Bachelor-Projekt
Network-Security-Übungsblatt 6

Mader, Burmester · SoSe 2016

Kryptographie

1. Absicherung des TCP-Chats mit SSL

Zur Absicherung des TCP-Chats wurden die bestehenden Sockets im Server und im Client durch SSlSockets, bzw. SSLServerSockets ausgetauscht (s. Anhang A). Die Authentifizierung per Benutzername und Passwort wurde entfernt, stattdessen benötigte der Client von nun an das richtige Zertifikat.

Dieses konnte mit folgendem Befehl erzeugt werden:

```
keytool -genkey -keystore mySrvKeystore -keyalg RSA
```

Und musste dann beim Starten des Servers und des Klienten angegeben werden.

```
java -Djavax.net.ssl.trustStore=mySrvKeystore -Djavax.net.ssl.
trustStorePassword=123456 javaLanguage.basic.socket.SSLSocketClient
```

```
java -Djavax.net.ssl.keyStore=mySrvKeystore -Djavax.net.ssl.
keyStorePassword=123456 javaLanguage.basic.socket.SSLSocketServer
```

Nun konnte die Chat-Kommunikation mittels Wireshark nicht mehr mitgelesen werden.

2. CAs und Webserver-Zertifikate

2. **Selbstsignierte Zertifikate**

    1. `openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days XXX`
    Erzeugt ein selbstsigniertes Server-Zertifikat. In der Datei `/etc/apache2/sites-available/default` musste nun noch der Pfad zu dem neuen Zertifikat angegeben werden. Nach einem Neustart des Apache-Servers konnte mit dem neuen Zertifikat gearbeitet werden.
    2. In den Firefox-Einstellungen konnte unter Advanced → Encryption → View Certificates → Authorities → Import: Das Zertifikat `cert.pem` als sicher eingestuft werden.
    3. Selbstsignierte Zertifikate könnten auch durch Schadsoftware oder andere Nutzer ohne großen Aufwand eingetragen werden.

3. **HTTPS-Weiterleitung**

    Mit der Rewrite-Regel:

    ```
    RewriteEngine On
    RewriteCond %HTTPS !=on
    RewriteRule ^/?(.*) https://%{SERVER_NAME}/$1 [R,L]
    ```

in der .htaccess Datei, werden alle Verbindungsversuche über `http` Serverweit auf eine gesicherte `https` Verbindung umgeleitet.

4. **sslstrip**

    1. Da das Programm `sslstrip` auf unserer Ubuntu-Version auch nach dem bearbeiten der Archiv-Paketquellen nicht installiert werden konnte, da die Python-Version ebenfalls veraltet war und sich nicht aktualisieren wollte, haben wir `sslstrip` auf einem privaten Computer installiert.
    2. Das aufrufen von `http://svs.informatik.uni-hamburg.de/ssldemo/` führte zu einer Weiterleitung auf eine `https` Verbindung.
    3. Wireshark ist nicht imstande die verschlüsselten Anmeldedaten darzustellen.
    4. **sslstrip -p -l 8080** Startet `sslstrip` als HTTP-Proxy auf dem Port 8080.
    5. In den Verbindungseinstellungen von Firefox kann nun der Proxy-Server `localhost` mit Port 8080 angegeben werden, um die Daten über `sslstrip` zu leiten.
    6. Folglich wird der aufruf einer ungesicherten `http-Verbindung` nicht mehr auf `https` umgeleitet.
    7. Die Verbindung zu `http://svs.informatik.uni-hamburg.de/ssldemo/` konnte über den privaten Computer nicht hergestellt werden, auf dem wir `sslstrip` verwendet haben. Ohne den Proxy konnte sie zwar aufgerufen werden, mit `sslstrip` kam jedoch keine Verbindung zustande. Stattdessen testeten wir den Angriff auf der Webseite `http://amazon.com`.
    8. Die Webseite von Amazon konnte hier jedoch von `sslstrip` nur stark verändert ausgegeben werden, sodass ein Nutzer die Änderung nicht übersehen könnte. Wir haben dennoch Testdaten eingegeben, und konnten diese nach Beendigung von `sslstrip` in dem Log-File wiederfinden (s. Anhang B).
    9. Die einfache Umleitung auf `https`, wie in Aufgabe 2.3 angewandt, kann auf diese Weise also mit wenigen Schritten umgangen werden.
    10. Die **HTTP Strict Transport Security** (HSTS) hat das Ziel den Klienten vor Downgrade-Attacken wie `sslstrip` zu schützen. Hierbei wird über den HTTP response header die Information mitgeliefert, dass der Klient, für eine mitgegebene Zeit, ausschließlich über eine verschlüsselte Verbindung kommunizieren sollte. Damit wird die Gefahr eines Man-in-the-Middle-Angriffes stark verringert. Ein Angriff mit `sslstrip` könnte dadurch verhindert werden, solange dieser nicht in der Lage ist den HTTP response header zu manipulieren.

Bachelor-Projekt
Network-Security-Übungsblatt 6

Mader, Burmester · SoSe 2016

3. Unsichere selbstentwickelte Verschlüsselungsalgorithmen

1. **BaziCrypt**

   Durch die Padding-Bytes (0x00) konnte der Schlüssel aus den verschlüsselten Nachrichten einfach herausgelesen werden. Eine Verknüpfung eines Bits a XOR 0 mit a $\in 0, 1$ hat a als Ausgabe. Da sich der der Schlüssel immer wiederholt, ist er in der verschlüsselten Nachricht gut sichtbar.

   Wenden wir nun auf die Nachricht eine bitweise XOR-Verknüpfung mit dem extrahierten und wiederholtem Schlüssel an, so erhalten wir die entschlüsselte Nachricht.

   Die erste Nachricht wurde manuell entschlüsselt. Dazu wurde der Schlüssel `6535336B396478653533` herausgelesen und die zu entschlüsselnde Nachricht mit dem wiederholten Schlüssel XOR-Verknüpft. Das Ergebnis war der Klartext.

   ```
   Hallo Peter. Endlich koennen wir geheim kommunizieren! Bis bald,
   Max
   ```

   Folgend wurden die Nachrichten mit einem Python-Script (s. Anhang C) entschlüsselt.

   Schlüssel Nachricht 2: `37646D4B646E37646D4B`
   ```
   Hi Max! Super, Sicherheitsbewusstsein ist ja extrem wichtig! Schoene
   Gruesse, Peter.
   ```

   Schlüssel Nachricht 3: `526C39734E336334526C`
   ```
   Hi Peter, hast du einen Geheimtipp fuer ein gutes Buch fuer mich?
   Gruss, Max
   ```

2. **AdvaziCrypt - Denksport**

   Mit dem PKSC7-Padding wird der zu füllende Bereich mit dem Hexadezimalen Wert der Anzahl zu füllender Bytes aufgefüllt. Werden also zum Beispiel 10 weitere Bytes benötigt, so wird dieser Bereich mit dem Wert A aufgefüllt. Das Knacken einer Verschlüsselung mit PKSC7-Padding ist nun aufwändiger, da das Ende der eigentlichen Nachricht herausgefunden werden muss. Da sich jedoch auch hier wieder ein wiederholender Wert verbirgt, kann zumindest der ungefähre Endpunkt der Nachricht ermittelt und fortan mit wenigen Versuchen die exakte Lösung gefunden werden.

   Bei der ersten Nachricht wurde mit dem Wert 19 aufgefüllt. Der Codierte Schlüssel kann nun mittels XOR 1919191919... entschlüsselt werden.

   Der Schlüssel lautet: `71496B346E336F71496B`
   Und bietet uns die Möglichkeit die Nachricht komplett zu entschlüsseln.
   ```
   Hi Max, natuerlich: Kryptologie von A. Beutelspacher ist super.
   Gruss Peter
   ```

   Schlüssel Nachricht 2: `015c055b467e015c055b`
   ```
   Hi Peter, worum geht es in dem Buch? Ciao, Max.
   ```

Schlüssel Nachricht 3: `2b742b2b2f6b7d7b2b74`

```
Hi Max, das ist ein super Buch, das viele Krypto-Themen abdeckt.
Gruss Peter
```

3. **AdvaziCrypt - Angriff implementieren**

   Auch die implementierte Version (s. Anhang D) unseres Angriffes führte zu einem erfolgreichen Entschlüsseln der Nachrichten.

4. EasyEAS

Dem Meet-in-the-Middle-Angriff (s. Anlage E) liegt die Überlegung zugrunde $c1 = E\_k1(Verschluesselung)$ mit allen Möglichkeiten für $k1$ (s. Anlage F) zu generieren und in einer Hashtabelle zu speichern. Folgend wird dann der bekannte Schlüsseltext mit allen möglichen Schlüsseln entschlüsselt und überprüft ob dieser in der angelegten Hashtabelle enthalten ist. Auf diese Weise konnten die beiden verwendeten Schlüssel mit geringem Zeitaufwand gefunden werden. Mit den 7807081 möglichen Schlüsseln Müssen wir durch den Meet-in-the-Middle-Angriff lediglich 7807081 Verschlüsselungen durchführen und speichern, und noch einmal maximal 7807081 Entschlüsselungen durchführen und in der Hashtabelle nach diesen Suchen. Ist die Hashtabelle ideal aufgebaut, kann hier in konstanter Zeit gesucht werden. Der Aufwand des Angriffes liegt also bei n möglichen Schlüsseln bei $4 \cdot n$ Schritten und damit in $O(n)$. Bei einem naiven Angriff, in dem alle Möglichkeiten von $E\_k2(E\_k1(Klartext)$ gespeichert und mit dem bekannten Schlüsseltext verglichen werden, liegt die Komplexität bei $O(n^2)$ mit n als Anzahl der möglichen Schlüssel.

Die gefundenen Schlüssel lauten:

k1 = 000000f500000000000006300000000000
k2 = 0000000077000000b000000000000000

Probleme bereitete im Grunde lediglich die Handhabung mit den ver- und entschlüsselten Darstellungen, weshalb wir in einem ersten Durchgang die möglichen verschlüsselten Texte von "Verschlüsselung" und die möglichen entschlüsselten Texte des Schlüsseltextes in Dateien gespeichert haben, um sie in den folgenden Durchläufen lediglich einlesen und nicht neu generieren zu müssen. Dies brachte, bei den eingeschränkten Schlüsselmengen, eine erhebliche Zeitersparnis und einen annehmbaren Platzverbrauch.

5. Timing-Angriff auf Passwörter (Bonusaufgabe)

Bei einem Timing-Angriff wird die unterschiedliche Überprüfungszeit ausgenutzt, die für teilweise richtige oder falsche Passwörter gebraucht wird. In dem gegebenen Programm wird zunächst überprüft, ob die Länge des zu überprüfenden Passworts mit dem Korrekten übereinstimmt. Bei einem Passwort der richtigen Länge, benötigt das Programm so eine leicht höhere Ausführungszeit. Durch tausendfache Wiederholung, kann diese zu einem merkbaren Ergebnis aufaddiert werden. Folgend wird nach jedem falschen Zeichen abgebrochen. Auch hier kann mit dem Timing-Angriff angesetzt

werden. Ist bei der Iteration durch alle möglichen Zeichen an der ersten Stelle das Richtige gefunden, so wird die Überprüfung wieder etwas mehr Zeit benötigen. Auf diese Weise muss lediglich Stelle für Stelle überprüft werden, anstatt jegliche Kombination der Stellen.

Unser C-Programm konnte bis zum Abgabeschluss leider nicht bis zum endgültigen Knacken des Passworts implementiert werden. Die bisherige Version liegt in Anhang G.

Bachelor-Projekt
Network-Security-Übungsblatt 6

Mader, Burmester · SoSe 2016

A. SSL-TCP-Chat (Client und Server)

```
import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocket;
import javax.swing.*;
import java.io.*;
import java.net.*;
import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

class SSLSocketServer extends JFrame implements ActionListener {
  JButton button;
  JLabel label = new JLabel("Text received over socket:");
  JPanel panel;
  JTextArea textArea = new JTextArea();
  SSLServerSocket server = null;
  SSLSocket client;
  List<PrintWriter> outs;
  String line;
  Map<InetAddress, Long> _clientAddresses;

  SSLSocketServer() { // Begin Constructor
    button = new JButton("Click Me");
    button.addActionListener(this);
    panel = new JPanel();
    panel.setLayout(new BorderLayout());
    panel.setBackground(Color.white);
    getContentPane().add(panel);
    panel.add("North", label);
    panel.add("Center", textArea);
    panel.add("South", button);
    outs = new ArrayList<PrintWriter>();
    _clientAddresses = new HashMap<InetAddress, Long>();
  } // End Constructor
```

```
public void actionPerformed(ActionEvent event) {
  Object source = event.getSource();
  if (source == button) {
    textArea.setText(line);
  }
}

public void listenSocket() {
  try {
    // server = new ServerSocket(4444);
    SSLServerSocketFactory sslserversocketfactory =
       (SSLServerSocketFactory) SSLServerSocketFactory
        .getDefault();
    SSLServerSocket sslserversocket = (SSLServerSocket)
       sslserversocketfactory
        .createServerSocket(4444);
    // SSLSocket sslsocket = (SSLSocket)
       sslserversocket.accept();
    server = sslserversocket;
  } catch (IOException e) {
    System.out.println("Could not listen on port 4444");
    System.exit(-1);
  }
  while (true) {
    try {
      client = (SSLSocket) server.accept();
      Thread t = new listener(client, outs);
      t.start();
    } catch (IOException e) {
      System.out.println("Accept failed: 4444");
      System.exit(-1);
    }
  }

}

private class listener extends Thread {
  Socket client;
  BufferedReader in = null;
  List<PrintWriter> outs;
  String line;

  public listener(Socket client, List<PrintWriter> outs) {
    this.client = client;
    this.outs = outs;
  }
```

```
    @Override
    public void run() {
      PrintWriter out = null;
      try {
        in = new BufferedReader(new InputStreamReader(client
            .getInputStream()));
        out = new PrintWriter(client.getOutputStream(), true);

        outs.add(out);
      } catch (IOException e) {
        System.out.println("Accept failed: 4444");
        return;
      }
      while (true) {
        try {
          line = in.readLine();
          // Send data back to client
          for (PrintWriter o : outs) {
            if (!out.equals(o)) {
              if (line != null) {
                o.println(line);
              } else {
                break;
              }
            }
          }
        } catch (IOException e) {
          System.out.println("Read failed");
          System.exit(-1);
        }
      }
    }

    @Override
    protected void finalize() throws Throwable {
      in.close();
      client.close();
    }
  }

  protected void finalize() {
    // Clean up
    try {
      server.close();
    } catch (IOException e) {
```

```
      System.out.println("Could not close.");
      System.exit(-1);
    }
  }

  public static void main(String[] args) {
    SSLSocketServer frame = new SSLSocketServer();
    frame.setTitle("Server Program");
    WindowListener l = new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    };
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);
    frame.listenSocket();
  }
}


//Begin SSLSocketClient


import java.awt.Color;
import java.awt.BorderLayout;
import java.awt.event.*;
import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import javax.swing.*;
import java.io.*;
import java.net.*;

class SSLSocketClient extends JFrame implements ActionListener {
  JTextArea text;
  JLabel clicked;
  JButton button;
  JPanel panel;
  JTextField textField;
  JLabel history;
  SSLSocket socket = null;
  PrintWriter out = null;
  BufferedReader in = null;

  SSLSocketClient() { // Begin Constructor
    text = new JTextArea("");
```

```
    text.setEditable(false);
    textField = new JTextField(20);
    button = new JButton("Click Me");
    button.addActionListener(this);
    panel = new JPanel();
    panel.setLayout(new BorderLayout());
    panel.setBackground(Color.white);
    getContentPane().add(panel);
    panel.add("North", text);
    panel.add("Center", textField);
    panel.add("South", button);
  } // End Constructor

  public void actionPerformed(ActionEvent event) {
    Object source = event.getSource();
    Thread t = new Listener();
    t.start();
    if (source == button) {
      // Send data over socket
      String message = textField.getText();
      out.println(message);
      textField.setText(new String(""));
      text.append("You: " + message + "\n");
      // Receive text from server
    }
  }

  private class Listener extends Thread {
    @Override
    public void run() {
      while (true) {
        try {
          String line = in.readLine();
          text.append("Other: " + line + "\n");
          System.out.println("Text received: " + line);
        } catch (IOException e) {
          System.out.println("Read failed");
          System.exit(1);
        }
      }
    }
  }

  public void listenSocket() {
    // Create socket connection
    try {
```

```
      // socket = new Socket("kq6py", 4444);
      // SSLsocket = new Socket("localhost", 4444);
      SSLSocketFactory sslsocketfactory = (SSLSocketFactory)
         SSLSocketFactory
          .getDefault();
      SSLSocket sslsocket = (SSLSocket)
         sslsocketfactory.createSocket(
          "localhost", 4444);
      socket = sslsocket;
      out = new PrintWriter(socket.getOutputStream(), true);
      in = new BufferedReader(new InputStreamReader(socket
          .getInputStream()));
    } catch (UnknownHostException e) {
      System.out.println("Unknown host: kq6py.eng");
      System.exit(1);
    } catch (IOException e) {
      System.out.println("No I/O");
      System.exit(1);
    }
  }

  public static void main(String[] args) {
    SSLSocketClient frame = new SSLSocketClient();
    frame.setTitle("Client Program");
    WindowListener l = new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    };
    frame.addWindowListener(l);
    frame.pack();
    frame.setVisible(true);
    frame.listenSocket();
  }
}
```

B. Ausschnitt aus SSLStrip Logfile

```
2016-07-05 18:04:09,076 SECURE POST Data (www.amazon.de):
appActionToken=rIVbEovvj2FWgK213k9eGIQtyMR3Ej3D&appAction=
SIGNIN&openid.pape.max_auth_age=ape%3AMA%3D
%3D&openid.return_to=ape%3AaHR0cHM6Ly93d3cuYW1hem9uLmRlLz
9pZT1VVEY4JnJlZl89bmF2X3lhX3NpZ25pbg%3D
%3D&prevRID=ape%3AQlQ3N1hYV0c4U1lCODhDQkFNVE0%3D&openid.
```

```
identity=ape
%3AaHR0cDovL3NwZWNzLm9wZW5pZC5uZXQvYXV0aC8yLjAvaWRlbnRpZ
mllcl9zZWxlY3Q%3D&openid.assoc_handle=ape
%3AZGVmbGV4&openid.mode=ape%3AY2hlY2tpZF9zZXR1cA%3D%3D
&openid.ns.pape=ape
%3AaHR0cDovL3NwZWNzLm9wZW5pZC5uZXQvZXh0ZW5zaW9ucy9wYXBlLzEuMA
%3D%3D&openid.claimed_id=ape
%3AaHR0cDovL3NwZWNzLm9wZW5pZC5uZXQvYXV0aC8yLjAvaWRlbnRp
Zmllcl9zZWxlY3Q%3D&pageId=ape
%3AZGVmbGV4&openid.ns=ape
%3AaHR0cDovL3NwZWNzLm9wZW5pZC5uZXQvYXV0aC8yLjA
%3D&email=peter&create=0&password=user
```

C. baziCrack.py

```
import sys

f = open(sys.argv[1], 'r')
c = f.read().encode("hex")
k = sys.argv[2]
mult = int(len(c)/len(k))

for i in range(mult+1):
  k += k

nk = k[:len(c)]
print len(c)
print len(nk)

hx =  hex(int(c, 16) ^ int(nk, 16))
print "Length: " + str(len(hx[2:-1])) + "\n"
print hx
print "\n\n Ascii:\n"
print hx[2:-1].decode("hex")
```

D. advaziCrack.py

```
import sys

f = open(sys.argv[1], 'r')
c = f.read().encode("hex")
k = sys.argv[2]
pkc = sys.argv[3]
mult = int(len(c)/len(k))
pk = ""
```

```
ks = ""


for j in range(len(k)/2):
  pk += pkc

k = str(hex(int(k, 16) ^ int(pk, 16)))[2:-1]

for i in range(mult+1):
  ks += k

nk = ks[:len(c)]

hx =  hex(int(c, 16) ^ int(nk, 16))
print hx
print "\n\nAscii:\n"
print hx[2:-1].decode("hex")
```

E. MeetInTheMiddle.java

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.UnsupportedEncodingException;
import java.security.InvalidKeyException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Hashtable;
import java.util.Set;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.spec.SecretKeySpec;

import it.sauronsoftware.base64.Base64;

public class MeetInTheMiddle
{

    private static final byte[] KLARTEXT =
```

```
        "Verschluesselung".getBytes();
    private static final byte[] SCHLUESSELTEXT =
        hexStringToByteArray(
             "be393d39ca4e18f41fa9d88a9d47a574");
    private static final Hashtable<String, String> HASHTABLE =
        new Hashtable<String, String>();
    private static final Hashtable<String, String> DECRYPTTABLE =
        new Hashtable<String, String>();

    public static void main(String[] args)
            throws InterruptedException, IOException
    {
        long timeStart = System.currentTimeMillis();

        //fillHashFile(); //~50 sec.
        fillHashTableFromFile(); //~5 sec.

        long timeStop = System.currentTimeMillis();
        System.out.println("Verschlüsselung von " +
            HASHTABLE.size()
                 + " Einträgen in " + (timeStop - timeStart) + "ms
                    eingelesen");
        System.out.println("-----------------");

        timeStart = System.currentTimeMillis();

        //fillDecryptHashFile();//~50 sec.
        fillDecryptFromFile();//~30 sec.

        timeStop = System.currentTimeMillis();
        System.out.println("Entschlüsselung mit " +
            HASHTABLE.size()
                 + " Schlüsseln in " + (timeStop - timeStart) +
                    "ms eingelesen");
        System.out.println("-----------------");

        timeStart = System.currentTimeMillis();

        testHashSet();

        timeStop = System.currentTimeMillis();

        System.out.println("Prüfung in " + (timeStop - timeStart)
            + "ms");
        System.out.println("-----------------");
```

```
    }

    private static byte[] hexStringToByteArray(String s)
    {
        int len = s.length();
        byte[] data = new byte[len / 2];
        for (int i = 0; i < len; i += 2)
        {
            data[i / 2] = (byte) ((Character.digit(s.charAt(i),
               16) << 4)
                    + Character.digit(s.charAt(i + 1), 16));
        }
        return data;
    }

    private static void fillHashTableFromFile() throws IOException
    {
        FileReader fr = new FileReader("hashset");
        BufferedReader br = new BufferedReader(fr);

        String zeile;

        while ((zeile = br.readLine()) != null)
        {
            String[] line = zeile.split(":");

            if (line.length == 2)
            {
                HASHTABLE.put(line[1], line[0]);
            }
        }

        br.close();
    }

    private static void fillDecryptHashFile() throws IOException
    {
        Set<String> keys = HASHTABLE.keySet();

        FileWriter fw = new FileWriter("decrypt");
        BufferedWriter bw = new BufferedWriter(fw);

        for (String string : keys)
        {
            String key = HASHTABLE.get(string);
```

```
            bw.write(key + ":" + decrypt(key, SCHLUESSELTEXT) +
                "\n");
        }

        bw.close();
    }

    private static void fillDecryptFromFile() throws IOException
    {
        FileReader fr = new FileReader("decrypt");
        BufferedReader br = new BufferedReader(fr);

        String zeile;

        while ((zeile = br.readLine()) != null)
        {
            String[] line = zeile.split(":");

            if (line.length == 2)
            {
                DECRYPTTABLE.put(line[0], line[1]);
            }
        }

        br.close();
    }

    private static void testHashSet()
    {
        Set<String> keys = DECRYPTTABLE.keySet();

        for (String key : keys)
        {
            String s = DECRYPTTABLE.get(key);

            if (HASHTABLE.containsKey(s))
            {
                System.out
                    .println("Key1 : " + HASHTABLE.get(s) +
                        "\nKey2 : " + key);
                System.out.println("----------------");
                break;
            }
        }
    }
```

```
private static void fillHashFile() throws
   InterruptedException, IOException
{
    int byteLength = (16 * 16);

    ArrayList<KeyGenerator> key = new ArrayList<>();

    for (int x = 0; x < 16; x++)
    {
        key.add(new KeyGenerator());
    }

    FileWriter fw = new FileWriter("hashset");
    BufferedWriter bw = new BufferedWriter(fw);

    for (int positionOfFirst = 0; positionOfFirst < 15;
       positionOfFirst++)
    {
        for (int countFirst = 0; countFirst < byteLength;
           countFirst++)
        {
            for (int positionOfSecond = positionOfFirst
                    + 1; positionOfSecond < 16;
                       positionOfSecond++)
            {
                for (int countSecond = 0; countSecond <
                   byteLength; countSecond++)
                {
                    KeyGenerator actualSecondKG =
                       key.get(positionOfSecond);

                    String keyString =
                       keyGenListToString(key);

                    bw.write(keyString + ":" +
                       encrypt(keyString, KLARTEXT)
                            + "\n");

                    actualSecondKG.next(0);

                }

                key.get(positionOfSecond)
                    .setState("00");
            }
```

```
                key.get(positionOfFirst)
                    .next(0);

            }

            key.get(positionOfFirst)
                .setState("00");
        }
        bw.close();
    }

    private static String
       keyGenListToString(ArrayList<KeyGenerator> key)
    {

        String keyString = "";

        for (KeyGenerator keyGenerator : key)
        {
            keyString += keyGenerator.toString();
        }

        return new StringBuilder(keyString).reverse()
            .toString();
    }

    final protected static char[] hexArray =
       "0123456789ABCDEF".toCharArray();

    public static String byteArrayToHexString(byte[] bytes)
    {
        char[] hexChars = new char[bytes.length * 2];
        for (int j = 0; j < bytes.length; j++)
        {
            int v = bytes[j] & 0xFF;
            hexChars[j * 2] = hexArray[v >>> 4];
            hexChars[j * 2 + 1] = hexArray[v & 0x0F];
        }
        return new String(hexChars);
    }

    public static String encrypt(String key, byte[] value)
    {
        try
        {
            SecretKeySpec skeySpec = new SecretKeySpec(
```

```
                    hexStringToByteArray(key), "AES");

            Cipher cipher =
               Cipher.getInstance("AES/ECB/NoPadding");
            cipher.init(Cipher.ENCRYPT_MODE, skeySpec);

            byte[] encrypted = cipher.doFinal(value);

            return new String(Base64.encode(encrypted));

        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }

        return null;
    }

    public static String decrypt(String key, byte[] encrypted)
    {
        try
        {
            SecretKeySpec skeySpec = new SecretKeySpec(
                    hexStringToByteArray(key), "AES");

            Cipher cipher =
               Cipher.getInstance("AES/ECB/NoPadding");
            cipher.init(Cipher.DECRYPT_MODE, skeySpec);

            byte[] original = cipher.doFinal(encrypted);

            return new String(Base64.encode(original));
        }
        catch (Exception ex)
        {
            ex.printStackTrace();
        }

        return null;
    }

}
```

F. KeyGenerator.java

```
import java.util.ArrayList;
import java.util.List;

public class KeyGenerator
{

    private List<Character> _string;

    public KeyGenerator()
    {
        _string = new ArrayList<>();
        _string.add('0');
        _string.add('0');
    }

    public void next(int pos)
    {
        char c = _string.get(pos);

        if (c == '9')
        {
            _string.set(pos, 'a');
        }
        else if (c == 'f')
        {
            _string.set(pos, '0');

            if (_string.size() == pos + 1)
            {
                _string.add('0');
            }

            this.next(pos + 1);
        }

        else
        {
            _string.set(pos, (char) (_string.get(pos) + 1));
        }
    }
```

```
    @Override
    public String toString()
    {
        String ret = "";

        for (Character character : _string)
        {
            ret = character + ret;
        }

        return new StringBuilder(ret).reverse().toString();
    }

    public void setState(String s)
    {
        _string = new ArrayList<>();
        s = new StringBuilder(s).reverse()
            .toString();

        char[] charArray = s.toCharArray();

        for (char c : charArray)
        {
            _string.add(c);
        }

    }

}
```

G. TimeAttack.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>

#pragma clang diagnostic ignored "-Wunused-value"

const char pass[9] = "password";

/*
 * Given function to compare a given password with the
    "hardcoded" one.
```

```
 */

int password_compare(const char *password);
/*
 * Calls the password_compare function several times to get a
    somewhat compareable result.
 * TODO: Still very unreliable.
 */
double checker(const char *password, int loops) {
    int i = 0;
    clock_t begin = clock();
    for(i; i < loops; i++) {
        password_compare(password);
    }
    clock_t end = clock();

    return (end - begin);
}

/*
 * This function uses the checker to determine the next correct
    character of the password. It does so by calling the checker
    function with all the different possible characters at the
    first not known position in the password. The longest
    execution time is thought to be the longest correct start
    which is therefore the result. There is no more logic for
    determining if it is correct.
 *
 * @param len: The length of the password that should be checked.
 * @param known: The number of characters of the supplied
    password that we already know to be correct.
 * @param currPass: Pointer to the char array where the current
    password is. The same char array will hold the new password
    after execution.
 *
 * @return The execution time of the password with the longest
    part we think to be correct.
 */
double supplyPass(int len, int known, char *currPass, int loops) {
    double maxTime = 0;

    int i = 32;
    int j = 126;
    int extra[6] = {129, 132, 142, 148, 153, 154};
    for(i; i <= j; i++){
        char pass[len];
```

```
        if(known == 0) {
            memset(pass, i, len*sizeof(char));
        } else {
            strcpy(pass, currPass);
            pass[known] = i;
        }
        double x = checker(pass, loops);
        if(x > maxTime){
            maxTime = x;
            strcpy(currPass, pass);
        }
    }
    i = 0;
    for(i; i<6; i++){
        char pass[len];
        if(known == 0) {
            memset(pass, extra[i], len*sizeof(char));
        } else {
            strcpy(pass, currPass);
            pass[known] = i;
        }
        double x = checker(pass, loops);
        if(x > maxTime){
            maxTime = x;
            strcpy(currPass, pass);
        }
    }

    return maxTime;
}

/**
 *  Calculates the length of the stored password using the
    supplyPass function.
 *
 *  @param maxLength The maximum length of password that should
    be tried.
 *  @param pass The variable to hold the password. Afterwards a
    password with the correct length will be stored inside.
 *  @param loops The times each password length should be tried.
 *
 *  @return Length of the password
 */
int calcLenth(int maxLength, char *pass, int loops) {
    int i = 2;
    int maxTime = 0;
```

```
    int length = 1;
    for(i; i <= maxLength+1; i++) {
        pass = (char *) malloc(i);
        int time = supplyPass(i-1, 0, pass, loops);
        free(pass);
        if(time > maxTime) {
            maxTime = time;
            length = i-1;
        }
    }

    pass = (char *) malloc(length+1);
    return length;
}

int main(int argc, char *argv[]) {
    char *pass;
    pass = (char *) malloc(9);

    int loops;
    sscanf(argv[1], "%d", &loops);
    loops = 1000;

    memset(pass, 'p', 8*sizeof(char));
    int length = calcLenth(8, pass, loops);
    int x = 1;
    for (x; x < length; x++) {
        supplyPass(8, 1, pass, loops);
    }

    printf("String: %s\n", pass);
    free(pass);
    return 0;
}
```