University of Alberta

Master of Science
in Internetwork

Capstone Project - Doorbell System

Final Report

by
Zeqi Song

Abstract

Our doorbell system is a client-server architecture program to improve the security of SSH. Some port scanning software such as nmap can detect the listening port of SSH and thus the port is a target for hackers. With running the doorbell system and under its control, the listening port can be protected. Only the knocker who has the correct secret can enable the SSH service and connect the server. In this project, we have designed the doorbell protocol, implemented the doorbell system, and tested the program with various situations. We also explain the implementation of APIs used in the doorbell system and demonstrate a working system in this report.

Keywords: doorbell system, SSH, UDP, TCP

Table of Content

1 Introduction

1.1 Motivation

Nowadays, with computer networks, people are able to not only work on their local computers but also access a server remotely. SSH is such a kind of network protocol that provides a secure channel over the Internet and has been widely used in many areas of servers, from logging into a remote machine and executing commands, to transferring files over the secure channel. Typically, SSH uses public-private key pairs to encrypt a network connection. Given the fact that the encryption key of 1024 bits is difficult to crack, data transmitted over the SSH channel is relatively safe for now. However, the server running with SSH has a TCP listening port used to receive a client's connection requests. Some port scanning software such as nmap can detect this listening port and hackers are able to utilize this target port to apply attacks to the server. Even changing the default port(i.e. 22) will not provide a significant improvement in security, as the listening port is still existing and can be detected by scanning.

1.2 Objectives

The project was approved to design and build a "doorbell" system to reinforce the network security of SSH. A doorbell system is analogous to a house's doorbell which would make the door unlocked when visitors enter the correct code and get approved by the host. Running the doorbell system and under its control, the SSH port is initially kept closed. The system will open the SSH port only if clients send proper messages to a series of UDP ports and successfully pass the verification within a window of time. Any incorrect message or out-of-order delivery would keep the SSH port closed, with any ordering problems ultimately leading to a protocol time-out, and a retry. As UDP sockets in the doorbell system do not require a three-way handshake to establish connections, it will keep silent for any malicious scanning.

2 Background

In SSH, a TCP port is used for listening to a client' connection requests. TCP is a connection-oriented protocol which requires a three-way handshake to initiate a connection, giving hackers opportunities to obtain port information from the server and thus vulnerable to port-scanning attacks. In contrast, UDP is a connectionless and unreliable protocol, which means that there is no connection between servers and clients. Because of its connectionless feature, the UDP socket can keep silent for malicious attempts and will not provide any information to hackers who are scanning ports. An additional mechanism will be added to the doorbell system to provide the robustness given the unreliability of the UDP protocol. For example, all doorbell actions have time-outs to support a simple time-out and retry strategy.

3 Specification & Design

3.1 Protocol Design

3.1.1 Message Format

As the doorbell system needs to keep silent for port scanning and malicious attempts, a proper message format is essential. If the doorbell system detects an invalid message, it will raise an exception and make the system keep silent. There are 4 types of messages used in exchanging information between clients and servers. All messages are encrypted by the client or server public key. Assuming that the server has port i_0, i_1, i_2 ... i_n:

- Message#1 containing a secret and a public key is sent to the server at port i_0, i_1, i_2 ... i_{n-1} to initialize the verification function for the client.

Name	Message#1
Direction	Client → Server
Format	Secret@@@ClientPublicKey

- Message#2 contains a one-time hash created by the server. Once the server receives the required number of Message#1s and the arriving order of the messages is correct, the server will send Message#2 at port i_{n-1} to the client.

Name	Message#2
Direction	Server → Client
Format	HashValue

- Message#3 is the one-time hash received from the server and re-encrypted by the client. Message#3 is sent to the server at port i_n.

Name	Message#3
Direction	Client → Server
Format	HashValue

- Message#4 contains SSH listening port and time-to-live of the port.

Name	Message#4
Direction	Server → Client
Format	SSH_port@@@SSH_TTL

3.1.2 Doorbell Protocol

The doorbell protocol works as follows: assuming that there are UDP sockets at ports A, B, and C on the doorbell server and ports A, B, and C are not sequentially numbered. The secrets known to the doorbell knocker are the values of A, B, and C (which are analogous to a combination code) and a public key corresponding to a private key already stored on the doorbell server. The values of A, B, C, and the public and private keys are configured out-of-band beforehand, like public-private keys of SSH. The sequence of the doorbell protocol is shown in the Figure 3.1.2:

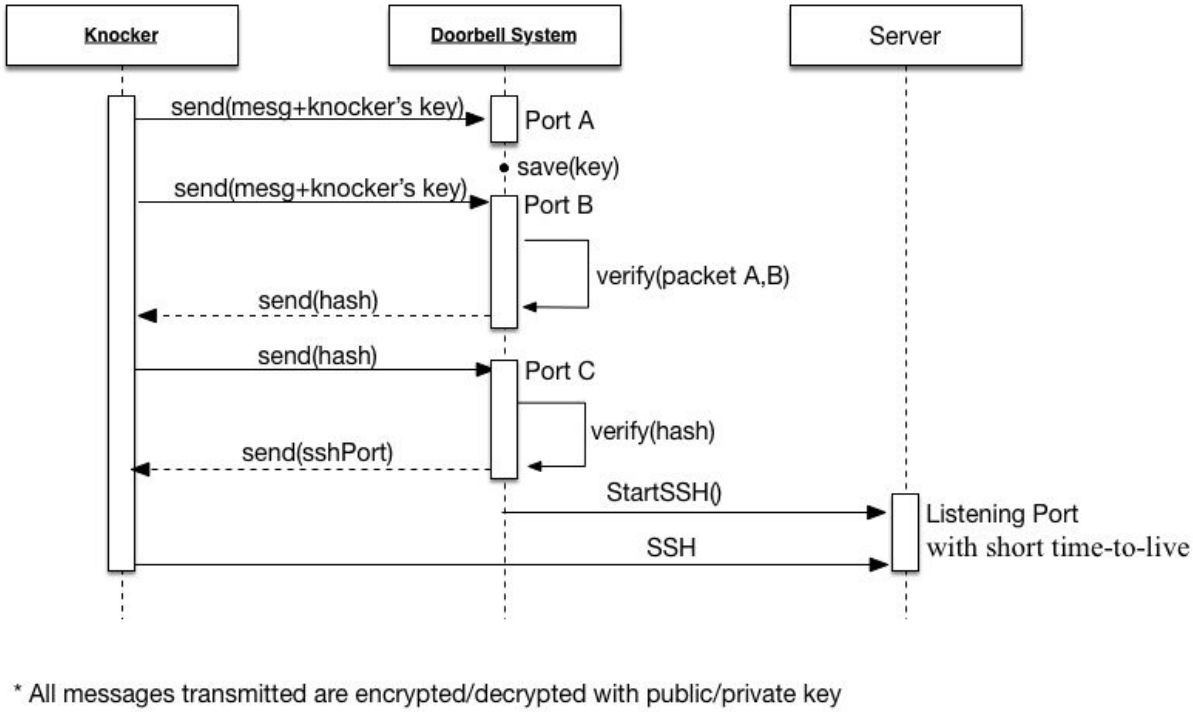


Figure 3.1.2. System Sequence Diagram

The **steps** of the protocol are:

1. The doorbell knocker first sends a UDP packet (Message#1) to port A containing a properly formed message encrypted with the doorbell server's public key. The encrypted message contains the doorbell knocker's own public key. The doorbell server **never** responds to packets received on port A (thus remaining silent to port-scanning), but the doorbell knocker's public key is saved on the server for a window of time.

2. Next, the doorbell knocker sends the **same** encrypted Message#1 in a UDP packet to port B.

3. Now, having seen the same Message#1 on both ports A and B (in that order) within a window of time, the doorbell server will respond via port B with a UDP packet containing a Message#2 encrypted with the knocker's public key previously sent to ports A and B as Message#1. Message#2 contains a one-time hash required for the next step of the

protocol. Note that the response from port B is only sent if the proper packets are observed at port A and then B, within a window of time. Thus, any attempt to scan "just" port A or port B will fail and will elicit no responding message from the doorbell server, because A and B are not sequential.

4. When the doorbell knocker receives Message#2 from port B, it decrypts the message (using the private key corresponding to the public key previously sent to ports A and B), re-encrypts the one-time hash from the doorbell server with the public key from the server (previously known and used for Step 1), and sends that Message#3 to port C.

5. Upon receiving the UDP packet on port C, the doorbell server decrypts Message#3, confirms the one-time hash is correct, and then the doorbell server responds with an encrypted UDP packet to the doorbell knocker, with a Message 4 as to which port to use for the SSH connection. Note that no response comes from port C unless it first receives a properly formed Message#3, which is based on a proper Message#2 received from port B. Thus, port C is silent to a typical port scan as well.

6. The doorbell knocker can decrypt Message#4 and initiate a normal SSH connection using the specified listening port (which could be port 22, but is likely to be a random port). The SSH listening port has a very short time-to-live. The actual SSH authentication itself remains unchanged.

There are time windows and limited time-to-live values associated with all the above steps, therefore any lost UDP packets or reordered messages will cause a protocol failure, problems can be detected after a sufficiently long time-out, and resources can be reclaimed. After the timeout, the protocol can be retried. A more efficient recovery strategy might be possible, but a simple timeout-and-retry strategy should work. After all, a doorbell is not usually a performance-oriented mechanism.

3.2 Software Design

3.2.1 UML

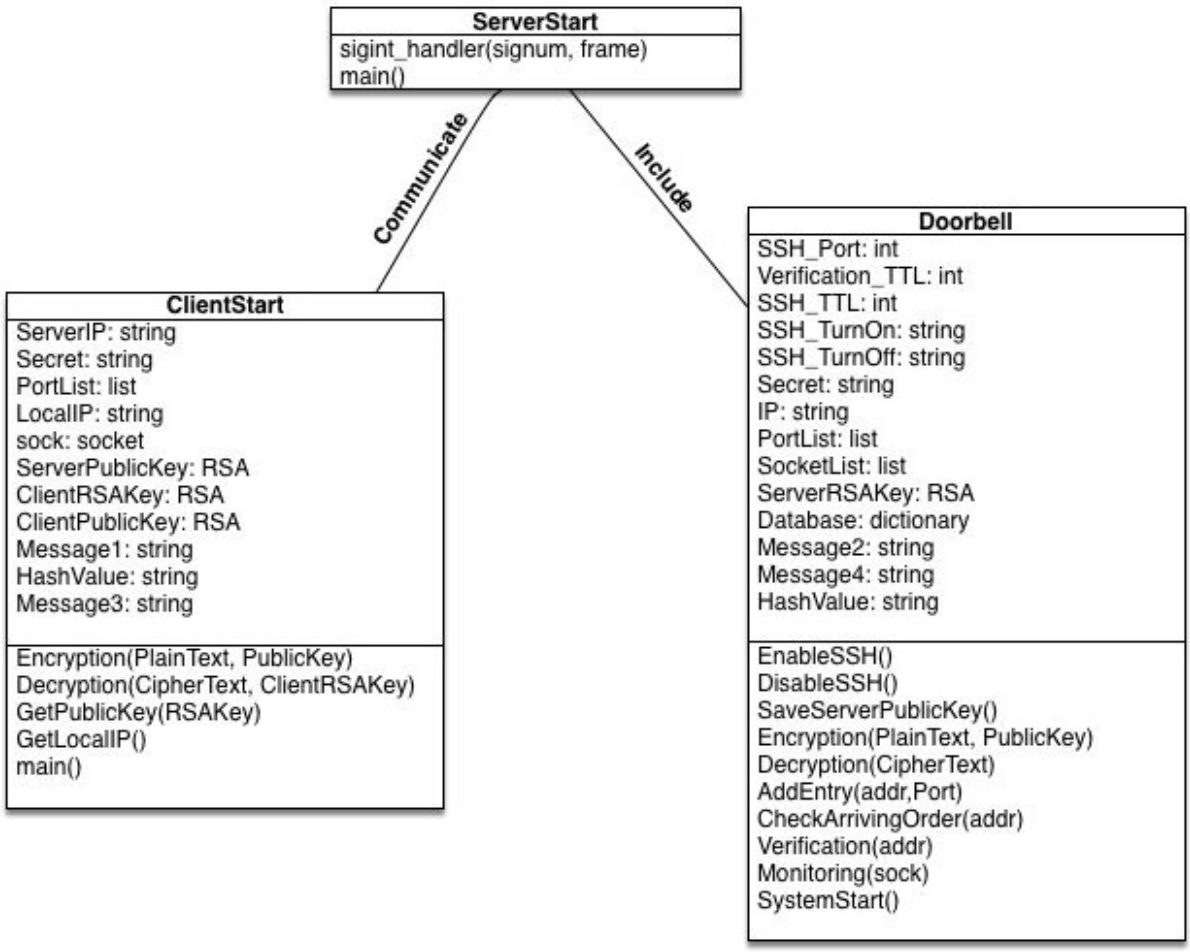


Figure 3.2.1. UML Diagram

3.3.2 API Design

This section illustrates the detail of functions according to the UML diagram described above. The project consists of two main modules: Doorbell module and ClientStart module. The Doorbell module is used in doorbell servers while the ClientStart module is provided for doorbell knockers.

Module: Doorbell	
Functions	
	Encryption(PlainText, PublicKey): Read a plaintext, encrypt the text by the public key and return encrypted bytes **Parameters:** ● **PlainText: String or Bytes.** The length of the plaintext should less than the length of the public key. ● **PublicKey: Crypto.PublicKey.RSA.** The RSA key used to encrypt the plaintext. **Returns:** ● **Bytes** **Raises:** ● **ValueError** - When the length of the plaintext is larger than the length of the public key, the function will raise a ValueError Exception.
	Decryption(CipherText): Read encrypted bytes, decrypt the ciphertext by the private key corresponding to the public key used to encrypt the ciphertext and return a plaintext. **Parameters:** ● **CipherText: Bytes.** The ciphertext is the message encrypted by a client using a server public key. **Returns:** ● **String** **Raises:** ● **ValueError** - When the ciphertext is not encrypted by the corresponding public key or a unencrypted data, the function will raise a ValueError Exception.
	EnableSSH(): Enable the SSH service **Parameters:** None **Returns:** None **Raises:** None
	DisableSSH(): Disable the SSH service **Parameters:** None **Returns:** None **Raises:** None
	SaveServerPublicKey(): Save the server public key to the local disk= /Server/Keys/PublicKey.pem **Parameters:** None

	Returns: None **Raises:** None
	AddEntry(addr,port): Add the packet information such as arriving port & time to the database. **Parameters:** ● **addr: String.** IP address of the client. ● **port: int.** The receiving port of the packet. **Returns:** None **Raises:** None
	CheckArrivingOrder(addr): Check the arriving order of Message#1s, if out-of-order, return false **Parameters:** ● **addr: String.** IP address of the client. **Returns:** ● **False:** If any order of Message#1s is wrong. ● **True:** The arriving order of Message#1s is correct. **Raises:** None
	Verification(addr): The verification function initiated by receiving a Message#1 at port A. In this function, the system uses a one-time hash to identify the client once more. If the client is identified, the doorbell system will enable the SSH service. **Parameters:** ● **Addr: String.** IP address of the client. **Returns:** None **Raises:** None
	Monitoring(sock): The Monitoring function containing a UDP socket is used to monitor the received packet at UDP ports of the doorbell system. If the packet is invalid or the secret in the packet is not correct, the function will discard the packet and keep silent to the client. **Parameters:** ● **sock: socket.** This is the UDP socket used in doorbell system. **Returns:** None **Raises:** None
	SystemStart(): Start the doorbell system. **Parameters:** None **Returns:** None **Raises:** None

Module: **ClientStart**	
Functions	
	Encryption(PlainText, PublicKey): Read a plaintext, encrypt the text by the public key and return encrypted bytes **Parameters:** ● **PlainText: String or Bytes.** The length of the plaintext should less than the length of the public key. ● **PublicKey: Crypto.PublicKey.RSA.** The RSA key used to encrypt the plaintext. **Returns:** ● **Bytes** **Raises:** ● **ValueError** - When the length of the plaintext is larger than the length of the public key, the function will raise a ValueError Exception.
	Decryption(CipherText): Read encrypted bytes, decrypt the ciphertext by the private key corresponding to the public key used to encrypt the ciphertext and return a plaintext. **Parameters:** ● **CipherText: Bytes.** The ciphertext is the message encrypted by a client using a server public key. **Returns:** ● **String** **Raises:** ● **ValueError** - When the ciphertext is not encrypted by the corresponding public key or a unencrypted data, the function will raise a ValueError Exception.
	GetPublicKey(RSAkey): Return a public key of the RSA key **Parameters:** None **Returns:** ● **Crypto.PublicKey** **Raises:** None
	GetLocalIP(): Return the local IP address. **Parameters:** None **Returns:** ● **String** **Raises:** None

	main(): The main function of the ClientStart module. **Parameters:** None **Returns:** None **Raises:** None

4 System Implementation

The doorbell system can be decoupled to three main components: Initialization, Monitoring and Verification.

4.1.1 Initialization

There are two tasks in Initialization. The first task is to check whether the required modules are installed. For instance, both doorbell servers and client tools need to encrypt and decrypt messages using RSA keys and therefore, a third-party RSA module is required. During Initialization, if the system detects that the required modules are missing, the doorbell system would prompt users to install the modules via pip. The second task of Initialization is that the system reads a secret and a list of numbers from the command line. The secret is used in Message#1 to knock the doorbell server and the list of numbers is the UDP ports used in the doorbell system. The function would also set up some important variables such as verification TTL, SSH port and SSH TTL during Initialization.

4.1.2 Monitoring

The next main component is Monitoring. The doorbell system creates several UDP sockets after Initialization. The doorbell system utilizes multi-threading to create several threads that each contains a monitoring function with a UDP socket. In this way, all the UDP sockets are running concurrently in its monitoring function. The architecture of the monitoring function is shown in the Figure 4.1.2:

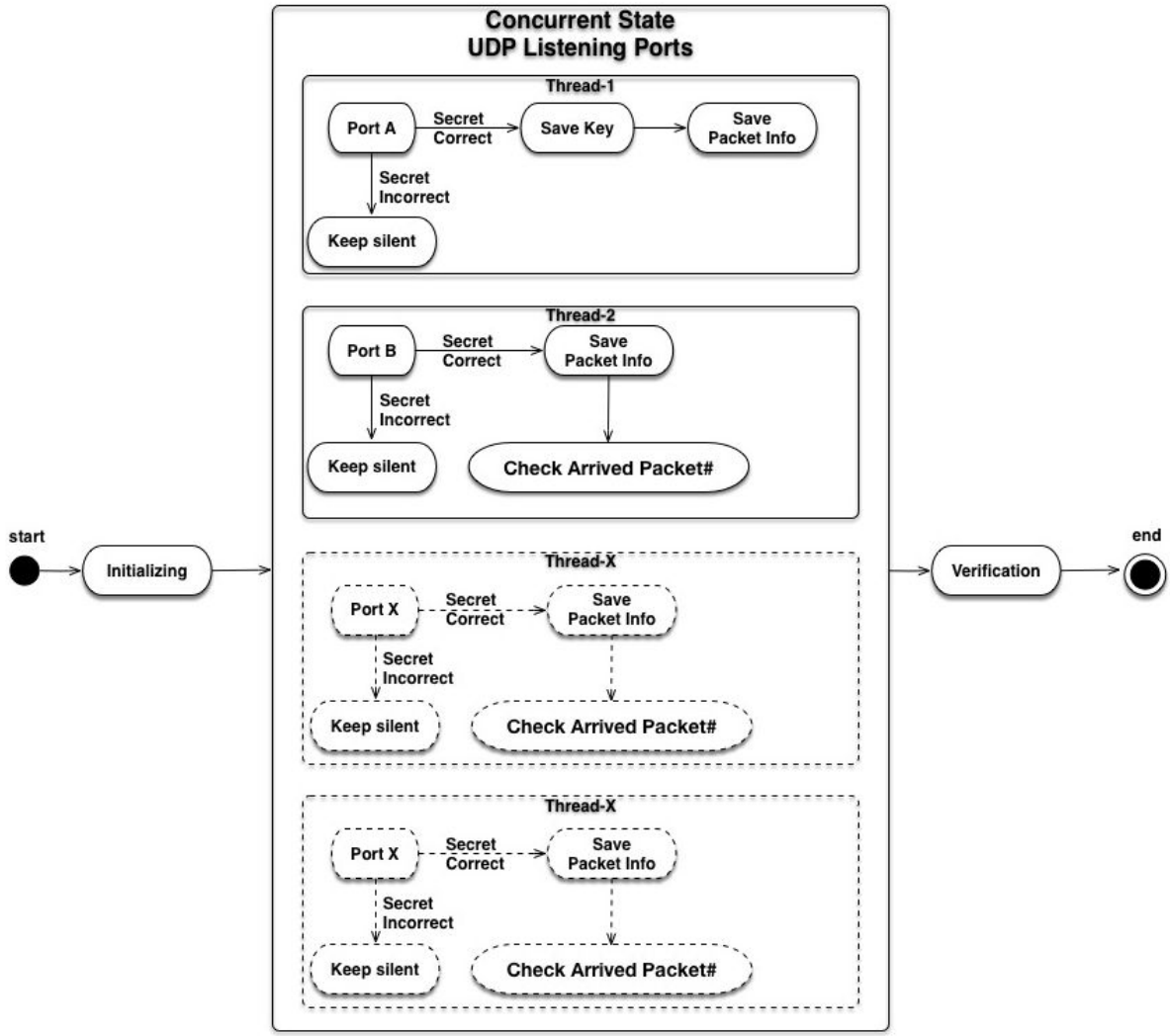


Figure.4.1.2

Assuming that the doorbell system has two UDP ports: ports A, B. These UDP ports are used to listen to clients' requests and receive Message#1s. Once a packet has arrived at port A, the doorbell system will decrypt the packet with the corresponding private key.

There are three cases in this process:

1. If the packet is sent from somewhere other than client tools, which means the packet may be a malicious attempt such as a port scanning. In this situation, the data should not be encrypted by the doorbell server's public key, and a ValueError exception will be raised when the system is trying to decrypt the data. The doorbell system would simply ignore the packet and keep silent.

2. If the received data is decrypted properly, however, the secret may be incorrect. The action is the same as the situation above, and the doorbell system would simply ignore the packet and keep silent.

3. If the message is parsing correctly and the secret is matching, the system will add a packet information to the server database. The packet information contains the packet's arriving time and receiving port, and the doorbell system can use this information to check the arriving order of each packet. When a packet has reached a port, there may be two cases:

    a. If a packet has arrived at port A, the doorbell system will save the client's public key and start the verification function.

    b. If a packet has arrived at another port, the system will check the number of received Message#1s and unlock Event (Event will be illustrated in next section) in the verification function once it receives the required number of Message#1s.

4.1.3 Verification

The last component is Verification. When a Message#1 has arrived at port A, the doorbell system would create a new thread to perform a verification function. In order to prevent transmission error, such as message out-of-order due to transmission delay, the verification function starts with an Event, which will block the function until the doorbell system has received the required number of Message#1s. Meanwhile, Event sets a timeout value. If the timer is expired, the verification function would be automatically quit. Every time when a packet arrives at the doorbell server, the system will check the number of received Message#1s. If the system has received the required number of Message#1s, it will unlock the Event and then check the arriving order of received Message#1s.

If the order is correct, a one-time hash would be created as Message#2 and sent to the client. Then the system would wait for Message#3 from the client. There is also a timer here. When the timer is expired, the verification function would

be quit. Having received Message#3, the system would check if the hash received is correct. If yes, the doorbell system would enable the SSH service and send the SSH port and time-to-live to the client as Message#4.

5 Result & Evaluation

5.1 Environment

Both doorbell systems and client tools have the similar system requirements, as listed below:

Library:

- Python3.5
- pip
- pycrypto - a RSA encryption/decryption module in PyPI
- SSH server installed on Linux

Operating System:

- The doorbell system should be running on Linux
- The client tool could be run on Linux, Mac OS and Windows

5.2 System Testing

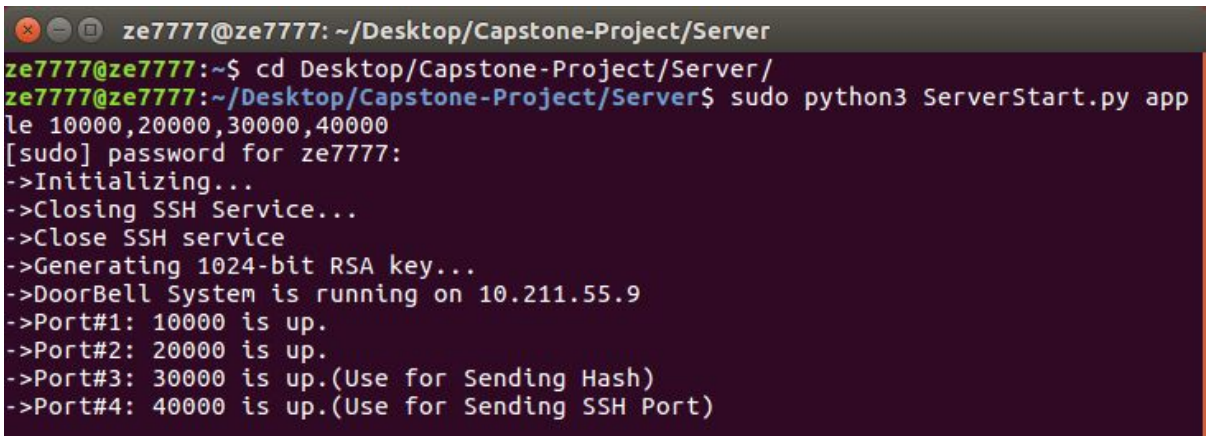
The system testing is to test the compatibility of various situations during the runtime of the doorbell system. Testing cases and goals are shown in the table below.

Test Number	**Test Goal**	**Test Result**
Case #1: Functionality Testing	Test overall functions of the doorbell system and client tool.	Work properly.
Case #2: Hundreds of Listening ports	Test the functionality of the doorbell system with hundreds of UDP listening ports.	Work properly. **Note:** The user should accordingly increase the timer's waiting time as

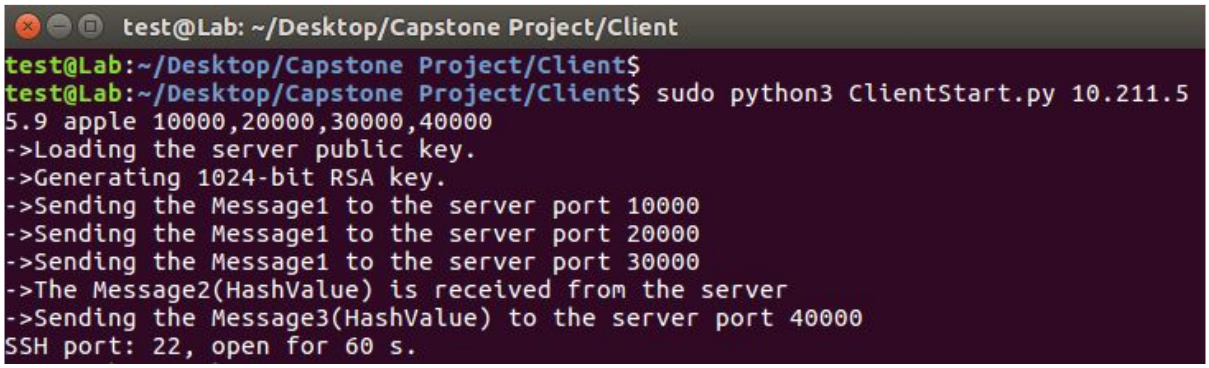
		port number increases.
Case #3: Incorrect Secret	Test the situation when the doorbell system receives a Message#1 with an incorrect secret. The system should ignore the packet and keep silent.	Work properly.
Case #4: Message out-of-order	Test the situation when the doorbell system receives all Message#1s but the arriving order of Message#1s is incorrect. The system should detect this condition and terminate the verification function for the client.	Work properly.
Case #5: Verification Time-out	Test the situation when the doorbell system doesn't receive the required number of Message#1s. After a certain time, the timer in the verification function will be expired and the verification is terminated.	Work properly.
Case #6: Invalid Message	Test the situation when the doorbell system receive an invalid message such as unencrypted data or encrypted message with incorrect key. The system should detect this condition and simply ignore the packet and keep silent.	Work properly.

Case #1: Normal Operation

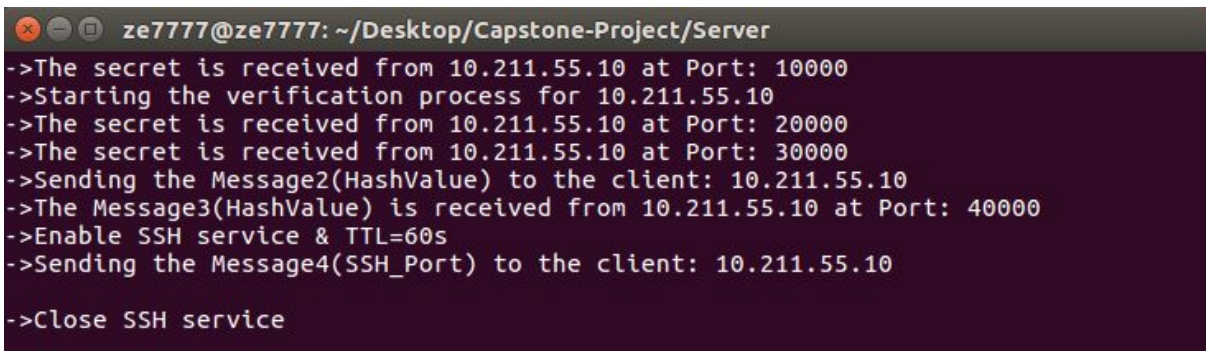
1. Server initialized with four ports: 10000, 20000, 30000, 40000:



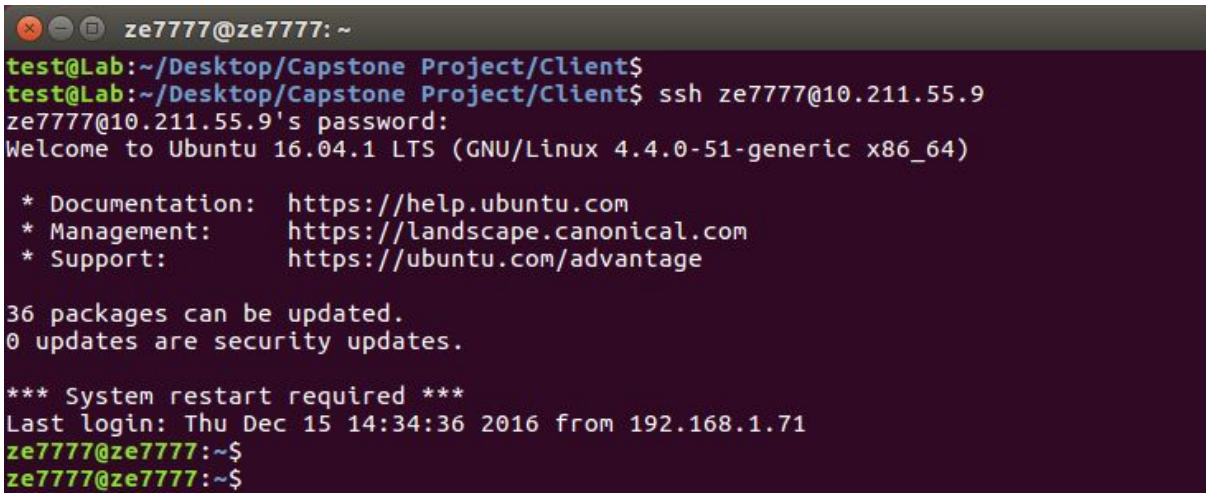
2. Client tries to knock the doorbell system:



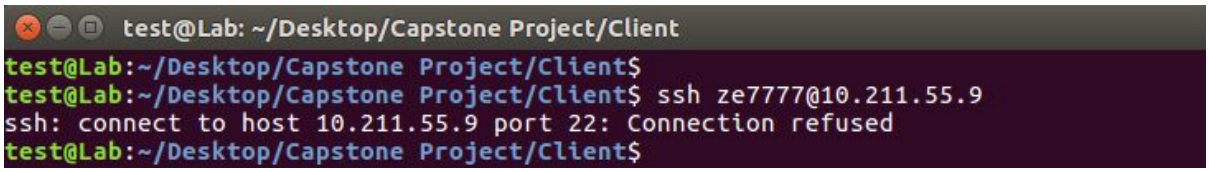
3. Server receives Message#1s from the client:



4. Client connects to the server via SSH:

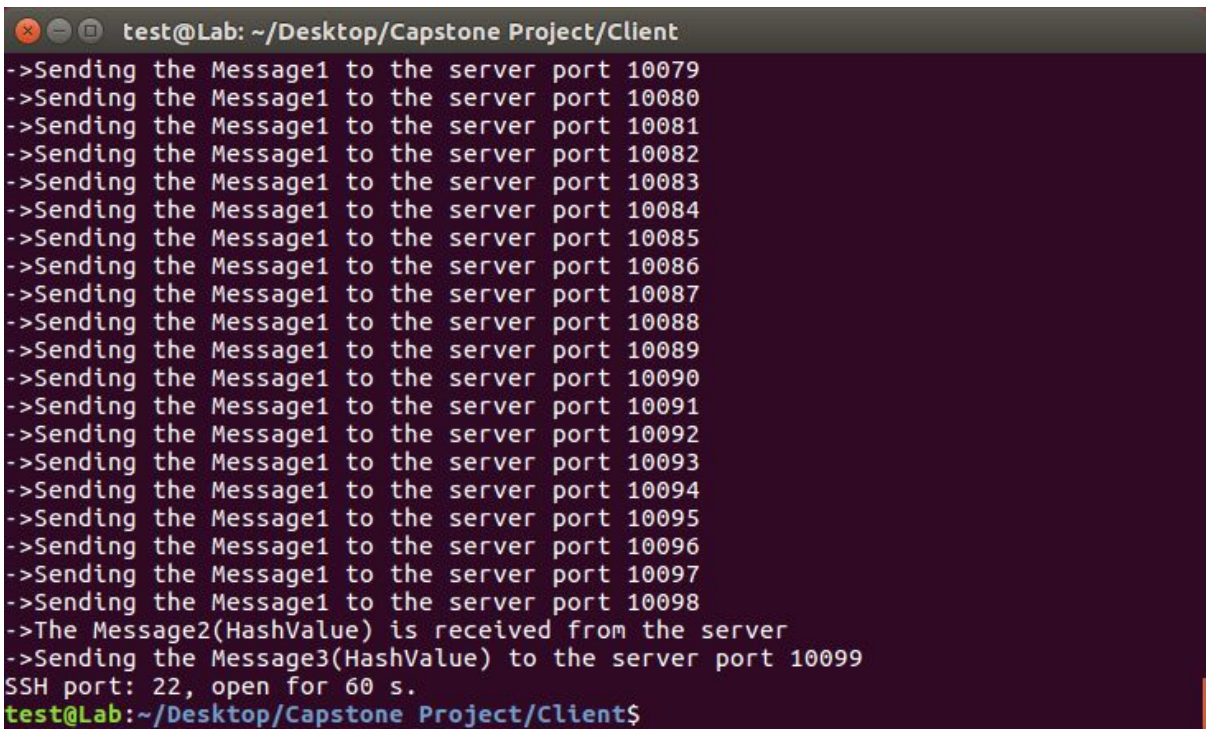


5. After 60s, the doorbell system automatically disables the SSH service

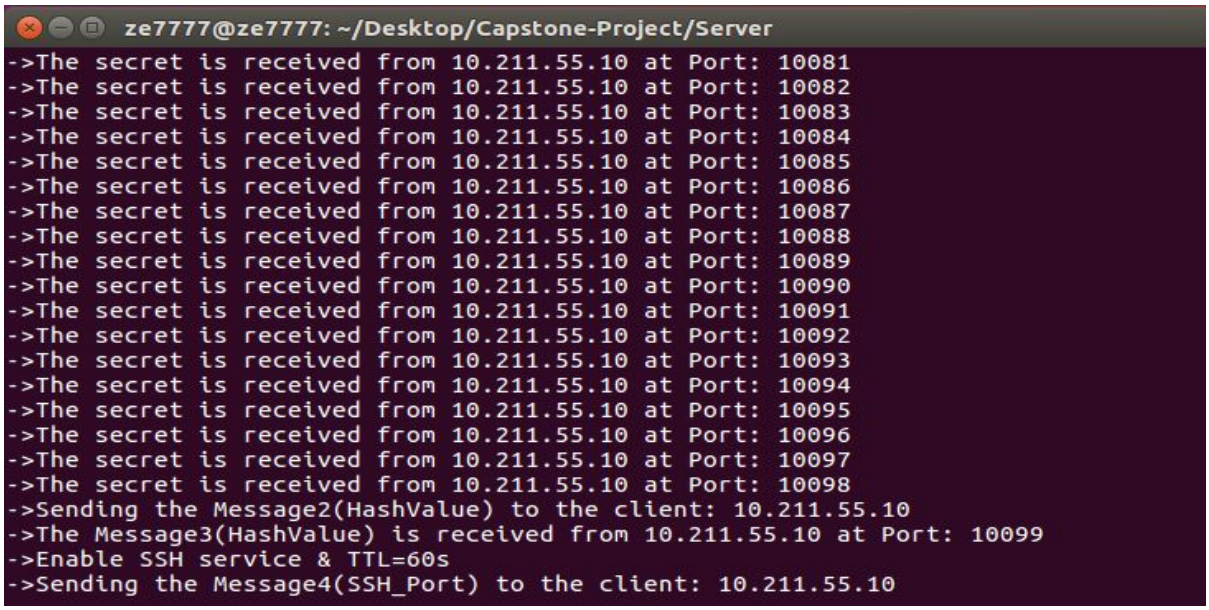


Case #2: Hundreds of Listening ports

1. Client tries to knock the doorbell system which is running with hundreds of ports:

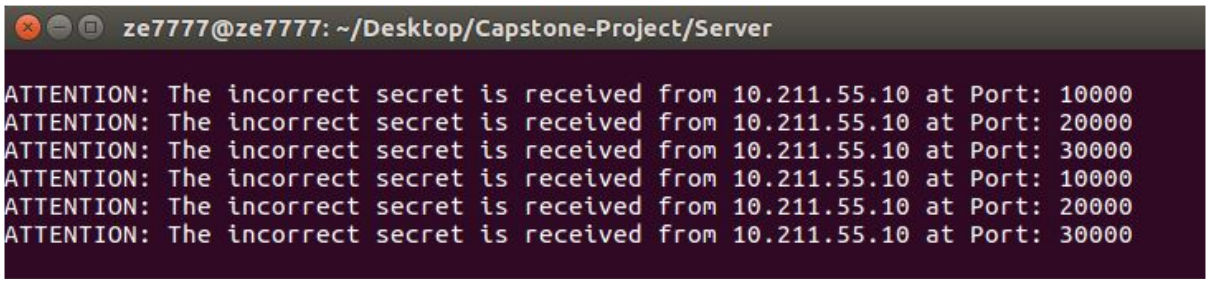


2. Server receives Message#1s from the client:

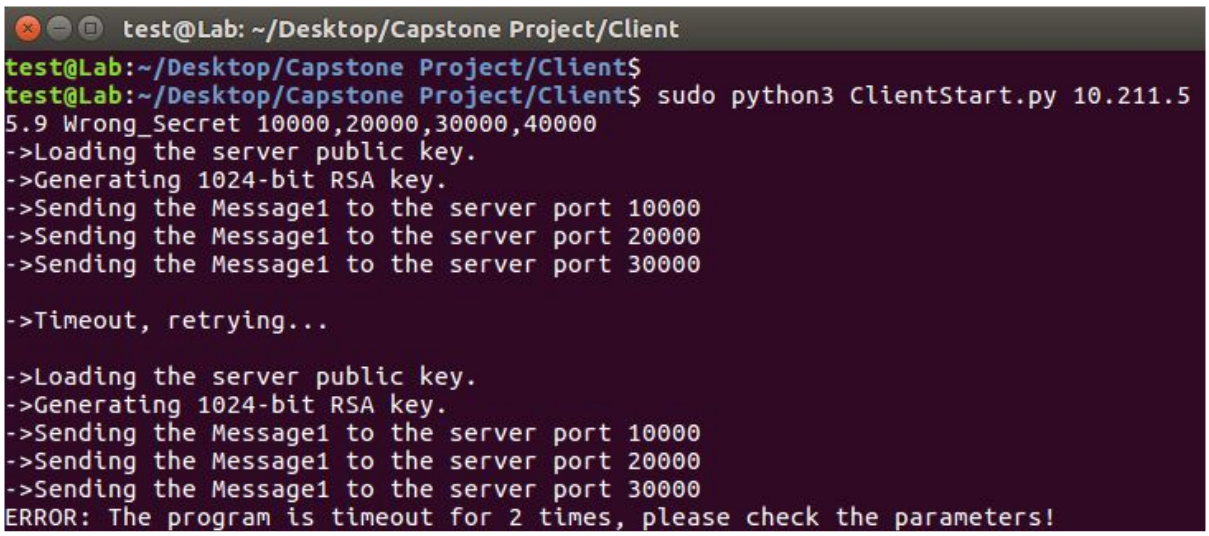


Case #3: Incorrect Secret

1. Server receives Message#1s with incorrect secret



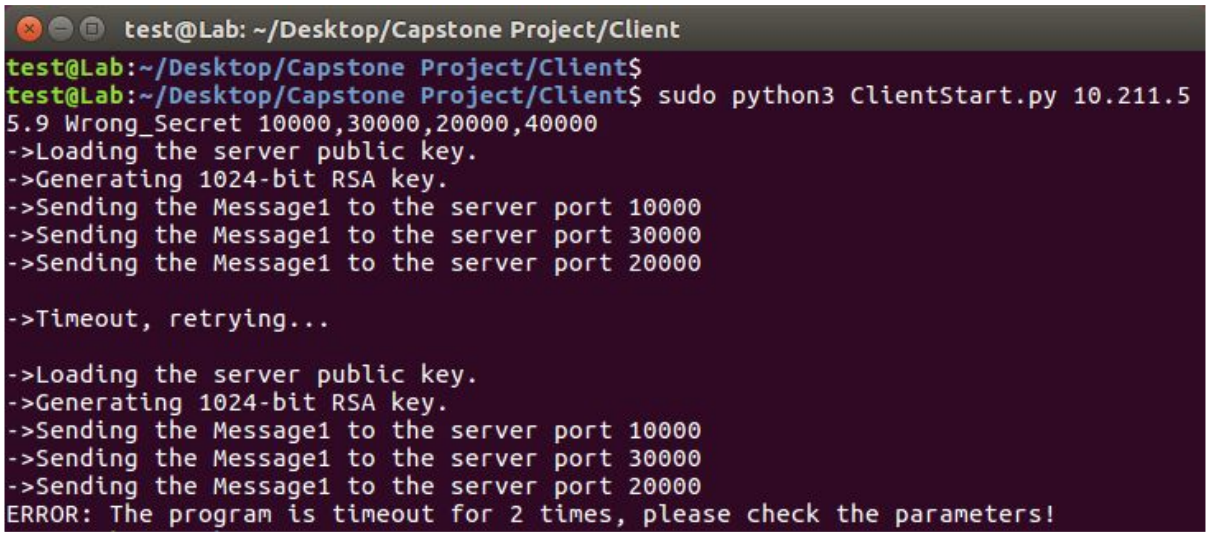
2. Client cannot receive response from the server; it will retry while the timer is expired:



Case #3: Message out-of-order

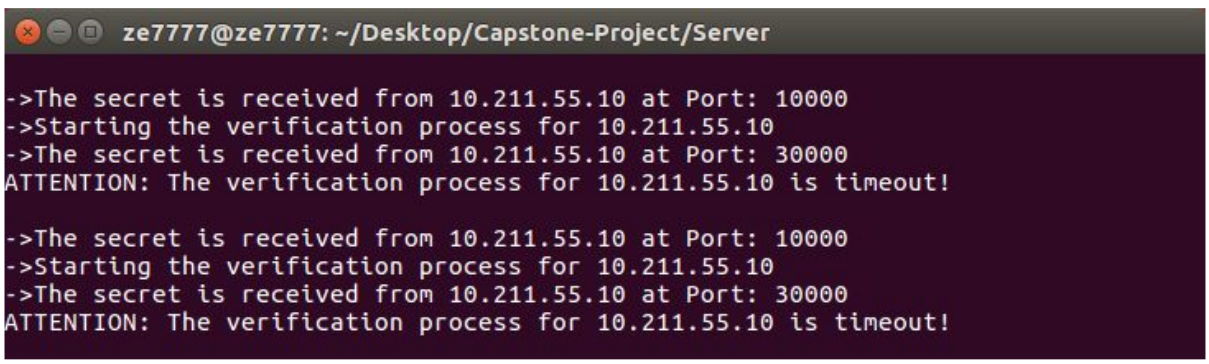
1. Server receives Message#1s but they are out-of-order

2. Client cannot receive response from the server; it will retry while the timer is expired:

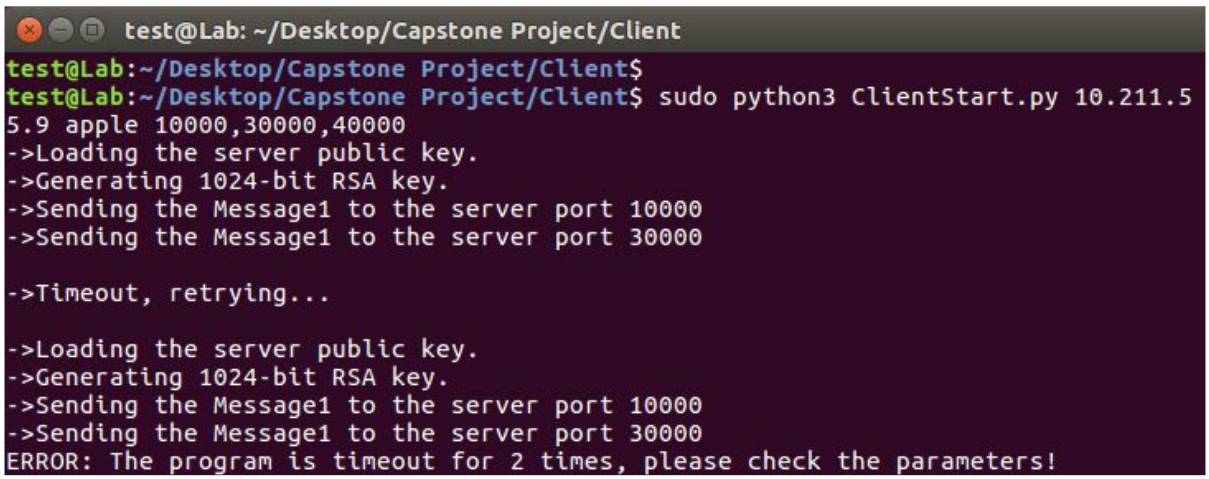


Case #4: Verification Time-out

1. Server doesn't receive required number of Message#1s; the verification timeout occurs after a certain time

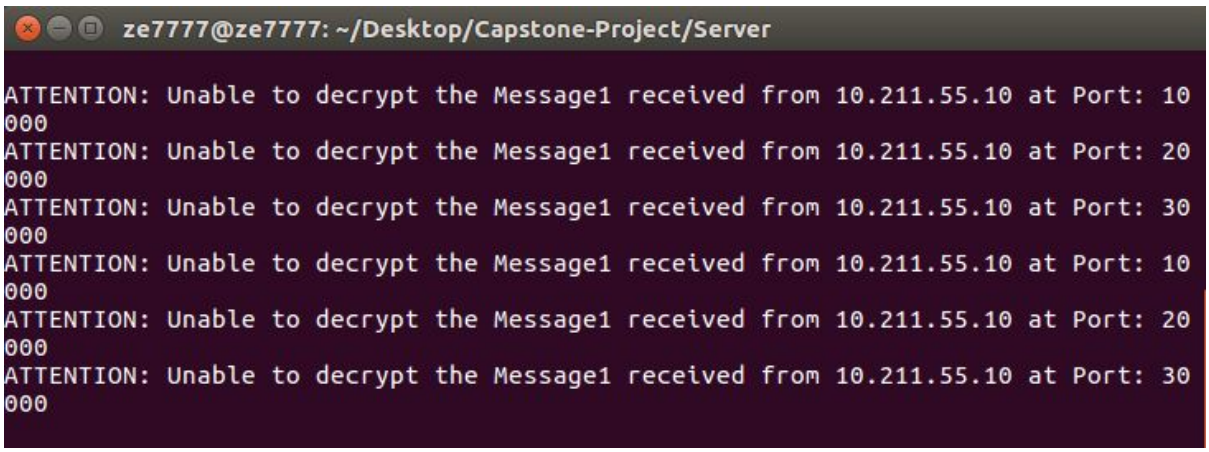


2. Client cannot receive response from the server; it will retry while the timer is expired:

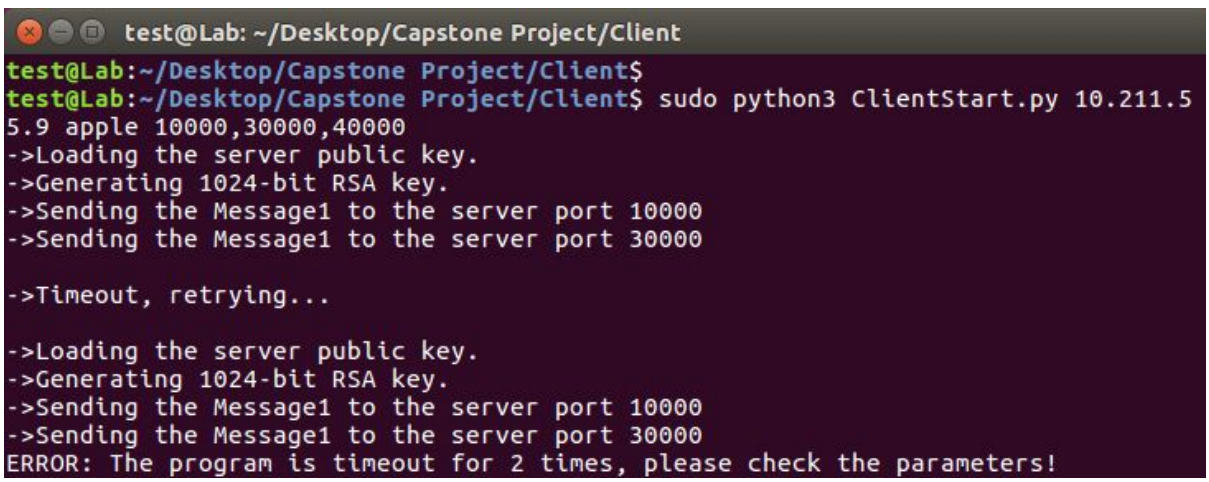


Case #5: Invalid Message

1. Server receives invalid Message#1s:



2. Client cannot receive response from the server; it will retry while the timer is expired:



6 Discussion & Conclusion

In this project, we have designed the doorbell protocol, implemented the doorbell system, and tested the program with various situations. The SSH listening port under the doorbell system's protection should be silent to port scanning attacks as the doorbell server does not respond in any way if any of the secret in Message#1, the arriving order of Message#1s, or the RSA key is compromised.

A replay attack based on sniffing Message#1 is ineffective because only a proper doorbell knocker can decrypt Message#2 to create a proper Message#3. (Message#2 is encrypted by the doorbell server using knocker's public key, and thus only the knocker can decrypt Message#2) Admittedly, an attacker can sniff Message#4 on the network and anticipate that some SSH port will become open, and start a port scan despite not being able to decrypt Message#4 to know the exact port. However, standard port scanning detection algorithms can be used to lock down the system, and the attacker would still need to break the baseline SSH protocol itself to get access to the system; the doorbell system only activates the baseline SSH server.

References

1. Secure Shell. (n.d.). In Wikipedia. Retrieved September 5, 2016, from https://en.wikipedia.org/wiki/Secure_Shell

2. UDP. (n.d.). In Wikipedia. Retrieved September 5, 2016, from https://en.wikipedia.org/wiki/User_Datagram_Protocol

3. TCP. (n.d.). In Wikipedia. Retrieved September 5, 2016, from https://en.wikipedia.org/wiki/Transmission_Control_Protocol

4. DDoS. (n.d.). In Wikipedia. Retrieved September 5, 2016, from https://en.wikipedia.org/wiki/Denial-of-service_attack

5.Man-in-the-middle_attack. (n.d.). In Wikipedia. Retrieved September 5, 2016, from https://en.wikipedia.org/wiki/Man-in-the-middle_attack