SANTA CLARA UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Date: June 13, 2019

I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION BY

Robert Herriott
Peter Paulson
Nathaniel Kragas

ENTITLED

DeadDropBox: A Time-Locked Safe for Data

BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF

BACHELOR OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

Thesis Advisor

Department Chair

DeadDropBox: A Time-Locked Safe for Data

by

Robert Herriott
Peter Paulson
Nathaniel Kragas

Submitted in partial fulfillment of the requirements
for the degree of
Bachelor of Science in Computer Science and Engineering
School of Engineering
Santa Clara University

Santa Clara, California
June 13, 2019

DeadDropBox: A Time-Locked Safe for Data

Robert Herriott
Peter Paulson
Nathaniel Kragas

Department of Computer Science and Engineering
Santa Clara University
June 13, 2019

ABSTRACT

This project is a proof-of-concept for software which would allow users to securely store sensitive data in such a way that it is difficult or impossible for third parties to compromise the data, even when the user is compelled to assist them. It will operate by storing data across multiple devices in a unreadable form so that it is inaccessible until the data is reunified. The user may specify the circumstances under which different pieces of the data may be accessed, so that it is impossible to access under circumstances of duress.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Problem Statement

1.1.1 Motivation

Even over their own citizens, countries hold excessive powers to compel people to turn over their own private information as they are crossing borders. In the United States, for example, the government can (and does) detain *its own citizens* for up to eight hours if they refuse to decrypt their devices as they return from a trip abroad. This, relative to the extensive powers reserved by other nations, does not seem too horrible, but it is nonetheless a legal loophole which is regularly abused by the state in order to extract confidential information from U.S. citizens without warrant. Even if one ignores the ethical component of these laws, one has to accept that the law is out of touch with the technological realities of the situation: data freely flows across most national borders (and we're hard at work on the exceptions) every millisecond, unimpeded and securely encrypted by those who are transmitting it. To have such an arcane regulation as to confiscate all information which a person can access simply because it happens to be in the same place as them as they are returning to their own country would be hilarious if it was not so irritating.

1.1.2 Solution

The solution to one legal loophole seems to be another loophole. Our proposed system serves as a "time-locked safe" for data, rendering the user incapable of exposing their data – even if they want to – for a limited time (or, in extended applications, until a specified set of requirements is met).

This system is designed in such a way as to both ameliorate the privacy threat and to demonstrate the anachronistic nature of the legal framework. It does so by simply removing from the user's power the ability to expose their data. By making it impossible to comply, we hope that legislators will recognize the law as impossible to apply, and thereby spur them to more rational and realistic legislation on the issue.

1.2   Background

Every year, tens of thousands(1) of phones, laptops, and other portable electronic devices are seized, unlocked, and searched by the U.S. Customs and Border patrol. Many of these searches would otherwise be illegal under 4th Amendment protections; however, the U.S. Government has decided that these rights should be suspended for national security reasons. We, as well as the American Civil Liberties Union, disagree with this decision, and where the ACLU seeks to challenge the practice in court, we wish to provide a solution that makes this practice impossible.

Moving data securely through a border is not a new problem. However, the solutions that currently exist do not provide the kinds of legal, safe-to-use security that we provide. Hiding the existence of data is dangerous, extremely difficult, and could potentially cause harm to users of our software, which should not happen with our current design.

1.3   Objectives

- Store data securely.

- Follow standard data protection measures.

- Provide an simple user experience.

- Make the system clearly unbreakable within the given time period.

Chapter 2

Requirements

2.1 Functional Requirements

The system will:

- Allow users to erase and later recover data

- Not allow user access to erased data before the predetermined requirements are met

- Not store any values that could be identified as encrypted data and matching key

- Provide users with options to retrieve their data

- Protect user data by following standard security practices

2.2 Nonfunctional Requirements

The system will be:

- Intuitive to use for nontechnical users

- Modular

- Portable

Chapter 3

Use Cases
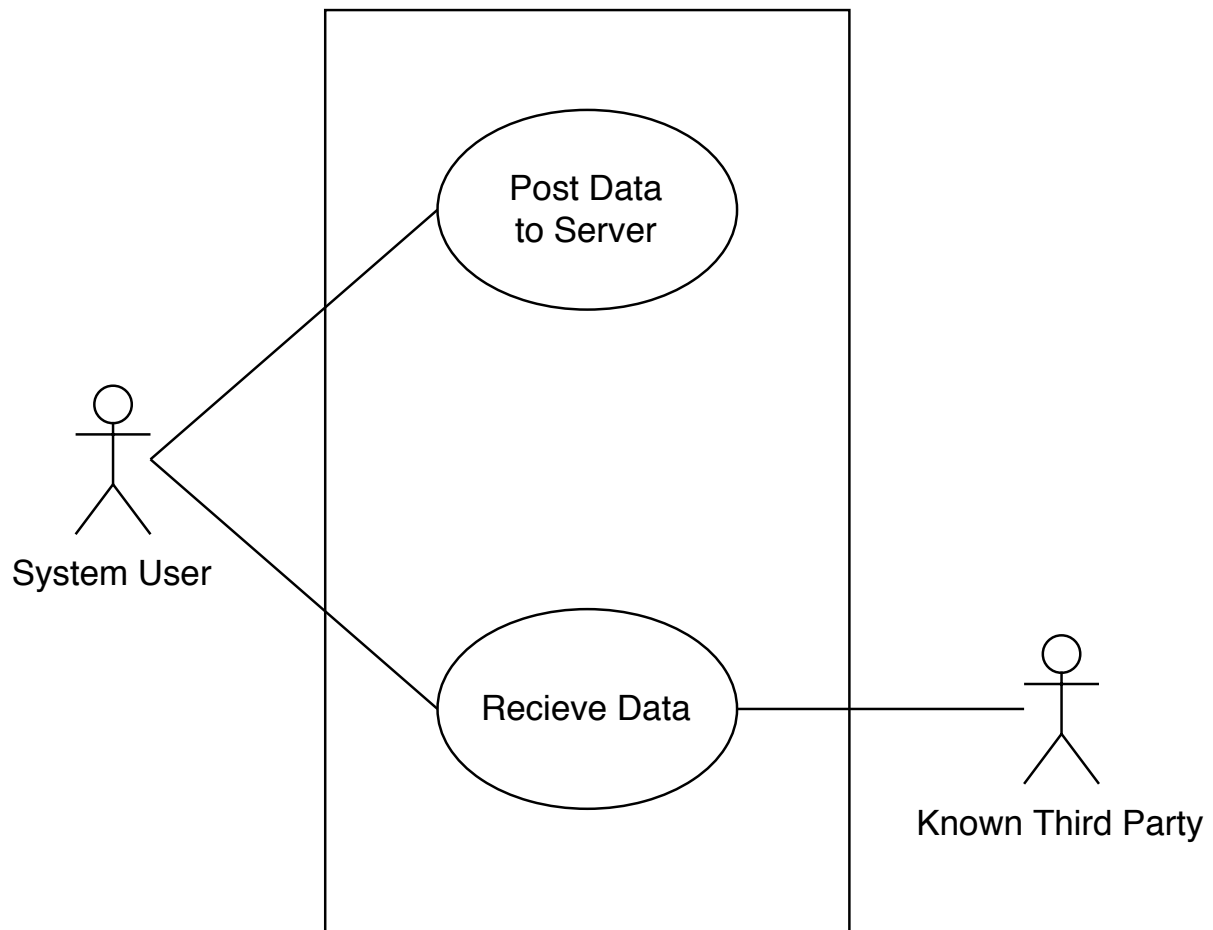
Figure 3.1: Use Case Diagram

Table 3.1: Use Cases

Actor	Use Case	Description
System Owner/User	Store Data	**Goal**: To secure their data beyond even their own grasp **Preconditions**: User possesses data and system client software on the same device; User has configured client to connect the system server which they will utilize; There is a viable connection from the client device to the server device. **Postconditions**: The user's client device no longer possesses the data; The client device now possesses a one-time-pad corresponding to one (or more) held by the server; Server has recorded User's preferences for the return of the data. **Exceptions**: If there is an interruption of the connection between client and server, the entire process is restarted; If the user indicate that they desire an atypical use where they are not the recipient of the one-time-pad, they shall not receive one.

Chapter 4

Activity Diagrams

4.1 DeadDropBox

The following diagrams show what prototypical interactions with the system look like:
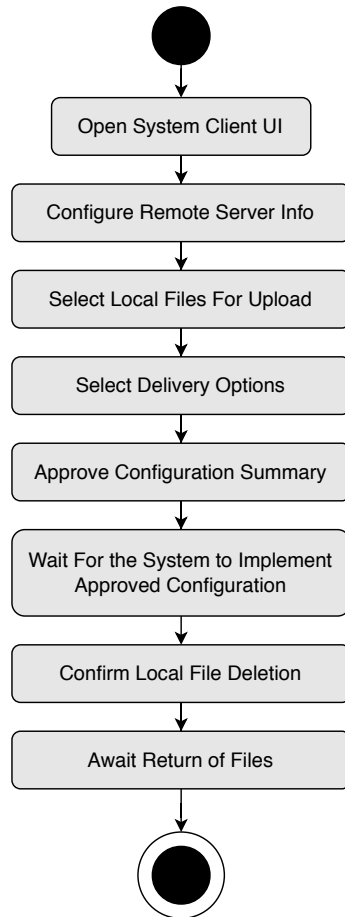
Figure 4.1: User Activity Diagram

Chapter 5

Technologies Used

5.1 Python

We chose to use Python because of its ability to run on a large number of platforms and its huge set of packages that we could access. Although we did not end up using a significant number of libraries, we were able to choose the libraries that we did use. In addition, Python is usually significantly more readable than equivalent C code, and we wanted to be able to keep the system as simple as possible.

5.1.1 PyCryptodome

PyCryptodome is a fork of the PyCrypto package that is currently deprecated. It provides a wealth of encryption functions. For this project, we use Crypto.RSA for key exchange, Crypto.AES for encryption, and Crypto.Random for key generation.

5.1.2 Sockets

We chose to use non-stream Python sockets because of the control that we have over them and our experience using almost identical sockets in C. In hindsight, our specific implementation - which doesn't not use stream sockets - is unnecessarily complex, but we were not aware of this ability during the majority of our implementation passes.

5.2   One-Time-Pad Encryption

Although we do use standard encryption techniques, the use of one-time-pad encryption was available because the use of the OTP is not to hide encrypted traffic, but rather to create two equal-length files, neither of which contain any information about the original without the other. Any other security is provided by AES encryption.

Chapter 6

Architectural Design

6.1   Dead Drop Box
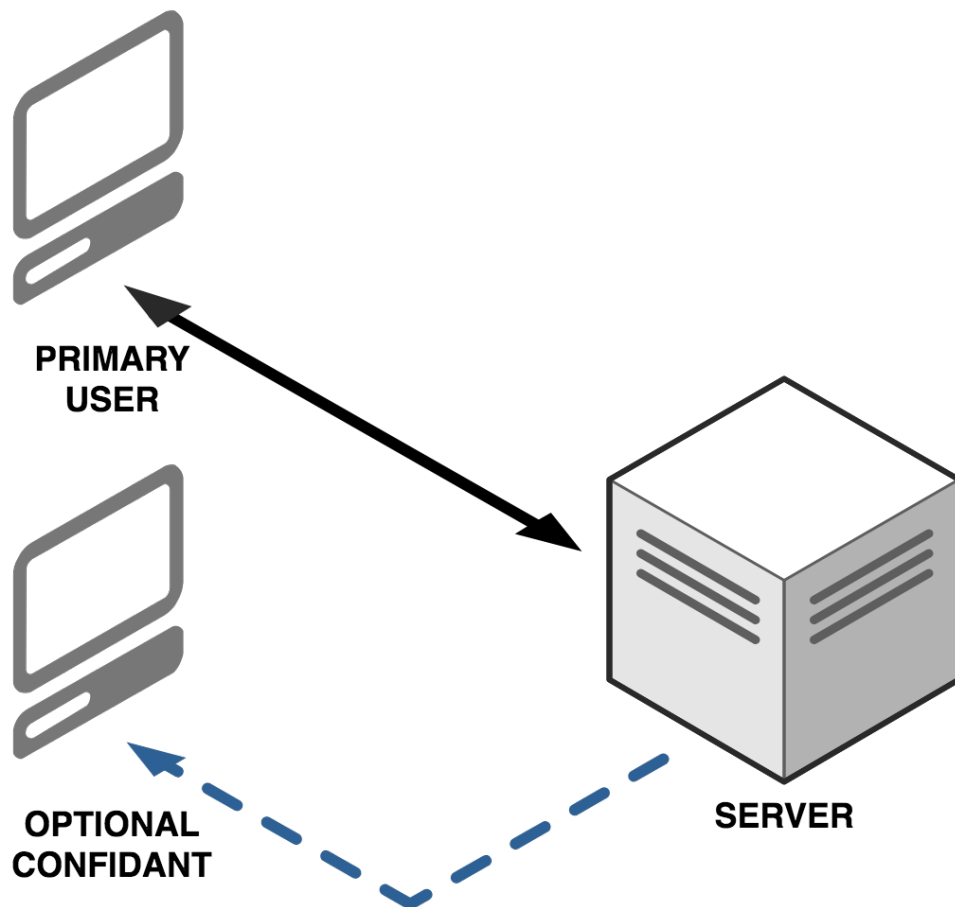
Figure 6.1: DeadDropBox Architecture Diagram

This system uses a client-server architecture, which we intend to serve as a template for future

iterations of this concept. Some processing is done on the client side, in order to hide the contents of the data stored from the server.

In the current implementation of the system, the client and server can be any platform capable of running Python 3. As we want the system to be usable without any specialized knowledge, and by anyone who would have data to store, we specifically made the system to be available to anyone that would need it.

Chapter 7

Design Rationale

7.1 Language Choice

The system is being written in Python for the sake of rapid development and cross-platform compatibility. While Python, as an interpreted language, is neither extremely fast nor secure, this project's status as a proof of concept and the intention of the designers to implement a maximum of variety in functionality made hardware-tailored low-level languages undesirable. Python provides vast libraries which implement many of the core cryptographic and networking tools which the system uses, and does so with little hassle. We also intend for our code to be very modular and expandable, and Python allows for that, not only within its own features but also through the many development frameworks which are built on it.

7.2 Hardware

As for hardware, we specifically built the system to be cross-platform, including declining the use of a library (pysendfile) because it would not have worked on Windows machines. The system should function on any system with Python 3 installed, which, to our knowledge, includes any modern machine.

Chapter 8

Risk Analysis

Risk	Likelihood	Severity	Impact	Prevention
Raspberry Pi Failure	0.2	4	0.8	Don't rely on having full set of Pis at all times
Power Supply Unit Failure	0.05	8	0.4	Have access to a backup power supply
Language Familiarity Issues	0.3	2	0.6	Familiarize with language before implementation begins
Data Loss	0.01	10	0.01	Use version control software such as Git
Obsolescence	0.001	5	0.005	Generalize the solution such as to be configurable to the legal situations of multiple countries

Chapter 9

Development Timeline

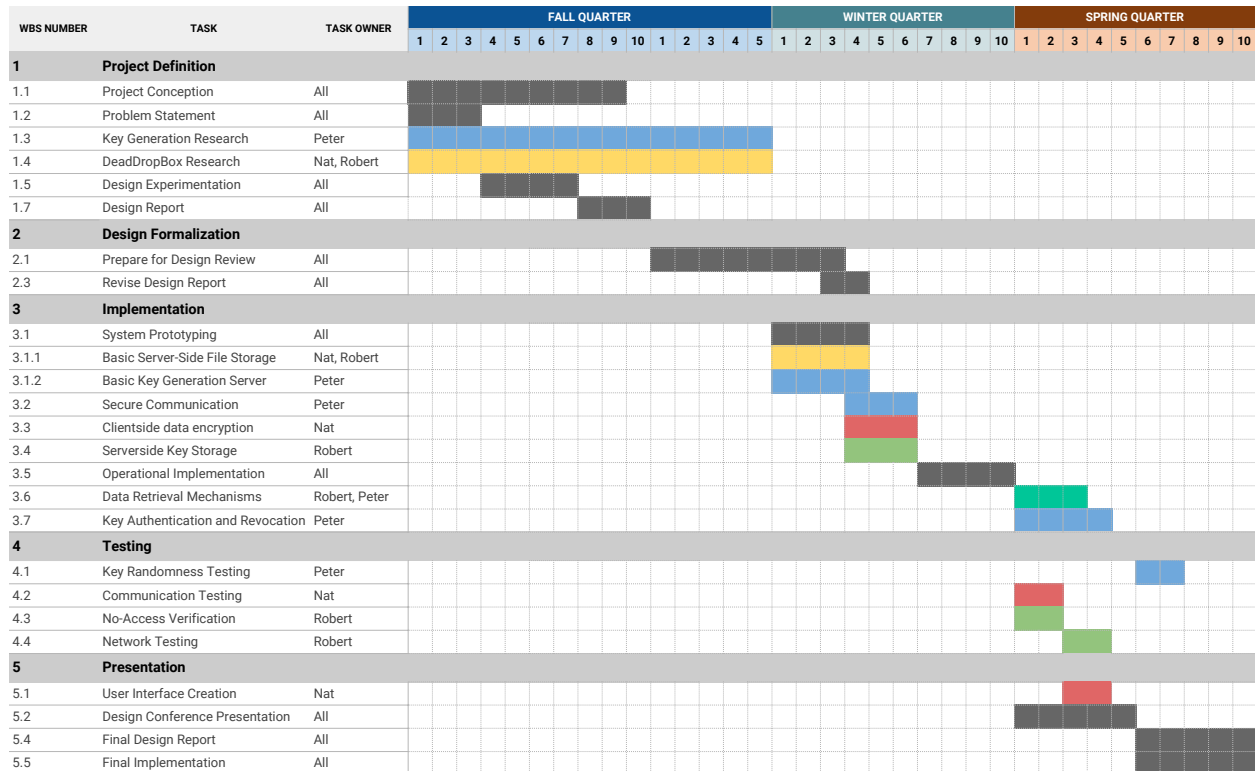| WBS NUMBER	TASK	TASK OWNER
1	**Project Definition**	
1.1	Project Conception	All
1.2	Problem Statement	All
1.3	Key Generation Research	Peter
1.4	DeadDropBox Research	Nat, Robert
1.5	Design Experimentation	All
1.7	Design Report	All
2	**Design Formalization**	
2.1	Prepare for Design Review	All
2.3	Revise Design Report	All
3	**Implementation**	
3.1	System Prototyping	All
3.1.1	Basic Server-Side File Storage	Nat, Robert
3.1.2	Basic Key Generation Server	Peter
3.2	Secure Communication	Peter
3.3	Clientside data encryption	Nat
3.4	Serverside Key Storage	Robert
3.5	Operational Implementation	All
3.6	Data Retrieval Mechanisms	Robert, Peter
3.7	Key Authentication and Revocation	Peter
4	**Testing**	
4.1	Key Randomness Testing	Peter
4.2	Communication Testing	Nat
4.3	No-Access Verification	Robert
4.4	Network Testing	Robert
5	**Presentation**	
5.1	User Interface Creation	Nat
5.2	Design Conference Presentation	All
5.4	Final Design Report	All
5.5	Final Implementation	All

Figure 9.1: Gantt Chart of Development Timeline

Chapter 10

Societal Issues

10.1 Ethical

Our system is designed to prevent an abuse of the current legal system - illegal search and seizure. We believe strongly that such abuses by the government are themselves extremely unethical, and as such, our attempt to prevent them from what we and others consider to be serious breaches of the Fourth Amendment - or at least open a discussion about how we should protect our borders in terms of electronic devices, instead of accepting a unilateral decision by a government who may not have the best interests of its citizens in mind.

10.2 Social

This system is designed solve to what we see as a social issue, so it has taken that social situation - legally grey areas of search and seizure created because of 21st century technology - and created a way that hopefully could force a serious discussion about it. By bringing this issue more towards the forefront and allowing people learn about it, we can prevent abuse of our civil liberties.

10.3 Political

Although this project does, in some ways, contradict the public (through disallowing government-created directives from being executed), we believe that these policies do not represent the views of an educated public.

10.4   Usability

We specifically designed our system such that anyone who had a reason to use our device (which requires a mobile device such as a laptop) would also, as a rule, have access to our software. Since the first device has to exist for the software to be needed, and the second can be almost anything, we feel that there should be no barriers to prevent people from using our system.

10.5   Lifelong learning

This project definitely taught us about the upsides of exploring new technologies. As part of this project, we both acquired new skills and noted that there are plenty of things that we have yet to learn, but are very important.

10.6   Compassion

Our system is designed to prevent people from being taken advantage of because they do know know information that they shouldn't be expected to know.

10.7   Manufacturability, Economic, Environmental Impact, Health and Safety, Sustainability, and Environmental Impact

As our product is not a physical product, and does not consume any physical resources to build, maintain, or modify, these issues do not apply.

Chapter 11

Conclusions

11.1   Summary

For this project, we created a simple, usable, and secure way for users to store data when crossing the American border. While our solution is not a viable end product that could be used on a wide scale, it does show that such system could absolutely be made. We hope that someone with access to greater resources takes our idea and spreads it, so that the issues that we discussed earlier can be brought to the attention of the public before the opportunity to debate them is lost.

11.2   What We Learned

One of the biggest things that we learned was that the initial design of a project needs to be flexible. Our initial design included a number of elements that ended up being significantly more complex for us to implement than we expected because of platform restrictions of the libraries we were planning on using. For example, we initially intended to use the pysendfile package before discovering that it did not support windows. Additionally, Peter did a significant amount of work to create a working RSA key generation algorithm before realizing that actually using that key in the standard Python encryption algorithms was not going to work without breaking the security of those packages.

11.3 Advantages and Disadvantages

Overall, we're quite happy with the core of our solution. It has all of the core features that we were aiming for, and was written such that extending the project should be relatively simple. In terms of problems that still remain, there are a few that we knew that we were not going to implement, such as hard deletion of the source file, a way to reconstruct an emailed data, persistence and recovery if the TCP connection is broken, multithreading, and self-implemented key generation.

11.4 Future Work

There is a significant amount of future work that could be done on this project. Implementing any of the missing features listed above would greatly increase the usefulness of the project. It would be very useful for anyone wanting to continue our work of they could get the project to a state where they could host a webserver that had the ability to act as a server while providing a user-friendly interface, as I think that the largest challenge with a project like this is getting people to use it, and while we have made it as simple as entering a few inputs, the less intimidating a system like this is, the better.

Bibliography

[1] Bhandari, Esha, and Nathan Freed. Can Border Agents Search Your Electronic Devices? It's Complicated. American Civil Liberties Union, American Civil Liberties Union, 14 Mar. 2017, www.aclu.org/blog/privacy-technology/privacy-borders-and-checkpoints/can-border-agents-search-your-electronic

[2] Schwartz, Matthew S. ACLU: Border Agents Violate Constitution When They Search Electronic Devices. NPR, NPR, 2 May 2019, www.npr.org/2019/05/02/719337356/aclu-border-agents-violate-constitution-when-they-search-electronic-devices.

Appendices

Appendix A

Code Listings

A.1 Client (client.py)

```
# Client recieves arguements
# Client opens connection with server
# Client encrypts file
# Client sends file, arguments
# Client waits for server response
# if server reports all clear, destroy original file
# else, throw execption, release file


#the core server code
import socket
import sys
import os
import datetime
#from threading import *
from lib import *
import hashlib
import keygen
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.Random import get_random_bytes
from email.utils import parseaddr


###DEFINES
fname = ''
MAXWAITS = 10
BUFSIZE = 4096
#HOST = "127.0.0.1"
#PORT = 4321 #I'm gonna keep this as the default port
DEFAULT_HOST = '172.22.173.33'
DEFAULT_PORT = 4321
###END DEFINES

def con(HOST = socket.gethostbyname(socket.gethostname()),PORT = 4321):
  #get socket
  s = socket.socket()
  #if any socket settings changes are needed, they go here
```

```python
    #connect to host
    HOST = input("Host? (default is localhost)") #may have to fix these due to the janky way that I did l
    PORT = input("Port? (default is 4321)")
    try:
        s.connect((HOST, int(PORT))) #https://docs.python.org/2/library/socket.html
    except:
        s.connect((DEFAULT_HOST, DEFAULT_PORT))
        print("Failed to connect to host, using defaults instead\n", flush = True)
    return s

def get_commands():
    #print("Commands not yet implemented")
    #sys.stdout.flush()
    inp = input("Email Address for return: ")
    com = "|EMAIL|"
    if not inp:
        print("No return address provided, quitting.\n")
        return
    email = parseaddr(inp)[1]
    if (not '@' in email or not '.' in email):
        print("Unacceptable email, quitting.\n")
        return
    com += email + '|'
    sleeptime = input("Time to sleep, in seconds: ")
    if not sleeptime:
        sleeptime = '10'
    com += sleeptime + '|'
    return com

def get_datablob():
    global fname
    fname = input("File: ")
    inf = open(fname,"rb")
    buf = inf.read(1)
    t = bytearray()
    while(buf):
        t += buf
        buf = inf.read(100)
    return t

def send_init(s,data):
    initpkt = InitPacket(commands,len(data))
    print("Sending InitPacket:\n\tCommands: "+ str(initpkt.commands) + "\n\tBlobsize: " + str(initpkt.blol
    print("Data: " + str(data))
    #Send Data
    s.sendall(str(initpkt.commands).encode('latin-1'))
    print("Sent commands.\n")
    s.recv(BUFSIZE)
    s.sendall(str(initpkt.blobsize).encode('latin-1'))
    print("Sent size.\n")
    s.recv(BUFSIZE)
    print("Sent InitPacket")
    return initpkt.blobsize
```

```python
def send_blob(s,data):

  dblob = DataBlob(data)
  print("Sending DataBlob:\n\tSize: "+str(dblob.size)+"\n\tHash: "+ str(dblob.md5hash) +"\n\tData: " + s

  for i in range(dblob.size//BUFSIZE + 1):
    print("Sent: part " + str(i))
    s.sendall(dblob.data[i*BUFSIZE:(i+1)*BUFSIZE].encode('latin-1'))
    s.recv(BUFSIZE)
  print("Sent DataBlob")



def recv_reply(s,data):
  rep = s.recv(BUFSIZE).decode('latin-1')
  print("Recieved ReplyPacket Hash: " + str(rep) + "\n")
  ##os.flush()
  return (rep == hashlib.md5(data).hexdigest())

def send_AES(sock):
  public_key = sock.recv(BUFSIZE).decode('latin-1')
  AES_key = get_random_bytes(16)
  encryped_AES_key = PKCS1_OAEP.new(RSA.import_key(public_key)).encrypt(AES_key)
  AES_key_object = AES.new(AES_key, AES.MODE_EAX)
  sock.sendall(encryped_AES_key)
  sock.recv(BUFSIZE)
  sock.sendall(str(AES_key_object.nonce).encode('latin-1'))
  return AES_key_object, AES_key

def recv_data(sock, numpacks):
  datastr = ""
  print("expecting " + str(numpacks) + " packets")
  for i in range(numpacks):
    print("Recieving part " + str(i))
    datastr = datastr + sock.recv(BUFSIZE).decode('latin-1')
    sock.sendall("ACK".encode('latin-1'))
  return datastr

def check_args():
  print("check_args() isn't finished!\n")
  def FAIL():
    print("Program was passed bad arguments\nCorrect arguments: <host> <port>\nQUITTING\n",flush=True)
    exit()
  #if (sys.argv[1]) #I'll just leave the hostname/ip formatting check to the exception catch in con() fo
  if (sys.argv[2].isnumeric() and 0 < int(sys.argv[1]) < 65535):
    pass
  else:
    FAIL()

if __name__ == "__main__":

  print ("Client started:", datetime.datetime.today().isoformat(),"\n")
  sys.stdout.flush()

  if len(sys.argv[1:]): #DEAL WITH COMMAND LINE USAGE HERE
```

23

```python
  CL = True
  check_args() #write this
  s = con(sys.args[1],sys.args[2])
else:
  CL = False
  s = con(DEFAULT_HOST, DEFAULT_PORT)


#connect
AES_init_bytes = get_random_bytes(16)
private_AES_key_object = AES.new(AES_init_bytes, AES.MODE_EAX)
AES_key_object, conn_AES = send_AES(s)
print("AES key: " + str(conn_AES))
print("AES nonce: " + str(AES_key_object.nonce))
sys.stdout.flush()


#Prompt user for commands
commands = get_commands()


#Create data object
data = AES_encrypt(private_AES_key_object, get_datablob())


size = send_init(s,data)
send_blob(s,data.decode('latin-1'))


#Wait for reply
valid = recv_reply(s,data)
print("Hash Comparison Check: " + str(valid) + "\n")
sys.stdout.flush()


#Recieve OTP
otp = recv_data(s, size//BUFSIZE + 1)
file = open("otp", 'w')
file.write(otp)
file.close()


#Wait


#Recieve OTP ^ Data
xor = recv_data(s, size//BUFSIZE + 1)


print("Recieved encoded: \n" + str(xor))
sys.stdout.flush()


otp_file = open("otp", 'r')
otp = otp_file.read()
data = ""
for a, b in zip(xor, otp):
  data += str(int(a) ^ int(b))


print("Read otp: \n" + otp)


print("Got data: \n" + data)
sys.stdout.flush()
```

```
  enc_data = ""
  for i in range(len(data)//8):
    enc_data += chr(int(data[i*8:i*8 + 8], 2))

  print("Encrypted data: " + enc_data)
  sys.stdout.flush()

  data = AES_decrypt(AES.new(AES_init_bytes, AES.MODE_EAX, private_AES_key_object.nonce), enc_data)
  print("File name: " + fname)
  sys.stdout.flush()
  file = open(fname.split('.')[0] + "_out." + fname.split('.')[1], 'wb')
  file.write(data);
  file.close()

  s.close()
  print("Connection Closed")
  sys.stdout.flush()
```

A.2   Server (server.py)

```
#the core server code
import socket
import sys
import os
import datetime
from threading import *
import lib
import hashlib
import keygen
import math
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES, PKCS1_OAEP
from Crypto.Random import get_random_bytes

###DEFINES
NUMTHREADS = 1 #actual running, 10 maybe? 1 for now since we just need to test that it works at all
MAXWAITS = 10
HOST = socket.gethostbyname(socket.gethostname())
PORT = 4321
BUFSIZE = 4096
OTP_KEY_EXTENSION = ".OTPkey"
GLOBAL_THREADNO = 0
###END DEFINES


def generateOTP(name,length): #returns name of file holding generated One Time Pad
length = math.ceil(length/8) * 8
otpname = name + OTP_KEY_EXTENSION
otpfile = open(otpname, "wb")
print("Length: " + str(length))
otp = str(bin(keygen.__getLargeRandom(length)))
print("Generated: " + otp)
otpfile.write(otp[2:].encode('latin-1'))
```

```python
otpfile.close()
return otpname

def applyOTP(name,otpname): #returns false if infile is empty, name of file holding result otherwise
resname = name + "_x_" + otpname
f1 = open(name,"rb")
f2 = open(otpname,"rb")
out = open(resname,"wb")
byte1 = f1.read(1)
byte2 = f2.read(1)
if (not byte1 or not byte2):
return False
while(byte1 and byte2):
bout = bytes([ord(byte1) ^ ord(byte2)])
out.write(bout)
byte1 = f1.read(1)
byte2 = f2.read(1)
if (not byte1 and byte2):
print("WARN: infile shorter than key!")
if (byte1 and not byte2):
print("WARN: key shorter than infile!")
#log_file.flush()
f1.close()
f2.close()
out.close()
return resname

def recieve_commands(conn):
init_commands = conn.recv(BUFSIZE).decode('latin-1')
print("Recieved commands: " + str(init_commands) + "\n")
conn.sendall("ACK".encode('latin-1'))
return init_commands

def recieve_size(conn):
init_size = conn.recv(BUFSIZE).decode('latin-1')
conn.sendall("ACK".encode('latin-1'))
print("Recieved size: " + str(init_size) + "\n")
return init_size

def recieve_data(conn, numpacks):
datastr = ""
for i in range(numpacks):
print("Recieving part " + str(i))
datastr = datastr + conn.recv(BUFSIZE).decode('latin-1')
conn.sendall("ACK".encode('latin-1'))
return datastr

def send_data(conn, data):
        print("Sending Data...")
        for i in range(len(data)//BUFSIZE + 1):
                print("Sent: part " + str(i))
                conn.sendall(data[i*BUFSIZE:(i+1)*BUFSIZE].encode('latin-1'))
                conn.recv(BUFSIZE)
        print("Sent all: " + data)
```

```python
def get_AES_key(conn):
        if not (os.path.isfile("public_key.pem")):
                RSA_key = RSA.generate(1024)
                private_key = RSA_key.export_key()
                private_file = open("private_key.pem", "wb")
                private_file.write(private_key)

                public_key = RSA_key.publickey().export_key()
                public_file = open("public_key.pem", "wb")
                public_file.write(public_key)

        public_key = open("public_key.pem").read()
        conn.sendall(public_key.encode('latin-1'))
        encrypted_AES_key = conn.recv(BUFSIZE)
        conn.sendall("0".encode('latin-1'))
        nonce = conn.recv(BUFSIZE)
        private_key = RSA.import_key(open("private_key.pem").read())
        k_AES = PKCS1_OAEP.new(private_key).decrypt(encrypted_AES_key)
        return AES.new(k_AES, AES.MODE_EAX, nonce), k_AES

def connHandler():
global GLOBAL_THREADNO
threadno = str(GLOBAL_THREADNO + 1)
GLOBAL_THREADNO += 1
def fail():
print ("Sending Fail Packet\n")
#conn.send(lib.ReplyPacket())

print(HOST)

s = socket.socket()
s.bind((HOST, PORT))
s.listen(MAXWAITS)
thisthread = str(current_thread())
print("Thread #" , thisthread ,": Handler started\n")
sys.stdout.flush()

conn, addr = s.accept()
print("Thread #", thisthread ,":",addr, "connected\n")
sys.stdout.flush()

        #Key Exchange
AES_key_object, k_AES = get_AES_key(conn)
print("AES key: " + str(k_AES))
print("AES nonce: " + str(AES_key_object.nonce))
sys.stdout.flush()

#Send initial packet
init_commands = recieve_commands(conn)
init_size = recieve_size(conn)
init_data = lib.InitPacket(init_commands, init_size)
if not init_data:
print("Thread #", thisthread ,":","Failed, no data recieved\n")
```

```
                return
        print("Thread #", thisthread ,":","Received connect from ", repr(addr), "\n")
        print("Thread #", thisthread ,":","\tblob size: ", init_data.blobsize)
        sys.stdout.flush()
        datastr = recieve_data(conn, int(init_data.blobsize)//BUFSIZE + 1)

                #Recieve file
        blob_data = lib.DataBlob(datastr)
        print("Created datablob\n")
        if not blob_data:
        print ("Thread #", thisthread ,":","Failed, blob data not recieved\n")
        fail()
        return

        #pull the data from the blob
        if (blob_data.md5hash != hashlib.md5(blob_data.data.encode('latin-1')).hexdigest()):
        print("Thread #", thisthread ,":","Failed, hashes do not match:", blob_data.md5hash, "vs." , hashlib.md5
        print("Data: " +  str(blob_data.data))
        fail()
        return
        sys.stdout.flush()
        #send reply
        conn.send(blob_data.md5hash.encode('latin-1'))
        print ("Thread #", thisthread ,":","Sent success packet\n")
        sys.stdout.flush()
        #Convert data to binary
        blob_data = lib.DataBlob(''.join(format(ord(i),'b').zfill(8) for i in blob_data.data))
        print("Data: " + blob_data.data)

        #Generate OTP of correct length
        generateOTP(threadno, blob_data.size)
        otpfile = open(threadno + OTP_KEY_EXTENSION, "r")
        otp = otpfile.read()
        print("OTP:   " + otp)
        otpfile.seek(0)

        outfile = open(str(threadno) + ".blobfile","w+b")
        data_bin = blob_data.data
        #if int(data_bin, 2) % 8 != 0:
        #print("Adding " + str(8 - (int(data_bin, 2) % 8)) + " 0s")
        #data_bin = "0" * (8 - (int(data_bin, 2) % 8)) + data_bin
        otp_bin = otp
        #data_bin = ''.join(format(ord(i),'b') for i in blob_data.data)
        #otp_bin = ''.join(format(ord(i),'b') for i in otp)
        print("Data_binary: " + data_bin)
        print("OTP_binary:  " + otp)
        sys.stdout.flush()
        encstr = ""
        for a, b in zip(data_bin, otp_bin):
        encstr += str(int(a) ˆ int(b))
        print("outp1:        " + encstr)
        outfile.write(encstr.encode('latin-1'))
        outfile.close()
        print ("Thread #", thisthread ,":","Wrote recieved data to file\n")
```

```python
sys.stdout.flush()
        #Send OTP
send_data(conn, otp_bin)
print("Sending ...")
sys.stdout.flush()
#Wait for some time
print("Waiting for " + init_data.commands.split('|')[3] + " seconds.\n")
sys.stdout.flush()
lib.wait_dhms(0,0,0,int(init_data.commands.split('|')[3]))

#Reread file
infile = open(str(threadno) + ".blobfile","r")
enc = infile.read()
print("Sending encoded...")
sys.stdout.flush()
lib.send_file_email_quick(init_data.commands.split('|')[2], str(threadno) + ".blobfile")
send_data(conn, enc)

conn.close()
s.close()
print ("Thread #", thisthread ,":","closed\n")
sys.stdout.flush()
#log_file.flush()

#def listenerThreads():
#th = []
#for i in range(NUMTHREADS): #for the constant version of this use threading.activeCount() in a loop
#thr = Thread(target=connHandler, args = ())
#thr.start()
#th.append(thr)
#log_file.flush()
#for thread in th:
# while thread.isAlive():
# pass


if __name__ == "__main__":
#log_file = open("log_" + datetime.datetime.today().isoformat().replace(":","-") + ".txt","w") #I think
#sys.stdout = log_file #all "print"s go to a logfile
print ("Server started:", datetime.datetime.today().isoformat(),"\n")
sys.stdout.flush()
#os.flush()
#listenerThreads()
connHandler()
print("Connection Closed\n")
sys.stdout.flush()
#os.flush()
sys.stdout = sys.__stdout__
#log_file.close()
```

A.3 Library Functions (lib.py)

```python
import sys
import hashlib
import smtplib
import time
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.base import MIMEBase
from email import encoders
from Crypto.Cipher import AES

class InitPacket:
  #this should store the details about the file to be sent & what to do with it
  def __init__(self,commands='',blobsize=0):
    self.commands = commands
    self.blobsize = blobsize

class DataBlob:
  def __init__(self,data = None):
    if data is None:
      self.data = []
      self.size = 0
      self.md5hash = 0
    else:
      self.data = data
      self.size = len(data)
      self.md5hash = hashlib.md5(data.encode('latin-1')).hexdigest()
  def update(self,new):
    self.data += new
    self.size = len(data)
    self.md5hash = hashlib.md5(data.encode('latin-1')).hexdigest()

class ReplyPacket:
  def __init__(self, success = False, ret_hash = hashlib.md5("error".encode('latin-1')).hexdigest()):
    self.success = success
    self.ret_hash = ret_hash


def AES_encrypt(key, data):
  a = key.encrypt(data)
  print(type(a))
  return a


def AES_decrypt(key, data):
  return key.decrypt(data.encode('latin-1'))

def send_file_email(email_server,email_port,email_user,email_pass,email_to,email_subj,email_msg,email_f:
  mail = MIMEMultipart()
  mail['From'] = email_user
  mail['To'] = email_to
  mail['Subject'] = email_subj
  mail.attach(MIMEText(email_msg,'plain'))
```

```
    atmt = open(email_filename,'rb')
    part = MIMEBase("application",'octet-stream')
    part.set_payload(atmt.read())
    encoders.encode_base64(part)
    part.add_header('Content-Disposition','attachment; filename='+email_filename)
    mail.attach(part)
    email_contents = mail.as_string()
    srv = smtplib.SMTP(email_server,email_port)
    srv.starttls()
    srv.login(email_user,email_pass)
    srv.sendmail(email_user,email_to,email_contents)
    srv.quit()

def send_file_email_quick(address,filename):
 send_file_email('smtp.gmail.com',587,'tt463'+'1309'+'@gm'+'ail.com','ddb_'+'sr'+'c_pass',address,'Your

def mail_test():
 send_file_email('smtp.gmail.com',587,'tt4'+'631309@gm'+'ail.com','dd'+'b_src_p'+'ass','tt46'+'31309@'+

def wait_dhms(days,hours,minutes,seconds):
  hours = hours + days*24
  minutes = minutes + hours*60
  seconds = seconds + minutes*60
  time.sleep(seconds)
```

A.4   Key Generation (keygen.py)

```
from os import urandom
import random

DEBUG = False

def __getRandomBitstream(bits):
    return urandom(bits//8)

def __isPrime(n, trials):
    """
    Miller-Rabin primality test.

    A return value of False means n is certainly not prime. A return value of
    True means n is very likely a prime.

    Source: RosettaCode.com
    """
    if n!=int(n):
        return False
    n=int(n)

    s = 0
    d = n-1
    while d%2==0:
        d>>=1
```

```python
        s+=1
    assert(2**s * d == n-1)

    def trial_composite(a):
        if pow(a, d, n) == 1:
            return False
        for i in range(s):
            if pow(a, 2**i * d, n) == n-1:
                return False
        return True

    for i in range(trials):
        a = random.randrange(2, n)
        if trial_composite(a):
            return False

    return True
'''
Generates a cryptographically random stream of bits.
Returns a str
'''
def __getLargeRandom(keylen):
    key = "";
    keyarr = __getRandomBitstream(keylen)
    for i in range(keylen//8):
        key = key + str(bin(keyarr[i])[2:]).zfill(8)
    return int(key, 2)

def __getLargeRandomPrime(length):
    i = 0
    while(True):
        i += 1
        num = __getLargeRandom(length)
        if i % 10 == 0:
            print('.', end="")
        if(__isPrime(num, 30)):
            print("Processed " + str(i) + " numbers.")
            return num


'''
Generates a cryptographically random stream of bits for use in the
AES encryption standard.
Recommended keylengths: 128, 192, 256
Returns a binary number
'''
def getAESKey(keylen):
    return __getLargeRandom(keylen)


'''
Uses pseudocode from Wikipedia
'''
def __euclideanGCD(num, mod):
```

```python
        div = mod // num
        rem = mod % num
        if DEBUG:
            print(str(mod) + " = " + str(div) + " * " + str(num) + " + " + str(rem))
        if rem == 0:
            return num, list()
        else:
            a, eqs = __euclideanGCD(rem, num)
            eqs.append([mod, div, num])
            return a, eqs

def __modInverse(num, mod):
    gcd, eqlist = __euclideanGCD(num, mod)

    if gcd != 1:
        return -1

    factorA = (1, eqlist[0][0])
    factorB = (-eqlist[0][1], eqlist[0][2])
    if DEBUG:
        print(str(gcd), end="")
        print(" = " + str(factorA[0]) + " * " + str(factorA[1]), end="")
        print(" + " + str(factorB[0]) + " * " + str(factorB[1]))

    for i in range(len(eqlist) - 1):
        eq = eqlist[i + 1]
        factorB = (factorB[0], eq[0])
        factorA = (factorA[0] - eq[1] * factorB[0], factorA[1])
        if DEBUG:
            print(str(gcd), end="")
            print(" = " + str(factorA[0]) + " * " + str(factorA[1]), end="")
            print(" + " + str(factorB[0]) + " * " + str(factorB[1]))
        if(factorA[1] < factorB[1]):
            temp = tuple(factorA)
            factorA = tuple(factorB)
            factorB = tuple(temp)

    inv = factorA[0] if factorA[1] != mod else factorB[0]
    return inv if inv > 0 else inv + mod

'''
Returns a secure RSA keypair and modulus.

'''
def getRSAKeypair(keylen):
    p = __getLargeRandomPrime(keylen)
    q = __getLargeRandomPrime(keylen)

    n = p * q
    mod = (p-1)*(q-1)
    pub = 65537
    priv = __modInverse(pub, mod)
    return pub, mod, priv
```

```
if __name__ == "__main__":
    if(DEBUG):
        #print(str(__modInverse(42, 2017)))
        #print(str(__modInverse(11, 14)))
        #print(str(__modInverse(17, 780)))
        print(getRSAKeypair(1024))
        print(getAESKey(128))
```