

Bachelorarbeit

Flexible – Eine erweiterbare GUI für den FlexiProvider
Frontend



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Marco Ghiglieri

Fachgebiet Kryptographie und Computeralgebra
Prof. Dr. Johannes Buchmann
Betreuer: Erik Dahmen und Martin Döring

20. Juli 2007

Inhaltsverzeichnis

1	Einleitung	1
2	Layout	2
3	Keystorebereich	4
3.1	Zuletzt geöffnete Keystores	4
3.2	Keystore	4
3.2.1	A.Symmetric keys	5
3.2.2	B.Asymmetric keys	7
3.2.3	C.CA	13
3.2.4	D.Waiting signing requests	13
4	Aktionsbereich und Aktionsleiste	15
4.1	Aktionsleiste	15
4.2	Aktionsbereich	17
4.3	Clientmodus	17
4.3.1	Verschlüsseln von Dateien	19
4.3.2	Entschlüsseln von Dateien	19
4.3.3	Signieren von Dateien	20
4.3.4	Verifizieren von Dateien	20
4.4	Administratormodus	21
4.4.1	CSR bearbeiten	21
4.4.2	Ausgestellte Zertifikate	22
5	Weitere Bereiche	23
5.1	Menüleiste	23
5.1.1	Algorithmen	23
5.2	Informationsleiste	24
5.2.1	Veränderung im Aktionsbereich	24
5.3	Dateimanager	25
6	Technische Grundlagen	26
6.1	Technischer Aufbau	26
6.2	Control-Klasse	26
6.3	Fehlerbehandlung	27
6.4	Keystore Format	27

7	Einschränkungen	29
7.1	Speicherung von Kennwörtern	29
7.2	Erstellung von Zertifikatsketten	29
7.3	Darstellung Keystore	29
7.4	Kompatibilität	29
7.5	Plattformunabhängigkeit	30
8	Anhang	31
8.1	Glossar	31
8.2	Quellen	31

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 20. Juli 2007

Kapitel 1

Einleitung

Die Möglichkeiten, an fremde Daten durch Abhören des Datenstroms zu gelangen, sind mit der Einführung des Internet um ein Vielfaches gestiegen. Um dem Schutzziel Vertraulichkeit gerecht zu werden, mussten hier schnell kryptographische Verfahren zum Einsatz kommen, die eine Ver- und Entschlüsselung ermöglichen.

Desweiteren wollte man garantieren, dass Daten auf dem Weg von einem Benutzer zum anderen Benutzer nicht manipuliert werden. Hier spielt das Schutzziel Integrität eine große Rolle. Um dieses zu gewährleisten, mussten gute Verfahren zur Signaturerstellung eingeführt werden. Viele Anwendungsentwickler implementierten ihre eigenen Verfahren, die sich als nicht sicher erwiesen. Eine gute Alternative, um standardisierte Verfahren in Anwendungen zu implementieren, stellt der FlexiProvider der Technischen Universität Darmstadt da.

Der FlexiProvider ist eine Java-Bibliothek von kryptographischen Modulen. Das Ziel des FlexiProviders^[2] ist es, schnelle und sichere Implementierungen von kryptographischen Algorithmen für Anwendungen bereitzustellen.

Jeder, der mit dem FlexiProvider ver- und entschlüsseln möchte, muss also zunächst Kenntnisse über den FlexiProvider erwerben, um diese Möglichkeit zu nutzen.

An dieser Stelle kommt Flexible ins Spiel. Flexible ist eine graphische Bedienoberfläche für den FlexiProvider. Sie soll es ermöglichen, ohne Vorkenntnisse vom FlexiProvider – nur Kenntnisse in Kryptographie – die Funktionen Verschlüsseln/Entschlüsseln/Signieren und Verifizieren benutzen zu können.

Bei Flexible wurde besonders darauf geachtet, dass es plattformunabhängig und möglichst zu nachfolgenden FlexiProvider Versionen (mit gleichbleibenden Schnittstellen) kompatibel ist.

Die Plattformunabhängigkeit wird durch das von der Eclipse Foundation entwickelte SWT-Framework gewährleistet. Dieses Framework wurde so entwickelt, dass es möglich ist, auf fast jedem Betriebssystem die typische Optik ohne Änderung am Quellcode zu erzeugen. SWT ist für viele Betriebssysteme verfügbar.

Da Flexible auf einem SVN vom Fachbereich Informatik der Technischen Universität Darmstadt untergebracht ist, bezieht sich dieses Dokument auf die Revision 889.^[1]

Kapitel 2

Layout

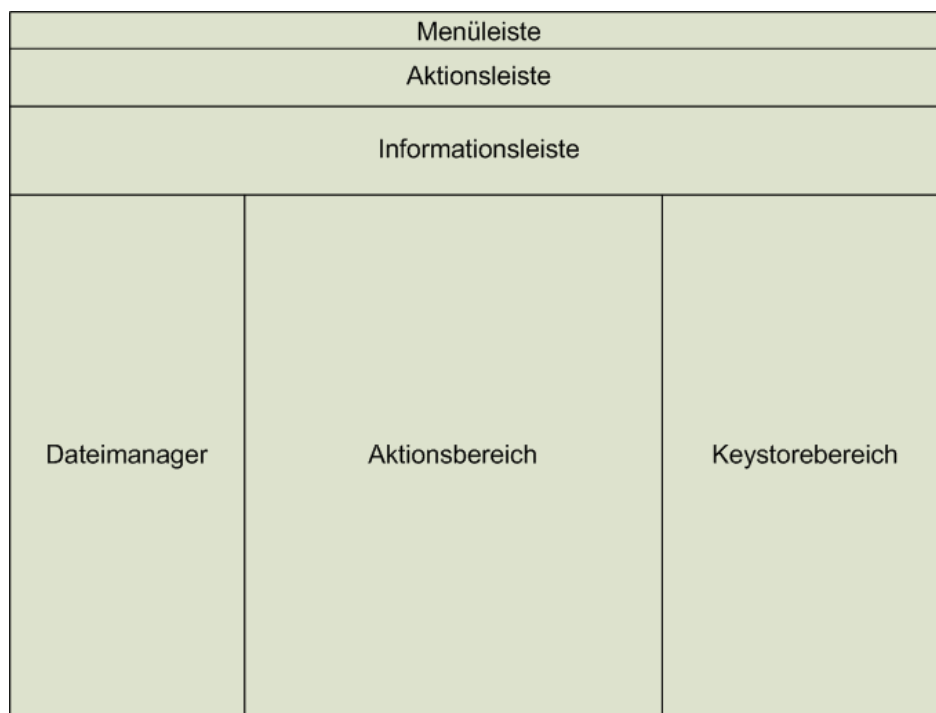


Abbildung 2.1: Groblayout der Bedienoberfläche

Die Bedienoberfläche ist in folgende sechs Bereiche unterteilt:

- Menüleiste
- Aktionsleiste
- Informationsleiste
- Dateimanager
- Aktionsbereich
- Keystorebereich

Jeder Bereich befindet sich – wie in Abbildung 2.1 erkennbar – an einer festen Stelle der Bedienoberfläche. Eine Veränderung der Größenverhältnisse ist nur zwischen den Bereichen Dateimanager, Aktionsbereich und Keystore möglich.

Es gibt einige Prozesse, die in keinem dieser Bereiche dargestellt werden. Diese nennen wir im folgenden Dialoge. Funktionen, die in den Bereichen dargestellt werden, heißen Composites.

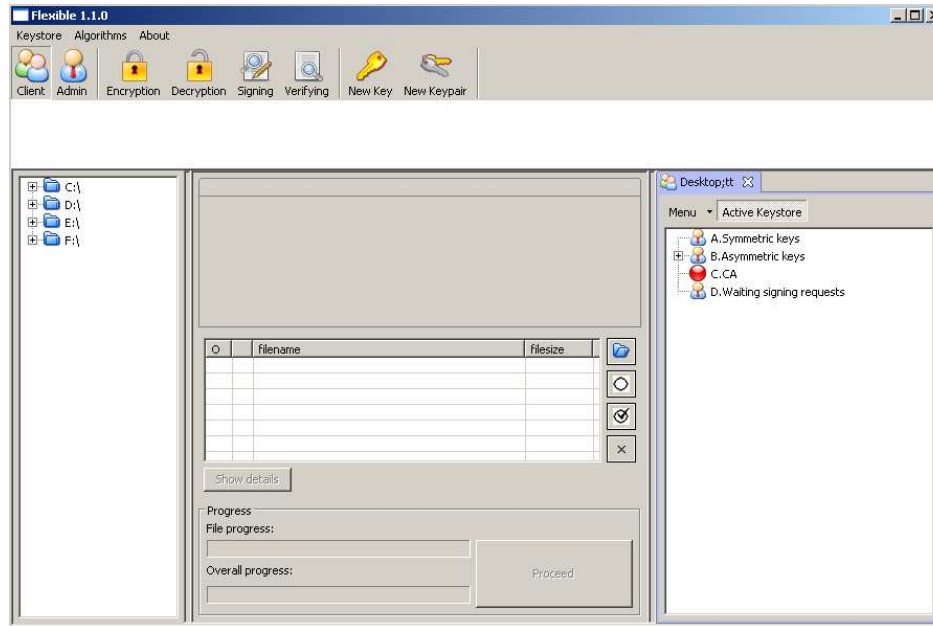


Abbildung 2.2: Flexible

Kapitel 3

Keystorebereich

Der Keystorebereich befindet sich im rechten Bereich von Flexible. Dieser ist mit Registern, die es ermöglichen mehrere Keystores geöffnet zu haben, organisiert. Die Unterscheidung geschieht dabei durch den Dateinamen der einzelnen Keystores.

Ein Keystore stellt das Herzstück der Schlüsselverwaltung da. Hier werden alle Schlüssel, Zertifikate und wartende CSRs angezeigt.

3.1 Zuletzt geöffnete Keystores

Beim Start von Flexible wird ein Dialog zum Auswählen von zuletzt geöffneten Keystores angezeigt. Es werden nur die Keystores angezeigt, die auf dem Datenträger vorhanden sind. Nicht in der Liste vorhandene Keystores müssen durch *Open other...* geöffnet werden. Ein neuer Keystore kann durch Wählen von *New* erzeugt werden.

Dieser Dialog ist auch im Menü *Keystore* → *Open keystore* zu finden.

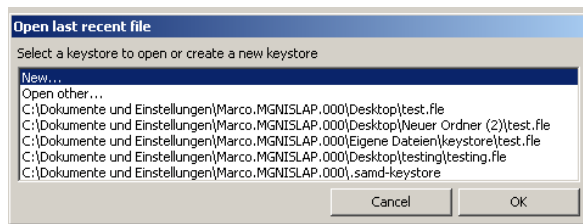


Abbildung 3.1: Zuletzt geöffnete Keystores

3.2 Keystore

Die geöffneten Keystores werden in Reitern im Keystorebereich dargestellt. Jeder Reiter und jeder geöffnete Keystore besitzt ein Menü, welches unter dem Button *Menu* zu finden ist. Daneben befindet sich ein Button, der angibt, ob der Keystore gerade aktiv gewählt ist. Alle Funktionen aus dem Aktionsbereich werden nur im aktiven Keystore ausgeführt. Dagegen werden Funktionen, die über ein Menü innerhalb des Keystores ausgeführt werden, auch in diesem bearbeitet.

In der Keystore-Ansicht darunter wird der Inhalt des Keystores graphisch dargestellt. Unterschieden werden die Kategorien:

- *A.Symmetric keys* – Symmetrische Schlüssel
- *B.Asymmetric keys* – Asymmetrische Schlüssel
- *C.CA* – Zertifizierungsstelle
- *D.Waiting signing requests* – Wartende Zertifikatsanforderungen

Deaktivierte Kategorien sind durch einen roten Punkt neben der Kategoriebezeichnung markiert.

Administratormodus und Clientmodus unterscheiden sich durch verschiedene Funktionalitäten in den einzelnen Kategorien.

Die Kategorien sind durch eine Baumstruktur dargestellt.

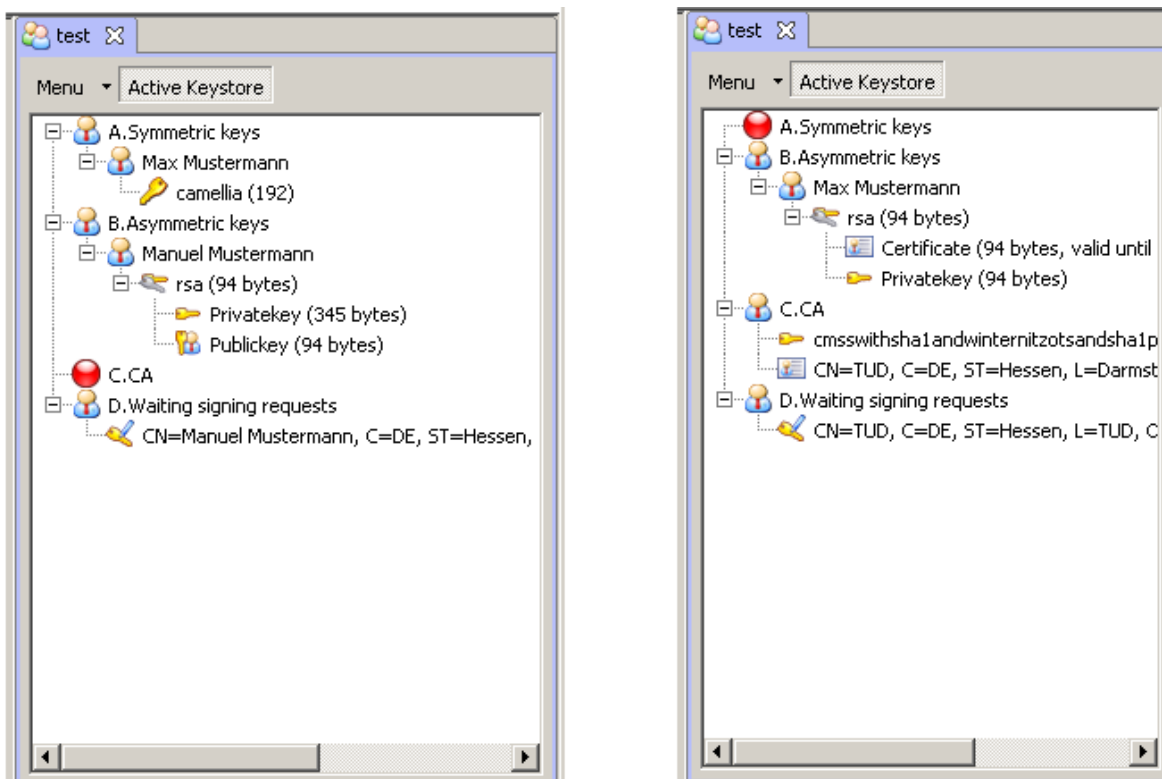


Abbildung 3.2: Keystore links: Clientmodus, rechts: Administratormodus

3.2.1 A.Symmetric keys

Im Administratormodus ist die Kategorie *A.Symmetric keys* deaktiviert.

Im Clientmodus bekommt der Benutzer alle von ihm erstellten oder importierten symmetrischen Schlüssel, die im gewählten Keystore gespeichert sind, angezeigt. In der ersten Ebene der Ansicht werden Schlüssel nach den vergebenen Kontaktnamen sortiert. Eine Ebene tiefer befinden sich die Schlüssel, die dem Kontakt zugeordnet sind. Die Beschriftung der Schlüssel

besteht immer aus dem Algorithmusnamen und der Bytelänge des Schlüssels.

Das Kontextmenü, welches durch Rechtsklick erscheint, bietet in der obersten Ebene die Funktionen zum Erstellen eines neuen symmetrischen Schlüssels (siehe 3.2.1.1) und das Importieren eines symmetrischen Schlüssels aus einer Datei in den Keystore (siehe 3.2.1.2).

Zusätzlich bietet das Kontextmenü bei Rechtsklick auf den Kontaktnamen noch die Möglichkeit zum Löschen des kompletten Kontakts. Es werden also alle darunterliegenden Schlüssel gelöscht.

Auf dem Schlüssel selbst werden die Funktionen Exportieren des Schlüssels (siehe 3.2.1.3) und Löschen des Schlüssels (3.2.1.4) angeboten.

3.2.1.1 Erstellen eines neuen symmetrischen Schlüssels

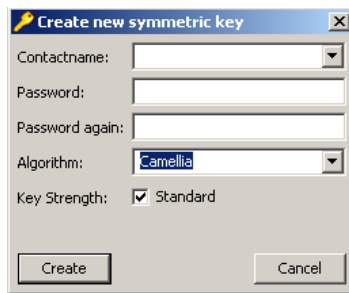


Abbildung 3.3: Neuer symmetrischer Schlüssel

Um einen neuen symmetrischen Schlüssel zu erzeugen, gibt man in das Dialogfenster den Kontaktnamen (Auswahl aus den bestehenden Kontakten möglich) und ein Passwort ein. Den Algorithmus wählt man über das Auswahlfenster *Algorithm*. Als Schlüssellänge kann entweder eine *Standard*-Schlüssellänge oder eine benutzerdefinierte Länge benutzt werden.

Das eingegebene Kennwort wird als Keystore-Kennwort zur Speicherung des symmetrischen Schlüssels verwendet.

Der generierte Schlüssel wird unter *A.Symmetric keys* abgelegt.

Wenn ein Schlüssel über das Kontextmenü eines Kontaktnamens erstellt wird, ist das Feld *Contact* schon ausgefüllt, kann aber verändert werden.

Technischer Ablauf

1. Dialog zur Erstellung eines symmetrischen Schlüssels anzeigen
2. Benutzerdaten validieren
3. Wenn *Standard* Schlüssellänge gewählt ist, an den Algorithmus -1 ausliefern, sonst den eingegebenen Wert
4. Schlüssel erstellen
5. Erstellten Schlüssel in den Keystore mit dem *KeyStoreTyp.SECRETKEY* speichern

3.2.1.2 Importieren eines symmetrischen Schlüssels

Importiert wird ein symmetrischer Schlüssel, in dem er mit dem *Datei öffnen*-Dialog ausgewählt wird. Daraufhin muss das Transportkennwort eingegeben werden. Wurde der Importvorgang über das Kontextmenü eines Kontakts gestartet, wird der symmetrische Schlüssel dort gespeichert. Ansonsten erhält man die Aufforderung, einen *Contact* auszuwählen bzw. einen neuen durch Eingabe eines Namens zu erstellen.

Ein Speicherkennwort für den Keystore muss nun noch zugewiesen werden.

Laden des Schlüssels

Die Datei wird als *FileInputStream* eingelesen. Wenn das Entschlüsseln mit einem leeren Kennwort mit dem Algorithmus *PBEWithMD5AndDES* fehl schlägt, wird die *Exception* abgefangen und ein Kennwort vom Benutzer erwartet. Der nun erhaltene Bytestrom ist die serialisierte *SecretKey* Klasse und kann weiterverarbeitet werden.

3.2.1.3 Exportieren eines symmetrischen Schlüssels

Zum Exportieren eines symmetrischen Schlüssels muss der Benutzer nach Wahl des Speicherorts durch einen *Speichern unter*-Dialog ein Kennwort eingeben. Dieses dient als Transportkennwort.

Speichern des Schlüssels

Da es für symmetrische Schlüssel kein standardisiertes Speicherformat gibt, wird beim Exportieren die Schlüsselklasse *SecretKey* mit Hilfe eines *ObjectOutputStream* serialisiert und als Bytestrom ausgegeben. Damit dieser Strom über das Transportkennwort gesichert ist, wird er, bevor er auf den Datenträger geschrieben wird, mit dem Algorithmus *PBEWithMD5AndDES* verschlüsselt.

3.2.1.4 Löschen von symmetrischen Schlüsseln

Das Löschen eines symmetrischen Schlüssel kann direkt anhand des *KeystoreFormat* (siehe Kapitel 6.4) durchgeführt werden. Der Benutzer wird mit einer Sicherheitsfrage auf den Löschvorgang hingewiesen.

3.2.2 B.Asymmetric keys

In der Keystorekategorie *B.Asymmetric keys* werden im Administrator- und im Clientmodus alle erstellten und importierten Schlüssel und Zertifikate, die im gewählten Keystore gespeichert sind, angezeigt.

In der ersten Ebene der Ansicht werden Schlüssel und Zertifikate nach den vorgegebenen Kontaktnamen sortiert. Eine Ebene tiefer befinden sich die Algorithmusnamen der gespeicherten Schlüssel. Unter dem Algorithmusnamen befinden sich die zusammengehörigen privaten Schlüssel und öffentlichen Schlüssel oder Zertifikate.

Finden von zusammengehörigen Schlüsseln oder Zertifikaten

Die Schlüssel im Java-Keystore werden mit Hilfe des *KeystoreFormat* (siehe Kapitel 6.4) so gespeichert, dass der Alias im Java-Keystore jedes Schlüssels oder Zertifikates den MD5-Hashwert des öffentlichen Schlüssels beinhaltet. Damit können problemlos zusammengehörige Schlüssel/Zertifikate gefunden und zugeordnet werden.

Im Clientmodus bietet das Kontextmenü vom Kategorienamen und dem Kontakt die Möglichkeit ein asymmetrisches Schlüsselpaar zu erstellen (siehe 3.2.2.1) und ein Zertifikat (siehe 3.2.2.2), sowie PKCS#12 zu importieren (siehe 3.2.2.3). Zusätzlich können alle Kontaktschlüssel auf dem Kontakt gelöscht werden (siehe 3.2.2.4).

Das Schlüsselpaar bietet die Möglichkeit zum Erstellen von einem CSR (siehe 3.2.2.5) und zum Exportieren als PKCS#12 (siehe 3.2.2.6) wie auch das Löschen aller darunter liegenden Zertifikate und Schlüssel (siehe 3.2.2.7).

Zertifikate haben die Funktionen Exportieren (siehe 3.2.2.8), Löschen (siehe 3.2.2.9) und zusätzliche Details anzeigen (siehe 3.2.2.10).

Im Administratormodus wird im Kontextmenü des Kategorienamens die Erstellung von Benutzerzertifikaten angeboten (siehe 3.2.2.11). Die Zertifikatsfunktionen sind im Vergleich zum Clientmodus um Zertifikatserneuerung erweitert (siehe 3.2.2.12).

3.2.2.1 Erstellen eines neuen asymmetrischen Schlüsselpaars

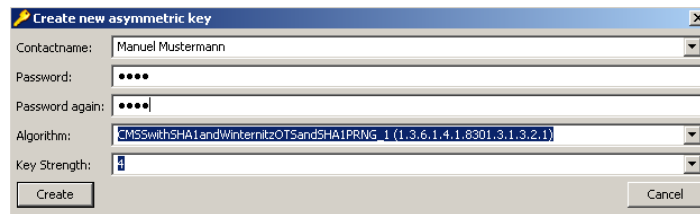


Abbildung 3.4: Neues asymmetrisches Schlüsselpaar

Das Erstellen eines asymmetrischen Schlüsselpaars läuft ähnlich wie das Erstellen eines symmetrischen Schlüssels ab. Neben Kontaktnamen, Passwort, Algorithmus und Schlüssellänge wird die OID mit angegeben.

Das eingegebene Passwort wird als Keystore-Kennwort für den privaten Schlüssel verwendet. Das generierte Schlüsselpaar wird unter *B.Asymmetric keys* abgelegt.

Wenn ein Schlüsselpaar über das Kontextmenü eines Kontaktnamens erstellt wird, ist das Feld *Contact* schon ausgefüllt, kann aber verändert werden.

Anzeige der Algorithmen

In dem Auswahlfenster *Algorithm* werden alle Algorithmen angezeigt, die im FlexiProvider eine OID registriert haben. Wenn keine OID registriert ist, können sie nicht richtig vom Java-Keystore verarbeitet werden. Deswegen werden sie ausgeblendet.

Technischer Ablauf

1. Dialog zur Erstellung eines asymmetrischen Schlüsselpaars anzeigen
2. Benutzerdaten validieren
3. Schlüsselpaar erstellen
4. Speichere öffentlichen Schlüssel mit dem *KeystoreTyp.PUBLICKEY* im Keystore
5. Speichere privaten Schlüssel mit eingegebenem Passwort und dem *Keystore-Typ.PRIVATEKEY* im Keystore

3.2.2.2 Import von Zertifikaten

Der Import von Zertifikaten wird im DER-kodiertem und im Base64-kodiertem Format unterstützt.

Das Zertifikat wählt man mit einem *Datei öffnen*-Dialog. Daraufhin wird das Zertifikat importiert.

Sollte kein Schlüssel im Keystore gefunden werden, der den gleichen Hash, wie der Schlüssel im Zertifikat hat, wird nach einem *Contact* gefragt.

Eine Besonderheit beim Import ist, dass Zertifikate mit einem öffentlichem Schlüssel, der zu einem noch offenen CSR gehört, automatisch zugeordnet werden und somit der CSR gelöscht wird.

Technischer Ablauf

1. Öffnen des systemeigenen *Datei öffnen*-Dialogs
2. Datei validieren
3. Import des Zertifikats
4. Prüfen, ob Hash des öffentlichen Schlüssels aus dem Zertifikat mit einem Hash eines öffentlichen Schlüssels von einem CSR übereinstimmt
5. Wenn ja, CSR löschen
6. Prüfen, ob Hash des öffentlichen Schlüssels aus dem Zertifikat mit einem Hash eines Schlüssels im Keystore übereinstimmt und Kontaktnamen übernehmen
7. Wenn nein, Dialog zur Eingabe des Kontaktnamens öffnen
8. Speichern des Zertifikates im Keystore als *textitKeystoreTyp.CERTIFICATE*

3.2.2.3 Import als PKCS#12

Der Import von Schlüsseln und Zertifikaten kann als PKCS#12 geschehen. Nach Auswahl der zu importierenden .p12-Datei wird das Transportkennwort abgefragt. Das Zertifikat und der private Schlüssel werden in einem wählbaren Kontakt gespeichert.

Technischer Ablauf

1. Öffnen des systemeigenen Datei öffnen Dialogs
2. Datei validieren
3. Import des PKCS#12
4. PKCS#12 mit Transportkennwort entschlüsseln
5. Extrahieren der Schlüssel und des Zertifikats
6. Dialog zur Kontakteingabe öffnen
7. Speichern des Zertifikates im Keystore als `textitKeystoreTyp.CERTIFICATE`
8. Speichern des privaten Schlüssels mit frei wählbarem Kennwort als `Keystore-Typ.PRIVATEKEY`

3.2.2.4 Löschen eines Kontakts

Es werden alle Schlüssel mit dem gleichen Alias gesucht und gelöscht.

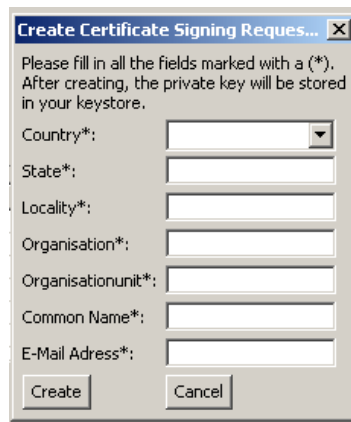
3.2.2.5 Erstellen eines CSR

Abbildung 3.5: Neuer CSR

In dem Dialogfenster müssen nun alle für ein Zertifikate relevanten Felder ausgefüllt werden: Land, Bundesland, Stadt, Organisation, Organisationseinheit, Common Name und E-Mail Adresse.

Das fertige CSR wird unter *D.Waiting signing requests* abgelegt.

Speicherung eines CSR im Java-Keystore

Im Java-Keystore können keine beliebigen Einträge gespeichert werden, deswegen werden CSRs immer als unsignierte Dummy-Zertifikate im Java-Keystore gespeichert.

Technischer Ablauf

1. Dialog zum Erstellen eines CSR öffnen
2. Benutzereingaben validieren
3. Laden des öffentlichen Schlüssels direkt oder aus dem Zertifikat
4. Erstellen eines Dummy-Zertifikats ohne Signatur mit dem öffentlichen Schlüssel
5. Sichern des Zertifikats als *KeystoreTyp.CSR*

3.2.2.6 Export als PKCS#12

Der Export als PKCS#12 kann nur durchgeführt werden, wenn ein privater Schlüssel und ein Zertifikat verfügbar sind. Der Benutzer wird aufgefordert ein Transportkennwort einzugeben. Der *Datei speichern*-Dialog wird geöffnet und die Datei im p12 Format auf den Datenträger geschrieben.

3.2.2.7 Löschen eines Schlüsselpaars

Es werden hier alle Schlüssel und Zertifikate gelöscht, die den Hash des öffentlichen Schlüssels haben. Der Benutzer wird mit einer Sicherheitsfrage auf den Löschvorgang hingewiesen.

3.2.2.8 Export eines Zertifikat

Der Export von Zertifikaten wird im DER-kodiertem und im Base64-kodiertem Format unterstützt. Ein Zertifikat wird aus dem Keystore exportiert indem der Benutzer ein Ziel und ein Format wählt.

3.2.2.9 Zertifikat löschen

Das Löschen eines Zertifikats kann direkt anhand des KeystoreFormat (siehe Kapitel 6.4) durchgeführt werden. Der Benutzer wird mit einer Sicherheitsfrage auf den Löschvorgang hingewiesen.

3.2.2.10 Zertifikatsinformationen anzeigen

Diese Funktion zeigt im Keystorebereich alle relevanten X.509 Zertifikatsinformationen an (siehe Abbildung 3.6).

3.2.2.11 Benutzerzertifikat erstellen

Benutzerzertifikate können im Administratormodus direkt erstellt werden. Der Umweg über ein CSR ist nicht nötig. Der Ablauf besteht aus Erstellung eines asymmetrischen Schlüsselpaars und Erstellung des Zertifikats.

Diese Funktion kann nur ausgeführt werden, wenn ein CA-Zertifikat im Keystore existiert.

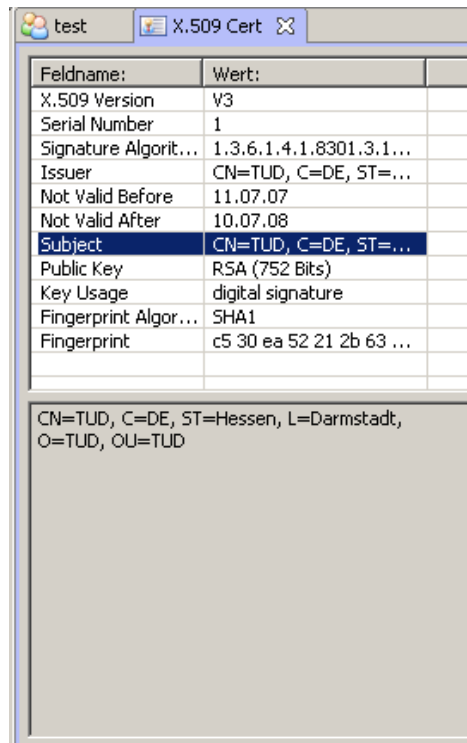


Abbildung 3.6: Zertifikatsinformationen anzeigen

Technischer Ablauf

1. Dialog für Erstellung eines asymmetrischen Schlüsselpaars öffnen
2. Benutzereingaben validieren
3. Schlüsselpaar erzeugen
4. Dialog für Eingabe von Zertifikatsinformationen und Signaturalgorithmus öffnen
5. Benutzereingaben validieren
6. Erzeugen des X.509 Zertifikats mit dem erzeugten Schlüsselpaar
7. Signieren des Zertifikats mit dem privaten CA-Schlüssel
8. Speichern des Zertifikates im Keystore als *KeystoreTyp.CERTIFICATE*

3.2.2.12 Erneuern von Zertifikaten

Die Erneuerung von Zertifikaten gibt es nicht als Funktion. Es wird ein neues Benutzerzertifikat mit dem gleichen Distinguished Name und dem gleichen Schlüssel erzeugt. Das Datum kann neu eingegeben werden. Der Ablauf ist analog zum Erstellen eines Benutzerzertifikats. Der einzige Unterschied ist, dass kein neues Schlüsselpaar erstellt, sondern das vorhandene

Schlüsselpaar wieder verwendet wird.

3.2.3 C.CA

Im Clientmodus ist die Kategorie *C.CA* deaktiviert.

Im Administratormodus wird hier das CA-Zertifikat mit dem privaten Schlüssel angezeigt. Es darf nur ein CA-Zertifikat und nur ein privater Schlüssel im Keystore existieren.

Das Kontextmenü, welches bei Klick auf die Kategoriebezeichnung erscheint, bietet die Funktion ein CA-Zertifikat und einen privaten Schlüssel zu erstellen, falls diese nicht existieren (siehe 3.2.3.1)

Das CA-Zertifikat kann exportiert (siehe 3.2.2.8) oder erneuert werden (siehe 3.2.2.12). Außerdem kann man sich Details anzeigen lassen (siehe 3.2.2.10)

3.2.3.1 CA-Zertifikat erstellen

Es wird ein Benutzerzertifikat erstellt, welches als CA markiert ist und die entsprechenden Key-USages automatisch gesetzt bekommt. Der Vorgang unterscheidet sich nicht zum Erstellen von Benutzerzertifikaten.

3.2.4 D.Waiting signing requests

In der Kategorie *Waiting signing requests* befinden sich alle CSRs.

Im Administratormodus können diese mit *Proceed CSR* bearbeitet werden (siehe 3.2.4.3).

Der Clientmodus bietet Funktionen zum Exportieren auf einen Datenträger (siehe 3.2.4.1) und zum Löschen eines CSRs an (siehe 3.2.4.2).

3.2.4.1 Exportieren eines CSRs

CSRs sind im Keystore als unsignierte Dummy-Zertifikate gespeichert. Da laut PKCS#10-Standard CSRs signiert sein sollen, wird der Benutzer beim Exportvorgang aufgefordert, das Kennwort des privaten Schlüssels, der zum öffentlichen Schlüssel im Dummy-Zertifikat passt, einzugeben. Der Speicherort wird über einen *Datei speichern*-Dialog ausgewählt.

Technischer Ablauf

1. Suchen des privaten Schlüssels zu dem im CSR gespeicherten öffentlichen Schlüssel
2. Abfrage des Kennworts des privaten Schlüssels
3. Angabe des Signaturalgorithmus
4. Öffnen des *Datei speichern*-Dialogs des Betriebssystems
5. Ausgabeformat und Name im Dialog wählen
6. Export

3.2.4.2 Löschen eines CSRs

Das Löschen eines CSRs, kann direkt anhand des KeystoreFormat (siehe Technische Grundlagen: Kapitel 6.4) durchgeführt werden. Der Benutzer wird mit einer Sicherheitsfrage auf den Löschvorgang hingewiesen.

3.2.4.3 CSR bearbeiten

Die Funktion CSR bearbeiten wird nicht im Keystorebereich ausgeführt, sondern im Aktionsbereich (siehe Kapitel 4). Es findet lediglich eine Übernahme des Schlüsselnamens zwischen Keystorebereich und Aktionsbereich statt. Die CSR-Bearbeitung wird dort durchgeführt (siehe Aktionsbereich).

Kapitel 4

Aktionsbereich und Aktionsleiste

4.1 Aktionsleiste



Abbildung 4.1: Aktionsleiste Clientmodus



Abbildung 4.2: Aktionsleiste Administratormodus

Die Aktionsleiste besteht aus einem statischen und einem dynamischen Teil. Der statische Teil beinhaltet die beiden linken Menüpunkte. Diese Menüpunkte sind im Benutzermodus Administrator und Client sichtbar. Sie werden benutzt, um zwischen den beiden Modi umzuschalten. Die Buttons haben ein gedrücktes Aussehen, wenn sie aktiv sind. Es kann immer nur einer der beiden Buttons aktiv sein. Rechts neben dem statischen Teil folgt der dynamische Teil mit den Funktionen für den jeweiligen Benutzermodus. Im Clientmodus stehen die folgenden Funktionen zur Verfügung:

- *Encryption* – Entschlüsseln von Dateien
- *Decryption* – Verschlüsseln von Dateien
- *Signing* – Signieren von Dateien
- *Verifying* – Verifizieren von Dateien
- *New Key* – neuer symmetrischer Schlüssel
- *New Keypair* – neues asymmetrisches Schlüsselpaar

Bis auf die beiden Schlüsselerstellungsdialoge (*New Key* und *New Keypair*) öffnen sich die Funktionen im Aktionsbereich. Funktionen, die bereits geöffnet sind, werden in den Vordergrund gebracht.

Die Funktionen *New Key* und *New Keypair* öffnen einen Dialog und haben keine Auswirkungen auf den Aktionsbereich.

Im Administratormodus gibt es die Möglichkeit Benutzerzertifikate (*New Usercertificate*) zu erstellen. Diese Funktion wird ebenfalls als Dialog geöffnet.

Ein geöffneter Keystore ist Voraussetzung für alle Funktionen. Falls kein Keystore geöffnet ist, wird eine entsprechende Fehlermeldung ausgegeben.

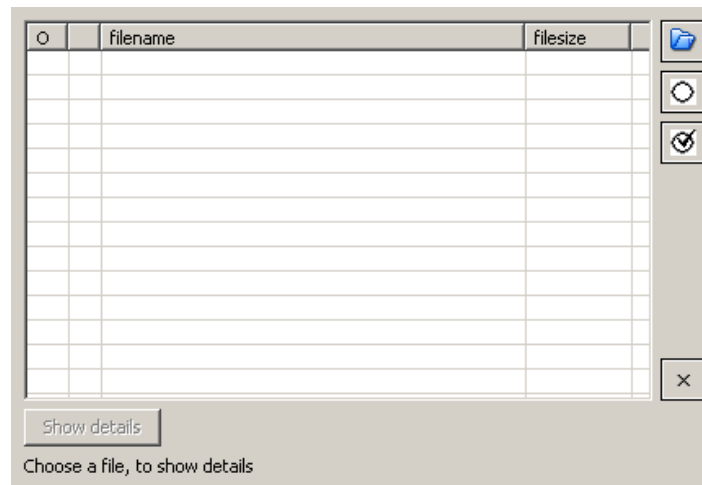


Abbildung 4.4: Dateiliste

Dateiliste In der Dateiliste werden alle Dateien angezeigt, die bei Klick auf *Proceed* mit der aktiven Funktion bearbeitet werden sollen. Neben dem Einfügen von Dateien durch Doppelklick im Dateimanager, bietet die Dateiliste auch eine Funktion zum Hinzufügen an. Diese Funktion wird mit einem systemeigenen Dialog angezeigt.

Außerdem existieren Buttons zum Markieren/Demarkieren von Dateien und zum Löschen der markierten Dateien.

Die Dateien werden mit Dateigröße, Dateinamen und Icon angezeigt. Markierungen erkennt man an der Checkbox neben dem Namen.

Ist eine Datei markiert, werden zusätzliche Details durch Klick auf *Show details* eingeblendet.

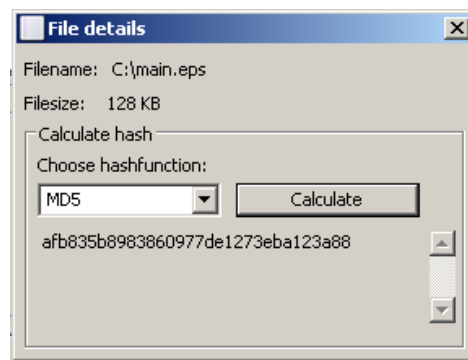


Abbildung 4.5: Details anzeigen

Details anzeigen Im sich öffnenden Dialog *File details* werden Dateiname sowie Dateigröße angezeigt. Weiterhin kann man einen Hash der Datei berechnen. Dazu wählt man eine Hashfunktion und klickt auf *Calculate*. Daraufhin erscheint der Hashwert im Textfeld.

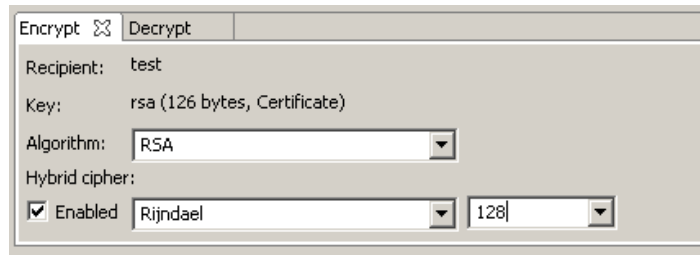


Abbildung 4.6: Verschlüsseln

4.3.1 Verschlüsseln von Dateien

Die zu verschlüsselnden Dateien müssen zuerst in die Dateiliste eingefügt werden. Das Einfügen von Dateien kann durch Doppelklick im Dateimanager oder durch den *Datei öffnen*-Dialog von der Dateiliste erfolgen.

Anschließend muss im Keystore ein symmetrischer Schlüssel oder ein Zertifikat mit Doppelklick ausgewählt werden. Informationen über den Schlüssel oder das Zertifikat werden unter *Key* angezeigt.

Wenn es ein symmetrischer Schlüssel ist, stimmt der Algorithmus mit dem des Schlüssels überein. Bei asymmetrischen Verfahren muss ein Algorithmus gewählt werden.

Hybride Verschlüsselung wird nur bei Zertifikaten unterstützt. Zusätzlich muss dann ein symmetrischer Algorithmus und eine Schlüssellänge gewählt werden.

Sollte ein Zertifikat nicht zum Verschlüsseln geeignet sein, wird eine entsprechende Fehlermeldung erzeugt.

Mit Klick auf den Button *Proceed* startet das Verschlüsseln, es muss lediglich das Ausgabeverzeichnis gewählt werden. Dort werden nun die verschlüsselten Dateien und eine Session-XML abgelegt.

4.3.2 Entschlüsseln von Dateien

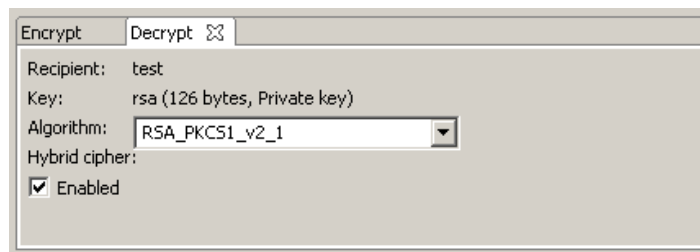


Abbildung 4.7: Entschlüsseln

Für das Entschlüsseln von Dateien benötigt man einen privaten oder symmetrischen Schlüssel. Den Schlüssel wählt man mit Doppelklick aus dem Keystore aus. Die Schlüsselinformationen werden angezeigt.

Der Algorithmus entspricht bei symmetrischen Verfahren dem Schlüsselalgorithmus und bei privaten Verfahren muss dieser unter *Algorithm* gewählt werden. Hybride Entschlüsselung wird nur bei privaten Schlüsseln angeboten. Eine Angabe des hybriden Verfahrens ist nicht notwendig und wird automatisch ermittelt.

Eine Alternative zur manuellen Konfiguration ist die beim Verschlüsseln erstellte Session-XML Datei. Sobald diese der Dateiliste hinzugefügt wurde, versucht Flexible, automatisch die Einstellungen vorzunehmen. Sollte dies nicht möglich sein, wird eine Fehlermeldung angezeigt.

Beim Klick auf den Button *Proceed* startet das Entschlüsseln, es muss lediglich das Ausgabeverzeichnis gewählt werden. Dort werden nun die entschlüsselten Dateien abgelegt.

4.3.3 Signieren von Dateien

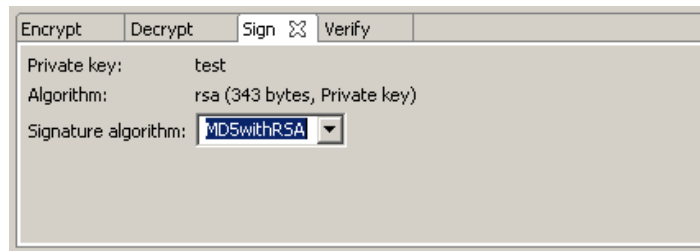


Abbildung 4.8: Signieren

Zum Signieren von Dateien benötigt man einen privaten Schlüssel, den man mit einem Doppelklick aus dem Keystore auswählen kann, und einen Signaturalgorithmus. Beim Klick auf den Button *Proceed* werden alle in der Dateiliste befindlichen Dateien signiert. Zur einfacheren Verifizierung wird ebenfalls eine Session-XML erzeugt.

4.3.4 Verifizieren von Dateien

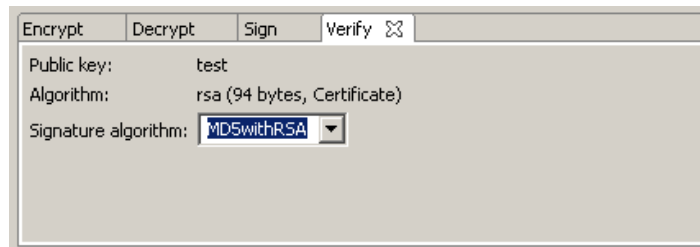


Abbildung 4.9: Verifizieren

Zum Verifizieren von Dateien benötigt man ein Zertifikat, das mit einem Doppelklick aus dem Keystore ausgewählt wird.

Alternativ kann auch die Session-XML in die Dateiliste gezogen werden und Flexible versucht die Einstellungen automatisch vorzunehmen.

Beim Klick auf *Proceed* werden die Dateien verifiziert und nach Abschluss wird eine Liste mit den Ergebnissen der einzelnen Dateien angezeigt.

Kapitel 5

Weitere Bereiche

5.1 Menüleiste

Die Menüleiste ist der oberste Bereich von Flexible.

Sie enthält die Menüpunkte *Keystore*, *Algorithms* und *About*.

Unter *Keystore* findet man die Funktion zum Öffnen der zuletzt geöffneten Keystores (siehe Kapitel 3.1).

Den Algorithmen Editor findet man unter *Algorithms* (siehe 5.1.1). Dort besteht auch die Möglichkeit, die Konfigurationsdatei für Algorithmen zurückzusetzen.

Beim Klick auf *About* öffnet sich ein Dialog mit allen wichtigen Programminformationen über Flexible.



Keystore Algorithms About

Abbildung 5.1: Menüleiste

5.1.1 Algorithmen

Da die möglichen Schlüssellängen der einzelnen Schlüssel nicht aus dem FlexiProvider ausgelesen werden können, stellt Flexible eine Oberfläche zum Bearbeiten der vom Benutzer eingegebenen Werte zur Verfügung. Eine Datei mit oft verwendeten Schlüssellängen wird mitgeliefert.

In dem Auswahlfeld sind alle verfügbaren Algorithmen aus dem FlexiProvider enthalten. Dort wählt man den zu bearbeitenden Algorithmus und mit Klick auf *Choose* wird das Textfenster mit dem XML Snippet des jeweiligen Algorithmus gefüllt.

Eine Schlüssellänge kann hinzugefügt werden, in dem man in das Textfeld die Länge eingibt und *Add size* klickt. Eine Schlüssellänge löscht man, in dem man im Auswahlfenster die entsprechende Schlüssellänge auswählt und auf *Delete size* klickt.

Die hier eingegebenen Schlüssellängen werden in den Schlüsselerstellungs-Dialogen von Flexible angezeigt.

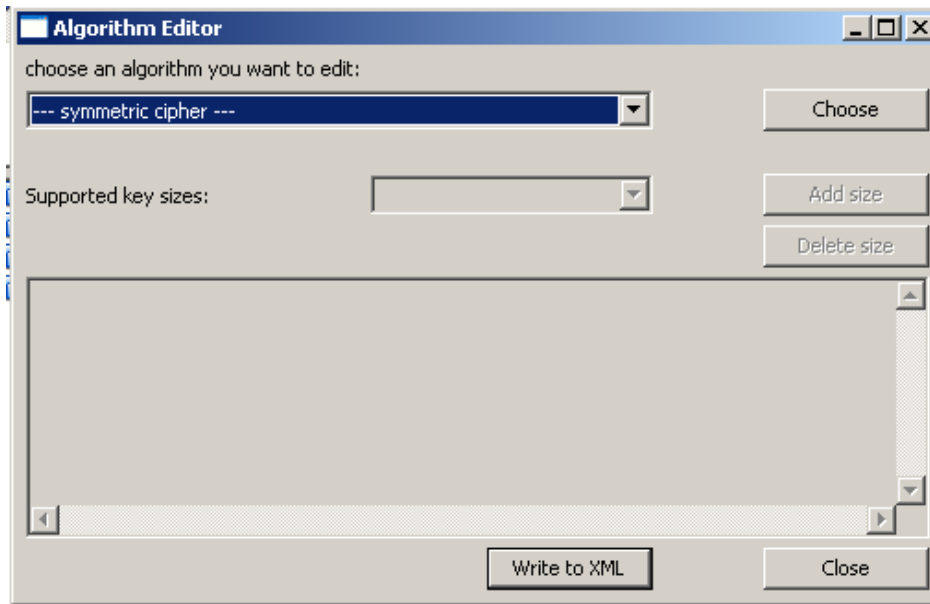


Abbildung 5.2: Editor für Algorithmen

5.2 Informationsleiste

Da der Hintergrund komplett weiß ist, hebt sich dieser Bereich deutlich von der restlichen Bedienoberfläche ab.

In dieser Leiste erscheinen Informationstexte, die eine kurze Einführung in die Bedienung der im Aktionsbereich geöffneten Funktion geben. Die aktive Funktion wird an der rechten Seite durch ein Icon gekennzeichnet. Die Icons sind die gleichen, die auch in der Aktionsleiste angezeigt werden.

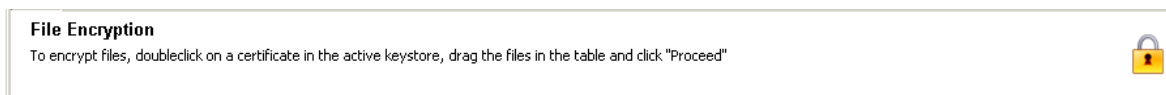


Abbildung 5.3: Informationsleiste

5.2.1 Veränderung im Aktionsbereich

Die Informationsleiste wird durch einen Listener, der Änderungen des Aktionsbereiches überwacht, geändert.

Die angezeigten Informationstexte sowie die Icons^[4] sind in einer extra Klasse untergebracht.

5.3 Dateimanager

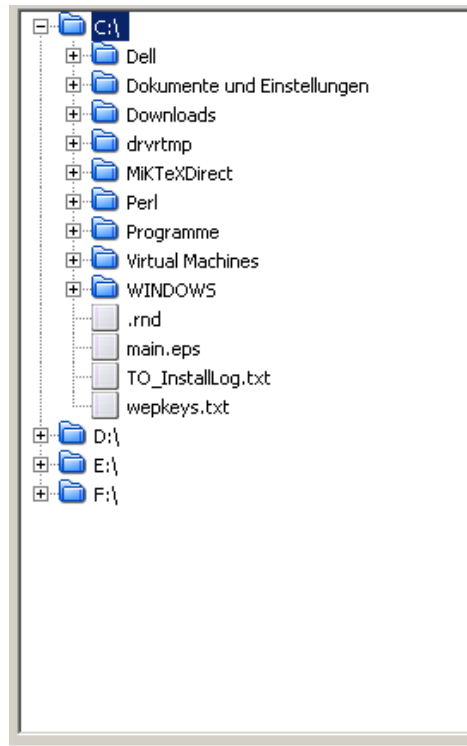


Abbildung 5.4: Dateimanager

Der Dateimanager befindet sich an der linken Seite von Flexible. Der Dateimanager bietet die Möglichkeit durch Doppelklick auf einen Dateinamen diesen in die Dateiliste des Aktionsbereiches zu bringen. Die Navigation im Dateimanager ist intuitiv mit Klick auf den zu öffnenden Bereich realisiert. Sonderfunktionen bietet der Dateimanager keine.

Kapitel 6

Technische Grundlagen

6.1 Technischer Aufbau

Alle wichtigen Grundfunktionalitäten für den Zugriff auf den Keystore sind über eine Basis-Klasse implementiert. Diese sind (in eckigen Klammern sind die Methoden angegeben):

- Berechnen des Hashes für das Speichern von Schlüsseln. Benutzt wird der MD5-Algorithmus aus dem Flexiprovider. [*calcHash(byte[] value)*]
- Erzeugen einer Eingabebox für Kennwörter. Diese wird benutzt für die Eingabe von Kennwörtern bei privaten oder sicheren symmetrischen Schlüsseln. [*getInputPasswd()*]
- Laden eines privaten oder symmetrischen Schlüssels mit Kennwort aus dem Keystore. [*getKey(KeystoreFormat ksf)*]
- Referenzieren auf das aktuelle KeystoreManagerBackend^[5]. [*getKsmb()*]
- Liefern des aktuellen Keystore-Kennworts für das mit [*getKsmb()*] referenzierte KeystoreManagerBackend ^[5]. [*getPasswd()*]
- Referenzieren auf Control. [*getControl()*]

Die Methoden für die Grundfunktionalitäten werden von den Basic-Klassen BasicDialog, BasicComposite und BasicAction bereitgestellt.

BasicDialog wird bei Dialogen verwendet. Dialoge greifen mit Hilfe dieser Methoden auf die oben genannten Funktionen zu.

BasicComposite wird von Composites innerhalb der Bedienoberfläche benutzt.

Keystorefunktionen implementieren BasicAction.

6.2 Control-Klasse

Die Control-Klasse besitzt keine graphische Repräsentation. Alle Einstellungen, die für mehrere Teile der Oberfläche sinnvoll sind, werden dort gespeichert. Jedes Composite und jeder Dialog muss die Instanz der Control übergeben bekommen, ansonsten ist kein Zugriff auf den Keystore möglich. Diese Arbeit übernehmen die abstrakten Klassen BasicDialog und BasicComposite.

Die Control-Klasse stellt einen Observer bereit, um so allen beteiligten Bereichen mitzuteilen, wenn sich relevante Änderungen ergeben haben. Derzeit wird über den Observer nur die Keystore-Aktualisierung durchgeführt.

Für den Transport der Update-Informationen vom Keystore wurde eine extra Klasse namens *KeystoreTransport* angelegt, die es ermöglicht, jeden Bereich einzeln oder alle Bereiche über den Namen anzusprechen.

Die Control-Klasse ist als *ControlMain* abgelegt und implementiert das Interface *ControlInterface*. Folgende Funktionalitäten werden von der Control-Klasse zur Verfügung gestellt:

- Observer-Funktionen: Hinzufügen/Löschen und Keystore-Aktualisierung
- Aktionsbereich-Funktionen: direkter Zugriff auf Bereichsfunktionen, um dort für eine Weiterverarbeitung die nötigen Werte auszulesen.
- Keystore-Handling: Auslesen des aktiven Keystores, physische Pfadangabe, Name des Keystores innerhalb von Flexible.

Eine sehr wichtige Funktionalität der Control-Klasse stellt das Keystore-Handling da. Innerhalb der Control-Klasse werden alle Verweise auf die Keystores gespeichert. Nur über diese Klasse ist der Zugriff problemlos möglich. Auch Funktionen, die im aktiven Keystore arbeiten, lesen den aktiven Keystore aus der Control-Klasse aus.

6.3 Fehlerbehandlung

Fehler, die in der Bedienoberfläche auftreten, werden durch eine spezielle Exception-Klasse *GUIException* behandelt. Bei Fehlern, die nicht näher spezifiziert sind, werden die systemeigenen Meldungen an den Benutzer weitergegeben.

Wenn die Fehlermeldung bis zu einer Oberflächenklasse weitergegeben wurde, wird er mit Hilfe von einer *ErrorMessage* ausgegeben.

6.4 Keystore Format

Der Java Keystore kann nur Schlüssel ohne weitere Attribute speichern. Dazu wurde eine Namensvergabe des Schlüssels festgelegt:

- **Kontaktname** – Hier wird der Kontaktnamen in Unicode gespeichert. In diesem Feld erwartet Flexible eine Hexadezimalzahl mit gerader Länge, da jeweils zwei Zeichen einem Zeichen umgewandelt entsprechen.
- **Schlüsseltyp** – Da der Java Keystore nur asymmetrische Schlüssel und symmetrische Schlüssel speichern kann, gibt es folgende Schlüsseltypen:
 - Symmetrischer Schlüssel (SECRETKEY)
 - Öffentlicher Schlüssel (PUBLICKEY)
 - Privater Schlüssel (PRIVATEKEY)
 - Zertifikat (CERTIFICATE)

- Öffentlicher Schlüssel einer CA (CA)
- Privater Schlüssel der CA (CAPRIVATE)
- **Algorithmus** – in diesem Feld wird der von `getAlgorithm()` zurückgegebene Name eines Schlüssels/Zertifikats gespeichert. Gewöhnlicherweise ist das der Algorithmus des Schlüssels oder die OID.
- **Schlüssellänge** – die Schlüssellänge ergibt sich aus der Länge des jeweiligen Schlüssels.
- **Hash** - das Feld Hash hat bei Schlüssel, jeder Art den Hash des öffentlichen Schlüssel bzw. des symmetrischen Schlüssels. Bei Zertifikaten wird der Hash des öffentlichen Schlüssels im Zertifikat hier gespeichert.

Kapitel 7

Einschränkungen

In diesem Kapitel werden Einschränkungen hinsichtlich der Sicherheit und der Funktionalität erläutert.

7.1 Speicherung von Kennwörtern

Kennwörter werden in einer Liste von String-Datentypen gespeichert. Dadurch ist es möglich, diese aus dem Speicher auszulesen, wenn das Programm aktiv ist.

Es gibt praktisch keine Lösung, dieses Problem zu beseitigen, ohne zusätzliche Hilfsmittel einzusetzen. Denn jedes *Geheimnis* muss im Klartext im Speicher vorhanden sein, sonst ist es nicht verwendbar. Eine Möglichkeit wäre der Einsatz eines Chipkartenlesers der Klasse 2 oder Klasse 3, dadurch wird dem *Geheimnis* ein Maximum an Sicherheit gegen Keystroke-Attacken geboten. Allerdings wird das Kennwort irgendwann trotzdem im Klartext im Speicher zu finden sein, weil es sonst nicht verwendbar ist.

7.2 Erstellung von Zertifikatsketten

Das Erstellen von Zertifikatsketten ist nicht möglich, da Flexible nur ein CA-Zertifikat zulässt.

7.3 Darstellung Keystore

Nach einer Änderung im Keystore schließt sich die geöffnete Baumstruktur.

Um eine bessere Benutzerfreundlichkeit zu gewährleisten, müsste der Keystore eine Aktualisierung der einzelnen Elemente bieten. Dann muss der Keystore nicht immer neu aufgebaut werden, sondern es wird nur das geänderte Element hinzugefügt. Kompatible Datentypen werden benötigt.

7.4 Kompatibilität

Eine Kompilierung ist nur mit einer Java Version von Sun ab Version 5.0 möglich. Außerdem sollte der Patch *Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files* von Sun installiert sein. Zu finden ist dieser auf den Webseiten von Sun.

7.5 Plattformunabhängigkeit

Die Plattformunabhängigkeit hängt stark mit der Entwicklung des SWT-Frameworks ab. Weitere Informationen sind auf der Projekt-Website <http://www.eclipse.org/swt> verfügbar.

Glossar

CA	Certificate Authority - Zertifizierungsstelle	13
CRL	Certificate Revocation List - Zertifikatssperrliste	21
CSR	Certificate Sign Request - Zertifikatsanforderung bei einer Zertifizierungsstelle	4
DN	Distinguished Name - Subjectname im Zertifikat	21
Keystore	Klasse in Java zum Speichern von Zertifikaten/privaten Schlüsseln/symmetrischen Schlüsseln	4
MD5	Hashverfahren	7
OID	Object Identifier	28
Subversion	Ein Open Source Tool zur Versionsverwaltung von Dateien	1
SWT	Das Standard Widget Toolkit (SWT) ist eine Bibliothek für die Erstellung grafischer Oberflächen mit Java.	1

Quellen

- [1] TU Darmstadt. Fachbereich Informatik - Fachgebiet CDC. <https://svn.cdc.informatik.tu-darmstadt.de/svn/repos/bp-2006-g5/Bachelorarbeit>, 2007. [Revision 889 vom 19.07.2007].
- [2] TU Darmstadt. FlexiProvider. <http://www.flexiprovider.de>, 2007.
- [3] Eclipse Foundation. Swt. <http://www.eclipse.org/SWT>, 2007. [Online; zugegriffen am 01.07.2007].
- [4] Icon-King. Nuvola. <http://icon-king.com/?p=15>, 2007. [Online; zugegriffen am 01.07.2007].
- [5] Andreas Roth. Flexible - Eine erweiterbare GUI für den FlexiProvider - Backend. Bachelorarbeit, TU Darmstadt, Fachbereich Informatik, Fachgebiet CDC, Darmstadt, Germany, 2007.