SVS Bachelor-Projekt Network Security

Blatt 6: Kryptographie

Louis Kobras
6658699

Utz Pöhlmann
6663579

1 Absicherung des TCP-Chats mit SSL

Anmerkung: Bei dieser Aufgabe haben wir uns Hilfe von `SS16G06` (Andre, Katharina) geholt, da wir nach wie vor keine funktionierende Chat-Implementierung hatten.

- Erstellen von Keys und Zertifikaten nach [2]
- I/O-Dialog vgl. [keytool-Dialog (S. 4)]
- Sourcecode für Client und Server siehe ebenda.

Es wird ein Zertifikat für den Public Key des Servers benötigt. Dieses wurde mit `keytool` erstellt.

Es wird ein Zertifikat für die Verbindung des Clients zum Server benötigt. Dieses wurde mit `openssl` erstellt.

2 CAs und Webserver-Zertifikate

2.2 Selbstsignierte Zertifikate

Es wurde in mehreren Läufen die Fallstudie durchgearbeitet. Und zwar mehrmals und sowohl zusammen als auch einzeln und unabhängig voneinander. Mit dem Ergebnis, dass der Apache2-Server nicht funktioniert. Die Gruppe neben uns, denen wir inzwsichen bestimmt mega auf den Keks gehen und denen ich als Wiedergutmachung ein Eis mitgebracht habe, konnte uns leider auch nicht helfen. Unsere certs und pems und reqs und keys und csrs wurden alle ordnungsgemäß erstellt und Schritt für Schritt, Wort für Wort nach der Fallstudie erzeugt und bearbeitet. Es ist kaputt. Selbst Reboots und Reinstallationen helfen nicht. Folglich funktionieren Aufgabe 2.1ff nicht. Mal wieder. Sind wir echt so blöd oder liegt vielleicht ein Fehler auf unserer Maschine vor?

2.3 HTTPS-Weiterleitung

folgefehlend.

2.4 sslstrip

sslstrip wurde nach [1] installiert und gestartet.
Die Verbindung wurde am "(svs.informatik.uni-hamburg.de)" in der Logdatei erkannt. (IP: 134.100.15.55)
Die Browsereinstellungen wurde unter `Edit` → `Preferences` → `Advanced` → `Network` → `Connection` → `Settings...` auf `localhost` und Port `8080` gesetzt. Zudem wurden die `No Proxy for`-Einstellungen entfernt.
Der Inhalt der Datei: vgl. [ssllog (S. 9)] Die Lösung aus Aufgabe 2.3 ist somit definitiv ein Plus an Sicherheit. Der Sinn von HSTS ist, sich vor sog. "downgrade Attacks" zu schützen. Hierbei wird der Client dazu gezwungen, statt einer "modernen" sicheren Verbindung eine "alte" unsichere Verbindung aufzubauen. (Bsp.: HTTP statt HTTPS) Ein weiterer Nutzen ist, "Session Hijacking" zu unterbinden. Hier wird ein Authentifikationscookie abgefangen und so ein Man-In-The-Middle-Angriff gestartet. Da HSTS Webservern erlaubt, auf Browser den Zwang einer sicheren Verbindung (via HTTPS) auszuüben, sind alle Server, die diese Möglichkeit nutzen, auch sicher vor SSL-Stripping-Angriffen.

3 Unsichere selbstentwickelte Verschlüsselungsalgorithmen

3.1 BaziCrypt

Quellcode siehe [Knacking BaziCrypt (S. 10)].

Das Programm nimmt die zu entschlüsselnden Dateien als Parameter entgegen:

```
$ python bazidec.py n01.txt.enc n02.txt.enc n03.txt.enc
message 1: Hallo Peter. Endlich koennen wir geheim kommunizieren! Bis bald, Max
message 2: Hi Max! Super, Sicherheitsbewusstsein ist ja extrem wichtig! Schoene Gruesse, P
eter.
message 3: Hi Peter, hast du einen Geheimtipp fuer ein gutes Buch fuer mich? Gruss, Max

Process finished with exit code 0
```

3.2 AdvaziCrypt - Denksport

Beim PKCS7-Padding wird jede zu verschlüsselnde Nachricht mit Länge L_i auf eine konstante Länge L_{max} aufgestockt.

Die Pading-Bytes sind allerdings nicht 0-Bytes, wie bei Bazi-Crypt, sondern werden aus der Differenz von L_{max} und L_i berechnet.

Beispiel:
$L_max - L_i = 1$
Pading-Bytes: 0x01
$L_max - L_i = 2$
Pading-Bytes: 0x0202
$L_max - L_i = 3$
Pading-Bytes: 0x030303
$L_max - L_i = 4$
Pading-Bytes: 0x04040404
$L_max - L_i = 5$
Pading-Bytes: 0x0505050505
usw.

Es gibt insgesamt $0xFF = 16 \cdot 16 = 2^4 \cdot 2^4 = 2^8 = 256$ verschiedene mögliche Padding-Bytes, sofern L_{max} lang genug gewählt wurde.

3.3 AdvaziCrypt - Angriff implementieren

Quellcode siehe [advazicrypt-Angriff (S. 12)].

Auch AdvaziDec nimmt die Dateien als Parameter entgegen. Leider wird derzeit bei einem Aufruf von AdvaziDec auch BaziDec ausgeführt, weswegen zuerst alle Nachrichten einmal (fehlerhaft) mit BaziDec entschlüsselt ausgedruckt werden, bevor die korrekte Übersetzung erfolgt.

```
$ python advazidec.py n04.txt.enc n05.txt.enc n06.txt.enc
message 1: Hi Max, natuerlich: Kryptologie von A. Beutelspacher ist super. Gruss Peter
message 2: Hi Peter, worum geht es in dem Buch? Ciao, Max.
message 3: Hi Max, das ist ein super Buch, das viele Krypto-Themen abdeckt. Gruss Peter

Process finished with exit code 0
```

Die BaziDec-Nachrichten wurden der Leserlichkeit halber ausgelassen. Zusätzlich ist zu sagen, dass eigentlich die Nachricht aufgrund der Natur von AdvaziCrypt mit einem beliebigen Zeichen aufgefüllt wird, bis die Standardlänge erreicht ist. Da LaTeX diese Zeichen u.U. nicht codieren kann, wurden auch sie hier weggelassen.

Fazit: Max und Peter freuen sich über (pseudo)sichere Kommunikation. Vielleicht sollten sie das Buch tatsächlich mal lesen, über das sie reden, dann fällt ihnen auf, wie leicht Bazi und Advazi zu knacken sind.

4 EasyAES

Quellcode für diese Aufgabe vgl. [EasyAES (S. 13)].

Leider hatten wir keine Zeit mehr, den Code vernünftig auszuprobieren. Wir haben nur begrenzt viel Rechenkapazität (mehr als 2GB RAM ist nicht drin) und damit hat es immer noch zu lange gedauert. Wir hatten bei dem Brute-Force Angriff auf Zettel 2 Aufgabe 2.2 ja auch schon Laufzeitschwierigkeiten...

Zum Starten die Klasse `AES.java` aufrufen.

5 Timing-Angriff auf Passwörter (Bonusaufgabe)

Quellcode für einen Timing-Angriff in Anlehnung an die gegebene Methode vgl. [Timing-Angriff (S. 19)]. Eine vertrauenswürdige Quelle hat uns gesagt, dass dieser Quellcode einen vollständigen Timing-Angriff durchführt.

Literatur

[1] `https://moxie.org/software/sslstrip/`

[2] `http://alvinalexander.com/java/java-keytool-keystore-certificates`

ANHANG

keytool-Dialog

IO-Dialog

```
pwd
# $(pwd) -> /home/apollo
# generate directory
mkdir PJNS
mkdir SSL-Server
cd PJNS/SSL-Server/
pwd
# $(pwd) -> /home/apollo/PJNS/SSL-Server
# generate keystore
keytool -genkey -alias server -keystore privateKeys.store
#### password: 'foobar'
#### first and last name: 'Louis Kobras'
#### orga unit: 'SVS'
#### organization: 'Uni Hamburg'
#### City: 'Hamburg'
#### State: 'Hamburg'
#### two-letter country code: 'DE'
#### correct: 'yes'
#### key password: '123xyz'
# generate temporary certificate
keytool -export -alias server -file cert.crt -keystore privateKeys.store
#### keystore password: 'foobar'
# import certificate in public keystore
keytool -import -alias public-server -file cert.crt -keystore publicKeys.
    store
#### store password: 'barfoo'
#### do i trust this?: 'yes'
# create openssl stuff
## server stuff
openssl genrsa -out server-ca.key 4096
openssl req -new -x509 -days 365 -key server-ca.key -out server-ca.crt
#### country code: 'DE'
#### state: 'Hamburg'
#### locale: 'Hamburg'
#### organization: 'Uni Hamburg'
#### unit: 'SVS'
#### common name: 'localhost'
#### email: '4kobras@inf...'
## client stuff
openssl genrsa -out client.key 4096
openssl -req -new -key client.key -out client.crt
openssl req -new -key client.key -out client.crt
# sign certificate
openssl x509 -req -days 365 -in client.crt -CA server-ca.crt -CAkey server-
    ca.key -set_serial 01 -out client-cert.crt
# switch into java workspace and open connection
cd /home/apollo/IntelliJProjects/FS4/src/PJNS/B06/A1/
openssl s_client -connect localhost:1337 -cert ~/PJNS/SSL-Server/client-
    cert.crt -key ~/PJNS/SSL-Server/client.key -CAfile ~/PJNS/SSL-Server/
    server-ca.crt
# start server program and connect to it with the client program
```

Server.java

```
1  package PJNS.B06.A1;
2
3  import java.awt.*;
4  import java.awt.event.*;
5  import javax.net.ssl.*;
6  import javax.net.ServerSocketFactory;
7  import javax.swing.*;
8  import java.util.*;
9  import java.util.stream.Collectors;
10 import java.util.concurrent.ConcurrentHashMap;
11 import java.io.IOException;
12
13 public class Server extends JFrame
14 {
15
16     private static final int PORT = 1337;
17     private final JTextArea textArea = new JTextArea();
18     private SSLServerSocket server = null;
19     private final Map<String, Client> clients = new ConcurrentHashMap<>();
20
21     public Server() throws HeadlessException
22     {
23         super();
24         final JPanel panel = new JPanel();
25         panel.setLayout(new BorderLayout());
26         panel.setBackground(Color.white);
27         getContentPane().add(panel);
28         // UI auf deutsch weil ist benutzerfreundlicher in Deutschland
29         panel.add("North", new JLabel("Empfangener Text:"));
30         panel.add("Center", textArea);
31     }
32
33     private void listenSocket()
34     {
35         try
36         {
37             final ServerSocketFactory socketFactory =
                   SSLServerSocketFactory.getDefault();
38             server = (SSLServerSocket) socketFactory.createServerSocket(
                   PORT);
39             server.setNeedClientAuth(true);
40         }
41         catch (IOException e)
42         {
43             // Fehlermeldungen auf Englisch weil ist Profi
44             System.out.println("Could not listen on port " + PORT);
45             System.exit(- 1);
46         }
47         while (true)
48         {
49             accept();
50         }
51     }
52
53     private void accept()
54     {
55         Client w;
```

```
56          try
57          {
58              w = new Client((SSLSocket) server.accept(), textArea);
59              Thread t = new Thread(w);
60              t.start();
61          }
62          catch (IOException e)
63          {
64              System.out.println("Accept␣failed:␣" + PORT);
65              System.exit(- 1);
66          }
67      }
68
69      protected void finalize() throws Throwable
70      {
71          super.finalize();
72          try
73          {
74              server.close();
75          }
76          catch (IOException e)
77          {
78              System.out.println("Could␣not␣close␣socket");
79              System.exit(- 1);
80          }
81      }
82
83      public void registerClient(final String clientName, final Client client
            )
84      {
85          clients.put(clientName, client);
86      }
87
88      protected boolean hasClient(final String clientName)
89      {
90          return clients.containsKey(clientName);
91      }
92
93      protected void broadcast(final String message)
94      {
95          final java.util.List<Client> clients = this.clients.entrySet()
96                                                             .stream()
97                                                             .map(Map.Entry::
                                                                 getValue)
98                                                             .collect(Collectors.
                                                                 toList());
99          for (final Client client : clients)
100         {
101             client.sendMessage(message);
102         }
103     }
104
105     public static void main(final String... args)
106     {
107         System.setProperty("javax.net.ssl.keyStore", "/Users/apollo/PJNS/
                SSL-Server/privateKeys.store");
108         System.setProperty("javax.net.ssl.keyStorePassword", "foobar");
109         System.setProperty("javax.net.ssl.trustStore", "/Users/apollo/PJNS/
                SSL-Server/publicKeys.store");
```

```
110         System.setProperty("javax.net.ssl.trustStorePassword", "barfoo");
111         Server socketServer = new Server();
112         socketServer.setTitle("Fancy␣SSL␣PJNS.B06.A1.Server");
113         WindowListener windowListener = new WindowAdapter()
114         {
115             public void windowClosing(WindowEvent e)
116             {
117                 System.exit(0);
118             }
119         };
120         socketServer.addWindowListener(windowListener);
121         socketServer.pack();
122         socketServer.setVisible(true);
123         socketServer.listenSocket();
124 
125     }
126 
127 }
```

Client.java

```
1  package PJNS.B06.A1;
2  
3  import java.io.*;
4  import java.util.logging.*;
5  import javax.swing.*;
6  import javax.naming.ldap.*;
7  import javax.naming.InvalidNameException;
8  import javax.net.ssl.*;
9  import javax.security.cert.X509Certificate;
10 
11 public class Client implements Runnable
12 {
13 
14     private static final Logger LOGGER = Logger.getLogger("PJNS.B06.A1.
       Client");
15     private SSLSocket _sslSocket;
16     private JTextArea _text;
17     private PrintWriter _pwOut = null;
18     private String _client;
19     private Server _server = new Server();
20 
21     public Client(SSLSocket socket, JTextArea textArea)
22     {
23         _sslSocket = socket;
24         _text = textArea;
25     }
26 
27     @Override
28     public void run()
29     {
30         BufferedReader brIn;
31         final SSLSession sslSession = _sslSocket.getSession();
32         if (sslSession.isValid())
33         {
34             try
35             {
36                 final InputStream is = _sslSocket.getInputStream();
```

```
37                  brIn = new BufferedReader(new InputStreamReader(is));
38                  _pwOut = new PrintWriter(_sslSocket.getOutputStream(), true
                        );
39
40                  getUsername(sslSession);
41                  stuff(brIn);
42              }
43              catch (IOException | InvalidNameException e)
44              {
45                  logError(e);
46              }
47          }
48      }
49
50      /**
51       * SOURCE: https://stackoverflow.com/questions/2914521/how-to-extract-
             cn-from-x509certificate-in-java
52       */
53      private void getUsername(SSLSession session) throws
            InvalidNameException
54      {
55          try
56          {
57              final X509Certificate[] x509Certificates = session.
                    getPeerCertificateChain();
58              final String name = x509Certificates[0].getSubjectDN().getName
                    ();
59              final LdapName lname = new LdapName(name);
60              for (final Rdn round : lname.getRdns())
61              {
62                  if (round.getType().equalsIgnoreCase("CN"))
63                  {
64                      _client = (String) round.getValue();
65                      logInfo("Username␣is:␣" + _client);
66                  }
67              }
68          }
69          catch (SSLPeerUnverifiedException e) {
70              logError(e);
71          }
72      }
73
74      private void stuff(BufferedReader brIn) throws IOException
75      {
76          String message;
77          if (_client != null && ! _client.isEmpty())
78          {
79              if (! _server.hasClient(_client))
80              {
81                  _server.registerClient(_client, this);
82                  while (true)
83                  {
84                      message = brIn.readLine();
85                      if (message != null)
86                      {
87                          broadcast(message);
88                      }
89                      else
90                      {
```

```
91                          break;
92                      }
93                  }
94              }
95              else
96              {
97                  logInfo("PJNS.B06.A1.Client␣already␣registered");
98              }
99          }
100     }
101
102     /**
103      * Log an Info message
104      *
105      * @param s Info to log
106      */
107     private void logInfo(String s)
108     {
109         LOGGER.log(Level.INFO, s);
110     }
111
112     /**
113      * Log an Error or an Exception as Severe
114      *
115      * @param e Error/Exception to log
116      */
117     private void logError(Exception e)
118     {
119         LOGGER.log(Level.SEVERE, "1", e);
120         System.exit(1);
121     }
122
123     /**
124      * Broadcasts a message to both yourself and the server
125      *
126      * @param msg the message to broadcast
127      */
128     private void broadcast(String msg)
129     {
130         _text.append(msg + "\n");
131         _server.broadcast(msg);
132     }
133
134     /**
135      * Sends a message to the output stream
136      *
137      * @param msg the message to send
138      */
139     public void sendMessage(final String msg)
140     {
141         _pwOut.println(msg);
142     }
143 }
```

ssllog

Anmerkung: Aufgrund der Länge der Zeilen wurden nachträglich manuell Zeilenumbrüche eingefügt.

```
2016-06-23 15:45:58,171 POST Data (safebrowsing.clients.google.com):
goog-malware-shavar;a:239451-244530:s
    :234828-234868,234872-234874,234876-234888,234890-234895,
     234897-234901,234903-234904,234906,234910-234915,234917-234927,
     234929-235103,235105-235168,235170-235176,235178-235198,235200-235256,
     235258-235274,235276-235307,235309-235473,235477,235479-235485,
     235487-235807,235809-235974,235976-236189,236191-236363,236365-236366,
     236368-237764,237766-238163,238165-238181,238183-238196,238198-239044,
     239046-239996:mac
goog-phish-shavar;a:448570-450992:s
    :268799-268858,268860-269092,269096-269182,
     269184-269212,269214-269222,269224-269265,269267-269291,269293-269295,
     269297-269311,269313,269316-269347,269349-269351,269353-269356,
     269359-269360,269362-269368,269370,269390-269392,269408-269513,
     269515-269518,269521-269575,269577-269606,269608-269615,269617,
     269619-269628,269630-269694,269696-269734,269736-269843,269845,
     269847-269902,269904-269926,269928-269937,269939-269995,269997-270023,
     270025-270092,270094-270115,270117-270151,270153-270163,270165-270190,
     270192-270205,270207-270234,270236,270238-270275,270277-270374,
     270376-270402,270404-271027,271030-271036,271038-271049,271051-271118,
     271120-271194,271196-271212,271214-271227,271229-271391,271393-271394,
     271398-271434,271436-271442,271444-271449,271452-271472,271475-271526,
     271528-271595:mac

2016-06-23 15:47:15,606 SECURE POST Data (svs.informatik.uni-hamburg.de):
username=admin&password=password
```

Knacking BaziCrypt

```
import sys


def task_three_point_one(msg):
    """
␣␣␣␣task_three_point_two␣function␣of␣the␣program.␣takes␣a␣HEX-encrypted␣
    message␣and␣decrypts␣it
␣␣␣␣:param␣msg:␣the␣message␣to␣decrypt
␣␣␣␣:return:␣the␣decrypted␣message
␣␣␣␣"""
    # find the key that was used to encrypt
    """
␣␣␣␣absuses␣the␣nature␣of␣the␣encryption␣method␣used␣by␣taking␣the␣last␣10␣
    chars␣of␣the␣message,␣which␣are␣the␣key
␣␣␣␣"""
    key = msg[-20:]
    return get_message(key, msg)


def get_message(key, msg):
    # decrypt the message using the found key
    """
␣␣␣␣␣␣␣␣due␣to␣the␣nature␣of␣the␣encryption␣method␣used,t␣he␣message,␣being
    ␣symmetrically␣encrypted,
␣␣␣␣␣␣␣␣can␣be␣decrypted␣by␣XORing␣the␣message␣with␣the␣encryption␣key.
␣␣␣␣␣␣␣␣So,␣bring␣the␣key␣to␣the␣same␣length␣as␣the␣message␣and␣xor␣them
␣␣␣␣␣␣␣␣"""
    key *= 10  # takes advantage of the fact that each key had a length of
        10 chars
```

```
26      # and each message had a length of 100 chars, setting the length factor
            to 10
27      msg = xor(msg, key)
28      msg = get_chars(msg)
29      return ''.join(msg)
30
31
32  def xor(first_string, second_string):
33      """
34  ␣␣␣␣given␣two␣HEX␣strings,␣uses␣the␣XOR␣function␣between␣them␣by␣converting
        ␣them␣to␣lists␣of␣integers
35  ␣␣␣␣:param␣first_string:
36  ␣␣␣␣:param␣second_string:
37  ␣␣␣␣:return:␣the␣result␣of␣XOR
38  ␣␣␣␣"""
39      # convert the first HEX string to integers
40      first_list = []
41      for c in first_string:
42          b = int(c, 16)
43          first_list.append(b)
44      # converts the second HEX string to integers
45      second_list = []
46      for c in second_string:
47          b = int(c, 16)
48          second_list.append(b)
49      result = []
50      # XORs the lists
51      for i in range(0, len(first_list)):
52          j = first_list[i] ^ second_list[i]
53          j = hex(j)  # converts the XORd integer to a HEX
54          j = j[2:]  # cut the HEX '0x' notation given by the built-in
                function cast hex(1)
55          result.append(j)
56      result = ''.join(result)  # joins the XORd list to a string
57      return result
58
59
60  def get_chars(lst):
61      """
62  ␣␣␣␣given␣a␣list␣of␣HEX␣values,␣this␣function␣returns␣the␣char␣values␣of␣
        each␣position
63  ␣␣␣␣:param␣lst:␣a␣list␣of␣hex␣values
64  ␣␣␣␣:return:␣a␣list␣of␣char␣values␣as␣integer
65  ␣␣␣␣"""
66      res = []
67      for i in xrange(0, len(lst) - 1, 2):
68          a = int(lst[i], 16)  # get first place as int
69          a *= 16  # multiply first place by 16
70          b = int(lst[i + 1], 16)  # get second place as int
71          b += a  # add values together to get ASCII char code
72          res.append(chr(b))
73      return res
74
75
76  for j in range(1, len(sys.argv)):
77      f = open(sys.argv[j], 'r')
78      f = f.read()
79      f = f.encode('hex')
80      f = task_three_point_one(f)
```

```
81      print "message␣%d:␣%s" % (j, f)
```

advazicrypt-Angriff

```
1  from bazidec import xor
2  from bazidec import get_message
3
4
5  def key(i):
6      if i < 256:
7          appendix = hex(i)[2:]
8          if len(appendix) < 2:
9              appendix = '0%s' % appendix
10     dec = '0x%s' % (appendix * i)
11     return dec
12
13
14 def task_three_point_three(msg, padding):
15     """
16 ␣␣␣␣task_three_point_two␣function␣of␣the␣program.␣takes␣a␣HEX-encrypted␣
       message␣and␣decrypts␣it
17 ␣␣␣␣:param␣msg:␣the␣message␣to␣decrypt
18 ␣␣␣␣:return:␣the␣decrypted␣message
19 ␣␣␣␣"""
20     # find the key that was used to encrypt
21     """
22 ␣␣␣␣absuses␣the␣nature␣of␣the␣encryption␣method␣used␣by␣taking␣the␣last␣10␣
       chars␣of␣the␣message,␣which␣are␣the␣key
23 ␣␣␣␣"""
24     message_bytes = msg[-20:]
25     padding_bytes = padding[-20:]
26     key = xor(message_bytes, padding_bytes)
27     return get_message(key, msg)
28
29
30 min_range = 10
31 max_range = 100
32 """
33 We␣can't␣get␣any␣intel␣about␣the␣first␣part␣of␣the␣key␣if␣the␣padding␣is␣
       smaller␣than␣the␣key␣length
34 with␣only␣part␣of␣the␣key,␣we␣can␣only␣decrypt␣parts␣of␣the␣message␣in␣
       periodic␣intervals
35 thus␣we␣start␣with␣a␣length␣of␣10,␣which␣is␣required␣for␣a␣complete␣key
36 the␣max␣range␣is␣the␣length␣of␣each␣message␣(all␣of␣them␣are␣100␣bytes␣long
       )
37 to␣improve␣runtime,␣switch␣the␣for␣loops␣(the␣answers␣then␣have␣to␣be␣
       sorted␣manually)
38 """
39 msgs = []
40 for j in range(1, len(sys.argv)):
41     for key_iterator in range(min_range, max_range):
42         padding_bytes = key(key_iterator)[2:]
43         msg = open(sys.argv[j], 'r').read().encode('hex')
44         str = task_three_point_three(msg, padding_bytes)
45         if str.__contains__("Max") or str.__contains__("Peter"):
46             print "message␣%d:␣%s" % (j, str)
```

EasyAES

SchluesselVersuchZwei.java

```
1  package B06.A4;
2
3  import java.util.ArrayList;
4
5  public class SchluesselVersuchZwei
6  {
7      private ArrayList<Key> key;
8
9      public SchluesselVersuchZwei()
10     {
11         key = generiereAlleSchluessel();
12         System.out.println(key); //TODO: wegen debug
13     }
14
15     public static void main(String[] args)
16     {
17         new SchluesselVersuchZwei();
18     }
19
20     public ArrayList<Key> getKeys()
21     {
22         return key;
23     }
24
25     private ArrayList<Key> generiereAlleSchluessel()
26     {
27         ArrayList<Key> schluesselListe = new ArrayList<Key>();
28         Key schluessel = new Key();
29         schluesselListe.add(schluessel);//generiere schlüssel für 0
               nichtNullBytes
30         ArrayList<Key> schluesselListe1 = generiereSchluessel1NNB();//
               generiere schlüssel für 1 nichtNullByte
31         ArrayList<Key> schluesselListe2 = generiereSchluessel2NNB();//
               generiere schlüssel für 2 nichtNullBytes
32         schluesselListe.addAll(schluesselListe1);
33         schluesselListe.addAll(schluesselListe2);
34         return schluesselListe;
35     }
36
37     private ArrayList<Key> generiereSchluessel1NNB()
38     {
39         ArrayList<Key> schluesselListe = new ArrayList<Key>();
40         for (int index = 0; index < 16; index++) //Für jedes Feld (0-15)
41         {
42             Key schluessel = new Key();
43             for (int wert = 1; wert < 256; wert++) //Für jeden Schlü
                   sselwert (0x00 - 0xff)
44             {
45                 schluessel.setKey(index, wert); //Gib Ergebnis
46                 schluesselListe.add(schluessel);
47             }
48
49         }
50         return schluesselListe;
51     }
```

```
52
53      private ArrayList<Key> generiereSchluessel2NNB()
54      {
55          ArrayList<Key> schluesselListe = new ArrayList<Key>();
56          for (int index1 = 0; index1 < 16; index1++) //Für jedes Feld (0-14)
57          {
58              for (int wert1 = 1; wert1 < 256; wert1++) //Für jeden Schlü
                    sselwert (0x00 - 0xff)
59              {
60                  for (int index2 = index1+1; index2 < 16; index2++) //Für
                        jedes Feld (0-15)
61                  {
62                      for (int wert2 = 1; wert2 < 256; wert2++) //Für jeden
                            Schlüsselwert (0x00 - 0xff)
63                      {
64                          Key schluessel = new Key();
65                          schluessel.setKey(index1, wert1); //Gib Ergebnis
66                          schluessel.setKey(index2, wert2); //Gib Ergebnis
67                          schluesselListe.add(schluessel);
68                      }
69
70                  }
71              }
72
73          }
74
75          return schluesselListe;
76      }
77
78  }
```

Key.java

```
1   package B06.A4;
2
3   public class Key
4   {
5
6       private Hex[] _key;
7
8       public Key()
9       {
10          _key = new Hex[16];
11          for (int index = 0; index < 16; index++)
12          {
13              _key[index] = new Hex(0);
14          }
15      }
16
17      public static void main(String args[])
18      {
19          int[] werte = new int[16];
20          for(int i=0; i<werte.length; i++){
21              werte[i] = i;
22          }
23          System.out.println(new Key(werte).toString());
24      }
25
```

```
26      public Key(int[] werte)
27      {
28          assert werte.length == 16;
29          _key = new Hex[16];
30          for (int index = 0; index < 16; index++)
31          {
32              _key[index] = new Hex(werte[index]);
33          }
34      }
35
36      public void setKey(int index, int wert)
37      {
38          assert index < 16;
39          assert index > -1;
40          assert wert < 256;
41          assert wert > -1;
42          _key[index] = new Hex(wert);
43      }
44
45      public Hex getWert(int index)
46      {
47          assert index < 16;
48          assert index > -1;
49          return _key[index];
50      }
51
52      @Override
53      public String toString()
54      {
55          StringBuilder sb = new StringBuilder();
56          for (int index = 0; index < _key.length-1; index++)
57     {
58              sb.append(_key[index]);
59              sb.append(":");
60          }
61          sb.append(_key[_key.length-1]);
62
63          return sb.toString();
64      }
65
66  }
```

Hex.java

```
1   package B06.A4;
2
3   public class Hex
4   {
5       int _wert;
6       /**
7        * Es wird davon ausgegangen dass nur Werte zwischen 0 und 255
             eigegeben werden.
8        *
9        * @param wert
10       */
11      public Hex(int wert){
12          assert wert < 256;
13          assert wert > -1;
```

```
14          _wert = wert;
15      }
16
17      public static void main(String args[])
18      {
19          System.out.println(new Hex(42).toString());
20      }
21
22      @Override
23      public String toString(){
24          if(_wert < 16) return "0".concat(Integer.toHexString(_wert));
25          else return Integer.toHexString(_wert);
26      }
27  }
```

AES.java

```
1   package B06.A4;
2
3   import java.security.MessageDigest;
4   import java.util.ArrayList;
5   import java.util.Arrays;
6
7   import javax.crypto.Cipher;
8   import javax.crypto.spec.SecretKeySpec;
9
10  import sun.misc.BASE64Decoder;
11  import sun.misc.BASE64Encoder;
12
13  /**
14   * Eine Klasse, die eine doppelte AES-Verschlüsselung durch einen Meet-In-
         The-Middle-Angriff knackt
15   * gegeben einen Klartext und einen verschluesselten Schluesseltext
16   *
17   *
18   * @author Alexander Gräsel, Louis Kobras, Utz Pöhlmann
19   * Quelle: http://blog.axxg.de/java-aes-verschluesselung-mit-beispiel/
20   */
21  public class AES
22  {
23
24      private static final String _klartext = "Verschluesselung"; //Der
            Klartext
25      private final static String _schluesseltext = "
            be393d39ca4e18f41fa9d88a9d47a574"; //Der doppelt verschlüsselte
            Klartext
26      private SchluesselVersuchZwei _generator;
27
28      /**
29       * Versucht, die beiden Schlüssel E_k1 und E_k2 heauszufinden mit
30       * _schluesseltext = E_k2(E_k1(_klartext))
31       * und scheibt sie auf die Konsole
32       *
33       * @param args
34       * @throws Exception
35       */
36      public static void main(String[] args) throws Exception
37      {
```

```
38          AES aes = new AES();
39          ArrayList<String> gefundeneSchluessel = aes.findeSchluessel(
                _klartext,
40                  _schluesseltext);
41          System.out.println(gefundeneSchluessel);
42      }
43
44      /**
45       * Versucht, die beiden Schlüssel E_k1 und E_k2 heauszufinden mit
46       * _schluesseltext = E_k2(E_k1(_klartext))
47       * durch einen Meet-In-The-Middle-Angriff
48       *
49       * @param klartext
50       * @param schluesseltext
51       * @return eine ArrayList von Typ String mit folgenden Elementen:
52       *      "Folgende Schluessel wurden gefunden:"
53       *      "E_k1: " + schluessel1.toString()
54       *      "E_k2: " + schluessel2.toString()
55       * @throws Exception
56       */
57      public ArrayList<String> findeSchluessel(String klartext,
58              String schluesselText)
59      {
60          String textStufeEins1;
61          String textStufeEins2;
62          ArrayList<String> ergebnis = new ArrayList<String>();
63          try
64          {
65              ArrayList<SecretKeySpec> schluesselListe = generiereSchluessel
                    ();
66              System.out.println(schluesselListe);
67              for (SecretKeySpec schluesselEnc : schluesselListe)
68              {
69                  textStufeEins1 = verschluesseln(klartext, schluesselEnc);
70                  for (SecretKeySpec schluesselDec : schluesselListe)
71                  {
72                      textStufeEins2 = entschluesseln(schluesselText,
73                              schluesselDec);
74                      System.out.println(textStufeEins1 + ":" +
                            textStufeEins2);
75                      if (textStufeEins1.equals(textStufeEins2))
76                      {
77                          ergebnis.add("Folgende␣Schluessel␣wurden␣gefunden:"
                                );
78                          ergebnis.add("E_k1:␣" + schluesselEnc.toString());
79                          ergebnis.add("E_k2:␣" + schluesselDec.toString());
80                      }
81                  }
82              }
83          }
84          catch (Exception e)
85          {
86              System.out.println(e.getMessage());
87          }
88          return ergebnis;
89      }
90
91      /**
```

```
92      * Generiert alle möglichen Schlüssel, auf die die Vorgaben (16 Byte,
            maximal an 2 Stellen keine 0-Bytes) zutreffen
93      *
94      *  @return die Schluessel Liste als Liste von SecretKeySpec
95      * @throws Exception
96      */
97     private ArrayList<SecretKeySpec> generiereSchluessel() throws Exception
98     {
99         ArrayList<SecretKeySpec> raffinierteSchluesselListe = new ArrayList
               <SecretKeySpec>();
100        _generator = new SchluesselVersuchZwei();
101        ArrayList<Key> schluesselListe = _generator.getKeys();
102        for (Key key : schluesselListe)
103        {
104            SecretKeySpec schluessel = bereiteSchluesselVor(key);
105            raffinierteSchluesselListe.add(schluessel);
106        }
107        return raffinierteSchluesselListe;
108    }
109
110    /**
111     * Bringt einen Schluessel in ein Format mit dem das Programm weiter
            arbeiten kann (SecretKeySpec)
112     *
113     * @param schluessel
114     * @return der fertige Schluessel als SecretKeySpec
115     * @throws Exception
116     */
117    private SecretKeySpec bereiteSchluesselVor(Key schluessel) throws
           Exception
118    {
119        // Das Passwort bzw der Schluesseltext
120        String schluesselString = schluessel.toString();
121        // byte-Array erzeugen
122        byte[] key = (schluesselString).getBytes("UTF-8");
123        // aus dem Array einen Hash-Wert erzeugen mit MD5 oder SHA
124        MessageDigest sha = MessageDigest.getInstance("SHA-256");
125        key = sha.digest(key);
126        // nur die ersten 128 bit nutzen
127        key = Arrays.copyOf(key, 16);
128        // der fertige Schluessel
129        return new SecretKeySpec(key, "AES");
130    }
131
132    /**
133     * Verschluesselt einen Klatext nach AES-Verfahren
134     *
135     * @param klartext
136     * @return der verschluesselte Klartext
137     * @throws Exception
138     */
139    private String verschluesseln(String klartext, SecretKeySpec schluessel
           )
140            throws Exception
141    {
142        // Verschluesseln
143        Cipher cipher = Cipher.getInstance("AES");
144        cipher.init(Cipher.ENCRYPT_MODE, schluessel);
145        byte[] encrypted = cipher.doFinal(klartext.getBytes());
```

```
146
147         // bytes zu Base64-String konvertieren (dient der Lesbarkeit)
148         BASE64Encoder myEncoder = new BASE64Encoder();
149         return myEncoder.encode(encrypted);
150     }
151
152     /**
153      * Entschluesselt einen Klatext nach AES-Verfahren
154      *
155      * @param schluesselText
156      * @return der entschluesselte schluesselText
157      * @throws Exception
158      */
159     private String entschluesseln(String schluesselText,
160             SecretKeySpec schluessel) throws Exception
161     {
162         // BASE64 String zu Byte-Array konvertieren
163         BASE64Decoder myDecoder2 = new BASE64Decoder();
164         byte[] crypted2 = myDecoder2.decodeBuffer(schluesselText);
165
166         // Entschluesseln
167         Cipher cipher2 = Cipher.getInstance("AES");
168         cipher2.init(Cipher.DECRYPT_MODE, schluessel);
169         byte[] cipherData2 = cipher2.doFinal(crypted2);
170         return new String(cipherData2);
171     }
172
173 }
```

Timing-Angriff

Anmerkung: String _symbole enthält nicht die Sonderzeichen,

1. weil LaTeXnicht alle codieren kann und
2. weil die Aufgabe schwammig gestellt ist. Ich habe eine deutsche Tastatur, kann aber, weil Linux, deutlich mehr Sonderzeichen über AltGr erreichen als ein Windows Keyboard. Außerdem kann man, sobald ein NumBlock angeschlossen ist, faktisch jedes ASCII-Zeichen über Tastatur erreichen.

```
1  public class Timer
2  {
3      private final char[] _passwort = "abcdefgh".toCharArray();
4      private boolean _gefunden = false;
5      private int _passwortLaenge;
6      private final char[] _symbole = "abcdefghijklmnopqrstuvwxyz1234567890"
7          .toCharArray();
8      private final int _maxPasswortLaenge = 20;
9
10     public static void main(String[] args)
11     {
12         Timer timer = new Timer();
13         timer._passwortLaenge = timer.findePasswortLaenge();
14         System.out.println(timer._passwortLaenge);
15         System.out.println(timer.findePasswort());
16     }
17
18     /**
19     * Findet das Passwort. Period.
```

```
20      * @return Das Passwort.
21      * @requirements: Die Passwortlaenge wurde bestimmt.
22      */
23      private String findePasswort()
24      {
25          String laufPasswort = "";
26          String bisherigesPasswort = "";
27          long[] zeiten = new long[_passwortLaenge];
28          long startZeit;
29          long endZeit;
30          do
31          {
32              // Zaehlt einen Zaehler bis zur bestimmen Passwortlaenge
33              for (int laenge = 0; laenge < _passwortLaenge; laenge++)
34              {
35                  // Zaehlt einen Zaehler bis zur Laenge des Eingabealphabets
36                  for (int zaehler = 0; zaehler < _symbole.length; zaehler++)
37                  {
38                      // haengt das aktuelle Laufsymbol an das Passwort
39                      laufPasswort = bisherigesPasswort + _symbole[zaehler];
40                      startZeit = System.nanoTime();
41                      // Hier wird das Passwort geprueft
42                      _gefunden = passwordCompare(laufPasswort.toCharArray(),
                            _passwort);
43                      endZeit = System.nanoTime();
44                      // Berechnet Zeitdifferenz fuer aktuelles Passwort
45                      zeiten[zaehler] = endZeit - startZeit;
46                  }
47                  // haengt an das bisherige Passwort dasjenige Symbol an,
                        fuer welches die groesste Zeit gebraucht wurde
48                  bisherigesPasswort += _symbole[gibIndexVonMaximum(zeiten)];
49              }
50          }
51          while (!_gefunden);
52          return bisherigesPasswort;
53      }
54
55      /**
56      * Bestimmt die Laenge eines Passworts
57      * @return die Laenge eines Passworts
58      */
59      private int findePasswortLaenge()
60      {
61          String passwort = "";
62          long[] zeiten = new long[_maxPasswortLaenge];
63          long startZeit;
64          long endZeit;
65          for (int zaehler = 1; zaehler < _maxPasswortLaenge; zaehler++)
66          {
67              passwort.concat("a");
68              startZeit = System.nanoTime();
69  //          System.out.println("Startzeit: " + startZeit);
70              // prueft, ob das Passwort korrekt ist
71              passwordCompare(passwort.toCharArray(), _passwort);
72              endZeit = System.nanoTime();
73  //          System.out.println("Endzeit: " + endZeit);
74              zeiten[zaehler] = endZeit - startZeit;
75              System.out.println("Zeitdifferenz:␣" + zeiten[zaehler]);
76          }
```

```
77          return gibIndexVonMaximum(zeiten);
78      }
79
80      /**
81      * Gibt den Index des hoechsten Wertes zurueck
82      * @param array Eingabe, das nach dem hoechsten Wert durchsucht werden
            soll
83      * @return den Index des hoechsten Wertes
84      */
85      private int gibIndexVonMaximum(long[] array)
86      {
87          int index = 0;
88          for (int i = 0; i < array.length; i++)
89          {
90              if (array[i] > array[index])
91              {
92                  index = i;
93              }
94          }
95          return index;
96      }
97
98      /**
99      * Vergleicht zwei Char-Arrays
100     * @param a erstes Array
101     * @param b zweites Array
102     * @return true, wenn sie gleich sind; false sonst
103     */
104     boolean passwordCompare(char[] a, char[] b)
105     {
106         int i;
107         if (a.length != b.length) return false;
108         for (i = 0; i < a.length && a[i] == b[i]; i++)
109             ;
110         return i == a.length;
111     }
112 }
```