0 200 OK  (text/html)
4307 27.116476   192.168.0.9     24.55.106.49    HTTP       493 OPTIONS /lights HTTP/1.1
4320 27.158014   24.55.106.49    192.168.0.9     HTTP       409 HTTP/1.0 200 OK
4329 27.163255   192.168.0.9     24.55.106.49    HTTP       523 PUT /lights HTTP/1.1  (application/json)
4339 27.212870   24.55.106.49    192.168.0.9     HTTP       495 HTTP/1.0 200 OK  (text/html)
4382 27.541442   192.168.0.9     24.55.106.49    HTTP       523 PUT /lights HTTP/1.1  (application/json)
4392 27.588311   24.55.106.49    192.168.0.9     HTTP       495 HTTP/1.0 200 OK  (text/html)
4615 29.084081   192.168.0.9     24.55.106.49    HTTP       523 PUT /lights HTTP/1.1  (application/json)

> Frame 2696: 603 bytes on wire (4824 bits), 603 bytes captured (4824 bits) on interface 0
> Ethernet II, Src: ArrisGro_3c:a3:79 (64:55:b1:3c:a3:79), Dst: Micro-St_95:3b:1b (4c:cc:6a:95:3b:1b)
> Internet Protocol Version 4, Src: 24.55.106.49, Dst: 192.168.0.9
> Transmission Control Protocol, Src Port: 5000, Dst Port: 58723, Seq: 18, Ack: 351, Len: 549
> [2 Reassembled TCP Segments (566 bytes): #2694(17), #2696(549)]
> Hypertext Transfer Protocol
v Line-based text data: text/html (1 lines)
    [truncated]S6Vye1LG+F92m0Hsf7L7TFGUEnH3S/8yV9hc80AceWNE2lHDAWa5+qY9OVf7ShkuNtkCZxpJvhs7MEEu6k3HG3lnmmonaluSsmvNRavmuWagXDV1k5Lr58f
```

Figure 10

AES Key Encrypted Using RSA Public Key Encryption as Seen in Wireshark

Conclusion

This project is a great way for students and STEM professionals interested in IoT to be able to develop a homemade IoT device while learning important concepts in network security. The project can be considered a homemade version of Open SSL [8]. There are a lot of software designed for Raspberry Pi that serve as IoT controllers, but it takes away the users experience to learn about security. The project was made using inexpensive items so that students with limited resources could follow along and learn about network security for IoT devices. By using HTTP, the user avoids spending money on SSL certificates, and they gain knowledge on how they perform the encryption used in said technology. Overall, the researcher is very proud of the project and thinks every student or STEM professional interested in learning about homemade IoT devices and HTTP communication security should give this project a try.

Future Work

The system is far from being completely secure. This project focused on the HTTP communication security. Although other security measures to the Raspberry Pi were added, such as the Iptables firewall and disguising the ports used with port forwarding, the system needs other security measures. The first thing missing is authentication for the system. As of right now, anyone that has the client can turn on and off the LEDs without authenticating. Another thing listed as future work would be to create a more complex and useful circuit to better demonstrate the importance of applying encryption to the communication.

References

[1] Raspberrypi.org. (2019). *Raspberry Pi 3 Model B – Raspberry Pi* [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-mode-b-plus/.

[2] J. Clark. (2017, Sept. 19). *What is the Internet of Things, and how does it work?* [Online]. Available: https://www.ibm.com/blogs/internet-of-things/what-is-the-iot/.

[3] Python Package Index. (2019). *Pycrypto* [Online}. Available: https://pypi.org/project/pycrypto/.

[4] Brix. (2019, Jan. 4). *Brix/crypto-is* [Online]. Available: https://github.com/brix/crypto-js.

[5] Wireshark.org. (2019). *Wireshark* [Online]. Available: https://www.wireshark.org/.

[6] Fritzing.org. (2019). *Fritzing* [Online]. Available: http://fritzing.org/.

[7] Flask.pocoo.org. (2019). *Flask* [Online]. Available: http://flask.pocoo.org/.

[8] *SSL Certificate Digital Certificate Authority*, 2019. [Online]. Available: https://www.ssl.com/.