Encrypted Wi-Fi Keyboard

Andrew Lu and Qi Guo, CMPSC 176B, Winter 2020

Introduction

The goal of this project is to create a wireless keyboard capable of securely communicating keystrokes over long distances.

Most keyboards on the market today communicate with devices through either a physical connection (e.g. USB and PS/2) or through Bluetooth / Bluetooth Low Energy (BLE). However, both methods place a limit on how far the keyboard can be from the connected computer, as physical connections are limited by cable length and Bluetooth connections can only transmit up to 100 meters, depending on the class used. This project aims to circumvent this limitation by instead using the user's internet connection to communicate keystrokes via a client-server socket communication application.

Implementation

The Wi-Fi keyboard will feature a wired keyboard connected to a Raspberry Pi Zero W that runs a client application that listens for keystrokes on the attached keyboard. If keystrokes are detected, the client (keyboard) will encrypt them using public-key encryption to ensure safety of information. The keystrokes will then be sent via Wi-Fi to a computer running the server application, which will then decrypt the message and apply the decrypted keystrokes on the server computer. Messages will be sent using either a TCP or UDP socket connection, to be determined after a performance analysis is done comparing both methods.

We used C++ to implement the keyboard (client side) application. C++ was chosen because we were unsuccessful in getting our originally researched Python keyloggers to perform as expected. We have instead opted to build our own keylogger by reading and decoding the raw USB input from the keyboard. To accomplish this, we made a mapping between the key name and the HID scan code representation. C++ was also chosen for its ability to fully manage memory and perform low-level socket programming.

We used Python to implement the computer (server) side of the project. Python was selected for its ability to perform low-level socket programming natively using its libraries, and the many keystroke emulator packages available. Python was also selected for its ability to execute shell commands.

Messages are sent between the two devices using the following specification:

- Every message packet (before encryption) is two bytes long.
- The first byte contains a number, between 0 and 255, that represents the HID scan code of the key pressed.
- The second byte contains the action, i.e. pressed(1), held(2), or released (3).

Keystroke emulation was accomplished using the Win32 API, since we both use Windows computers. We will be using PyWin32, which is a Python wrapper for the Win32 APIs. PyWin32 was selected over other keyboard emulators like pyautogui for its agility. Unlike pyautogui, pywin32 does not have any noticeable delay, and was able to keep up with my 100 WPM typing speed. However, PyWin32 only works on Windows, whereas pyautogui is a cross-platform library. We decided that speed was more important than cross-platform functionality.

RSA public-key cryptography was performed using self-coded RSA algorithms that match the spec described in the lecture. On both the computer and keyboard, we first found two relatively small prime numbers, which we kept small because of our small packet sizes. We then used those numbers to generate a public and private key pair on each device.

If encryption is enabled, communication between the devices starts by exchanging the public keys. Once the public keys are exchanged, the keyboard (client) will generate a symmetric key of two bytes that will be used to encrypt all data packets. (The key is two bytes because our packet size is two bytes. This may not provide the most security, but it at least provides some). The key is then encrypted twice, first using the keyboard's private key, then using the computer's public key. The encrypted message, now 8 bytes long, is sent over to the computer (server), which decrypts it using first the computer's private key, then the keyboard's public key.

Once the initial exchange of the symmetric key is accomplished, all messages sent from that point on will be encrypted using the shared key. The shared key encryption is accomplished by simply performing an XOR operation between the packet and the key, allowing it to retain its two-byte size.

Execution

This project is composed of two applications: a client (that runs on the keyboard) and a server (that runs on the computer).

Keyboard Requirements:

- Linux computer (we used a Raspberry Pi 3 B+)
- Attached USB keyboard
- Access to the internet
- Administrative privileges to read from direct USB input

Computer Requirements:

- Windows Computer (we used Windows 10)
- Win32 APIs installed
- Python 3 with the Pywin32 package
- Administrative privileges
- Access to the internet, with a port of your choosing available

Instructions:

- On the computer side:
    - Ensure that you have administrative privileges to your computer.
    - Install the Win32 API, Python 3, and Pywin32, if not already installed.
    - If your computer is behind a NAT, and you wish to access it from a different network, port forward a port of your choosing to be accessible outside of the NAT.
    - Clone the following Git repo: https://github.com/andrewhlu/wifi-keyboard
    - Open Command Prompt, and cd into the directory.
    - Run the python script as follows:

    ```
    python computer.py <port> <type> <encryption>
    ```

    where <port> is a port number, <type> is 'UDP' or 'TCP', and <encryption> is 'true' or 'false'.

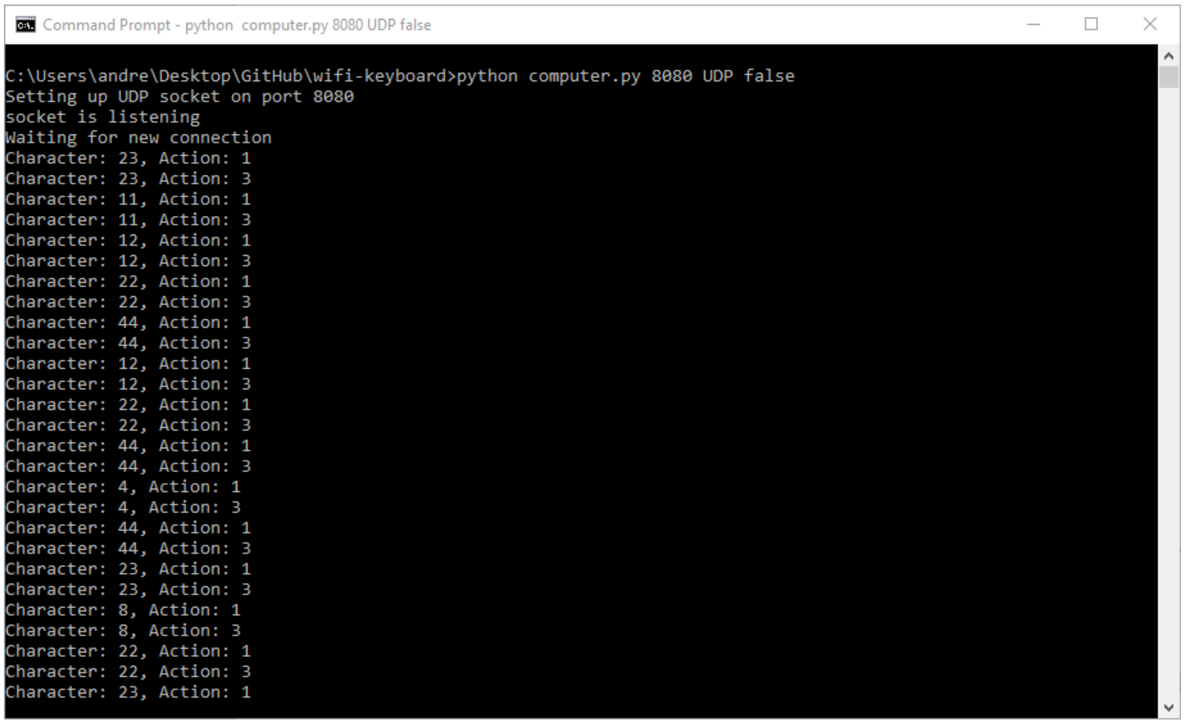
- On the keyboard side:
    - Attach the keyboard to a USB port on your computer.
    - Ensure that you have administrative privileges to your computer.
    - Clone the following Git repo: https://github.com/andrewhlu/wifi-keyboard
    - Open a terminal, and cd into the directory.
    - Build the code using `make all`.
    - Run the C++ executable as follows:

    ```
    ./input <ip> <port> <type> <encryption>
    ```

    where <ip> is the server computer's IP address, <port> is the port number used above, <type> is 'UDP' or 'TCP' and <encryption> is 'true' or 'false'.

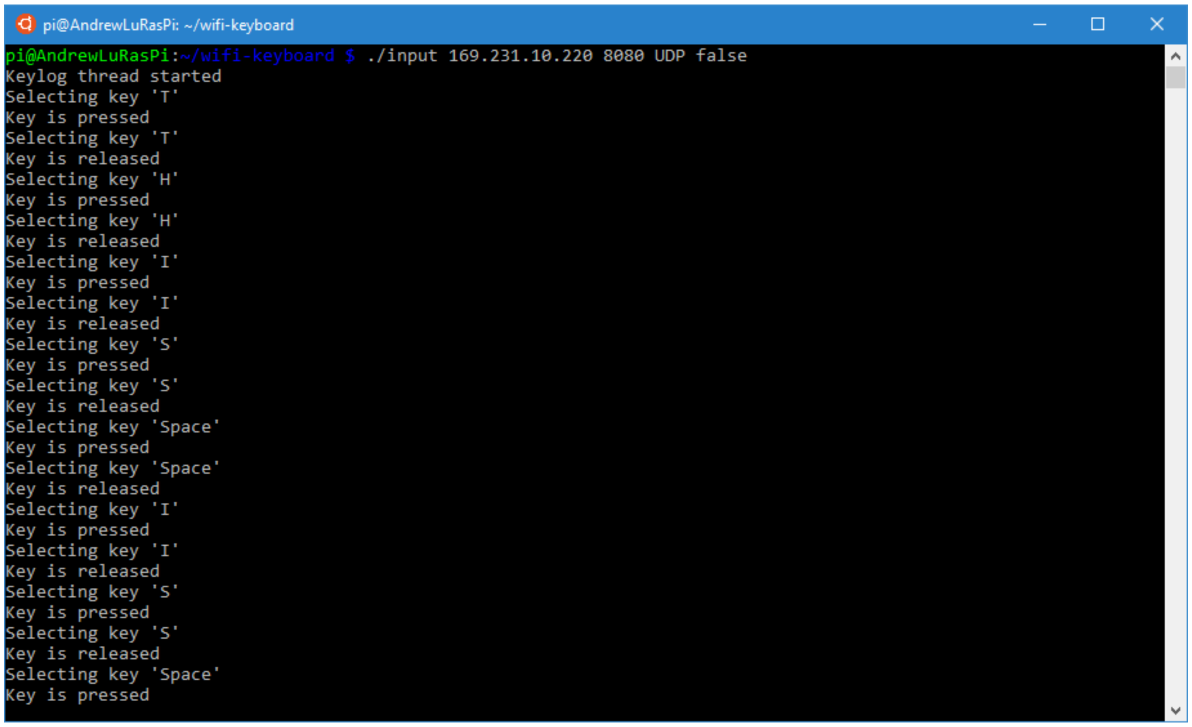
Sample Execution Screenshots

Computer Side, UDP, without encryption:



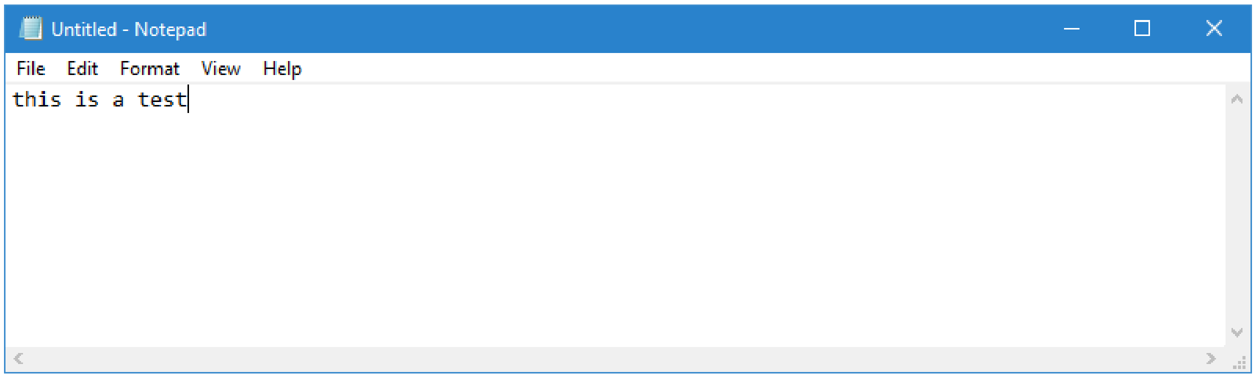
```
C:\Users\andre\Desktop\GitHub\wifi-keyboard>python computer.py 8080 UDP false
Setting up UDP socket on port 8080
socket is listening
Waiting for new connection
Character: 23, Action: 1
Character: 23, Action: 3
Character: 11, Action: 1
Character: 11, Action: 3
Character: 12, Action: 1
Character: 12, Action: 3
Character: 22, Action: 1
Character: 22, Action: 3
Character: 44, Action: 1
Character: 44, Action: 3
Character: 12, Action: 1
Character: 12, Action: 3
Character: 22, Action: 1
Character: 22, Action: 3
Character: 44, Action: 1
Character: 44, Action: 3
Character: 4, Action: 1
Character: 4, Action: 3
Character: 44, Action: 1
Character: 44, Action: 3
Character: 23, Action: 1
Character: 23, Action: 3
Character: 8, Action: 1
Character: 8, Action: 3
Character: 22, Action: 1
Character: 22, Action: 3
Character: 23, Action: 1
```

Keyboard Side, UDP, without encryption:



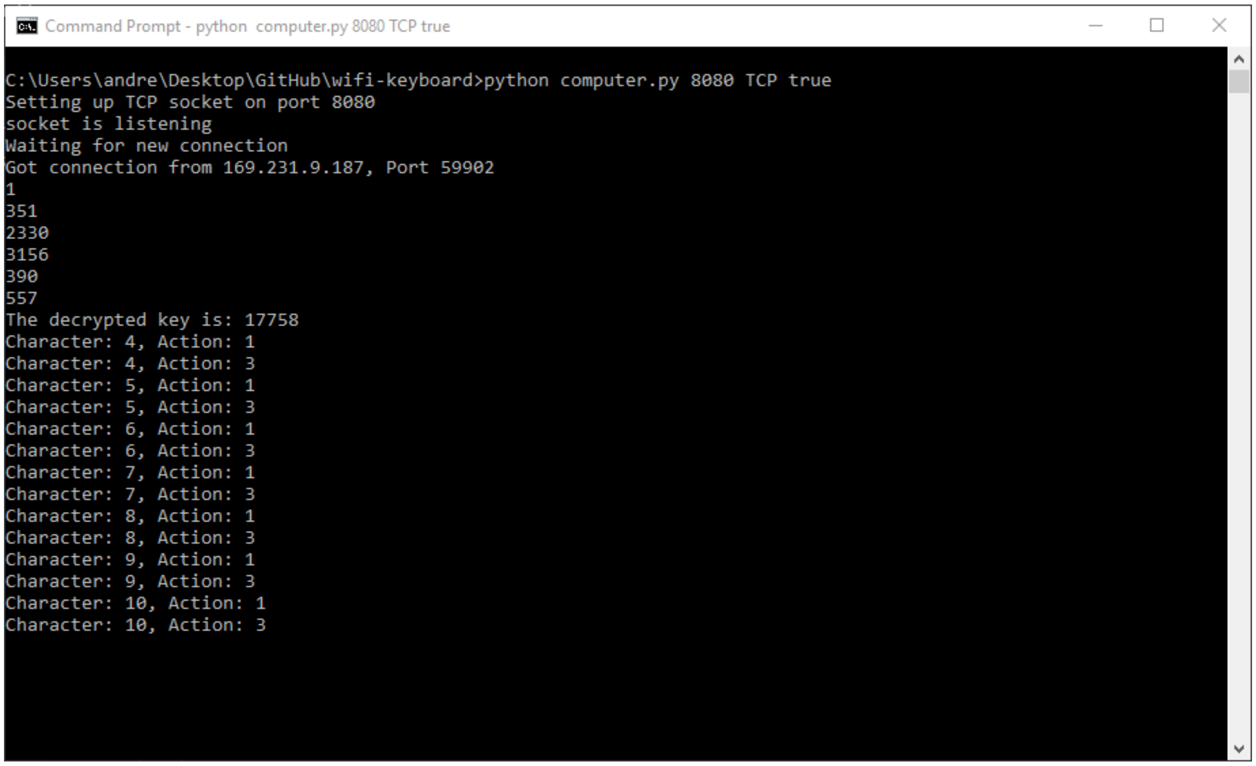
```
pi@AndrewLuRasPi:~/wifi-keyboard $ ./input 169.231.10.220 8080 UDP false
Keylog thread started
Selecting key 'T'
Key is pressed
Selecting key 'T'
Key is released
Selecting key 'H'
Key is pressed
Selecting key 'H'
Key is released
Selecting key 'I'
Key is pressed
Selecting key 'I'
Key is released
Selecting key 'S'
Key is pressed
Selecting key 'S'
Key is released
Selecting key 'Space'
Key is pressed
Selecting key 'Space'
Key is released
Selecting key 'I'
Key is pressed
Selecting key 'I'
Key is released
Selecting key 'S'
Key is pressed
Selecting key 'S'
Key is released
Selecting key 'Space'
Key is pressed
```

Sample Output into Notepad for above case:



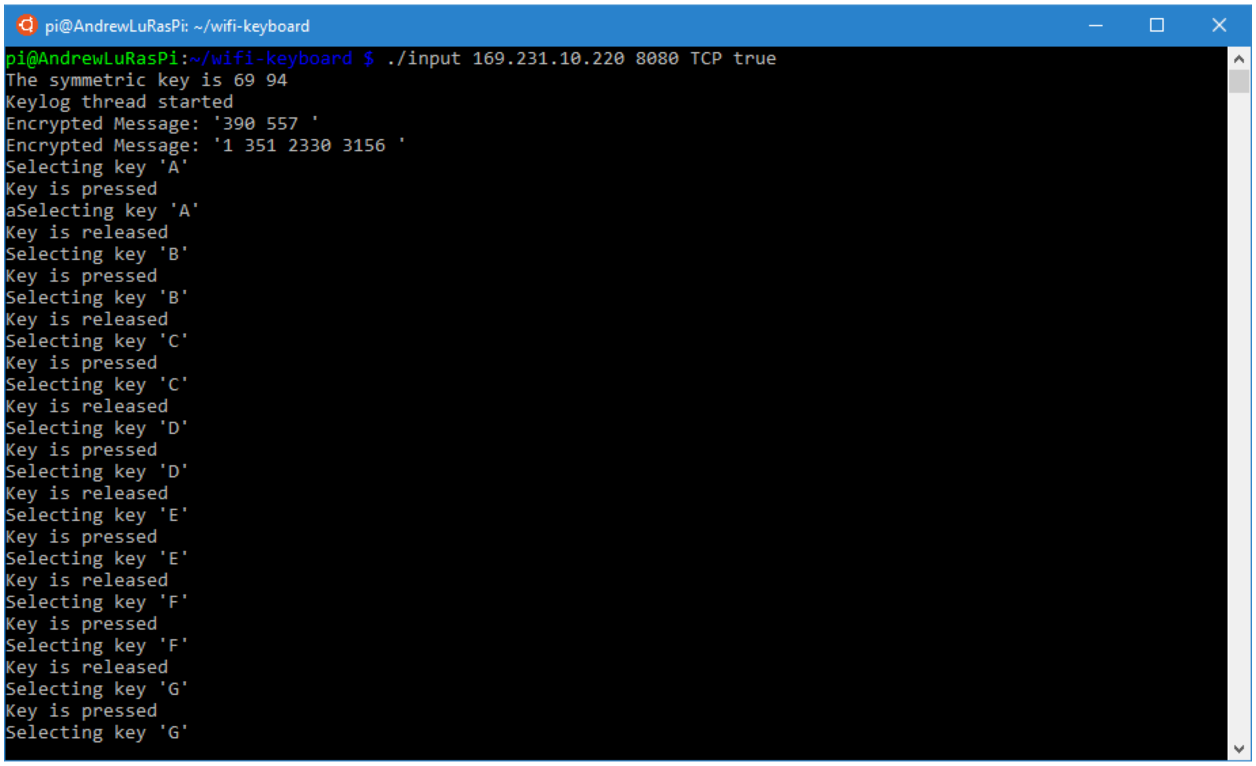
```
this is a test
```

Computer Side, TCP, with encryption:



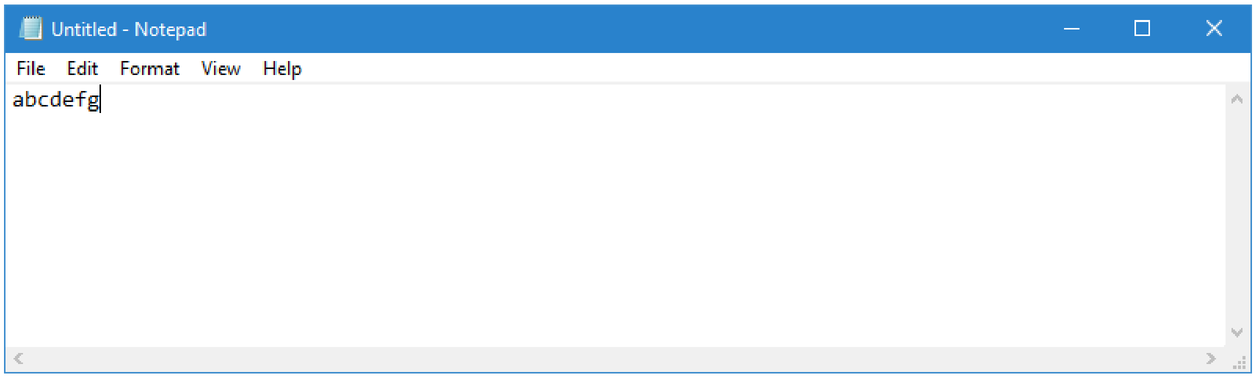
```
C:\Users\andre\Desktop\GitHub\wifi-keyboard>python computer.py 8080 TCP true
Setting up TCP socket on port 8080
socket is listening
Waiting for new connection
Got connection from 169.231.9.187, Port 59902
1
351
2330
3156
390
557
The decrypted key is: 17758
Character: 4, Action: 1
Character: 4, Action: 3
Character: 5, Action: 1
Character: 5, Action: 3
Character: 6, Action: 1
Character: 6, Action: 3
Character: 7, Action: 1
Character: 7, Action: 3
Character: 8, Action: 1
Character: 8, Action: 3
Character: 9, Action: 1
Character: 9, Action: 3
Character: 10, Action: 1
Character: 10, Action: 3
```

Keyboard Side, TCP, with encryption:


pi@AndrewLuRasPi: ~/wifi-keyboard

```
pi@AndrewLuRasPi:~/wifi-keyboard $ ./input 169.231.10.220 8080 TCP true
The symmetric key is 69 94
Keylog thread started
Encrypted Message: '390 557 '
Encrypted Message: '1 351 2330 3156 '
Selecting key 'A'
Key is pressed
aSelecting key 'A'
Key is released
Selecting key 'B'
Key is pressed
Selecting key 'B'
Key is released
Selecting key 'C'
Key is pressed
Selecting key 'C'
Key is released
Selecting key 'D'
Key is pressed
Selecting key 'D'
Key is released
Selecting key 'E'
Key is pressed
Selecting key 'E'
Key is released
Selecting key 'F'
Key is pressed
Selecting key 'F'
Key is released
Selecting key 'G'
Key is pressed
Selecting key 'G'
```

Sample Output into Notepad for above case:


Untitled - Notepad

File Edit Format View Help

```
abcdefg
```

- The actual symmetric keys will be hidden in the final code.
- The symmetric keys are equivalent – the keyboard side just displays the decimal representations of each byte, whereas the computer side displays the decimal representation of the whole key.

  69 * 256 + 94 = 17758

Timeline

Milestone 1:

- Develop client and server applications capable of sending messages using both TCP and UDP sockets.
- Perform a performance evaluation comparing the two methods.

Milestone 2:

- Develop a client application capable of recording keystrokes from a Linux machine.
- Develop a server application capable of taking those keystrokes and applying them on a Windows machine.

Milestone 3:

- Merge the two client applications and the two server applications together to allow us to send keystrokes unencrypted over a network.
- Perform a performance evaluation comparing the two communication protocols, along with Bluetooth using an existing Bluetooth keyboard.

Milestone 4:

- Implement public-key encryption on the client and server applications, allowing data to be transmitted in an encrypted state.
- Perform a final performance evaluation comparing encrypted versus non-encrypted messages on both TCP and UDP sockets.

Future Improvements

- There are some known issues with our program, described below. We hope to fix those in the future.
- We will also try to implement automatic device discovery in the future, given that both devices are on the same network.

Known Issues

- Occasionally, when typing too fast while using any TCP configuration, you may experience a KeyError error from the Win32 API.
- The UDP implementation for asymmetric encryption is not completely functional, as we were unable to get a working implementation for the key-pair exchange in time.
- Some very specific keys, like "Fn" and media keys, do not work with this application.

Performance Evaluation

To evaluate the performance of our Wi-Fi keyboard, we can measure the latency and the success rate of input keystrokes at a fixed distance from a host computer for each of the various methods of communication. We hope to be able to compare the following four methods of wireless communication:

- TCP, without encryption
- TCP, with public-key encryption
- UDP, without encryption
- Bluetooth (using a standard Bluetooth keyboard)

We had originally planned to measure the latency for keypresses as a method of evaluating performance, However, since the latency is too small to be noticeable for all methods, timing such latency was near impossible.

We will instead perform a performance evaluation based on the success rate of keypresses. The success rate will be computed by applying 100 keystrokes on the client computer and measuring the number of keystrokes that were applied on the server computer. This will be performed twice for each method. For the Bluetooth method, I will stand two rooms away and try the test.

The phrase typed will be "I like to eat pie! " five times. I will type this at my normal typing speed (around 100 WPM).

The results are tabulated below:

Communication Method	Trial 1 Success Rate	Trial 2 Success Rate	Average
TCP, without encryption	91/100	92/100	91.5/100
TCP, with encryption	96/100	95/100	95.5/100
UDP, without encryption	100/100	100100	100/100
Bluetooth	100/100	100/100	100/100

From this, we can see that UDP, without encryption, was the best performing option. This is likely because it does not require a full handshake for every keystroke, and as a result, is able to keep up with the fast typing speeds. The UDP implementation performed as well as the standard Bluetooth keyboard in our test, but I envision that the UDP implementation will work much better from longer distances. While some people may appreciate the added security with TCP + encryption, I value accuracy and speed more importantly, so I would prefer the UDP implementation over the TCP + encryption implementation.

References

Socket Programming using Python

- Basic Tutorial: https://www.geeksforgeeks.org/socket-programming-python/
- Socket Library Documentation: https://docs.python.org/3/library/socket.html
- CMPSC 176A Homework 2 Starter Code

Custom Keylogger

- USB Scan Codes: https://www.win.tue.nl/~aeb/linux/kbd/scancodes-14.html
- HID Usage Tables: https://www.usb.org/sites/default/files/documents/hut1_12v2.pdf
- Shell Commands in C: https://linux.die.net/man/3/system

Keyboard Emulator

- PyWin32 for Windows: https://pypi.org/project/pywin32/
- pyautogui: https://pyautogui.readthedocs.io/en/latest/
- SendKeys Command (to apply keystrokes): https://ss64.com/vb/sendkeys.html
- Shell Commands in Python: https://janakiev.com/blog/python-shell-commands/

Public Key Encryption

- How it Works: https://www.cloudflare.com/learning/ssl/how-does-public-key-encryption-work/
- Encryption using RSA in Python (not used): https://medium.com/@ashiqgiga07/asymmetric-cryptography-with-python-5eed86772731
- RSA Encryption in C++: http://www.trytoprogram.com/cpp-examples/cplusplus-program-encrypt-decrypt-string/
- Feb-03 Lecture: https://sites.cs.ucsb.edu/~almeroth/classes/W20.176B/lectures/Lecture-02-03.pdf