CS6053 - Network Security

Stack Attack

April 17, 2014

Contents

List of Figures

List of Tables

1 Introduction

The purpose of this lab was to create a program on the client end that connects to a server. On the server end is another program that is running called the `Monitor`. The `Monitor` listens for an open socket on an open port for incoming connections. As long as the `Monitor` is able to authenticate the client, the `Monitor` will reward points. The group with the highest points wins the contest. The challenging part of the contest is to remain a secure connection between the client and the server so the group is able to be rewarded with points. In order to maintain a secure connection we have implemented a few algorithms: Diffie-Hellman, Karn Symmetric Cyrptosystem, and the Fiat-Shamir Algorithm.

2 Implementation

2.1 Authentication of Monitor

In order to authenticate with the `Monitor` we first needed to implement a shared secret key. We did this by implementing the Diffie-Hellman protocol. The Diffie-Hellman key exchange works as shown below:

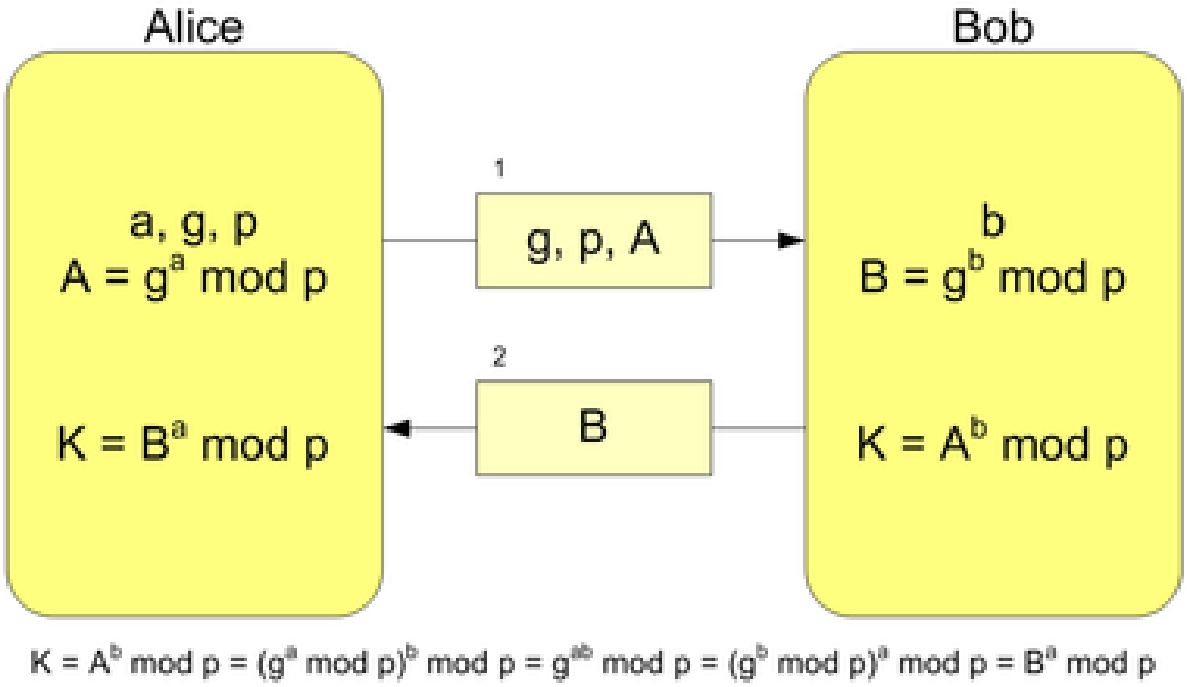

Figure 1: Diffie-Hellman Key Exchange

In the image above, we can see that the `g,p` are just the generator and the modulo prime number respectively. `A` is calculated using the function (g^a mod p) sent over to `Bob`. Note that g,p, and A are all seen by everyone. The trick here is that `Alice` and `Bob` both generate there own secret key locally (this key is never sent). The way we can authenticate can be shown in the example below:

$$A = g^a \ mod \ p,$$
$$B = g^b \ mod \ p,$$
$$A = 3^{15} \ mod \ 17 \equiv 6$$
$$B = 3^{13} \ mod \ 17 \equiv 12$$

where a = 15 (Alice's secret key), g = 3, p = 17 and
where b = 13 (Bob's secret key), g = 3, p = 17

Therefore, `Bob` receives this value from `Alice` along with p and g. Now `Bob` computes his result and sends `12` to Alice. Now the heart of the trick lies here: Alice will take Bob's message and use her private key to get the correct message from Bob:

$$A = 12^{15} \ mod \ 17 \equiv 10$$
$$B = 6^{13} \ mod \ 17 \equiv 10$$

The reason this works is because when you take an exponent to another exponent, these values get multiplied:

$$3^{13^{15}} \ mod \ 17 \equiv 3^{15^{13}} \ mod \ 17$$

Therefore, even if Eve performed a man-in-the-middle attack, all Eve would have is just the encrypted message and the prime and the generator number–making it rather difficult to decrypt the message via brute-forcing due to the discrete logarithm problem.

2.2 Karn Symmetric Cryptosystem

After implementing the Diffie-Hellman protocol we then needed to implement the Karn encryption scheme. The Karn encryption scheme uses the Secure Hash Algorithm (SHA-1) in order to generate the message digests. This algorithm uses the shared keys generated by the Diffie-Hellman protocol. It initially splits the bits of the shared secret key into two halves: a left and a right half. The encryption algorithm works as described below:

- Add "guard" byte of value 42,
- During each clock of plaintext, divide block of plaintext into left/right half,
- Find the message digest of the concatenation of the left plaintext half and the left key half,
- XOR the digest with the right plaintext half. (This produces the right ciphertext half),
- Find the message digest of the ciphertext right half concatenated with the key right half,
- XOR the digest with the plaintext left half.

Once this is complete we have both halves, therefore we can just output both the left and right half of the ciphertext.

In order to decrypt the message we needed to follow the steps shown below:

- Strip guard byte,
- For each ciphertext block, split the block into a left half and a right half,
- Find the message digest of the ciphertext right half concatenated with the key right half,
- XOR the result with the ciphertext left half to obtain the plaintext left half,
- Find the message digest of the plaintext left half and the key left half,
- XOR the result with the ciphertext right half to get the plaintext right half.

2.3 Zero-Knowledge Proof

2.4 Diffie-Hellman

2.5 Maintaining Security

3 Appendix

3.1 Python Code

```
import sys
import socket
import argparse
from printer import Printer
from config import Config
from diffie_hellman import DHE
from karn import Karn
from fiat import Prover
from base32 import base32
import parse_log

config = Config()
printer = Printer("client")
dhe = DHE()
prover = Prover()
authenticated = False
karn = None
transfer = None
exit = False
only_do_alive = False

def generate_response(line):
    global karn
    global authenticated
    global transfer
    global exit

    line = line.strip()
    directive, args = [x.strip() for x in line.split(':', 1)]

    if directive == "REQUIRE":
        if args == "IDENT":             return "IDENT %s %s" % (Config.ident, dhe.public_key)
        elif args == "PASSWORD":        return "PASSWORD %s" % Config.password
        elif args == "ALIVE":           return "ALIVE %s" % Config.cookie
        elif args == "HOST_PORT":       return "HOST_PORT %s %s" % (Config.server_ip, Config.server_port)
        elif args == "PUBLIC_KEY":      return "PUBLIC_KEY %s %s" % (base32(prover.v), base32(prover.n))
        elif args == "AUTHORIZE_SET":   return "AUTHORIZE_SET %s" % prover.authorize_set()
        elif args == "SUBSET_J":        return "SUBSET_J %s" % prover.subset_j()
        elif args == "SUBSET_K":        return "SUBSET_K %s" % prover.subset_k()
        else:                           printer.error("Unknown require: " + line)
    elif directive == "RESULT":
        args = args.split(' ', 1)
        if args[0] == "IDENT":
            dhe.monitor_key(args[1])
            karn = Karn(dhe.secret)
            printer.info("Setup secret key")
        elif args[0] == "PASSWORD":
            Config.cookie = args[1]
            printer.info("Got cookie: " + Config.cookie)
        elif args[0] == "HOST_PORT":
            printer.info("Login successful! (%s)" % args[1])
            if transfer is None and not manual_mode:
                exit = True
                return ""
        elif args[0] == "ALIVE" and args[1] == "Identity has been verified.":
            printer.info("Alive verified")
            authenticated = True
            if only_do_alive:
                exit = True
                return ""
        elif args[0] == "TRANSFER_REQUEST":
            printer.info("Transfer was %s" % args[1])
        elif args[0] == "ROUNDS":
            prover.rounds = int(args[1])
        elif args[0] == "SUBSET_A":
            prover.subset_a = [int(x) for x in args[1].split()]
        elif args[0] == "TRANSFER_RESPONSE" and not manual_mode:
            exit = True
            return ""
        else:
            printer.error("Unknown result: %s (args: %r)" % (line, args))
    elif directive == "WAITING":
        if transfer and authenticated:
            rt = "TRANSFER_REQUEST %s %s FROM %s\n" % tuple(transfer)
            transfer = None
```

```
76              return rt
77          if manual_mode and authenticated:
78              return raw_input('Command: ')
79      elif directive == "COMMENT":
80          pass
81      elif directive == "COMMAND_ERROR":
82          exit = True
83          printer.error(line)
84          return ""
85      else:
86          exit = True
87          printer.error("Unknown directive: " + line)
88          return ""
89
90      return None
91
92  if __name__ == "__main__":
93
94      parser = argparse.ArgumentParser()
95      parser.add_argument('--ident', default="mtest16")
96      parser.add_argument('--transfer', nargs=3, metavar=('TO', 'AMOUNT', 'FROM'))
97      parser.add_argument('--manual', action='store_true')
98      parser.add_argument('--alive', action='store_true')
99      parser.add_argument('--logfile', default="/home/httpd/html/final.log.8180")
100     args = parser.parse_args()
101
102     manual_mode = args.manual
103     transfer = args.transfer
104     only_do_alive = args.alive
105
106     if args.ident not in Config.accounts:
107         print "Invalid ident"
108         sys.exit(1)
109
110     Config.ident = args.ident
111     Config.server_port = Config.accounts[args.ident].port
112
113     parse_log.parse(args.logfile)
114     Config.cookie = parse_log.cookies[args.ident.lower()]
115     Config.password = parse_log.passes[args.ident.lower()]
116
117     print "Using cookie: '%s'" % Config.cookie
118
119     sock = socket.create_connection((Config.monitor_ip, Config.monitor_port))
120
121     for line in sock.makefile():
122
123         # check if this line is encrypted
124         if line.startswith('1a'):
125             line = karn.decrypt(line)
126             if line is None: continue
127             printer.directive(line, encrypted=True)
128         else:
129             printer.directive(line)
130
131         # generate the response for this line
132         response = generate_response(line)
133
134         # if there is no response go get another line
135         if response is None: continue
136
137         printer.command(response + '\n', encrypted=karn)
138
139         # if we have a valid karn we can encrypt the response
140         if karn:
141             response = karn.encrypt(response)
142
143         # add the newline and send the response
144         response += '\n'
145         sock.send(response)
146
147         if exit: break
148
149     sock.close()
```

Listing 1: Client

```
1  import argparse
2  import sys
```

```
3  import socket
4  import SocketServer
5  import platform
6  import subprocess
7  from config import Config
8  from config import checksums
9  from printer import Printer
10 from diffie_hellman import DHE
11 from karn import Karn
12 from fiat import Verifier
13 import parse_log
14
15 class tcp_handler(SocketServer.StreamRequestHandler):
16
17     def setup(self):
18
19         self.dhe = DHE()
20         self.karn = None
21         self.verifier = Verifier()
22
23         self.printer = Printer("server")
24
25         self.exit = False
26
27         SocketServer.StreamRequestHandler.setup(self)
28
29     def finish(self):
30
31         SocketServer.StreamRequestHandler.finish(self)
32
33     def generate_response(self, line):
34
35         line = line.strip()
36         directive, args = [x.strip() for x in line.split(':', 1)]
37
38         if directive == "REQUIRE":
39             if args == "IDENT":
40                 return "IDENT %s %s" % (Config.ident, self.dhe.public_key)
41             elif args == "ALIVE":
42                 return "ALIVE %s" % Config.cookie
43             elif args == "ROUNDS":
44                 return "ROUNDS %s" % Config.num_rounds
45             elif args == "SUBSET_A":
46                 return "SUBSET_A %s" % ' '.join(str(x) for x in self.verifier.subset_a)
47             elif args == "TRANSFER_RESPONSE":
48                 if self.verifier.good():
49                     return "TRANSFER_RESPONSE ACCEPT"
50                 else:
51                     return "TRANSFER_RESPONSE DECLINE"
52             elif args == "QUIT":
53                 return "QUIT"
54             else:
55                 self.printer.error("Unknown require: " + line)
56         elif directive == "RESULT":
57             args = args.split(' ', 1)
58             if args[0] == "IDENT":
59                 self.dhe.monitor_key(args[1])
60                 self.karn = Karn(self.dhe.secret)
61                 self.printer.info("Setup secret key")
62             elif args[0] == "ALIVE" and args[1] == "Identity has been verified.":
63                 self.printer.info("Alive verified")
64             elif args[0] == "QUIT":
65                 self.printer.info("server has quit")
66             elif args[0] == "SUBSET_K":
67                 self.verifier.subset_k = [int(x) for x in args[1].split()]
68             elif args[0] == "SUBSET_J":
69                 self.verifier.subset_j = [int(x) for x in args[1].split()]
70             elif args[0] == "PUBLIC_KEY":
71                 self.verifier.v = int(args[1].split()[0], 32)
72                 self.verifier.n = int(args[1].split()[1], 32)
73             elif args[0] == "AUTHORIZE_SET":
74                 self.verifier.authorize_set = [int(x) for x in args[1].split()]
75             else:
76                 self.printer.error("Unknown result")
77         elif directive == "PARTICIPANT_PASSWORD_CHECKSUM":
78             self.printer.info("Got checksum: %s" % args)
79             if args not in checksums:
80                 self.printer.error("INVALID CHECKSUM")
81                 self.exit = True
82                 return ""
83         elif directive == "WAITING":
```

```
            pass
        elif directive == "COMMENT":
            pass
        else:
            self.printer.error("Unknown directive: " + line)

        return None

    def handle(self):

        for line in self.rfile:

            # check if this line is encrypted
            if line.startswith('1a'):
                line = self.karn.decrypt(line)
                if line is None: continue
                self.printer.directive(line, encrypted=True)
            else:
                self.printer.directive(line)

            # generate the response for this line
            response = self.generate_response(line)

            # if there is no response go get another line
            if response is None: continue

            self.printer.command(response + '\n', encrypted=self.karn)

            # if we have a valid karn we can encrypt the response
            if self.karn:
                response = self.karn.encrypt(response)

            # add the newline and send the response
            response += '\n'
            self.wfile.write(response)

            if self.exit:
                self.printer.error("Exiting!")
                break

if __name__ == "__main__":

    parser = argparse.ArgumentParser()
    parser.add_argument('--ident', default="mtest16")
    parser.add_argument('--logfile', default="/home/httpd/html/final.log.8180")
    args = parser.parse_args()

    if args.ident not in Config.accounts:
        print "Invalid ident"
        sys.exit(1)

    Config.ident       = Config.accounts[args.ident].ident
    Config.server_port = Config.accounts[args.ident].port

    parse_log.parse(args.logfile)
    Config.cookie = parse_log.cookies[args.ident.lower()]
    Config.password = parse_log.passes[args.ident.lower()]
    print "Using cookie: '%s'" % Config.cookie

    print "Starting server on %s:%s" % (Config.server_ip, Config.server_port)
    server = SocketServer.ThreadingTCPServer((Config.server_ip, Config.server_port), tcp_handler)
    server.serve_forever()
```

Listing 2: Server

```
from random import getrandbits
from base32 import base32

class DHE:

    def __init__(self, key=None):

        self.p = 0x96C99B60C4F823707B47A848472345230C5B25103DC37412A701833E8FF5C567A53A41D0B37B10F0060D50F4131C57CF1FD
        self.g = 0x2C900DF142E2B839E521725585A92DC0C45D6702A48004A917F74B73DB26391F20AEAE4C6797DD5ABFF0BFCAECB29554248

        """
        Generate a new private key if it wasn't given
        """
        if (key):
```

```
            self.private_key = int(key, 32)
        else:
            self.private_key = getrandbits(512)

        """
        Public Key = g**x % p
        """
        self.public_key = base32(pow(self.g, self.private_key, self.p))

    def monitor_key(self, key):
        """
        Takes in the monitors public key in Base 32
        """
        self.secret = pow(int(key,32), self.private_key, self.p)
```

Listing 3: Diffie Hellman

```
from hashlib import sha1
from itertools import izip_longest
import struct
import base64
import string
from printer import Printer

BLOCK_SIZE = 40
GUARD_BYTE = 42

class Karn:

    def __init__(self, key):

        # Convert the integer key into a hex string
        # If the length of it is odd we need to left pad a '0'
        key_hex = '%x' % key
        if len(key_hex) & 1:
            key_hex = '0' + key_hex

        # Convert the hex key string into byte array
        key = bytearray.fromhex(key_hex)

        # java BigInteger.toByteArray() is signed so it pads if MSB is 1
        # http://stackoverflow.com/a/8544521/253650
        if key[0] & (1<<7):
            key = bytearray([0]) + key

        # Split key into two halfs
        # Monitor drops last byte if len is odd (monitor/Cipher.java:87)
        self.key_left  = key[:len(key)/2]
        self.key_right = key[len(key)/2:(len(key)/2)*2]

        self.printer = Printer("karn")

    def encrypt(self, message):

        # Start the output with the guard byte
        output = bytearray([GUARD_BYTE])

        # Break message into blocks and process each one
        for block in self._grouper(message, BLOCK_SIZE, '\0'):

            # Convert the block into an array of bytes
            block = bytearray(block)

            # Divide block into left and right half
            block_left  = block[:BLOCK_SIZE/2]
            block_right = block[BLOCK_SIZE/2:]

            # Hash the left plaintext plus the left key
            digest = sha1(block_left + self.key_left).digest()

            # XOR the digest with the right plaintext
            cipher_right = bytearray([ord(d)^b for d,b in zip(digest, block_right)])

            # Hash the right cipher plus the right key
            digest = sha1(cipher_right + self.key_right).digest()

            # XOR the digest with the left plaintext
            cipher_left = bytearray([ord(d)^b for d,b in zip(digest, block_left)])

```

```
63              output += cipher_left
64              output += cipher_right
65
66          # Convert output to a hex string
67          output = '0x'+''.join('%02X' % x for x in output)
68
69          # Convert the hex string to an integer and then conver to base 32
70          return self._baseN(int(output, 16), 32)
71
72      def decrypt(self, message):
73
74          # Convert the message from base 32 to hex
75          message = '%x' % int(message, 32)
76          if len(message) & 1:
77              message = '0' + message
78
79          # Convert the hex string to a byte array
80          message = bytearray.fromhex(message)
81
82          # If the first byte isn't the guard byte we are done
83          if message[0] != GUARD_BYTE:
84              self.printer.error("Did not find guard_byte!")
85              return None
86
87          # Remove the guard byte
88          message = message[1:]
89
90          output = bytearray()
91
92          # Break message into blocks and process each one
93          for block in self._grouper(message, BLOCK_SIZE, '\0'):
94
95              # Convert the block into an array of bytes
96              block = bytearray(block)
97
98              # Divide block into left and right half
99              block_left  = block[:BLOCK_SIZE/2]
100             block_right = block[BLOCK_SIZE/2:]
101
102             # Find digest of cipher right and key right
103             digest = sha1(block_right + self.key_right).digest()
104
105             # XOR the digest with cipher left
106             text_left = bytearray([ord(d)^b for d,b in zip(digest, block_left)])
107
108             # Find the digest of text left and key left
109             digest = sha1(text_left + self.key_left).digest()
110
111             # XOR the digest with cipher right
112             text_right = bytearray([ord(d)^b for d,b in zip(digest, block_right)])
113
114             output += text_left
115             output += text_right
116
117         # Remove the padding
118         output = str(output).split('\0')[0]
119
120         # Make sure the plaintext is normal printable text
121         if not all(x in string.printable for x in output):
122             self.printer.error("Unable to decrypt: %r" % output)
123             return None
124
125         return output
126
127     def _grouper(self, iterable, n, fillvalue=None):
128         """
129         Collect data into fixed-length chunks or blocks
130         grouper('ABCDEFG', 3, 'x') --> ABC DEF Gxx
131         http://docs.python.org/2/library/itertools.html#recipes
132         """
133         args = [iter(iterable)] * n
134         return izip_longest(fillvalue=fillvalue, *args)
135
136     def _baseN(self, num,b,numerals="0123456789abcdefghijklmnopqrstuvwxyz"):
137         """
138         Return 'num' in base 'b'
139         http://stackoverflow.com/a/2267428
140         """
141         return ((num == 0) and numerals[0]) or (self._baseN(num // b, b, numerals).lstrip(numerals[0]) + numerals[num
142
143 if __name__ == "__main__":
```

```
144
145     text = "this is a test"
146     key = 123456789
147     enc =  "1avfbcuej96oo1m2vc04rnlin6rpurc2pu7ac8h42dhr8l13ahdfcbev3a5sj74o85"
148
149     k = Karn(key)
150
151     print "--- Testing Correct ---"
152     e = k.encrypt(text)
153     assert e == enc
154     d = k.decrypt(e)
155     assert d == text
156
157     print "--- Testing No Guard Byte ---"
158     e = '0' + enc[1:]
159     d = k.decrypt(e)
160     assert d == None
161
162     print "--- Testing Corrupted Message ---"
163     e = enc[:10] + "00000" + enc[15:]
164     d = k.decrypt(e)
165     assert d == None
166
167     print "--- Done ---"
```

Listing 4: Karn

```
1   import random
2   from config import Config
3
4   class Prover:
5
6       '''
7       http://pages.swcp.com/~mccurley/talks/msri2/node24.html
8       http://friedo.szm.com/krypto/AC/ch21/21-01.html
9       '''
10
11      p = 18446744073709551253
12      q = 18446744073709551557
13      n = p * q
14
15      def __init__(self):
16
17          self.s = random.randrange(2<<64)
18          self.v = pow(self.s, 2, self.n)
19          self.subset_a = []
20          self.r_set = []
21          self.rounds = 10
22
23      def authorize_set(self):
24          # generate the random set
25          self.r_set = [random.randrange(2<<64) for _ in range(self.rounds)]
26          # calculate the authorize set
27          auth_set = [pow(r, 2, self.n) for r in self.r_set]
28          # return it as a string
29          return ' '.join(str(x) for x in auth_set)
30
31      def subset_k(self):
32          k = [(self.s * self.r_set[i]) % self.n for i in self.subset_a]
33          return ' '.join(str(x) for x in k)
34
35      def subset_j(self):
36          j = [self.r_set[i] % self.n for i in set(range(self.rounds)) - set(self.subset_a)]
37          return ' '.join(str(x) for x in j)
38
39
40  class Verifier:
41
42      def __init__(self):
43
44          self.rounds = Config.num_rounds
45          self.v = 0
46          self.n = 0
47
48          self.authorize_set = []
49          self.subset_j = []
50          self.subset_k = []
51
52          self.subset_a = sorted(random.sample(range(self.rounds), self.rounds // 2))
```

```
53
54      def good(self):
55
56          # check that the public key is our own
57          if self.n != Prover.n:
58              print "public keys dont match!"
59              return False
60
61          j = iter(self.subset_j)
62          k = iter(self.subset_k)
63          for i in range(self.rounds):
64              if i in self.subset_a:
65                  if pow(next(k), 2, self.n) != (self.v * self.authorize_set[i]) % self.n:
66                      return False
67              elif pow(next(j), 2, self.n) != self.authorize_set[i]:
68                  return False
69
70          return True
```

Listing 5: Fiat-Shamir - Zero Knowledge Proof

```
1   from config import Config
2
3   class Printer:
4
5       BOLD    = '\033[1m'
6       BLACK   = '\033[30m'
7       RED     = '\033[31m'
8       GREEN   = '\033[32m'
9       YELLOW  = '\033[33m'
10      BLUE    = '\033[34m'
11      PURPLE  = '\033[35m'
12      CYAN    = '\033[36m'
13      WHITE   = '\033[37m'
14      DEFAULT = '\033[0m'
15
16      def __init__(self, who=""):
17          self.who = who
18
19      def directive(self, string, encrypted=False):
20          if encrypted:
21              label = "<E<"
22          else:
23              label = "<<<"
24          self._print(string, label, self.GREEN)
25
26      def command(self, string, encrypted=False):
27          if encrypted:
28              label = ">E>"
29          else:
30              label = ">>>"
31          self._print(string, label, self.BLUE)
32
33      def error(self, string):
34          string = string.strip()
35          self._print(string, "---", self.RED)
36
37      def info(self, string):
38          string = string.strip()
39          self._print(string, "---", self.YELLOW)
40
41      def _print(self, string, label, color):
42          string = string.strip()
43          if string == "": return
44          print "[" + Config.ident + "] " + label + "  " + color + string + self.DEFAULT
```

Listing 6: Printer

```
1   def base32(num):
2       return _baseN(num, 32)
3
4   def _baseN(num, b, numerals="0123456789abcdefghijklmnopqrstuvwxyz"):
5       """
6       Return 'num' in base 'b'
7       http://stackoverflow.com/a/2267428
8       """
9       return ((num == 0) and numerals[0]) or (_baseN(num // b, b, numerals).lstrip(numerals[0]) + numerals[num % b])
```

Listing 7: Base 32

```
import hashlib
import socket

checksums = []

class Account:
    def __init__(self, ident, password, cookie, port):
        global checksums
        self.ident = ident
        self.password = password
        self.cookie = cookie
        self.port = port
        checksums.append(hashlib.sha1(password).hexdigest())

class Config:
    monitor_dns = "gauss.ececs.uc.edu"
    monitor_ip = socket.gethostbyname(monitor_dns)
    monitor_port = 8180
    server_ip = socket.gethostbyname(socket.getfqdn())
    server_port = 20167
    accounts = {
            "EDSNOWDEN":     Account("EDSNOWDEN", "8Y6]%D[:T%2!.P^$6AGDEHMCZICN;OOX-]UOJU=^G-", "ATPL5QCPINFHAWYQWXN",
            "GROUND_WATER": Account("GROUND_WATER", "Y9JUY8A`L?+;RK9O$;SWRLO_M'I-[9MP)IK=*]4-E!", "QNU0DWVPNQFGOSD1TC
            "CORNHOLIO":     Account("CORNHOLIO", "-S_:9J7R9X*/&7UDL'SX-'2?R=FOW2!T'I8.Q7F?PA", "W6M2D5OCZAJZWTOV3M8",
            "mtest16":       Account("mtest16", "12345", "C5K8GNM22KQ5XCFKVHF", 11121),
            "mtest17":       Account("mtest17", "12345", "J98Q82H1C458X6YAKAI", 11122),
            "mtest18":       Account("mtest18", "12345", "MGYKSTOKL4T106N0175", 11123),
    }

    ident = ""
    password = ""
    cookie = ""

    num_rounds = 5
```

Listing 8: Configuration File

```
import random
import gmpy2

'''
def is_prime(n):
    for k in range(50):
        a = random.randrange(1, n-1)
        a = gmpy2.powmod(a, n-1, n)

        if (a == 1):
            print "prime = %i\n" % (n)
            return True
        else:
            return False

def find_prime(lower_bound, upper_bound):
    for _ in range(1000000):
        if (~is_prime(lower_bound)):
            lower_bound = lower_bound + 1
            continue
        if (~is_prime(upper_bound)):
            upper_bound = upper_bound + 1
            continue
        n = random.randrange(lower_bound | 1, upper_bound, 2)
        if is_prime(n):
            print "%d | %d | %d\n" %(lower_bound, upper_bound, n)
            return

find_prime(2**32-1,2**64-1)
'''

def is_prime(n, k=50):
    '''Use Miller-Rabin Primality test to test if an integer is probably prime.

    The probability that this test will report a prime number as composite is 4**-k'''
    if n == 2 or n == 3:
```

```
37          return True
38      if n < 2:
39          return False
40
41      #divide by some small primes first to fast track most composite numbers
42      for i in (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43 ,47):
43          if n % i == 0:
44              return False
45
46      s = 1
47      d = n - 1
48
49      while d % 2 == 0:
50          s += 1
51          d = d >> 1
52
53      for i in xrange(k):
54          x = pow(random.randint(2,n-2), d, n)
55          if x != 1 and x != n - 1:
56              for _ in xrange(1, s):
57                  x = pow(x, 2, n)
58                  if x == 1:
59                      return False
60                  if x == n - 1:
61                      break
62              else:
63                  return False
64      return True
65
66  def find_prime(lower_bound, upper_bound):
67      for _ in range(1000):
68          n = random.randrange(lower_bound | 1, upper_bound, 2)
69          print "%d | %d | %d\n" %(lower_bound, upper_bound, n)
70          if is_prime(n):
71              return n
72      raise FiatShamirError('Could not find a prime number.')
73
74  find_prime(2**64-363,2**64-59)
```

Listing 9: Test 1

```
1   from random import randrange
2
3   class fiat:
4       def __init__(self):
5           '''
6           http://pages.swcp.com/~mccurley/talks/msri2/node24.html
7           http://friedo.szm.com/krypto/AC/ch21/21-01.html
8           '''
9           p = 18446744073709551253
10          q = 18446744073709551557
11          n = p * q
12          self.s = randrange(self.n)
13          self.v = pow(self.s, 2, self.n)
14          self.rounds = 50
15
16      # client
17      def prover(self):
18
19          self.x = 0
20          self.y = 0
21          self.r = 0
22          self.z_n = ()
23          self.z_n = tuple(randrange(self.n) for _ in xrange(self.rounds))
24
25          # A: picks random r in Z_n *, sends x=r^2 mod n to B
26          def public_authen(self):
27              for self.r in self.z_n:
28                  self.x = pow(r, 2, self.n)
29                  yield self.x
30
31          #a A sends y to B, where If c=0, y=r, else y=rs mod n
32          def send(self):
33              if (~self.c):
34                  self.y = self.r
35                  yield self.y
36              else:
37                  self.y = (self.r*self.s) % self.n
38                  yield self.y
```

```
39
40      # server
41      def verifier(self):
42
43          self.c = 0
44          self.autorization = ()
45          self.key_i = ()
46          self.key_j = ()
47
48          # B checks x!=0 and sends random c in {0,1} to A
49          def check(self):
50              if (self.x):
51                  self.c = randrange(0,1)
52                  yield self.c
53
54          # B accept if y^2 congruent xv^c mod n
55          def accept(self):
56              if (self.n == fiat.n): return True
57
58              if (pow(self.y,2,self.n) == (self.x*pow(self.v,self.c))):
59                  return True
60
61              i = iter(self.key_i)
62              j = iter(self.key_j)
63              for r in xrange(self.rounds):
64                  if pow(next(i), 2, self.n) == ((self.v * self.autorization[i]) % self.n):
65                      return True
66                  elif pow(next(j), 2, self.n) == (self.autorization[i]):
67                      return True
68                  else:
69                      return False
```

Listing 10: Test 2

References

[1] http://www.digicert.com/sha-2-ssl-certificates.htm