X.509 Certificate Factory

Project Documentation

Author:
T. Lauchenauer

Table of Contents

1 Abstract

The goal of this project was to write a new GUI for requesting certificates to replace the one implemented in the OpenCA package. This project is based on a previous project by M. Grieder and S. Zehnder called “LDAP-Server für X.509 Zertifikate”. In that project the OpenCA and the OpenSSL project had been installed. The current project takes all installations and settings of the previous project as given.
The goal in itself, the creation of a new GUI, was not reached over the course of this project. The main task was broken down into three separate tasks. The first consisted of creating an RSA keypair and a certificate from a Java applet. The public key would be sent with the rest of the request data to the OpenCA package. This is the area where no solution was found. Java does not yet provide classes to support RSA keypairs, and as of Java SDK 1.3 only reading certificates is supported and not the creation of certificates.
The second task consisted of getting permission for the applet to write files to the local computer and to open a URL connection to a remote host other than the one it came from. These two aspects are important to allow the applet to save the created certificate (if the creation was possible). The ability to open connections to remote hosts would allow the applet to be hosted on a computer separate from the OpenCA installation, thus separating the two from each other. A solution to this was found in the form of RSA signed applets. Java SDK 1.2.2 introduced this form of signed applets. The creation of such an applet is not as simple as the SUN documentation wants to make believe. To create such an applet a developer certificate from a recognized Certification Authority (CA) is necessary. It is possible to create one with OpenSSL, though the process is quite complicated. In addition to having such a certificate, it is necessary to have the Java Plug-In 1.2.2 or better installed on the computer running the applet. If the applet is set up with the <EMBED> or <OBJECT> tag instead of the <APPLET> tag in an HTML page, then the user trying to run the applet is presented with a window requesting him to grant the applet all permissions. If the user grants these permissions the applet can then be run correctly.

The third task consisted of “fooling” OpenCA into thinking that the data it was receiving came from its own Perl script. To do this the Perl script had to be adjusted somewhat (the checking for the browser was removed). A separate class (the class “Poster”) was written to accomplish the task of sending and receiving data to the Perl script. An important thing to notice is that the “Poster” class sends the data using the content type “text/plain”. If another content type is used the class doesn’t work correctly. Another important issue is that the applet has to be run from a browser. The OpenCA package uses a secure connection (“https” protocol) which the appletviewer does not support up to now.
The final applet contains all the features that were possible to implement. The sending of the data via the “Poster” class is included, though for testing purposes that code part was commented. Also no parts of the generation of the keypair or the certificate were implemented. The final applet has not been signed, as there is still a lot of work to be done on it (implementing the certificate creation).
For detailed information on the above mentioned tasks and different approaches to the problems read the complete report in Chapter 4.

2 Task Description

For the original Task Description see:
\Supporting Documents\Task Description\PA_2000_Sna05.htm

The basis for this project is a previous project by M. Grieder and S. Zehnder called “LDAP-Server für X.509 Zertifikate”. There, among many other tasks, the open-source packages OpenSSL and OpenCA had been installed and configured. The installed version of OpenCA only supports Netscape Browser because the <KEYGEN> tag is used to create the private and public keys for the certificate. Due to that limitation and the lack of versatility in requesting certificates, the idea was to write a new GUI for requesting certificates.

Tasks:

- get to know the OpenSSL and OpenCA packages
- get to know the sequence of events in the process of requesting and receiving a certificate
- decide on the tools and programming language needed to create the GUI
- write a versatile GUI that works with the OpenCA package and runs on Netscape and Microsoft Browsers
- testing and documenting the Software

3 Approach

The project started on Tuesday the 23. of May 2000. After receiving the task description the first step consisted of determining what exactly was to be accomplished. The previous project (see [1]) provided a lot of information on what had been done before. The OpenSSL and the OpenCA packages had been installed and set up, so one task was getting to know these two open-source packages (at least superficially at this point). Especially the OpenCA package is still in development and new releases come out every few months, so that left a lot of room for improvement. The previous project had uncovered quite a few features that were still lacking.
Due to the lack of Linux and Perl experience of the author the project task was soon narrowed down to setting up a better and more versatile GUI for requesting certificates. This allowed a part of the work to be accomplished in Windows. The OpenCA package processes certificate requests with a Perl script that only works on Netscape browsers (see chapter 4 for more details). The new GUI was supposed to run on any browser. This requirement soon led to the idea of using Java as the programming language. By using a Java Plug-In most up-to-date browsers are able to run Java applets.
This led to the next step of refreshing the applet programming theory. SUN provides a good tutorial on this subject on their page (see [14]). Once the applet theory was reviewed the main task was broken down into several smaller tasks. These tasks consisted of the following:

1. **Generating a certificate and saving it**
   The applet has to generate the private and public RSA keys for the certificate and pass on the public one in the request. The private key and the user data is used to generate a certificate and save it as a *.p12 file.

2. **Applet security**
   An applet doesn't get permission to write a file to the client PC where the applet is run. It also doesn't get permission to open URL connections to hosts other than the one it came from. To get these permissions they have to be explicitly granted by the user.

3. **Client – Server Communication**
   To send out a certificate request, the applet has to communicate with the OpenCA package. This package consists of Perl CGI scripts. The applet has to "fool" these scripts to receive data from an applet and not generate the HTML pages through the script.

4. **Putting it all together**
   These main tasks have to work together somehow to then set up the final project.

From this point on most of these tasks ran more or less in a parallel mode. Depending on where difficulties were encountered or solved work continued on a different task. For documentation purposes, the different tasks have been kept together.

The Client-Server Communication was among the first that was tried. After searching the internet for information on how to post data to a Perl script (see [12]), first tests were made. The first mistake that was made at this stage was trying to open a URL connection to a different host. A normal applet does not allow connections to hosts other than the one where the applet comes from. Once this error had been taken care of, and the applet was running on the same host as the Perl script, the next hurdle was encountered. The OpenCA Perl script for requesting certificates is quite large. Again some tests were made that failed because of different causes. To work around this, a simple Perl script was used, instead of the complex OpenCA script. Errors were more easily located and soon the applet was able to communicate with the simple trial Perl script. The next step then consisted of trying to get the same applet to work with the OpenCA script. At first the sending of the data seemed to work, although receiving data from the Perl script generated exceptions in the applet. The strange part was that the sending didn't throw exceptions, but no information was received by the script. A step by step search for the position that caused the Perl script to abort started. Every few lines of code an output to a logfile was included in the script. The "error" was finally found in a standard library from Perl. The Perl script exited because of a wrong content type of the data. Once the content type was changed, the applet worked without problems. Even the receiving of the data, which had caused exceptions previously, now worked without problems. A lot of time was spent trying to locate these errors. Many different variations of the applet had been tried before locating the errors. Mr. Steffen's help was really appreciated in locating these errors.

The question of the applet security was also a task that used up a lot of time, until it finally worked. The main problem here was the lack of documentation of the Java security features. The first thing that was found were the policy files that SUN normally uses. Here the documentation was plentiful in every area, except on how to implement policy files into applets run from browsers. The big bulk of the documentation consisted of how to run policy files on the appletviewer (see [6]). This feature was useless for the project, because the appletviewer does not support the "https" protocol (which was discovered after a short test). The only way to implement policy files into applets that are run from browsers is to include the policy file into the security properties file of the Java Plug-In. This possibility was ruled out directly because most users have no rights to access the security properties file and it would have forced every user of the applet to download the policy file and install it before running the applet. Finally the subject of RSA signed applets was discovered. The Java SDK 1.2.2 introduced this topic. The documentation on this subject was even worse than on the previous issues. At first only two papers were encountered (see [9] and [10]). One even included instructions on how to set up an RSA signed applet. The idea of RSA signed applets seemed perfect. A dialog should appear requesting the user to grant the applet permission to access other hosts or write files. For an RSA signed applet the Netscape singtool has to be used (which includes good documentation [11]). One option of the signtool allows the creation of test certificate, with which it is possible to sign the applet.

The instructions were followed minutely, but tests soon showed that the nice idea didn't work. The idea was then laid to rest and taken up again towards the end of the project. First a new certificate was generated with OpenSSL, with no new results. Again it was Mr. Steffen, who found the solution. The problem had been, that to imbed the applet into an HTML page the <APPLET> tag had been used. This caused the browser to use its own Java Virtual Machine (JVM). To force the browser to use the Java Plug-In the <EMBED> or the <OBJECT> tag has to be used. Even with these new tags the signed applet wasn't recognized when it was signed with the test certificate generated by the signtool. Neither did it work with the second certificate generated by OpenSSL. Mr. Steffen then created a new certificate with a correct entry in the Certificate Revocation List and it worked! In the end he even found a documentation of the tags to be used for applets (see 7]). This was probably the most frustrating part of the project, as no documentation gave any hint why the whole thing didn't work, though finally it did work (thanks to Mr. Steffen).

The shortest task was the try to generate certificates and keypairs. Java includes different classes to do these jobs. First the generation of the keypair was checked out. Here Java has the KeyFactory class that makes this easy. Soon though the problem was encountered. The Netscape <KEYGEN> tag generates RSA keys. Java only has built in support for DSA keys. To create RSA keys a new algorithm Provider would have to be found.
Creating certificates is even less supported. Java currently includes classes that allow it to read certificates. Even those didn't seem to work correctly on certificates exported from IExplorer after a few quick tests.
This is where the main idea of using a Java applet for the GUI failed. Without sufficient support from Java for the needed security issues, it is not possible to conclude the applet.

In between of trying to solve the other problems, the final GUI was started as well. From the results taken from the communication tests with the OpenCA Perl script the Poster class was written (see the appendix for the listing). This class is responsible for sending the data to a URL and receiving data from it again. This class was specifically written separate from the GUI as to make future changes in the OpenCA package easy to implement into the GUI.
The second part of the GUI was the visuals. These were kept very similar to the way that the OpenCA handles certificate requests. For a complete description of this see section 4.4.

To finish the whole project, all these separate parts had to be documented, and all the used sources catalogued.

4 Report

As mentioned in “Chapter 2 : Task Description” the goal of this project was to write a GUI for requesting certificates that is versatile and runs on both Netscape Navigator and IExplorer. This GUI had to run with the already installed versions of OpenCA and OpenSSL. This limitation was given through the previous project [1], on which the current work is based. OpenSSL didn’t pose any problems because OpenCA takes charge of calling the functions from OpenSSL. This was different with OpenCA. This package sets up several limitations, among them the using of a secure connection for requesting certificates (https – protocol). Another limitation from OpenCA is the way that certificate requests were handled, these are first stored in a file and then processed further, but only through administrator interactions. For more information on the OpenSSL package see [4] and www.openssl.org. For more information on the OpenCA package see the above mentioned previous project [1] and www.openca.org.

Two possibilities were available to write this GUI, one was to extend the present Perl scripts, the second was to write an applet that more or less copied the functions of the Perl scripts from OpenCA. The first possibility posed several problems: first, the current Perl scripts depended on using the Netscape Navigator. Only Netscape supports the <KEYGEN> tag. This tag allows the generating of a public and a private key. The private key is stored in the local Netscape key database and the public key can be passed on with other form data (for detailed information on the <KEYGEN> tag see [2]). Another problem with using Perl was, that the author had no experience with Perl and the scripts are quite complicated. Thus extending the OpenCA script would have added an additional complication and the second possibility (writing an applet) was chosen. There would also have been the possibility of writing a Java (or any other programming language) application, though that would have added the necessity of the user downloading an application before being able to request a certificate. This last possibility was disregarded in the process of deciding between an applet and extending the Perl scripts.

Once that the decision was made to use a Java applet to create the GUI, the whole project was broken down into four main tasks:

1. **Generating a certificate and saving it**
   The applet has to generate the private and public RSA keys for the certificate and pass on the public one in the request. The private key and the user data is used to generate a certificate and save it as a *.p12 file.

2. **Applet security**
   An applet doesn't get permission to write a file to the client PC where the applet is run. It also doesn't get permission to open URL connections to hosts other than the one it came from. To get these permissions they have to be explicitly granted by the user.

3. **Client – Server Communication**
   To send out a certificate request, the applet has to communicate with the OpenCA package. This package consists of Perl CGI scripts. The applet has to "fool" these scripts to receive data from an applet and not generate the HTML pages through the script.

4. **Putting it all together**
   These main tasks have to work together somehow to then set up the final project.

Below these three tasks are covered in a more detailed form.

4.1 Generating a certificate and saving it

For the GUI applet to work it has to be able to generate a public and a private key. The public key has to be sent to the OpenCA Perl script. From the private key and the user data a certificate has to be generated and saved in a format that both the Netscape and the Microsoft browser can read.
The whole thing doesn't sound that complicated. The problem starts with the Java classes for these tasks. SUN has quite a few classes that deal with the security issue. A paper on the subject of Cryptography (which includes Key generation and certificates) can be found under [13].
Generating the private and public keys can be achieved by instantiating a KeyFactory class. The problem starts when an RSA keypair should be generated. The OpenCA script uses the standard <KEYGEN> tag with the RSA algorithm (for more information on the <KEYGEN> tag see [3]). SUN only provides built-in support for the DSA algorithm. If an RSA keypair is to be generated an external Provider has to be found that supports the RSA algorithm. This was the first unsolved problem in this task.
The next problem surfaced with the creation of a certificate. Java classes allow the loading of a certificate but do not provide a possibility to create a certificate. Neither is there support to save certificates in the "*.p12" format that Netscape and Microsoft browsers understand.

Java still does not have enough support for certificates and keypair generation. Either another approach has to be found to solve these problems (like using something else than an applet) or the classes that allow the generation of RSA keypairs and certificates have to be programmed from scratch.
This is where the idea of using an applet to substitute the certificate request part of the OpenCA package failed. Java simply doesn't provide enough support for these security issues (yet?).

4.2 Applet security

Applets have restricted permissions on the computer that they are run on. The term that is used, is sandbox. Every applet has its own sandbox in which it is run. The sandbox stands for a protected environment with limited options. The two main limitations in this case were the fact that an applet is not allowed to write a file to the computer on which it is run and that the applet is not allowed to open a URL connection to any other computer other than the one it came from.
Both of these limitations had to be removed to allow the applet to run without hindrance. The ability to write a file is crucial, the ability to open a connection to another computer would allow the applet to be hosted on a computer other than the one where the OpenCA package is installed.

SUN and Netscape have devised ways to circumvent or break out of the limitations of the applet sandbox. A quite thorough documentation on the basics of applet security can be found on the SUN site [5].

The way that SUN proposes, is the use of policy file. A policy file contains a list of specific permissions what the applet (or even the application) is allowed to do. These can include, requiring that the applet is signed (using the jarsigner tool), or a number of other requirements. The SUN site has a very good tutorial for security for applets (and applications) using policy files [6]. The problem with policy files, is the way to tell the applet to include the policy file. There are two ways to include such a policy file: use the appletviewer with the "–J-Djava.security.policy=mypolicy" addition. A step by step tutorial on how to use such policy files for the appletviewer can be found under [8]. The second possibility is to add the policy file to the security properties of the Java Runtime Environment (JRE). Both of these possibilities were ruled out in the project. The first was ruled out, because the appletviewer does not support secure connections. When trying to open a connection using the "https" protocol an exception "protocol unknown" is thrown. The second possibility was ruled out because of the way the servers at school are organized. The "normal" users can not edit the security properties file of the JRE. Thus the use of policy files fell away. An error easily made is the use of the <APPLET> tag. This tag does not force the browser to use the installed JRE but uses the Java Virtual Machine (JVM) that comes with the browser. In this case even setting the security properties file will not affect anything. To force the browser to use either the <OBJECT> or <EMBED> tag have to be used. These tags are fully documented in [7].
Netscape also provides a way to break out of the Java Virtual Machine (JVM) sandbox. This is only possible through the implementation of the nescape.security classes (see [5]). This possibility was also ruled out, because of lack of compatibility between Netscape and Microsoft.

A new solution is provided by the Java Plug-In1.2.2 from SUN. The Plug-In 1.2.2 has to be installed to the browser where the applet is to be run. SUN introduced the support for RSA signed applets with version 1.2.2. This means that now standard developer certificates can be used to sign an applet for Netscape and Microsoft browsers, instead of requiring a separate certificate for each browser supported. RSA signed applets, when correctly set up bring up a dialog asking the user whether he wants to grant the applet all permissions once, grant all permissions always, or deny permissions. Depending on which option the user chooses the applet can run without a problem.
The documentation on this topic is sparse at best and the newsgroups are full of questions on this subject. It is possible to make it work, though getting there can be very frustrating. Two papers from SUN on this subject can be found under [9] and [10]. Below are the steps that were used to create a working applet with the RSA support.

1. Write the applet
   In our case the applet consists of a “Send” button and a textfield for the outputs and errors. When the “Send” button is pressed a string is assembled. Then a secure URL connection is opened to the edited version of the Perl script from OpenCA. The string is then sent to that connection. If the sending went ok, the applet then listens to anything that is sent back from the Perl script (which should be a HTML page). Once all this is through the connection is closed again. Any errors that occur during this process terminate the program and the errormessage is drawn into the textfield.
   Here the source code (is also found in original form in **\Source Code\RSA Applet**):

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;

public class MailMe extends Applet implements ActionListener
{
 // Simple user interface
 TextArea field;
 Button btn;

 public void init()
 {
   // Init user interface
   setLayout(new BorderLayout());
   field = new TextArea("Applet Start", 10, 80);
    field.setEditable(false);
   btn = new Button("Send");
   btn.addActionListener(this);
   add(field,BorderLayout.NORTH);
   add(btn,BorderLayout.SOUTH);
   validate();
 }

  void addItem(String newWord)
  {
    //This used to append the string to the StringBuffer;
    //now it appends it to the TextField.
    String t = field.getText();
    System.out.println(newWord);
    field.setText(t + "\n" + newWord);
    repaint();
  }

 // On button click, send data
 public void actionPerformed(ActionEvent e)
  {
   URL url;
   URLConnection connection;
   try
   {
      url = new URL("https://ksy112.zhwin.ch/cgi-bin/request2");
      //url = new URL("https://ksy112.zhwin.ch/cgi-bin/backwards");
      connection = url.openConnection();
      connection.setRequestProperty("content-type", "text/plain");
      connection.setDoOutput(true);

      PrintWriter out = new PrintWriter(
                           connection.getOutputStream());
      //String mytext = "string=" + URLEncoder.encode("client-confirmed-form");
      String mytext = "operation=" + URLEncoder.encode("client-confirmed-form");
      mytext = mytext + '&' + "first_name=" + URLEncoder.encode("Tobias79");
      mytext = mytext + '&' + "last_name=" + URLEncoder.encode("Lauchenauer");
      mytext = mytext + '&' + "email=" + URLEncoder.encode("i7lauche@zhwin.ch");
      mytext = mytext + '&' + "country=" + URLEncoder.encode("CH");
      mytext = mytext + '&' + "ou=" + URLEncoder.encode("ZHW Developer");
      mytext = mytext + '&' + "passwd3=" + URLEncoder.encode("1234567890");
      mytext = mytext + '&' + "newkey=" + URLEncoder.encode("12121212122");
      out.println(mytext);
```

```
        addItem(mytext);

        out.close();
       addItem("Send OK \n");

        BufferedReader in = new BufferedReader(new InputStreamReader(
                                                connection.getInputStream()));

        addItem("Reader Init OK \n");
        String inputLine;
        while ((inputLine = in.readLine()) != null)
            addItem(inputLine);

        in.close();
        addItem("Reading finished OK \n");
      }
     catch (Exception ex)
      {
       addItem("Error....");
       addItem(ex.getMessage());
       repaint();
       return;  // what to do?
      }

   }
}
```

2. After the applet is compiled without a problem you have to make sure that you have a valid certificate. The certificate that was used for this project is called “laucheCert.p12” and is found in the directory **\Source Code\RSA Applet**, the password to it is i7lauche. When using certificates that are self-made, it is important to make sure that the Certification Authority (CA) is accepted (on creating your own certificates see [4]). To accept a CA edit the corresponding CA entry in the security settings of your browser.
   Once the certificate is imported, Netscape Signtool has to be installed. Actually, the installation consists of downloading and unzipping it. Once the signtool is set up the applet can be signed. To do this copy all the class files into a directory and use the following command:

   signtool –k “certificate name” –d “netscape settigns dir” –Z applet.jar “dir of the class files”

   For a complete documentation of the signtool see [11]. This puts all the class files into a jar file and then signs the jar archive.

   The signed jar archive can be found in **\Source Code\RSA Applet**

3. Set up an HTML page
   Once the applet is stored in a jar file and the jar file is signed, it can be hooked up to an HTML page. Be careful not to use the <APPLET> tag. This tag uses the built in JVM of the browser and not the Plug-In from SUN. To make sure that the Plug-In from sun is activated use the <EMBED> or the <OBJECT> tag. In out example the following worked:

   ```
   <embed type="application/x-java-applet;version=1.3"
     code="MailMe.class" archive="MailMe.jar" width="300" height="200">
   </embed>
   ```

   The utilized HTML page can be found under **\Source Code\RSA Applet**.

   For a complete documentation of the <EMBED> and <OBJECT> tag with applets see [7].

Another problem that can occur when running RSA applets is that the certificate is not correctly set up. This is especially the case when self-made certificates are used. In the case of this example an error that was encountered was a missing entry for the Certificate Revocation List (CRL) which caused the whole thing to abort without much of an error message.

When an RSA signed applet is run and all permissions are granted, an entry is made into the Plug-In Control Panel. This Control Panel can be found where all the Control Panels from the computer are.

Another thing with RSA signed applets, is that no specific permissions can be granted. Either all permissions are granted, or none at all. It is advisable to be careful when granting permissions!

Problems:
First, even finding out about the possibility of the RSA signed applets was a challenge. SUN mainly propagates the use of the policy files. Once some of the sparse documentation was found, the next big challenge was finding out that the <APPLET> tag cannot be used to run RSA signed applets. The <EMBED> tag was used (thank you Mr. Steffen!). An additional difficulty arose when the certificate didn't work properly. Again it was Mr. Steffen that found out that the CRL had to be correctly configured, else the whole thing wouldn't work.
It seems that SUN is still working on all those security issues, and are releasing them one by one without sufficient documentation. Maybe in one to two years the whole issue will begin to brighten, currently working with these issues is definitely frustrating.

4.3 Client – Server Communication

The OpenCA package uses one Perl script to walk the user through the whole process of entering and confirming the data. For an applet to take the part of confirming the data, it has to be able to "fool" the Perl script into accepting the data as confirmed.

As the first step to communicating with the script, some security functions from it had to be disabled. Since the resulting GUI is not to be browser dependent any code to do with browser detection had to be commented.

Here the Perl script tries to get the version and the name of the browser from which data is coming:

```
## Get the Browser Name and the Version
if ( $USER_AGENT =~ /(MSIE)/ ) {
my ($browser, $ver) =
      $USER_AGENT =~ /(MSIE)\ (\d+\.\d*|\.\d+)/;
} else {
my ($browser, $ver) =
      $USER_AGENT =~ m-(^[^/]+)/(\d+\.\d*|\.\d+)-;
};
```

Here the script makes sure that only Netscape browsers can continue on beyond this point. This too was commented.

```
if ($USER_AGENT !~ /Mozilla/i) {
      &sendPage( $browserError );
      exit;
```

```
}
elsif ($USER_AGENT =~ /MSIE/i) {
      &sendPage( $browserError );
      exit;
}
else {
}
```

The last part of code that was commented was the checking for other methods for sending data to the script. This implies, that the applet can use any method that it wants to send the data (the POST method was used).

```
## Checking for Boguous Systems ( POST method, REFERER doc, etc... )
if ($METHOD !~ /POST/i) {
      &sendPage( &protocolError );
};
```

The resulting Perl script was saved under a different name (request2) as to not interfere with the original OpenCA installation. A copy of it can be found under **\Source Code\Perl Script Request**

Once the Perl script was set up, the next task was the communication with it from the applet. An example was found on the net on how to do this (see [12]). On the basis of that, the class "Poster" was written. The complete source code of this class can be found under **\Source Code\Poster.** For the complete documentation of it, please see the javadoc pages included in the same directory.

What this class does, is piece together a data string from "name=value" pairs. With the "post()" method this string is then posted to the URL with which the class was initialized. It's important to notice that the "content-type" is set to "text/plain". Some Perl scripts may work with other content types, the OpenCA Perl script only works with this content type.
When the data is all posted the method continues to listen at the specified connection. Any data that is returned is collected in a string and returned from the method.

```
public class Poster
{
  URL url;
  String data;
  URLConnection connection;

  public void addPair(String descriptor, String content)
  {
      //append an encoded name=value pair to the data string
  }

  public String post() throws java.io.IOException
  {
    connection = url.openConnection();
    connection.setRequestProperty("content-type", "text/plain");
    connection.setDoOutput(true);
    PrintWriter out = new PrintWriter(connection.getOutputStream());

    out.println(data);
    out.close();

     BufferedReader in = new BufferedReader(
                             new InputStreamReader(connection.getInputStream()));
     StringBuffer buf = new StringBuffer();
     String line;
  while ((line=in.readLine()) != null)
    {
      buf.append(line);
      buf.append("\n");
```

```
            //return buf.toString();
          }
        in.close();
          // return it*/
         return buf.toString();
        }
    }
```

With this class, when initialized to https://ksy112.zhwin.ch/cgi-bin/request2, it is possible to communicate with the OpenCA Perl script. The following "name=value" pair are necessary for the script to accept a certificate request.

Name	Value
operation	client-confirmed-form
first_name	*the user's first name*
last_name	*the user's last name*
email	*the user's email*
country	*the user's country code (2 characters)*
ou	*the organization unit (a selection of the ones specified in the secure.cnf file from the OpenCA installation)*
passwd3	*the user's confirmed password*
newkey	*the public key, as generated by the <KEYGEN> tag from Netscape*

The "operation=client-confirmed-form" is important to tell the script, that the data has already been verified, and that the request can be processed. The "ou" value is a bit cryptic. The file secure.cnf (you can find a copy of it under **`\Source Code\Perl Script Request`**, defines different choices. Though it is unclear how OpenCA uses these different choices to create different certificates with OpenSSL. The documentation of OpenCA is severely lacking in this point. The "newkey" attribute is also important. The Perl script expects a key in the form generated by the <KEYGEN> tag of Netscape. For a description of this tag see [3].
Once all these entries are correct, the data can be posted to the "request2" script. If everything went correctly the script replies with a "thank you" page.

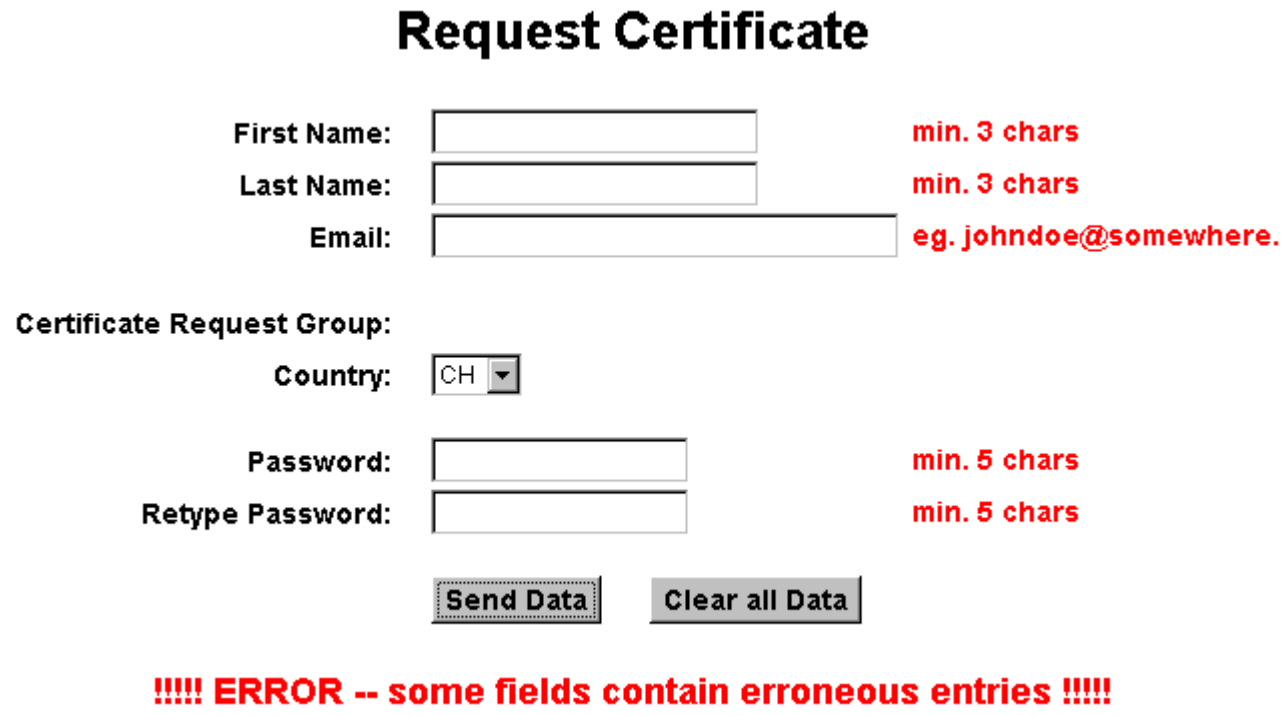
Problems:
One problem that was encountered, was finding out that the "content-type" had to be set to "text/plain". Since normal HTML forms use the content type "multipart/form-data" it was assumed that this would also work with the "request2" script. After a lot of stepping through perl scripts (thank you Mr. Steffen for the help) the error was located. A second problem was encountered when trying to generate a public key in the form expected by the script. For more details on that see section 4.1.

4.4 Putting it all together

An attempt was made to pull the three core tasks together into the final GUI. Since Java just doesn't provide enough support for certificates, the final GUI can't include these either. The other two sections can be included (the GUI was not converted into an RSA signed applet, although that would not be much of a problem). The communication with the OpenCA package works through the "Poster" class that was explained in section 4.3. This allows the communication to be completely separate from the other parts of the GUI. This makes changes in the OpenCA package easy to be included in the GUI, because only one class has to be changed without touching the rest of the GUI.

A simple layout very similar to the one presented by the OpenCA package was chosen:

There are no entries by the title “Certificate Request Group” because the way that OpenCA implements these is unclear. This is explained better in section 4.3.
The “Clear all Data” button empties all the entries. Pressing the “Send Data” button first checks if the data entered is correct. The following picture shows the applet with errors in all entries:



Every field with an error in it is highlighted with the red explanation text behind it. No data is sent while there are errors in any of the entries. Once that all entries check out as correct, the entries are sent to the “request2” script.
The name entries have to be at least 3, passswords at least 5 characters long. The email has to consist of at least the following: a@a.a . If these criteria are fulfilled the entries are said to be ok. When the data gets sent the applet has the following appearance:

In the current version of the GUI the code-line that sends the data to the Perl script has been commented, although all that is done, has been explained in the example from section 4.3.
The complete source code to the GUI can be found in the Appendix or in `\Source Code\Gui.`

In the time period of this project the GUI was not finished. Especially the problems illustrated in section 4.1 made it impossible to finish in the given time period. Once SUN brings out additional Java classes, or other providers bring out classes to handle RSA key and certificates, it will be possible to finish the GUI.
Also the lack of documentation in the OpenCA package make the choice of the Organization Unit somewhat difficult. To set up that part of the GUI correctly, it would have to be known how OpenCA calls the OpenSSL package. Then the possibility would be open to allow the user to choose what type of certificate should be requested (mail, developer, etc.).
Another option would be to rewrite the Perl scripts from OpenCA and implement new scripts from which the possibilities would all be known.

The current GUI leaves a lot of room for improvements and additions. Next steps would be to get the key generation and certificate generation to work. Then the OpenCA and OpenSSL communication would have to be studied to implement the request for different types of certificates. Once these steps are accomplished the GUI could be of use and extended further.

5 Conclusion & Prospects

Looking back on the project, the biggest task was searching for information. Nearly every aspect of the project consisted of searching the internet for ways to accomplish the task. Along the way, a lot of information was gathered (much of which is listed in the Bibliography).
The security aspect of Java was a very frustrating and discouraging and in the same time interesting subject. Finding all the different proposals on signing applets and giving applets permissions for certain tasks has given the author a much better understanding of the difficulties and possibilities of this subject. This whole area still has a lot of potential and SUN is developing new classes all the time. It will be very interesting to see what the next steps will be that SUN will take in the security issue concerning Java applications and Java applets.
The issue of Java classes that will be able to create their own certificates is also something that will be interesting to follow. It may well be possible that SUN will expand their current classes to include such support.
SUN had a good idea when they enabled other algorithm Providers to be included into their standard packages in creating keypairs and other security functions. Though it is not yet apparent if other firms have taken up on the idea and will be adding such new algorithms. SUN themselves only provide a limited set of algorithms and thus depend on others to pull along.
In the course of the project the author also got an introduction into Perl and the Linux OS. Especially Perl proved to be quite an interesting programming language for interaction with browsers and applets. The task of getting the communication between the GUI applet and the OpenCA Perl script to work took up a lot of time but in the end was an interesting task. The insights gained there will certainly be something that the author will be using again.

The work on this project in overall was interesting and a lot of times frustrating. Especially the times that one problem appeared after the next with no solutions in sight were tough. In those times Mr. Steffen was a really great help. Most of the solutions would not have been found without his help.

Even though the main task, the creation of a new GUI, was not accomplished, the project was still educational and provided new insights and better understanding into areas that the author had little experience before commencing the project.

The emphasis was laid on setting up the documentation in a way so that in the future it would be easy to continue the work started here. The GUI that was partly written here, can be finished once Java provides support for creating certificates and RSA key-generation. The code was written in such a way to allow changes to be made easily.
There is still room for improvement as well. Possibilities would be the finishing the GUI with the ability to specify exactly what kind of certificate is requested. Also expanding the GUI to include more than just the normal data fields necessary for a certificate would be a possibility.

All in all this project was a challenge that brought insight into the interesting area of security issues and certificates. It will be interesting to see what the future will bring in new technologies, algorithms, etc. for these issues.

6 *Appendix*

1. **Listing of the CARequests Class**
2. **Listing of the Poster Class**

6.1 Listing of the CARequests Class

```
// CARequests
// an applet  that allow to request a certificate
//
// The next two lines allow appletviewer to view the java file
// <applet code=MailMe.class width=500 height=300></applet>
//

import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class CARequests extends Applet implements ActionListener
{
  public void init()
  {
    // Init the Layoutmanager
    setLayout(null);

    // Init the fields for the GUI
    firstname = new TextField("", 20);
    lastname = new TextField("", 20);
    email = new TextField("", 30);
    pass1 = new TextField("", 15);
    pass1.setEchoChar('*');
    pass2 = new TextField("", 15);
    pass2.setEchoChar('*');
    country = new Choice();
    country.addItem("CH");
    ou = new Choice();
    ou.addItem("ZHW User");
    ou.addItem("ZHW Developer");
    ou.addItem("Web Server");
    ou.addItem("Network Server");
    sendbtn = new Button("Send Data");
    clearbtn = new Button("Clear all Data");
    f_normal = new Font("MS Sans Serif", Font.PLAIN, 12);
    f_fett = new Font("MS Sans Serif", Font.BOLD, 14);
    f_titel = new Font("MS Sans Serif", Font.BOLD, 26);
    f_klein = new Font("MS Sans Serif", Font.PLAIN, 7);
    f_fehler = new Font("MS Sans Serif", Font.BOLD, 18);

    GridBagLayout gbl = new GridBagLayout();
    GridBagConstraints gbc;
    setLayout(gbl);
    Label lbl;

    lbl = new Label("Request Certificate");
    lbl.setFont(f_titel);
    gbc = makegbc(0, 0, 6, 1);
    gbc.anchor = GridBagConstraints.CENTER;
    gbl.setConstraints(lbl, gbc);
    add(lbl);

    // add a little space
    lbl = new Label(" ");
    lbl.setFont(f_klein);
    gbc = makegbc(0, 1, 2, 1);
    gbc.anchor = GridBagConstraints.EAST;
    gbl.setConstraints(lbl, gbc);
    add(lbl);

    lbl = new Label("First Name:");
    lbl.setFont(f_fett);
    gbc = makegbc(0, 2, 2, 1);
    gbc.anchor = GridBagConstraints.EAST;
    gbl.setConstraints(lbl, gbc);
    add(lbl);

    lbl = new Label("Last Name:");
    lbl.setFont(f_fett);
```

```
gbc = makegbc(0, 3, 2, 1);
gbc.anchor = GridBagConstraints.EAST;
gbl.setConstraints(lbl, gbc);
add(lbl);

lbl = new Label("Email:");
lbl.setFont(f_fett);
gbc = makegbc(0, 4, 2, 1);
gbc.anchor = GridBagConstraints.EAST;
gbl.setConstraints(lbl, gbc);
add(lbl);

// add a little space
lbl = new Label(" ");
lbl.setFont(f_klein);
gbc = makegbc(0, 5, 2, 1);
gbc.anchor = GridBagConstraints.EAST;
gbl.setConstraints(lbl, gbc);
add(lbl);

lbl = new Label("Certificate Request Group:");
lbl.setFont(f_fett);
gbc = makegbc(0, 6, 2, 1);
gbc.anchor = GridBagConstraints.EAST;
gbl.setConstraints(lbl, gbc);
add(lbl);

lbl = new Label("Country:");
lbl.setFont(f_fett);
gbc = makegbc(0, 11, 2, 1);
gbc.anchor = GridBagConstraints.EAST;
gbl.setConstraints(lbl, gbc);
add(lbl);

// add a little space
lbl = new Label(" ");
lbl.setFont(f_klein);
gbc = makegbc(0, 12, 2, 1);
gbc.anchor = GridBagConstraints.EAST;
gbl.setConstraints(lbl, gbc);
add(lbl);

lbl = new Label("Password:");
lbl.setFont(f_fett);
gbc = makegbc(0, 13, 2, 1);
gbc.anchor = GridBagConstraints.EAST;
gbl.setConstraints(lbl, gbc);
add(lbl);

lbl = new Label("Retype Password:");
lbl.setFont(f_fett);
gbc = makegbc(0, 14, 2, 1);
gbc.anchor = GridBagConstraints.EAST;
gbl.setConstraints(lbl, gbc);
add(lbl);

// add a little space
lbl = new Label(" ");
lbl.setFont(f_klein);
gbc = makegbc(0, 15, 2, 1);
gbc.anchor = GridBagConstraints.EAST;
gbl.setConstraints(lbl, gbc);
add(lbl);

// add a little space
lbl = new Label(" ");
lbl.setFont(f_klein);
gbc = makegbc(0, 17, 2, 1);
gbc.anchor = GridBagConstraints.EAST;
gbl.setConstraints(lbl, gbc);
add(lbl);

firstname.setFont(f_normal);
```

```
gbc = makegbc(2, 2, 2, 1);
gbc.anchor = GridBagConstraints.WEST;
gbl.setConstraints(firstname, gbc);
add(firstname);

lastname.setFont(f_normal);
gbc = makegbc(2, 3, 2, 1);
gbc.anchor = GridBagConstraints.WEST;
gbl.setConstraints(lastname, gbc);
add(lastname);

email.setFont(f_normal);
gbc = makegbc(2, 4, 2, 1);
gbc.anchor = GridBagConstraints.WEST;
gbl.setConstraints(email, gbc);
add(email);

country.setFont(f_normal);
gbc = makegbc(2, 11, 2, 1);
gbc.anchor = GridBagConstraints.WEST;
gbl.setConstraints(country, gbc);
add(country);

pass1.setFont(f_normal);
gbc = makegbc(2, 13, 2, 1);
gbc.anchor = GridBagConstraints.WEST;
gbl.setConstraints(pass1, gbc);
add(pass1);

pass2.setFont(f_normal);
gbc = makegbc(2, 14, 2, 1);
gbc.anchor = GridBagConstraints.WEST;
gbl.setConstraints(pass2, gbc);
add(pass2);

lbl_firstname = new Label("min. 3 chars.");
lbl_firstname.setFont(f_normal);
gbc = makegbc(4, 2, 2, 1);
gbc.anchor = GridBagConstraints.WEST;
gbl.setConstraints(lbl_firstname, gbc);
add(lbl_firstname);

lbl_lastname = new Label("min. 3 chars.");
lbl_lastname.setFont(f_normal);
gbc = makegbc(4, 3, 2, 1);
gbc.anchor = GridBagConstraints.WEST;
gbl.setConstraints(lbl_lastname, gbc);
add(lbl_lastname);

lbl_email = new Label("eg. johndoe@somewhere.com");
lbl_email.setFont(f_normal);
gbc = makegbc(4, 4, 2, 1);
gbc.anchor = GridBagConstraints.WEST;
gbl.setConstraints(lbl_email, gbc);
add(lbl_email);

lbl_ou = new Label("");
lbl_ou.setFont(f_normal);
gbc = makegbc(4, 6, 2, 1);
gbc.anchor = GridBagConstraints.WEST;
gbl.setConstraints(lbl_ou, gbc);
add(lbl_ou);

lbl_country = new Label("");
lbl_country.setFont(f_normal);
gbc = makegbc(4, 11, 2, 1);
gbc.anchor = GridBagConstraints.WEST;
gbl.setConstraints(lbl_country, gbc);
add(lbl_country);

lbl_pass1 = new Label("min. 5 chars.");
lbl_pass1.setFont(f_normal);
gbc = makegbc(4, 13, 2, 1);
```

```
    gbc.anchor = GridBagConstraints.WEST;
    gbl.setConstraints(lbl_pass1, gbc);
    add(lbl_pass1);

    lbl_pass2 = new Label("min. 5 chars.");
    lbl_pass2.setFont(f_normal);
    gbc = makegbc(4, 14, 2, 1);
    gbc.anchor = GridBagConstraints.WEST;
    gbl.setConstraints(lbl_pass2, gbc);
    add(lbl_pass2);

    sendbtn.setFont(f_fett);
    gbc = makegbc(2, 16, 1, 1);
    gbc.anchor = GridBagConstraints.CENTER;
    gbl.setConstraints(sendbtn, gbc);
    add(sendbtn);
    sendbtn.addActionListener(this);

    clearbtn.setFont(f_fett);
    gbc = makegbc(3, 16, 1, 1);
    gbc.anchor = GridBagConstraints.CENTER;
    gbl.setConstraints(clearbtn, gbc);
    add(clearbtn);
    clearbtn.addActionListener(this);

    lbl_error = new Label("!!!!! ERROR -- some fields contain erroneous entries !!!!!");
    lbl_error.setFont(f_fehler);
    lbl_error.setForeground(Color.white);
    gbc = makegbc(0, 18, 6, 1);
    gbc.anchor = GridBagConstraints.CENTER;
    gbl.setConstraints(lbl_error, gbc);
    add(lbl_error);

   validate();
}

private GridBagConstraints makegbc(int x, int y, int width, int height)
{
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridwidth = width;
    gbc.gridheight = height;
    gbc.insets = new Insets(1, 2, 1, 4);
    return gbc;
}

// On button click, send data
public void actionPerformed(ActionEvent e)
{
if (e.getActionCommand().equals(clearbtn.getLabel()))
{
    // user pressed CLEAR button, clear all contents
    firstname.setText("");
    setLabelOK(lbl_firstname);
    lastname.setText("");
    setLabelOK(lbl_lastname);
    email.setText("");
    setLabelOK(lbl_email);
    pass1.setText("");
    setLabelOK(lbl_pass1);
    pass2.setText("");
    setLabelOK(lbl_pass2);
    repaint();
    setLabelOK(lbl_country);
    setLabelOK(lbl_ou);
    lbl_pass1.setText("min. 5 chars.");
      lbl_pass2.setText("min. 5 chars.");
    lbl_error.setForeground(Color.white);
}
else
{
    // user pressed SEND Button
```

```
    // check the data
    if (checkData())
    {
      // send the data via Poster if it checked out ok
      //sendData();
      lbl_error.setForeground(Color.blue);
        lbl_error.setText("!!!!! Your Request has been successfully sent !!!!!");
        repaint();
    }
  }
  }

  protected boolean checkData()
  {
    // everything's still ok
    boolean ok = true;

    // check for the length of the name fields
    if (firstname.getText().length() < 3)
    {
      ok = false;
      setLabelError(lbl_firstname);
    }
    else
    {
      setLabelOK(lbl_firstname);
    }
    if (lastname.getText().length() < 3)
    {
      ok = false;
      setLabelError(lbl_lastname);
    }
    else
    {
      setLabelOK(lbl_lastname);
    }

    // check for the length of the password fields
    lbl_pass1.setText("min. 5 chars.");
    lbl_pass2.setText("min. 5 chars.");
    if (pass1.getText().length() < 5)
    {
      ok = false;
      setLabelError(lbl_pass1);
    }
    else
    {
      setLabelOK(lbl_pass1);
    }
    if (pass2.getText().length() < 5)
    {
      ok = false;
      setLabelError(lbl_pass2);
    }
    else
    {
      setLabelOK(lbl_pass2);
    }

    if (ok)
    {
      // check if the two passwords match
      if (!pass1.getText().equals(pass2.getText()))
      {
        lbl_pass1.setText("don't match");
        lbl_pass2.setText("don't match");
        ok = false;
        setLabelError(lbl_pass1);
        setLabelError(lbl_pass2);
      }
      else
      {
        setLabelOK(lbl_pass1);
```

```
        setLabelOK(lbl_pass2);
      }
    }

    // check the email address
    if (!checkEmail(email.getText()))
    {
      ok = false;
      setLabelError(lbl_email);
    }
    else
    {
      setLabelOK(lbl_email);
    }

    // check whether to signal error or not
    if (!ok)
    {
      lbl_error.setForeground(Color.red);
      lbl_error.setText("!!!!! ERROR -- some fields contain erroneous entries !!!!!");
    }
    else
    {
      lbl_error.setForeground(Color.white);
    }

    repaint();
    return ok;
  }

  private void setLabelError(Label lbl)
  {
    lbl.setForeground(Color.red);
    lbl.setFont(f_fett);
  }

  private void setLabelOK(Label lbl)
  {
    lbl.setForeground(Color.black);
    lbl.setFont(f_normal);
  }

  // check for that at least the following exists: x@x.x
  private boolean checkEmail(String email)
  {
    boolean ok = true;
    int temp;
    String tempemail = new String();

    // check for the @ sign and string length before the @
    temp = email.indexOf('@');
    if (temp < 1)
    {
      ok = false;
    }

    if (ok)
    {
      // check for the dot after the @
      tempemail = email.substring(temp + 1);
      temp = tempemail.indexOf('.');
      if (temp < 1)
      {
        ok = false;
      }
    }

    if (ok)
    {
      // check for the domain
      tempemail = tempemail.substring(temp + 1);
      if (tempemail.length() < 1)
      {
```

```
            ok = false;
          }
        }

        return ok;
    }

    protected void sendData()
    {
      // button pushed
      Poster post;
      try
    {
       post = new Poster("https://ksy112.zhwin.ch/cgi-bin/request2");
      }
      catch (java.net.MalformedURLException ex)
      {
       firstname.setText(ex.getMessage());
       repaint();
       return;  // what to do?
      }
      post.addPair("operation", "client-confirmed-form");
      post.addPair("first_name", firstname.getText());
      post.addPair("last_name", lastname.getText());
      post.addPair("email", email.getText());
      post.addPair("country", country.getSelectedItem());
     post.addPair("ou", ou.getSelectedItem());
      post.addPair("passwd3", pass1.getText());
       post.addPair("newkey", "1231231231");
      try
    {
        feedback.setText(post.post());
        repaint();
      }
      catch (Exception ioe)
      {
        // what to do?
        feedback.setText(ioe.getMessage());
        repaint();
      }
    }


    TextField firstname;
    Label lbl_firstname;
    TextField lastname;
    Label lbl_lastname;
    TextField email;
    Label lbl_email;
    TextField pass1;
    Label lbl_pass1;
    TextField pass2;
    Label lbl_pass2;
    Label lbl_error;
    Choice country;
    Label lbl_country;
    Choice ou;
    Label lbl_ou;
    Button sendbtn;
    Button clearbtn;
    TextArea feedback;
    Font f_fett;
    Font f_normal;
    Font f_titel;
    Font f_klein;
    Font f_fehler;
}
```

6.2 Listing of the Poster Class

```
import java.net.*;
import java.util.*;
import java.io.*;


/**
 * This class takes care of setting up a connection to a url. To that connection a
 * string is sent with the command <code>post()</code>. The content-type is set
 * to text/plain.<br>
 * This class was written for the communication with the OpenCA perl script called
 * request2 (a slight variation to the original script request). It probably can be
 * used for communication with other scripts, although taht has not been tested.<br><br>
 *
 * @author  Tobias Lauchenauer
 * @version 1.0
 */
public class Poster
{
  URL url;
  String data;
  URLConnection connection;

  /**
   * Constructor of the class. <br>Given an already initialized <code>URL</code> it
   * copies the reference to it and sets up an emtpy string.<br><br>
   *
   * @param _url requires an initialized <code>URL</code>
   */
  public Poster(URL _url)
  {
  url=_url;
  data = new String();
  };

  /**
   * Constructor of the class. <br>Given a URL address (including the protocol) it
   * sets up the <code>URL</code> and an empty string.<br>
   * <br>example URL:<br>
   * https://ksy112.zhwin.ch:443<br><br>
   *
   * @param _url requires a URL address (including protocol)
   */
  public Poster(String _url) throws MalformedURLException
  {
  url = new URL(_url);
  data = new String();
  }

  /**
   * Clears the contents of the string.
   */
  public void clear()
  {
    data = new String();
  }

  /**
   * Add a data pair to the string. <br>
   * This Method takes the descriptor and the content and encodes them both with the
   * command: <code>URLEncoder.encode()</code>. Then the two strings are concatenated
   * with '=' in between. This new string is then appended to the already existing
   * with '&' as connection.<br><br>
   *
   * @param descriptor the name of the data element
   * @param content the value for the data element
   */
  public void addPair(String descriptor, String content)
  {
    if (!data.equals(""))
    {
```

```
        data = data + "&";
      }

      data = data + URLEncoder.encode(descriptor) + "=" + URLEncoder.encode(content);
    }

    /**
     * Posts the data in the string to the URL. <br>
     * Make sure that the string contains something. Posting with an empty string may
     * have unexpected results. After the string is posted, this method listens for
     * a return from the specified URL and collects the return in a String. This string
     * is the return value for the Method.<br><br>
     *
     * @return the data that is received from the connection after posting the string
     *         is returned
     * @exception IOException if something goes wrong while trying to set up the
     *                        connection or while posting this exception is thrown
     */
    public String post() throws java.io.IOException
    {
      connection = url.openConnection();
      connection.setRequestProperty("content-type", "text/plain");
      connection.setDoOutput(true);
      PrintWriter out = new PrintWriter(connection.getOutputStream());

      out.println(data);
      out.close();

      BufferedReader in = new BufferedReader(
                              new InputStreamReader(connection.getInputStream()));
      StringBuffer buf = new StringBuffer();
      String line;
      // join all input lines
    while ((line=in.readLine()) != null)
      {
        buf.append(line);
        buf.append("\n");
        //return buf.toString();
      }
    in.close();
      // return it*/
     return buf.toString();
    }
}
```

7 Bibliography

[1] Markus Grieder, Stefan Zehnder.
LDAP-Server für X.509 Zertifikate
Projektarbeit Sna 04 - 2000

[2] Larry Wall, Tom Christiansen, Randal L. Schwartz.
Programming Perl
O'Reilly, ISBN 1-56592-149-6

[3] Netscape
Netscape Extensions for User Key Generation Communicator4.0 Version
http://www.netscape.com/eng/security/comm4-keygen.html

[4] Universität Hamburg, FB Informatik – RZ.
Das OpenSSL Handbuch
http://www.pca.dfn.de/dfnpca/certify/ssl/handbuch/ossl095/ossl095.html

[5] SUN.
Fundamentals of Java Security
http://developer.java.sun.com/developer/onlineTraining/Security/Fundamentals/Security.html

[6] SUN.
Trail: Security in Java 2 SDK 1.2
http://java.sun.com/docs/books/tutorial/security1.2/index.html

[7] SUN.
JAVA PLUG-IN HTML SPECIFICATION
http://java.sun.com/products/plugin/1.3/docs/tags.html

[8] SUN.
Signed Applets, Browsers, and File Access
http://developer.java.sun.com/developer/technicalArticles/Security/Signed/index.html

[9] SUN.
DEPLOYING RSA SIGNED APPLETS IN JAVA PLUG-IN
http://java.sun.com/products/plugin/1.2/docs/nsobjsigning.html

[10] SUN.
HOW RSA SIGNED APPLET SUPPORT WORKS IN JAVA PLUG-IN
http://java.sun.com/products/plugin/1.2/docs/netscape.html

[11] Netscape.
Netscape Signing Tool
http://developer.netscape.com/docs/manuals/cms/41/adm_gide/app_sign.htm

[12] Al Williams.
The Postman Always Rings Twice
http://www.webtechniques.com/archives/2000/02/java/

[13] Netscape.
Java Cryptography Architecture API Specification & Reference
http://java.sun.com/j2se/1.3/docs/guide/security/CryptoSpec.html

[14] SUN.
Lesson: Overview of Applets
http://java.sun.com/docs/books/tutorial/applet/overview/index.html