# Readme

Zetao Yu, Chuanqi Xiong

## Introduction

This is a documentation for Rutgers CS352 Spring 2019 Project. The project demonstrates a simple version of Go-Back-N protocol similar to TCP. As implementing in Python, we built it based on UDP transport layer. Moreover, we added the basic encryption/decryption option for users to encrypt their messages. The encryption method is based on NACL (sodium) library.

Thank **Anarav Patel** and **Saurin Shah** for providing their code as reference!

## Usage and Arguments

Users should run server2.py and client2.py on different terminals (either on different machines or not)

Arguments:

-f <the file name to send or to save>
-d <the destination host to send to>
-u <the UDP port to use for receiving>
-v <the UDP port to use for sending> (optional)
-k <keyfile used for encryption/decryption>

Example Usage:

python server2.py -f recv.txt -u 8888 -v 9999 -k keychain.txt
python client2.py -d localhost -f send.txt -u 9999 -v 8888 -k keychain.txt

## Keychain file format

# this is an example keychain file for the CS 352 socket assignment #
lines with a # in the first word are comments
# the keys are labeled by if they are public, private, and the host and
# destination ports for each key.
# a '*' is a wildcard that can be used for all hosts and ports
# private keys are used to decrypt incomming packets and public keys are
# used to encrypt outbound packets
private * * 53fbcbb3b76e173f8241408b1f3dd8f9bf0d2a9f84d3db8fee2f38d0f2429729
public localhost 8888 78c7227cf5c637fbc2066070b6fa2662b5c94bbac6fcb1ba5d77ecd61f718574
public localhost 9999 78c7227cf5c637fbc2066070b6fa2662b5c94bbac6fcb1ba5d77ecd61f718574

## Detailed Description

In order to imitate TCP protocol in each step of socket communication, our functions in project map the existing Python socket built-in methods. For example, connect() and accept() function would first complete a “three way handshake” communication, then the socket connection would be established. In recv(), the server would only accept packets with correct sequence number, and send back ACKs. If any packets lost, the server would block and wait until the client retransmit the packet after timeout. In send(), we use the technique of multi-threading to allow the client sending packets and receiving ACKs simultaneously. When one thread realizes there is a timeout, it will notify the other thread immediately by modifying a shared global variable. And the sending thread will resend from the specific packets. In close() function, we implement “two

double handshakes" algorithm which ensures the termination can be done from both sides. Notice the termination would fail if the transmission is still going on.

Encryption is an option for users to encode their messages. Users should first determine whether they need encryption in the state of connection. On both sides of server and client, they need to search in keychain file to find a private/public key that has a matched pair of address and port. Private key is used to decrypt incoming packets, while public key is used to encrypt outgoing packets. If there is no external argument in connect() function or no matched keys, the program would send unencrypted messages by default. In sending the message, the send() function would first convert the whole message into binary/ascii format, encrypt the whole buffer, and then divide the buffers into pieces for transmission. In receiving the message, the recv() function would first re-assemble all the pieces together, and decrypt the whole buffer.

**sock352.py function list**

```
def init(UDPportTx, UDPportRx):
def readKeyChain(filename):
        return tuple(publicKeys, privateKeys)

class socket():
        def __init__(self):
        def connect(self, address, *args):
        def accept(self, *args):
                return (clientsocket, address)
        def close(self):
        def send(self, buffer):
                return len(buffer)
        def recv(self, nbytes):
                return bytesreceived
        def create_data_packets(self, buffer):
                return num_of_packets
        def recv_acks(self):
        def createPacket(self, flags=0x0, sequence_no=0x0, ack_no=0x0, payload_len=0x0)
                return packet
        def manage_recvd_data_packet(self, packet):
                return packet_data
```

**Dependencies**

-Python 2.x (https://www.python.org/downloads/release/python-272/) (using python3 may cause crash)

-PyNaCL (https://pypi.org/project/PyNaCl/) for encryption/decryption functionality.