



Technische Universität Darmstadt
Fachbereich 20 – Informatik
Fachgebiet Theoretische Informatik
Prof. Dr. Johannes Buchmann

Dirk Schramm

Entwicklung einer flexiblen Java-basierten S/MIME-Erweiterung für Microsoft Outlook

Diplomarbeit

Januar 2001

Autor:	Dirk Schramm
Studiengang:	Informatik
Hochschule:	TU Darmstadt
Matrikel-Nummer:	492362

Betreuer:	Markus Ruppert
-----------	----------------

INHALTSVERZEICHNIS

VORWORT	5
1 EINLEITUNG	7
1.1 GRUNDLAGEN DER KRYPTOGRAPHISCH GESICHERTEN KOMMUNIKATION PER E-MAIL	7
1.1.1 Signatur.....	7
1.1.2 Verschlüsselung.....	7
1.2 S/MIME ALS STANDARD FÜR SICHERE E-MAILS	8
1.2.1 Grundlagen.....	8
1.2.2 Signatur.....	9
1.2.3 Verschlüsselung.....	11
1.3 EXISTIERENDE S/MIME-LÖSUNGEN	11
1.4 MOTIVATION UND ZIELE FÜR EINE EIGENE S/MIME-IMPLEMENTIERUNG.....	12
2 LÖSUNGSANSÄTZE FÜR S/MIME-ERWEITERUNGEN	13
2.1 PROXY-SERVER FÜR SMTP UND POP3.....	13
2.2 PLUGIN	14
2.3 GEWÄHLTER ANSATZ	15
3 INTEGRATION VON S/MIME IN JAVAMAIL	16
3.1 GRUNDLAGEN	16
3.2 SIGNATUR.....	16
3.2.1 Die Schnittstelle SMimeSignature.....	16
3.2.2 Die Klasse SMimeSignedData	17
3.2.3 Die Klasse SMimeMultipartSigned.....	17
3.3 VERSCHLÜSSELUNG.....	18
3.3.1 Die Schnittstelle SMimeEnvelope.....	18
3.3.2 Die Klasse SMimeEnvelopedData	18
3.4 DIE KLASSE SMIME.....	20
3.5 DIE KLASSEN SIMPLEMIMEMESSAGE UND SIMPLEMIMEATTACHMENT	22
4 VERWALTUNG VON ZERTIFIKATEN UND PRIVATEN SCHLÜSSELN	23
4.1 GRUNDLAGEN	23
4.2 DIE KLASSE CERTIFICATEMANAGER	24
4.3 DIE SCHNITTSTELLE CERTIFICATEMANAGERSPI.....	25
4.4 TESTIMPLEMENTIERUNG AUF DER GRUNDLAGE DES KEYSTORE-MANAGERS.....	26
5 BENUTZEROBERFLÄCHE FÜR S/MIME-ERWEITERUNGEN	28
5.1 GRUNDLAGEN	28
5.2 DIE KLASSE SMIMEUI	28
5.3 DIE KLASSEN SMIMEOPTIONS UND SMIMEOPTIONSIALOG.....	29
5.4 NACHRICHTEN LESEN	30
5.4.1 Die Klasse SMimeReadMessageContext	30
5.4.2 Die Klassen SMimeMessageStatus und SMimeMessageStatusDialog.....	31

5.5	NACHRICHTEN SCHREIBEN	33
5.5.1	Die Klassen SMimeMessageOptions und SMimeMessageOptionsDialog	33
5.5.2	Die Klasse SMimeWriteMessageContext	34
5.6	MEHRSPRACHIGKEIT	35
5.7	ENTWICKLUNG EINES S/MIME-PLUGINS	36
6	JAVA NATIVE INTERFACE FRAMEWORK FÜR MICROSOFT VISUAL C++	38
6.1	GRUNDLAGEN	38
6.2	DIE KLASSE JOBJECT	39
6.3	DIE KLASSE JCLASS	39
6.4	DIE KLASSE JSTRING	40
6.5	DIE KLASSEN JARRAY UND JOBJECTARRAY	40
6.6	BISHER IMPLEMENTIERTE WRAPPER-KLASSEN	41
6.7	ANZEIGEN VON MODALEN SWING-DIALOGEN	43
6.8	AUSNAHMEN	44
7	S/MIME-ERWEITERUNG FÜR MICROSOFT OUTLOOK	45
7.1	GRUNDLAGEN	45
7.2	ERWEITERUNG DES OUTLOOK MENÜS UND DER WERKZEUGLEISTE	46
7.3	VERARBEITUNG DER NACHRICHTEN	48
7.3.1	Anpassen der Nachrichtenklasse	48
7.3.2	Konvertierung einer Nachricht zwischen MAPI und JavaMail	50
7.3.3	Verarbeitung einer Nachricht vor dem Anzeigen	51
7.3.4	Verarbeitung einer Nachricht vor dem Absenden	51
7.4	FEHLERBEHANDLUNG	52
7.5	INSTALLATION	54
7.6	KOMPATIBILITÄT	54
7.6.1	Hardwareplattform	55
7.6.2	Microsoft Windows	55
7.6.3	Java	55
7.6.4	Microsoft Outlook	55
7.6.5	Andere Mailprogramme	56
7.7	GESCHWINDIGKEIT	56
8	AUSBLICK	58
	LITERATUR	59

EHRENWÖRTLICHE ERKLÄRUNG

Hiermit versichere ich, die vorliegende Diplomarbeit 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 31. Januar 2001

VORWORT

Gegenstand dieser Diplomarbeit war die Entwicklung einer flexiblen und universell einsetzbaren S/MIME-Implementierung in Java auf der Grundlage der am Lehrstuhl von Prof. J. Buchmann im Rahmen des FlexiPKI Projektes entstandenen Komponenten. Hiermit sollte die Basis geschaffen werden, um vorhandene E-Mail-Programme möglichst einfach mit leistungsfähigen S/MIME-Funktionen zu erweitern, die insbesondere einen flexiblen Austausch der verwendeten kryptographischen Verfahren erlauben. Die Praxistauglichkeit des gewählten Ansatzes und damit auch des Gesamtkonzepts von FlexiPKI wurde abschließend mit der Entwicklung einer S/MIME-Erweiterung für Microsoft Outlook nachgewiesen.

Diese Arbeit ist in 8 Kapitel gegliedert, die sich wie folgt zusammenfassen lassen:

In Kapitel 1 findet sich eine Einführung in das Thema.

Kapitel 2 stellt unterschiedliche Lösungsansätze für S/MIME-Erweiterungen vor und beschreibt deren individuelle Vor- und Nachteile.

Kapitel 3 beschäftigt sich mit der Integration von S/MIME-Funktionen in JavaMail.

Kapitel 4 beschreibt die im Rahmen dieser Arbeit definierte Schnittstelle für ein Softwaremodul zur Verwaltung von Zertifikaten und privaten Schlüsseln.

Kapitel 5 erläutert das Konzept einer universell einsetzbaren Benutzeroberfläche für S/MIME-Erweiterungen

In Kapitel 6 werden die Verwendung des Java Native Interface zur Nutzung der Java Klassen von Visual C++ aus und das in diesem Zusammenhang entwickelte "Framework" beschrieben

Kapitel 7 beschäftigt sich mit der S/MIME-Erweiterung für Microsoft Outlook und den damit vorgenommenen Kompatibilitätstests.

Kapitel 8 gibt abschließend einen Ausblick auf die zukünftige Weiterentwicklung.

Ich danke an dieser Stelle Prof. Buchmann für die Möglichkeit, mich im Rahmen meiner Diplomarbeit mit diesem interessanten Thema zu beschäftigen, sowie Markus Ruppert und Markus Tak für ihre Unterstützung. Ein weiteres Dankeschön geht an Volker Roth für den guten Support im Zusammenhang mit dem von ihm entwickelten ASN.1 Codec.

Die Wiedergabe von Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. in dieser Diplomarbeit berechtigt auch ohne besondere Kennzeichnung nicht zu der Annahme, daß solche Namen im Sinne der Warenzeichen- und Markenschutz-Gesetzgebung als frei zu betrachten wären und daher von jedermann benutzt werden dürften.

1 EINLEITUNG

1.1 Grundlagen der kryptographisch gesicherten Kommunikation per E-Mail

In Zeiten, in denen die Kommunikation per E-Mail über offene Netze wie das Internet eine immer größere Rolle spielt und auch zunehmend Geschäftsprozesse auf diesem Wege abgewickelt werden, gewinnen Fragen der Sicherheit stark an Bedeutung.

Die Grundprobleme dabei bestehen darin, daß eine Nachricht beim Transport vom Absender zum Empfänger an jedem Netzknoten, den sie passiert, von Dritten gelesen und auf einfache Weise manipuliert werden kann. Zudem ist es mit geringem Aufwand möglich, Nachrichten unter beliebigen fremden Namen zu verschicken, ohne daß der Empfänger die Fälschung bemerkt. Dies läßt sich jedoch mit Hilfe der Public-Key-Kryptographie verhindern, die hierfür zwei Standardlösungen bietet, die im folgenden kurz vorgestellt werden:

1.1.1 Signatur

Beim kryptographischen Signieren einer Nachricht berechnet der Sender einen Hashwert über den gesamten Inhalt (den Text und die eventuellen Anhänge) und fügt diesen der Nachricht bei, nachdem er ihn mit seinem privaten Schlüssel, den nur er kennt, verschlüsselt hat.

Der Empfänger entschlüsselt den Hashwert mit dem öffentlichen Schlüssel des Absenders und vergleicht diesen mit dem Hashwert, den er selber berechnet hat. Stimmen beide Werte überein, kann er davon ausgehen, daß die Nachricht nach dem Signieren nicht verändert wurde, weil sich ein gutes Hashverfahren dadurch auszeichnet, daß sich keine zwei Texte mit dem gleichen Hashwert finden lassen.

Gleichzeitig kann der Empfänger darauf vertrauen, daß die Nachricht tatsächlich vom angegebenen Absender stammt, weil nur dieser den zu seinem öffentlichen Schlüssel passenden privaten Schlüssel kennt und somit nur er den Hashwert damit verschlüsseln kann. Dadurch ist eine Fälschung der Nachricht praktisch ausgeschlossen und der Absender kann auch nicht nachträglich bestreiten, sie geschrieben zu haben.

1.1.2 Verschlüsselung

Mit der Verschlüsselung einer Nachricht soll sichergestellt werden, daß ausschließlich der Empfänger sie lesen kann. Dazu wählt der Absender zunächst einen zufälligen geheimen Schlüssel für ein symmetrisches Verschlüsselungsverfahren, mit dem er den

Nachrichteninhalt chiffriert. Anschließend wird dieser geheime Schlüssel mit dem öffentlichen Schlüssel des Empfängers verschlüsselt und der Nachricht beigelegt.

Nur der Empfänger, der als einziger den zu seinem öffentlichen Schlüssel gehörenden privaten Schlüssel kennt, kann mit diesem den vom Absender erzeugten geheimen Schlüssel entschlüsseln und damit wiederum den Nachrichteninhalt dechiffrieren.

Eine derart geschützte Nachricht kann zwar immer noch während der Übertragung zum Empfänger von Dritten mitgelesen werden, jedoch erlangen diese dabei keine Kenntnis über den verschlüsselten Inhalt.

1.2 S/MIME als Standard für sichere E-Mails

Um kryptographische Verfahren zum Schutz von E-Mails in der Praxis effektiv nutzen zu können, ist es entscheidend, daß sie von möglichst vielen potentiellen Kommunikationspartnern unterstützt werden. Eine wichtige Voraussetzung dafür ist eine klare und offen zugängliche Spezifikation. Hierbei hat sich S/MIME (Secure Multipurpose Internet Mail Extensions) in den letzten Jahren im Internet zum de facto Standard entwickelt und ist inzwischen in fast allen leistungsfähigen Mailprogrammen integriert.

1.2.1 Grundlagen

S/MIME Version 3 ist in [RFC2632], [RFC2633] und [RFC2634] definiert und beschreibt im wesentlichen den Ablauf beim Signieren, Verschlüsseln und Entschlüsseln einer Nachricht, sowie das Verifizieren einer Signatur. Die Grundlage dafür bilden zwei weitere Spezifikationen:

MIME (Multipurpose Internet Mail Extensions)¹

MIME definiert in Ergänzung zu [RFC822] ein Format für den Inhalt von E-Mails. Dieses erlaubt die zuverlässige Übertragung von Sonderzeichen und binären Daten durch entsprechende Kodierungen (Quoted-Printable und Base64). Zusätzlich kann der Inhalt einer Nachricht aus mehreren Teilen zusammengesetzt werden (z.B. Text und angehängte Dateien), die sich beim Empfänger wieder eindeutig voneinander trennen lassen.

CMS (Cryptographic Message Syntax)²

CMS entspricht bis auf ein paar Details PKCS #7 und definiert verschiedene Formate zur Kodierung signierter oder verschlüsselter Daten, von denen die folgenden für S/MIME relevant sind:

¹ siehe [RFC2045], [RFC2046], [RFC2047], [RFC2048], [RFC2049]

² siehe [RFC2630]

1. SignedData

Diese Struktur enthält ein oder mehrere kryptographische Signaturen³ in Form von SignerInfo-Strukturen, sowie optional den signierten Inhalt und eine beliebige Anzahl von Zertifikaten. Die zum Signieren verwendeten kryptographischen Verfahren werden dabei mit Hilfe ihrer OIDs (Object Identifiers) eindeutig spezifiziert.

2. EnvelopedData

Diese Struktur enthält neben dem chiffrierten Inhalt für jeden Empfänger eine RecipientInfo-Struktur, die den verschlüsselten geheimen Schlüssel beinhaltet⁴. Die zum Verschlüsseln verwendeten kryptographischen Verfahren werden auch hier mit Hilfe ihrer OIDs (Object Identifiers) eindeutig spezifiziert.

3. ContentInfo

Diese Struktur umschließt einen beliebige weitere Struktur, deren Typ durch eine OID spezifiziert wird.

1.2.2 Signatur

S/MIME sieht zwei Möglichkeiten vor, eine Nachricht zu signieren:

1. *application/pkcs7-mime mit SignedData*

Hierbei erstellt der Absender eine SignedData-Struktur, die den signierten Nachrichteninhalte und die eigentliche Signatur enthält. Diese wird in eine ContentInfo-Struktur eingebettet, mittels Base64 kodiert und als Inhalt der Nachricht mit Content-Type "application/pkcs7-mime" verschickt.

Eine einfache Nachricht sieht dann so aus:

```
Content-Type: application/pkcs7-mime; smime-type=signed-data;
             name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
```

```
567GhIGfHfYT6ghyHhHUUjpfyF4f8HHGTrfvhJhjH776tbB9HG4VQbnj7
77n8HHGT9HG4VQpfyF467GhIGfHfYT6rfvbnj756tbBghyHhHUUjhJhjH
HUUjhJh4VQpfyF467GhIGfHfYGT6rfvbnjT6jH7756tbB9H7n8HHGghyHh
6YT64V0GhIGfHfQbnj75
```

(entnommen aus [RFC2633])

Der Nachteil dieser Variante besteht darin, daß ein Empfänger, dessen Mailprogramm kein S/MIME unterstützt, nur eine Datei mit dem Namen "smime.p7m" angezeigt bekommt, mit der er nichts anfangen kann. Damit in einem solchen Fall der signierte Inhalt der Nachricht trotzdem korrekt angezeigt wird, gibt es die folgende Alternative.

³ siehe Abschnitt 1.1.1

⁴ siehe Abschnitt 1.1.2

2. multipart/signed

Hierbei wird eine MIME-Multipart-Struktur verwendet, die das Versenden mehrerer unabhängiger Teile innerhalb einer Nachricht ermöglicht. Der erste Teil entspricht in diesem Fall dem zu signierenden Nachrichteninhalte. Der zweite Teil besteht aus einer Base64-kodierten ContentInfo-Struktur mit einer eingebetteten SignedData-Struktur. Diese enthält nur noch die Signatur ohne den eigentlichen Inhalt.

Eine entsprechende Nachricht könnte folgendermaßen aussehen:

```
Content-Type: multipart/signed;
  protocol="application/pkcs7-signature";
  micalg=sha1; boundary=boundary42

--boundary42
Content-Type: text/plain

This is a clear-signed message.

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s

ghyHhHUujhJhjh77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jh77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjh776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42--
```

(entnommen aus [RFC2633])

Hierbei ist der signierte Mailtext sogar ohne jede MIME-Unterstützung noch lesbar. Durch das Herauslösen des signierten Inhalts aus der SignedData-Struktur ergibt sich allerdings ein anderes Problem: Wenn eine Nachricht auf dem Weg zum Empfänger Zwischenstationen passiert, die alte oder fehlerhafte Software einsetzen, kann sie dabei z.B. durch Einfügen oder Löschen von Leerstellen am Ende einer Zeile verändert werden. Zudem ist eine Zerstörung der multipart/signed-Struktur beim Durchlaufen von Gateways in nicht-RFC822-kompatible Netze wie X.400 möglich. In beiden Fällen kann die Signatur vom Empfänger nicht mehr überprüft werden.

Ein weiteres Problem in diesem Zusammenhang besteht in der systemabhängigen Kodierung der Zeilenumbrüche. Hier schreibt S/MIME vor, daß bei der Berechnung des Hashwertes jede Zeile mit CR/LF enden muß. Andere Kodierungen sind vorher entsprechend zu konvertieren.

Die SignedData-Strukturen dürfen bei beiden Signaturvarianten eine beliebige Zahl von Zertifikaten enthalten, die für den Empfänger von Interesse sein könnten. Im Normalfall sind dies die Zertifikate, die vom Absender zum Signieren verwendet werden und jene, mit denen Nachrichten an ihn verschlüsselt werden sollen.

1.2.3 Verschlüsselung

Beim Verschlüsseln einer Nachricht erstellt der Absender eine EnvelopedData-Struktur, die den chiffrierten Nachrichteninhalte und für jeden Empfänger eine RecipientInfo-Struktur enthält. Diese wird in eine ContentInfo-Struktur eingebettet, mittels Base64 kodiert und als Inhalt der Nachricht mit Content-Type "application/pkcs7-mime" verschickt.

Eine einfache Nachricht sieht dann so aus:

```
Content-Type: application/pkcs7-mime;
             smime-type=enveloped-data; name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
```

```
rfvbnj756tbBghyHhHUujhJhjh77n8HHGT9HG4VQpfyF467GhIGfHfYT6
7n8HHGghyHhHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H
f8HHGTrfvhJhjh776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
0GhIGfHfQbnj756YT64V
```

(entnommen aus [RFC2633])

1.3 Existierende S/MIME-Lösungen

In den letzten Jahren wurden bereits viele S/MIME-Lösungen entwickelt. Diese lassen sich in zwei Gruppen einteilen:

1. E-Mail-Programme mit S/MIME-Unterstützung

Inzwischen bieten fast alle leistungsfähigeren Mailprogramme S/MIME-Funktionen an. Dazu gehören u.a. Outlook und Outlook Express von Microsoft sowie der Netscape Communicator, um die beiden wohl am weitesten verbreiteten Produkte zu nennen.

2. S/MIME-Erweiterungen

Zusätzlich werden diverse S/MIME-Erweiterungen für bestehende Mailprogramme angeboten, die selbst kein S/MIME unterstützen, oder deren S/MIME-Funktionen durch leistungsfähigere ersetzt werden sollen.

Hierzu gehört TrustedMIME der Firma SSE, das als Erweiterung für Microsoft Outlook, Microsoft Exchange und Lotus Notes verfügbar ist und u.a. die Verwendung von Smart Cards zum Speichern der persönlichen Zertifikate ermöglicht.

1.4 Motivation und Ziele für eine eigene S/MIME-Implementierung

Die bekannten S/MIME-Lösungen haben den entscheidenden Nachteil, daß sie nur die kryptographischen Standardverfahren für Signatur und Verschlüsselung anbieten und sich diese nicht ohne weiteres ergänzen oder durch andere ersetzen lassen. Insbesondere steht in der Regel nur RSA als Public-Key-Verschlüsselungsverfahren zur Verfügung. Dies stellt jedoch ein nicht unerhebliches Risiko dar, weil die Sicherheit von RSA bis heute nicht bewiesen ist. Somit könnte jederzeit ein wirksamer Angriff gefunden werden, der sämtliche auf RSA basierenden Sicherheitskonzepte zunichte macht.

Aus diesem Grund ist größtmögliche Flexibilität bei der Wahl der kryptographischen Verfahren eines der wichtigsten Ziele des am Lehrstuhl von Prof. Buchmann laufenden FlexiPKI-Projektes. Die Basis dafür bildet die Java Cryptography Architecture, die es ermöglicht, die Liste der zur Verfügung stehenden kryptographischen Algorithmen durch einfaches Hinzuladen von Providern nahezu beliebig zu erweitern. Für das Anwendungsprogramm ist es dabei transparent, welches konkrete Verfahren eingesetzt wird, so daß hier keine Änderungen erforderlich sind. Zudem werden Methoden entwickelt, um unsicher gewordene kryptographische Algorithmen ähnlich wie Zertifikate automatisch zurückziehen zu können und so die Sicherheit zusätzlich zu erhöhen.

Der Nachweis der Praxistauglichkeit dieses zunächst abstrakten Konzeptes anhand einer konkreten Anwendung (S/MIME) ist das primäre Ziel dieser Diplomarbeit.

2 LÖSUNGSANSÄTZE FÜR S/MIME-ERWEITERUNGEN

Moderne Mailprogramme wie Microsoft Outlook 2000 bieten inzwischen einen enormen Funktionsumfang, so daß die Entwicklung eines eigenen konkurrenzfähigen Mailprogramms mit einem extrem hohen Aufwand verbunden wäre. Daher wird im Rahmen dieser Arbeit der Ansatz einer S/MIME-Erweiterung für bestehende Mailprogramme verfolgt. Dies bietet darüber hinaus den Vorteil, daß die Anwender weiterhin mit ihrer gewohnten Software arbeiten können und trotzdem in den Genuß der leistungsfähigeren S/MIME-Funktionen kommen.

Dabei sind zwei unterschiedliche Lösungsansätze denkbar, die im folgenden kurz vorgestellt werden.

2.1 Proxy-Server für SMTP und POP3

Ein Proxy-Server ist ein eigenständiges Programm, das in diesem Fall auf dem Computer des Benutzers läuft und zwischen den Mailclient und den Mailserver "geschaltet" wird. Es verhält sich dem Mailclient gegenüber wie ein Mailserver, so daß dieser über die Standardprotokolle SMTP⁵ und POP3⁶ Nachrichten zum Proxy-Server senden und von dort empfangen kann. Der Proxy-Server seinerseits kommuniziert über die gleichen Protokolle mit dem eigentlichen Mailserver.

Bei dieser Lösung laufen also sämtliche gesendeten und empfangenen Nachrichten über den Proxy-Server, der dadurch die Möglichkeit hat, sie vor dem Weiterleiten zu verarbeiten. So kann er zu sendende Nachrichten zunächst signieren und/oder verschlüsseln, sowie empfangene Nachrichten falls nötig entschlüsseln und eine mögliche Signatur überprüfen und danach entfernen.

Der große Vorteil dieses Ansatzes besteht darin, daß er mit jedem Mailprogramm funktioniert, das die beiden genannten Standardprotokolle unterstützt, weil keine weiteren Schnittstellen benötigt werden.

Jedoch ergeben sich auch einige Nachteile. Üblicherweise werden verschlüsselte Nachrichten erst beim Anzeigen und dann auch nur temporär entschlüsselt, so daß Dritte selbst bei direktem Zugriff auf den Computer an den Nachrichteninhalt nicht herankommen, solange der private Schlüssel des Benutzers ausreichend geschützt ist. Diese zusätzliche Sicherheit entfällt, wenn die Nachricht bereits beim Empfang vom Proxy-Server entschlüsselt wurde.

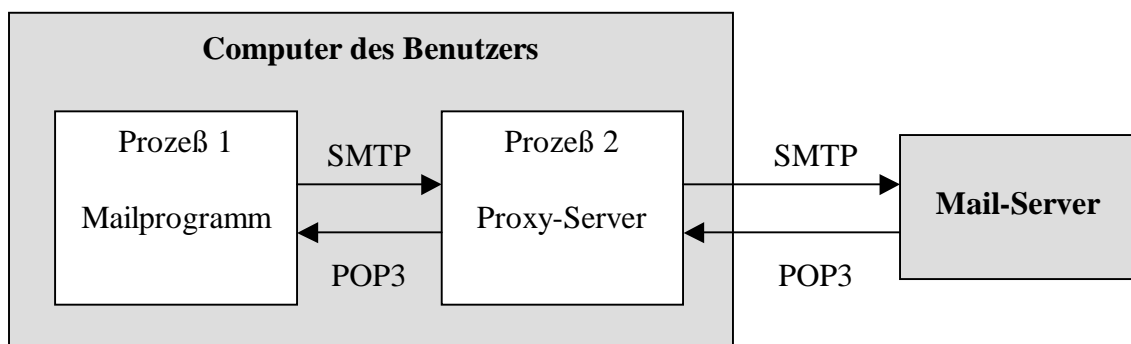
⁵ siehe [RFC821]

⁶ siehe [RFC1939]

Ein ähnliches Problem besteht bei der Überprüfung der Signatur eingehender Nachrichten. Diese wird nur einmal direkt nach dem Empfang vom Proxy-Server vorgenommen. Spätere Manipulationen lassen sich so nicht mehr erkennen.

Weitere Schwierigkeiten bereitet die Verlagerung eines Teils der normalerweise im Mailclient integrierten Benutzeroberfläche in den Proxy-Server. So muß der Benutzer mit einem separaten Programm arbeiten, um beispielsweise den zu verwendenden Mailserver oder die S/MIME-Einstellungen zu verändern oder seine Zertifikate zu verwalten. Zudem muß die Konfiguration des Mailclients auf die des Proxy-Servers abgestimmt werden, was unerfahrene Anwender vor Probleme stellen kann.

Die folgende Abbildung verdeutlicht noch einmal den vorgestellten Ansatz:



2.2 Plugin

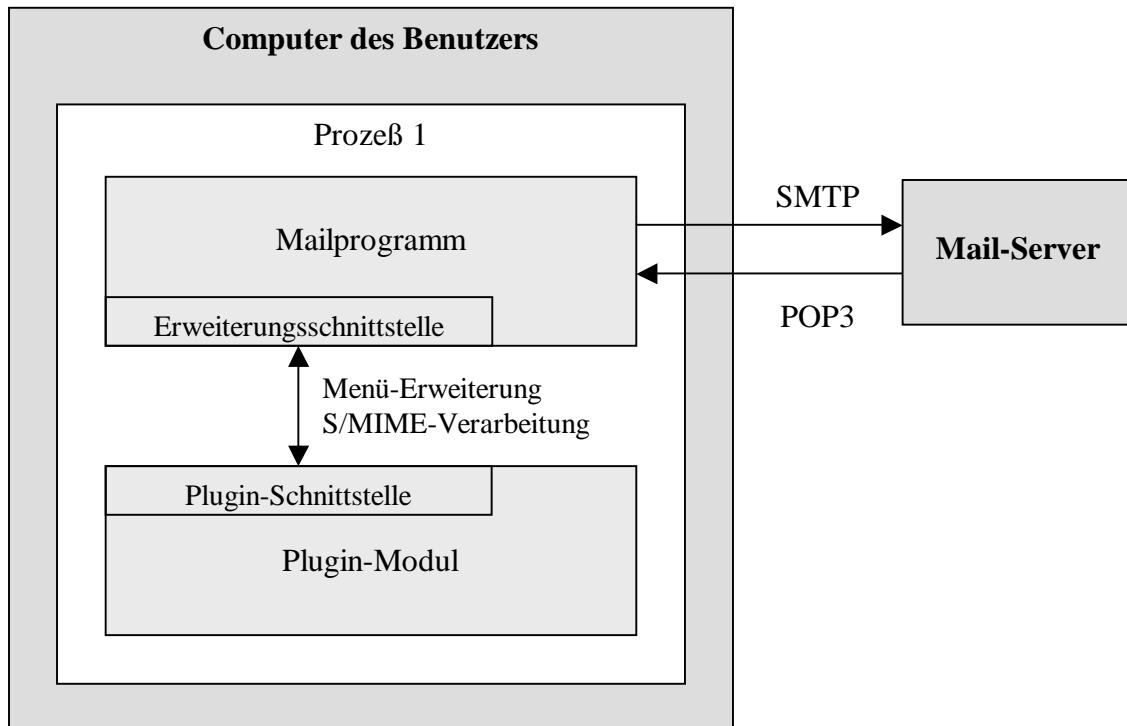
Viele leistungsfähige Mailprogramme wie Microsoft Outlook bieten definierte Schnittstellen, über die sich die Funktionalität der Software erweitern läßt. Die Möglichkeiten dabei sind vom jeweiligen Programm abhängig und reichen vom einfachen Hinzufügen neuer Menüpunkte und Werkzeugleistensymbolen bis zum Einlesen und Verändern von Nachrichten. Die dafür zu entwickelnden Softwarekomponenten werden u.a. als "Plugin", "Add-In" oder "Extension" bezeichnet.

Ein weiterer Ansatz für eine S/MIME-Erweiterung besteht somit darin, sie als "Plugin" zu realisieren. Voraussetzung dafür ist, daß die vom jeweiligen Mailprogramm zur Verfügung gestellte Plugin-Schnittstelle die Möglichkeiten bietet, Menüs und Werkzeugleisten zu erweitern und insbesondere vor dem Anzeigen oder Absenden einer Nachricht auf deren Darstellung im MIME-Format zuzugreifen und sie gegebenenfalls zu verändern.

Diese Lösung hat den großen Vorteil, daß sie eine elegante Integration der S/MIME-Funktionen in die Benutzeroberfläche eines vorhandenen Mailprogramms ermöglicht und dadurch eine einfache und intuitive Bedienung erreicht werden kann.

Die Nachteile bestehen allerdings darin, daß nicht jedes Mailprogramm eine passende Plugin-Schnittstelle besitzt und es zudem keinen Standard für derartige Schnittstellen gibt. Dadurch ist eine individuelle Anpassung für jedes Mailprogramm nötig.

Die folgende Abbildung veranschaulicht das beschriebene Konzept:



2.3 Gewählter Ansatz

Beide vorgestellten Varianten haben ihre individuellen Vor- und Nachteile. Wenn das Ziel allerdings ein professionelles Produkt für den täglichen Einsatz sein soll, das dem Benutzer größtmöglichen Bedienkomfort bietet, kann der Proxy-Server bestenfalls als universell einsetzbare Notlösung dienen.

Daher fiel im Rahmen dieser Diplomarbeit die Entscheidung auf die Entwicklung eines Plugins für Microsoft Outlook als einem der am häufigsten verwendeten Mailclients. Dabei sollen die einzelnen Softwarekomponenten jedoch so allgemein und flexibel wie möglich gehalten werden, um sie bei Plugins für andere Mailprogramme oder bei der Entwicklung eines Proxy-Servers direkt wiederverwenden zu können.

3 INTEGRATION VON S/MIME IN JAVAMAIL

Wie in Abschnitt 1.2 dargestellt wurde, bildet MIME die Basis für S/MIME. Es erscheint daher sinnvoll, die S/MIME-Funktionen in eine bereits vorhandene MIME-Implementierung zu integrieren. Hier bietet sich JavaMail an, weil es kostenlos von Sun verfügbar ist und auf einfachem Wege erweitert werden kann.

3.1 Grundlagen

JavaMail nutzt das JavaBeans Activation Framework, um Objekte zur Repräsentation des Nachrichteninhalts abhängig vom jeweiligen Content-Type zu erzeugen. Im Falle von "text/plain" ist dies beispielsweise ein einfacher String. Hierbei kommen sogenannte `DataContentHandler`-Klassen zum Einsatz, die jeweils einem bestimmten MIME-Typ zugeordnet sind und aus einem gegebenen `InputStream` der Rohdaten das entsprechende Objekt erzeugen. In der umgekehrten Richtung läßt sich auch der Inhalt eines solchen Objektes in einen `OutputStream` schreiben.

Neben dem Einlesen und Erzeugen von MIME-Nachrichten spielt für S/MIME, wie in Abschnitt 1.2 beschrieben, auch die Kodierung und Dekodierung von CMS-Strukturen eine wichtige Rolle. Hierbei kommt der von der Fraunhofer Gesellschaft entwickelte ASN.1-Codec zum Einsatz, der fertige Klassen zur Verarbeitung der benötigten Strukturen (`ContentInfo`, `SignedData`, `EnvelopedData`, etc.) liefert.

Im Folgenden werden die wichtigsten Klassen für die Integration von S/MIME in JavaMail vorgestellt.

3.2 Signatur

Bei der Signatur einer Nachricht erlaubt S/MIME, wie in Abschnitt 1.2.2 erläutert, zwei Varianten. Um hier in der späteren Anwendung möglichst wenige Fallunterscheidungen machen zu müssen, wird zunächst eine gemeinsame Schnittstelle definiert.

3.2.1 Die Schnittstelle `SMimeSignature`

Die Schnittstelle `SMimeSignature` enthält alle Funktionen, die zum Erzeugen oder Verarbeiten einer S/MIME-Signatur auf der Grundlage einer CMS `SignedData`-Struktur benötigt werden:

1. Erzeugen einer Signatur:

- Festlegen des zu signierenden Nachrichteninhalts
- Hinzufügen eines Zertifikats
- Signieren des Inhalts mit einem gegebenen Zertifikat und dem dazugehörigen privaten Schlüssel unter Verwendung eines bestimmten Algorithmus und mit anschließendem Hinzufügen der entsprechenden SignerInfo-Struktur

2. Verarbeiten einer Signatur:

- Auslesen des signierten Nachrichteninhalts
- Auslesen der enthaltenen Zertifikate
- Auslesen der enthaltenen SignerInfo-Strukturen
- Überprüfen der von einer SignerInfo-Struktur repräsentierten Signatur

Diese Schnittstelle wird von den beiden folgenden Klassen implementiert.

3.2.2 Die Klasse `SMimeSignedData`

Die Klasse `SMimeSignedData` repräsentiert eine S/MIME-Signatur basierend auf einer `SignedData`-Struktur, die sich in einem Nachrichtenteil mit den Content-Types "application/pkcs7-mime" oder "application/pkcs7-signature" befindet. Das Kodieren und Dekodieren der Daten erfolgt dabei mit Hilfe des ASN.1-Codec, dessen Klasse `SignedData` die Grundlage für die Implementierung der Schnittstelle `SMimeSignature` bildet.

3.2.3 Die Klasse `SMimeMultipartSigned`

Die Klasse `SMimeMultipartSigned` repräsentiert eine S/MIME-Signatur basierend auf einer MIME-Multipart-Struktur mit Content-Type "multipart/signed". Sie erbt dazu von der JavaMail-Klasse `MimeMultipart`, die zur Verarbeitung beliebiger Multipart-Typen genutzt werden kann, und paßt deren Funktionalität entsprechend an. So sind im Falle von "multipart/signed" genau zwei Teile notwendig, wobei der erste den zu signierenden Nachrichteninhalt und der zweite die eigentliche Signatur mit dem Content-Type "application/pkcs7-signature" enthält⁷.

Zur Verarbeitung des Signaturteils kommt wieder die Klasse `SMimeSignedData` zum Einsatz, an die bei der Implementierung der Schnittstelle `SMimeSignature` fast alle

⁷ siehe Abschnitt 1.2.2

Methodenaufrufe weitergeleitet werden. Nur das Auslesen oder Verändern des zu signierenden Inhalts erfolgt innerhalb der Klasse `SMimeMultipartSigned`.

3.3 Verschlüsselung

Aus Gründen der Symmetrie zur Signatur und für mögliche spätere Erweiterungen wurde auch im Falle der Verschlüsselung die Schnittstelle von der Implementierung getrennt.

3.3.1 Die Schnittstelle `SMimeEnvelope`

Die Schnittstelle `SMimeEnvelope` enthält alle Funktionen, die zum Erzeugen oder Verarbeiten einer S/MIME-Verschlüsselung auf der Grundlage einer CMS `EnvelopedData`-Struktur benötigt werden:

1. Erzeugen einer Verschlüsselung:

- Festlegen des zu verschlüsselnden Nachrichteninhalts
- Verschlüsseln des Inhalts unter Verwendung eines bestimmten Algorithmus
- Erzeugen und Hinzufügen von `RecipientInfo`-Strukturen mittels der Zertifikate der Empfänger

2. Verarbeiten einer Verschlüsselung

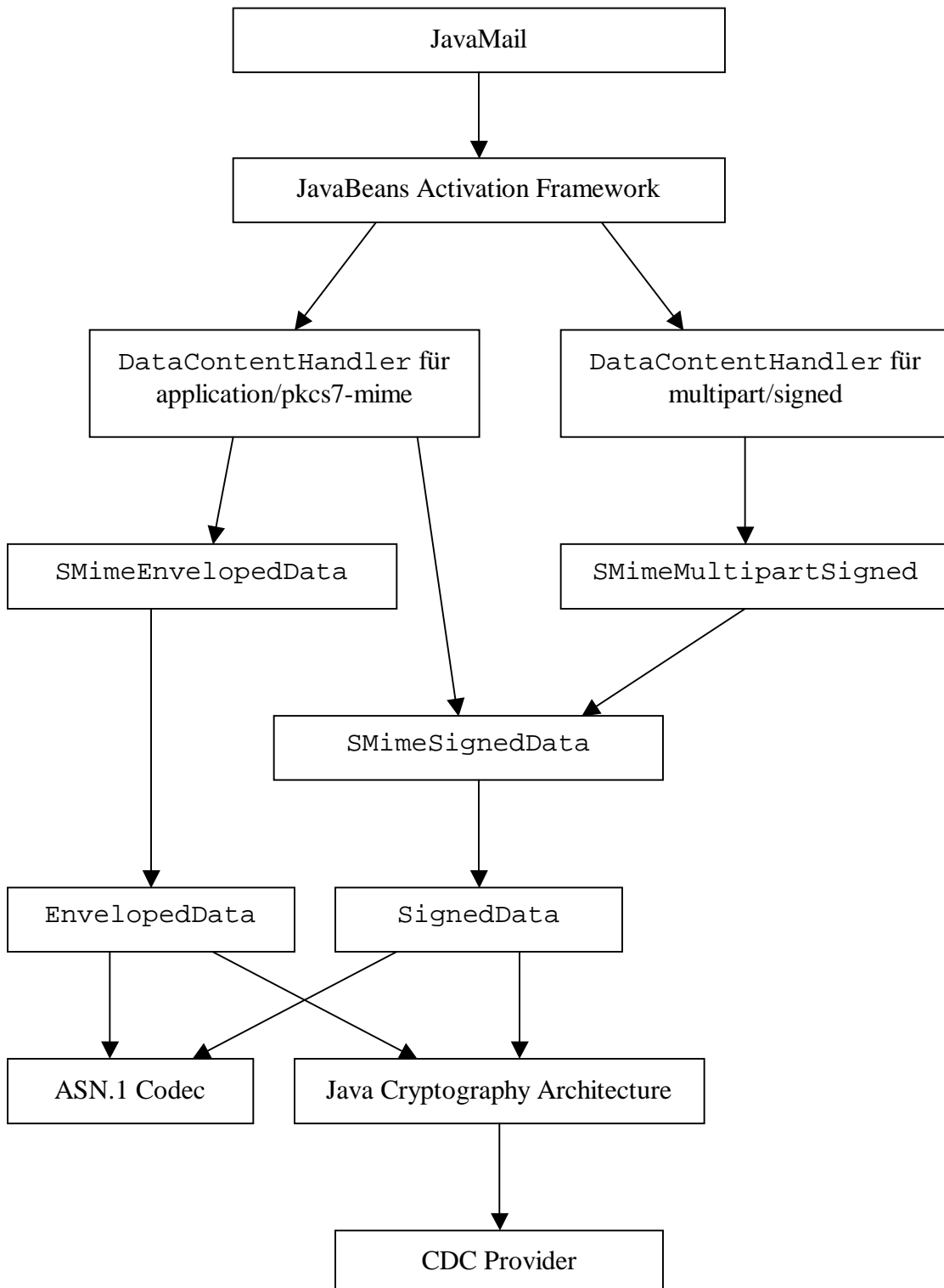
- Auslesen der `RecipientInfo`-Strukturen
- Entschlüsselung mittels eines passenden Zertifikats und des dazugehörigen privaten Schlüssels
- Auslesen des entschlüsselten Nachrichteninhalts

Diese Schnittstelle wird von der folgenden Klasse implementiert.

3.3.2 Die Klasse `SMimeEnvelopedData`

Die Klasse `SMimeEnvelopedData` repräsentiert eine S/MIME-Verschlüsselung basierend auf einer `EnvelopedData`-Struktur, die sich in einem Nachrichtenteil mit dem Content-Type "application/pkcs7-mime" befindet. Das Kodieren und Dekodieren der Daten erfolgt dabei mit Hilfe des ASN.1-Codec, dessen Klasse `EnvelopedData` die Grundlage für die Implementierung der Schnittstelle `SMimeEnvelope` bildet.

Die folgende Darstellung verdeutlicht das Zusammenspiel der vorgestellten Klassen:



3.4 Die Klasse `SMime`

Die Klasse `SMime` enthält viele statische Hilfsfunktionen, von denen einige für S/MIME-Anwendungen interessant sind. Vier davon werden in diesem Abschnitt vorgestellt:

1. `registerDataContentHandlers`

Wie der Name dieser Funktion bereits andeutet, dient sie zum Registrieren der `DataContentHandler`-Klassen für die S/MIME-spezifischen Content-Types innerhalb des JavaBeans Activation Frameworks und sollte daher vor jeder anderen S/MIME-Funktion aufgerufen werden.

2. `addSignature`

Diese Funktion fügt eine S/MIME-Signatur zu einer gegebenen Nachricht hinzu, die deren gesamten bisherigen Inhalt umfaßt. Dadurch lassen sich MIME-Nachrichten auf einfachste Weise signieren:

```
MimeMessage message;
X509Certificate certificate;
PrivateKey privateKey;
SMimeSignature signature;

[ Nachricht erzeugen ]
[ Zertifikat und privaten Schlüssel einlesen ]

signature = SMime.addSignature(message, true);
signature.sign(certificate, privateKey, "SHA1/RSA");
signature.addCertificate(certificate);

[ Nachricht absenden ]
```

3. `addEnvelope`

Diese Funktion fügt eine S/MIME-Verschlüsselung zu einer gegebenen Nachricht hinzu, die deren gesamten bisherigen Inhalt umfaßt. Dadurch lassen sich MIME-Nachrichten auf einfachste Weise verschlüsseln:

```
MimeMessage message;
X509Certificate certificate;
SMimeEnvelope envelope;

[ Nachricht erzeugen ]
[ Zertifikat des Empfängers einlesen ]

envelope = SMime.addEnvelope(message);
envelope.encrypt("DESede/CBC/PKCS5Padding");
envelope.addRecipient(certificate);

[ Nachricht absenden ]
```

4. *copyMimePart*

Diese Funktion kopiert den Inhalt eines Nachrichtenteils in einen anderen. Sie kann eingesetzt werden, um S/MIME-Signaturen und Verschlüsselungen wieder zu entfernen:

```
MimeMessage message;

[ Nachricht einlesen ]

while (true) {
    Object content = message.getContent();
    if (content instanceof SMimeSignature) {

        // S/MIME-Signatur verarbeiten

        SMimeSignature signature;
        Iterator iterator;

        signature = (SMimeSignature)content;
        iterator = signature.getSignerInfos().iterator();
        while (iterator.hasNext()) {
            SignerInfo signerInfo =
                (SignerInfo)iterator.next();
            if (!signature.verify(signerInfo)) {
                // Ungültige Signatur!
            }
        }
        SMime.copyMimePart(signature.getBodyPart(), message);
    } else if (content instanceof SMimeEnvelope) {

        // S/MIME-Verschlüsselung verarbeiten

        X509Certificate certificate;
        PrivateKey privateKey;
        SMimeEnvelope envelope;

        [ Zertifikat und privaten Schlüssel einlesen ]

        envelope = (SMimeEnvelope)content;
        envelope.decrypt(certificate, privateKey);
        SMime.copyMimePart(envelope.getBodyPart(), message);
    } else {
        break;
    }
}

[ Nachricht anzeigen ]
```

3.5 Die Klassen `SimpleMimeMessage` und `SimpleMimeAttachment`

Eine MIME-Nachricht kann beliebig komplex verschachtelte MIME-Multipart-Strukturen enthalten. Die Klasse `SimpleMimeMessage` liefert hier eine vereinfachte Sicht, in der eine Nachricht aus drei Hauptteilen besteht:

- Nachrichtenkopf (Absender, Empfänger, Betreff, etc.)
- Mailtext (ASCII und/oder HTML)
- eine beliebige Anzahl von Anhängen (repräsentiert durch Instanzen der Klasse `SimpleMimeAttachment`)

Diese Sichtweise entspricht der der meisten Mailprogramme.

Zudem enthalten beide Klassen diverse Hilfsfunktionen, die die Programmierung verglichen mit den JavaMail-Klassen `MimeMessage` und `MimeBodyPart` vereinfachen, und sind zusätzlich für eine effiziente Nutzung über das Java Native Interface⁸ optimiert.

Im Falle des Plugins für Outlook werden diese Klassen eingesetzt, um MAPI-Nachrichten in das MIME-Format zu konvertieren, sie dann zu verarbeiten und anschließend wieder zurückzuwandeln⁹.

Ausführlichere Informationen zu den in diesem Kapitel vorgestellten Schnittstellen und Klassen finden sich in der mit JavaDoc erzeugten Dokumentation.

⁸ siehe Kapitel 6

⁹ siehe Kapitel 7

4 VERWALTUNG VON ZERTIFIKATEN UND PRIVATEN SCHLÜSSELN

Bei kryptographischen Anwendungen wie S/MIME spielt die Verwaltung von Zertifikaten und privaten Schlüsseln eine entscheidende Rolle. Da jedoch im Rahmen des FlexiPKI-Projektes bisher keine entsprechende Softwarekomponente zur Verfügung steht, wurde als ein weiterer Teil dieser Diplomarbeit eine passende Schnittstelle definiert und testweise implementiert.

4.1 Grundlagen

Ähnlich wie bei der Wahl der kryptographischen Verfahren ist auch bei der Zertifikats- und Schlüsselverwaltung größtmögliche Flexibilität ein wichtiges Qualitätskriterium.

So gibt es zum Speichern von Zertifikaten und Schlüsseln auf der lokalen Festplatte als dem einfachsten Ansatz verschiedene Alternativen. Beispielsweise sind private Schlüssel innerhalb einer Chipkarte besser gegen unberechtigten Zugriff geschützt, so daß auch eine solche Lösung unterstützt werden sollte. In diesem Zusammenhang wäre sogar der Einsatz biometrischer Verfahren denkbar. Außerdem läßt sich der Komfort für den Anwender erhöhen, wenn Zertifikate anderer Personen, die lokal nicht gefunden wurden, bei Bedarf automatisch von Directory-Servern abgerufen werden können.

Auch die Online-Überprüfung der Gültigkeit eines Zertifikats mittels des OCSP (Online Certificate Status Protocol) stellt eine gute Alternative zur Verwendung der klassischen CRLs (Certificate Revocation Lists) dar.

Eine weitere wichtige Rolle neben den unterstützten Schnittstellen und Protokollen spielen flexible Konfigurationsmöglichkeiten. Insbesondere die Kriterien für die Bewertung der Vertrauenswürdigkeit eines Zertifikats und auch für den Umgang mit neuen Zertifikaten, die z.B. in einer S/MIME-Nachricht enthalten sind, sollten vom Anwender frei wählbar sein. Gleiches gilt für die Sicherheitseinstellungen der privaten Schlüssel. Hier sollte der Benutzer unter anderem entscheiden können, welche Anwendungen welche Schlüssel nutzen dürfen und ob bei jeder Verwendung eines Schlüssels ein Paßwort eingegeben werden muß oder nur einmal für eine in irgendeiner Form begrenzte Zeitspanne.

Es wird deutlich, daß dem Umfang und der Komplexität einer Zertifikats- und Schlüsselverwaltung fast keine Grenzen gesetzt sind. Daher ist es besonders wichtig, daß eine solche Softwarekomponente über eine flexible und leistungsfähige Programmierschnittstelle verfügt, die gleichzeitig so einfach und abstrakt wie möglich gehalten ist. Kryptographische Anwendungen sollen auf unkompliziertem Wege auf Zertifikate und Schlüssel zugreifen und die Vertrauenswürdigkeit von Zertifikaten überprüfen können, ohne dabei unnötige Details berücksichtigen zu müssen.

Mit diesem Ziel wurde die im folgenden beschriebene Schnittstelle entworfen.

4.2 Die Klasse `CertificateManager`

Die Klasse `CertificateManager` beinhaltet die zentrale Schnittstelle für alle Anwendungen, die mit Zertifikaten und Schlüsseln arbeiten. Sie bietet dabei folgende Gruppen von Funktionen:

1. *Zertifikate suchen*

Hierbei kommt zusätzlich die Klasse `CertificateSearchCriteria` zum Einsatz, mit deren Hilfe Suchkriterien wie Aussteller, Seriennummer, E-Mail-Adresse des Inhabers oder die Beschränkung auf Zertifikate für einen bestimmten Anwendungstyp oder solche, zu denen der private Schlüssel verfügbar ist, festgelegt werden können. Dabei ist es für die Anwendung völlig transparent, wie die Suche von der Zertifikatsverwaltung genau durchgeführt wird und welche Quellen dabei entsprechend der Konfiguration Berücksichtigung finden. Als Ergebnis wird lediglich eine Liste der Zertifikate, die für die Anwendung verfügbar sind und die Suchkriterien erfüllen, zurückgegeben.

Jedes Zertifikat wird dabei durch eine ID in Form eines `String`-Objekts repräsentiert, deren Format von der Zertifikatsverwaltung beliebig gewählt werden kann. Diese Indirektion bietet den Vorteil, daß die Anwendung die IDs als Verweise auf Zertifikate und private Schlüssel z.B. in Konfigurationsdateien abspeichern kann und dadurch vor allem die Schlüssel erst dann von der Zertifikatsverwaltung angefordert werden müssen, wenn sie tatsächlich benötigt werden (siehe Punkt 3).

Für den Fall, daß das zu einer extern gespeicherten ID gehörende Zertifikat in der Zwischenzeit gelöscht oder verschoben wurde, steht die Funktion `updateIDs` zur Verfügung. Diese aktualisiert die IDs in einer übergebenen Liste und ersetzt alle unbekanntenen durch `null`. Dabei kann das Wiederfinden eines Zertifikats nach dessen Verschiebung an einen anderen Speicherort durch eine entsprechende Kodierung der ID sichergestellt werden. Diese sollte neben einem Verweis auf die momentane Speicherstelle für den schnellen Zugriff auch eine eindeutige Kennung wie den Herausgeber und die Seriennummer des Zertifikats enthalten, nach der bei Bedarf gesucht werden kann.

2. *Zertifikate vom Benutzer auswählen lassen*

Neben der automatischen Suche steht auch ein Dialog zur Verfügung, über den der Benutzer ein oder mehrere Zertifikate einer durch die Anwendung mittels Suchkriterien vorgegebenen Menge auswählen kann. Hier kommen ebenfalls die IDs stellvertretend für die Zertifikate zum Einsatz.

3. *Zertifikate oder private Schlüssel auslesen*

Verschiedene Funktionen erlauben das Auslesen von Zertifikaten oder privaten Schlüsseln, die durch ihre IDs spezifiziert sind, in Form von `X509Certificate`- bzw. `PrivateKey`-Objekten. Dabei obliegt es der Zertifikatsverwaltung, falls nötig ein Paßwort vom Benutzer abzufragen, oder ihn zum Einlegen einer Chipkarte aufzufordern.

4. Überprüfen eines Zertifikats

Die Funktionen zum Überprüfen der Gültigkeit sowie Vertrauenswürdigkeit eines Zertifikats liefert einen `long`-Wert zurück, bei dem jedes Bit ein bestimmtes Problem (Zertifikat abgelaufen, zurückgezogen, unbekannt, etc.) anzeigt. Wie die Überprüfung genau abläuft, ist dabei der Zertifikatsverwaltung überlassen und auch von der jeweiligen Konfiguration abhängig.

5. Anzeigen von Zertifikaten

Eine weitere Gruppe von Funktionen öffnet ein Dialogfenster zur Anzeige eines oder mehrerer Zertifikate, die z.B. in einer S/MIME-Nachricht enthalten sind. Dabei sollte auch eine Überprüfung der Zertifikate auf Wunsch des Benutzers möglich sein.

6. Verarbeiten neuer Zertifikate

Mittels der Funktion `processCertificates` kann eine Anwendung Zertifikate, die z.B. in einer S/MIME-Nachricht enthalten waren und möglicherweise neu sind, zur weiteren Verarbeitung an die Zertifikatsverwaltung übergeben. Diese entscheidet dann auf der Grundlage der jeweiligen Konfiguration, wie weiter zu verfahren ist und ob neue Zertifikate gespeichert oder einfach ignoriert werden sollen.

7. Hilfsfunktionen

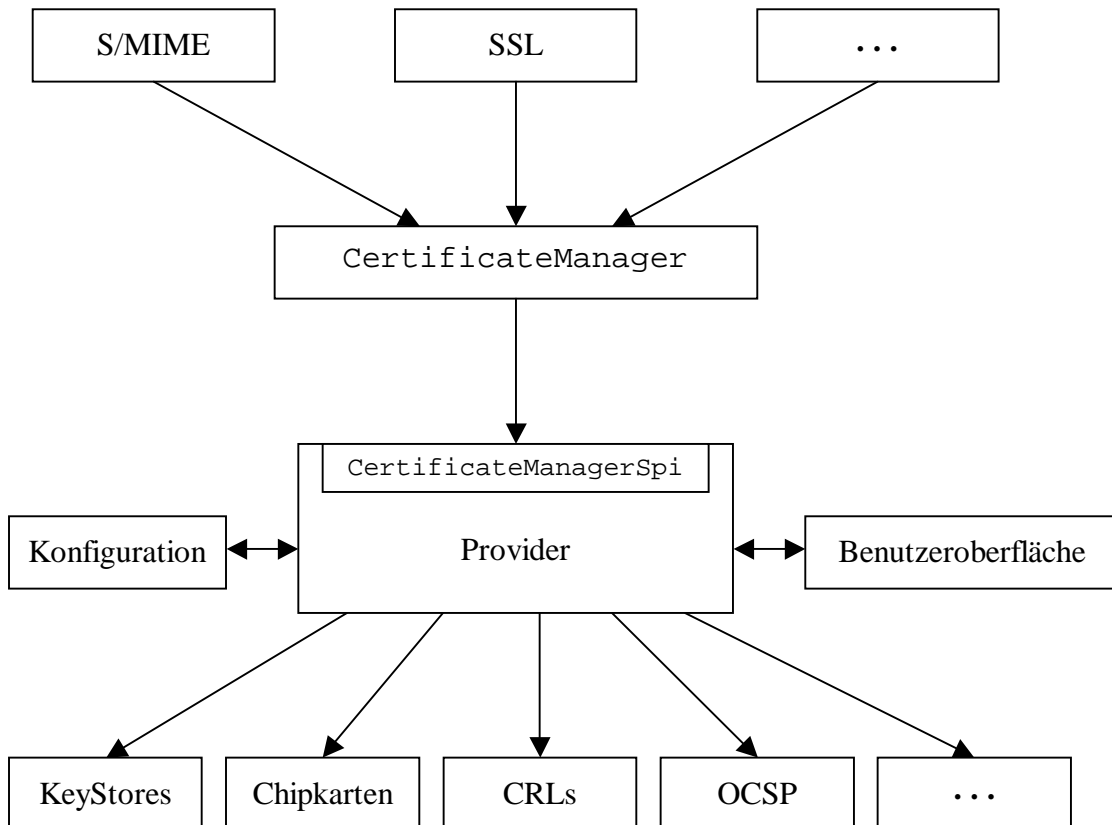
Zusätzlich existieren diverse Hilfsfunktionen, die die Verwendung der `CertificateManager`-Klasse in typischen Anwendungsfällen vereinfachen. So kann beispielsweise als Ergebnis einer Suche direkt die Liste der gefundenen `X509Certificate`-Objekte ohne den Umweg über die IDs geliefert werden oder der zu einem gegebenen `X509Certificate`-Objekt passende private Schlüssel gesucht und als `PrivateKey`-Objekt bereitgestellt werden.

4.3 Die Schnittstelle `CertificateManagerSpi`

Die eigentliche Implementierung der Zertifikatsverwaltung erfolgt außerhalb der Klasse `CertificateManager` durch Provider-Klassen, die die Schnittstelle `CertificateManagerSpi` bereitstellen. Dies ermöglicht die Realisierung unterschiedlicher Lösungen, die von ihrer Komplexität und Leistungsfähigkeit her besser auf bestimmte Anwendungsbereiche, von der privaten Nutzung bis zum Einsatz in einem großen Unternehmen, zugeschnitten werden können. Die Klasse `CertificateManager` leitet dabei die Methodenaufrufe intern an die entsprechende Provider-Klasse weiter, so daß ein Austausch des Providers für die Anwendungsprogramme transparent ist.

Die Schnittstelle `CertificateManagerSpi` ist zudem mit 14 Methoden deutlich kleiner und somit einfacher zu implementieren, als die der Klasse `CertificateManager` mit über 40 Methoden.

Die folgende Abbildung veranschaulicht noch einmal das beschriebene Konzept:



4.4 Testimplementierung auf der Grundlage des Keystore-Managers

Da die Entwicklung einer vollwertigen Zertifikatsverwaltung mit einem erheblichen Aufwand verbunden ist, der den Rahmen dieser Arbeit, die auch so schon rund 800 KB Quelltext beinhaltet, endgültig gesprengt hätte, wurde nur ein einfacher Provider zu Testzwecken implementiert.

Dieser basiert auf dem innerhalb des FlexiPKI-Projektes entwickelten Keystore-Manager, der eine einfache Verwaltung von Java-Keystores erlaubt. Die Funktionen der Zertifikatsverwaltung zum Suchen und Auswählen von Zertifikaten beschränken sich dabei auf die im Keystore-Manager eingetragenen Keystores. Neue Zertifikate werden automatisch in einen Keystore mit dem Namen "NewCerts.ks" übernommen. Eine Überprüfung der Vertrauenswürdigkeit von Zertifikaten ist nicht realisiert.

Ausführlichere Informationen zu den in diesem Kapitel vorgestellten Schnittstellen und Klassen finden sich in der mit JavaDoc erzeugten Dokumentation.

5 BENUTZEROBERFLÄCHE FÜR S/MIME-ERWEITERUNGEN

Neben den in Kapitel 3 vorgestellten Klassen zum Kodieren und Dekodieren von S/MIME-Nachrichten spielt bei einem S/MIME-Plugin auch die Benutzeroberfläche eine große Rolle. Dieses Kapitel befaßt sich mit den hierfür gewählten Konzepten.

5.1 Grundlagen

Für die Interaktion mit dem Benutzer werden Dialoge zum Einstellen der Konfiguration, sowie zum Anzeigen des Status einer Nachricht und möglicherweise auftretender Fehler benötigt. Diese sollten so allgemein gehalten sein, daß sie sich bei der Entwicklung von Plugins für andere Mailprogramme direkt wiederverwenden lassen. Die Unterstützung mehrerer Sprachen sollte ebenfalls vorgesehen werden.

Die Verarbeitung einer Nachricht vor dem Anzeigen oder Versenden ist so eng mit der Funktionalität der Benutzeroberfläche verbunden, daß hier keine klare Trennung vorgenommen werden kann. Deshalb wird in den Abschnitten 5.4 und 5.5 jeweils beides zusammen betrachtet.

5.2 Die Klasse `SMimeUI`

Die Klasse `SMimeUI` enthält diverse auf die Benutzeroberfläche bezogene statische Hilfsfunktionen, von denen die vier wichtigsten im folgenden vorgestellt werden.

1. initializeDefault

Diese Methode führt alle für S/MIME erforderlichen Initialisierungen durch und sollte daher vor jeder anderen Methode aufgerufen werden. Sie lädt dabei das systemspezifische "Look and Feel" für Swing, registriert die `DataContentHandler`-Klassen¹⁰ für S/MIME, sowie den am Lehrstuhl von Prof. Buchmann entwickelten CDC-Provider, der die benötigten kryptographischen Algorithmen über die Java Cryptography Architecture bereitstellt.

2. showExceptionDialog

Mit Hilfe dieser Methode lassen sich unerwartet aufgetretene Java-Ausnahmen dem Benutzer anzeigen. Dieser kann dabei auf Wunsch auch den mit der Ausnahme verbundenen Stack-Trace einsehen, was insbesondere bei der Fehlersuche von Vorteil ist.

¹⁰ siehe Abschnitt 3.4

3. *showCertificateManagerDialog*

Diese Methode öffnet den Hauptdialog der Zertifikatsverwaltung, über den der Benutzer im allgemeinen die Konfigurationseinstellungen bearbeiten, sowie Zertifikate und Schlüssel im- und exportieren kann. Die genaue Funktion ist dabei von der verwendeten Zertifikatsverwaltung abhängig.

4. *showOptionsDialog*

Diese Methode zeigt ein Dialogfenster, über das der Benutzer die S/MIME-Konfiguration¹¹ bearbeiten kann.

5.3 Die Klassen `SMimeOptions` und `SMimeOptionsDialog`

Die Klasse `SMimeOptions` enthält die S/MIME-spezifische Grundkonfiguration. Diese wird aus einer Datei mit dem Namen "FlexiSMIME.cfg" im Homedirectory des jeweiligen Benutzers eingelesen.

Folgende Einstellungen können dabei vorgenommen werden:

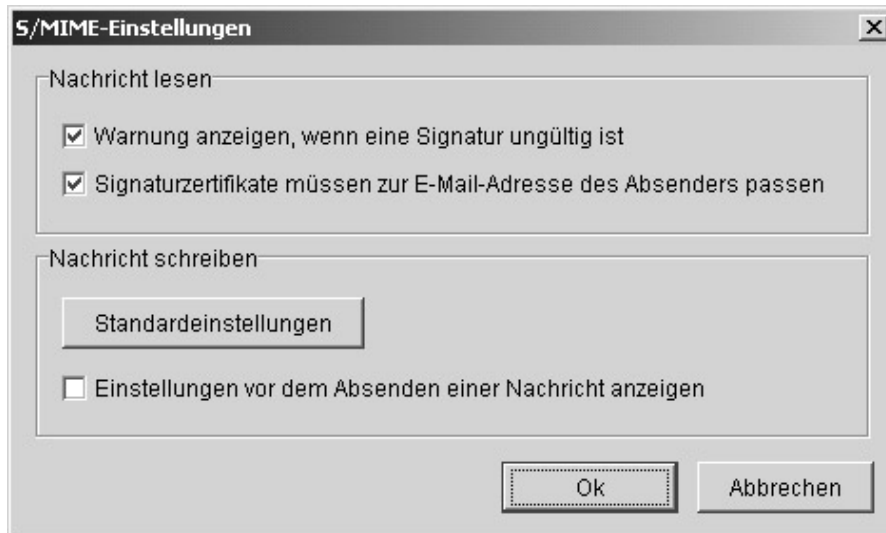
- automatisches Anzeigen einer Warnung beim Öffnen einer Nachricht mit ungültiger Signatur (ja / nein)
- die zum Signieren einer Nachricht verwendeten Zertifikate müssen zur E-Mail-Adresse des Absenders passen (ja / nein)
- automatisches Anzeigen der S/MIME-Einstellungen einer Nachricht¹² vor deren Absenden (ja / nein)
- Festlegen der Voreinstellungen für neue Nachrichten¹³

¹¹ siehe Abschnitt 5.3

¹² siehe Abschnitt 5.5.2

¹³ siehe Abschnitt 5.5.2

Die Klasse `SMimeOptionsDialog` implementiert den folgenden Swing-Dialog zum Anzeigen und Bearbeiten der S/MIME-Konfiguration auf der Grundlage eines `SMimeOptions`-Objekts.



5.4 Nachrichten lesen

Vor dem Anzeigen einer Nachricht durch das Mailprogramm muß zunächst überprüft werden, ob sie Signaturen oder verschlüsselte Teile enthält. In diesem Fall ist eine entsprechende Verarbeitung nötig, über deren Ergebnis (Signatur gültig? Entschlüsselung erfolgreich?) der Benutzer in geeigneter Form informiert werden sollte. Die im folgenden beschriebenen Klassen übernehmen diese Aufgaben.

5.4.1 Die Klasse `SMimeReadMessageContext`

Eine Instanz der Klasse `SMimeReadMessageContext` ermöglicht die Verarbeitung einer Nachricht vor dem Anzeigen. Sie hält danach die S/MIME-spezifischen Statusinformationen¹⁴, die dem Benutzer auf Wunsch in Form eines Dialogfensters angezeigt werden. Im Falle einer ungültigen Signatur oder beim Auftreten von Fehlern während der Verarbeitung erfolgt die Anzeige entsprechend der Konfiguration¹⁵ auch automatisch.

Im einfachsten Fall ist nur der Aufruf von zwei Methoden dieser Klasse nötig:

1. *processMessage*

Diese Methode verarbeitet eine gegebene Nachricht. Sie überprüft dabei enthaltene Signaturen, entschlüsselt chiffrierte Teile und aktualisiert die Statusinformationen. Dabei

¹⁴ siehe Abschnitt 5.4.2

¹⁵ siehe Abschnitt 5.3

werden alle S/MIME-Strukturen durch ihren jeweiligen signierten oder entschlüsselten Inhalt ersetzt, so daß die Nachricht später von jedem MIME-fähigen Mailclient korrekt angezeigt werden kann. Treten bei der Verarbeitung Fehler auf oder ist der für eine Entschlüsselung notwendige private Schlüssel nicht verfügbar, wird der jeweilige Teil aus der Nachricht entfernt.

2. *showMessagePropertiesDialog*

Diese Methode öffnet ein Dialogfenster, das den Benutzer über den S/MIME-Status der Nachricht¹⁶ informiert.

5.4.2 Die Klassen `SMimeMessageStatus` und `SMimeMessageStatusDialog`

Die Klasse `SMimeMessageStatus` enthält alle S/MIME-spezifischen Statusinformationen einer Nachricht. Dies sind im einzelnen:

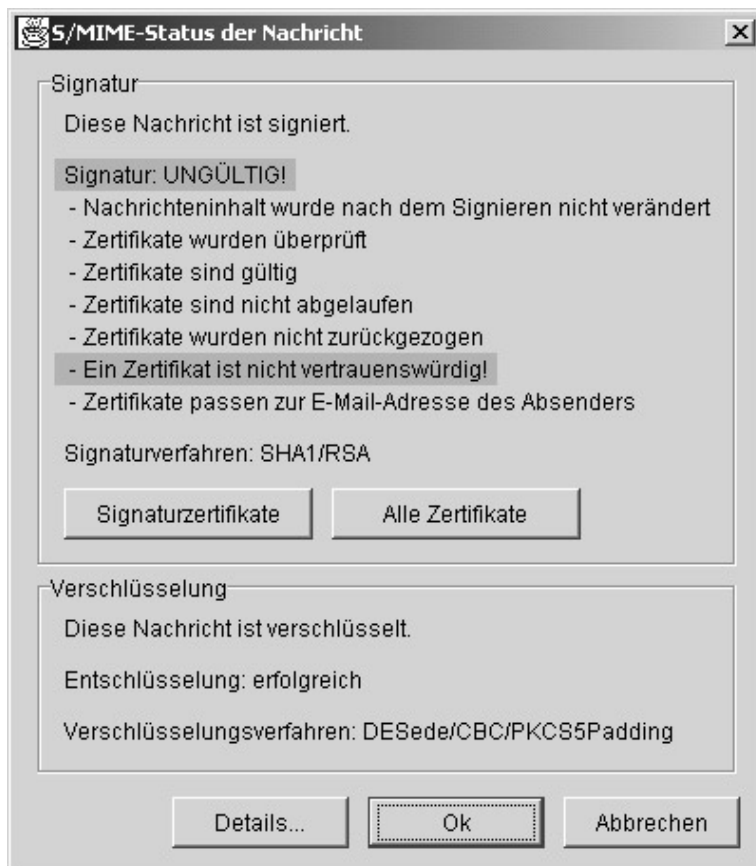
- Verarbeitung fehlerfrei (ja / nein)
- Nachricht signiert (ja / nein / teilweise)
- Hashwert der Signatur korrekt (ja / nein)
- Status der Zertifikate:
 - verfügbar (ja / nein)
 - gültig (ja / nein)
 - nicht abgelaufen (ja / nein)
 - nicht zurückgezogen (ja / nein)
 - vertrauenswürdig (ja / nein)
 - bekannt (ja / nein)
 - überprüfbar (ja / nein)
 - zur E-Mail-Adresse des Absenders passend (ja / nein)
- Signaturverfahren bekannt (ja / nein)
- Verifikation der Signaturen fehlerfrei (ja / nein)
- Liste der verwendeten Signaturverfahren
- Liste der zum Signieren verwendeten Zertifikate
- Liste aller in der Nachricht enthaltenen Zertifikate
- Nachricht verschlüsselt (ja / nein / teilweise)
- zur Entschlüsselung notwendige private Schlüssel verfügbar (ja / nein)

¹⁶ siehe Abschnitt 5.4.2

- Verschlüsselungsverfahren bekannt (ja / nein)
- Entschlüsselung nicht vom Benutzer abgebrochen (ja / nein)
- Entschlüsselung fehlerfrei (ja / nein)
- Liste der verwendeten Verschlüsselungsverfahren
- zusätzliche Detailinformationen in Textform (Struktur der Nachricht, Status jeder einzelnen Signatur, aufgetretene Exceptions, etc.)

Wenn eine Nachricht mehrere Signaturen oder mehrere verschlüsselte Teile enthält, geben die Statusinformationen die "Summe" der aufgetretenen Probleme wieder.

Die Klasse `SMimeMessageStatusDialog` implementiert den folgenden Swing-Dialog zum Anzeigen der Statusinformationen auf der Grundlage eines `SMimeMessageStatus`-Objekts.



5.5 Nachrichten schreiben

Der Benutzer sollte die Möglichkeit haben, während des Schreibens einer Nachricht deren S/MIME-Einstellungen¹⁷ zu bearbeiten. Wird dabei das Signieren oder Verschlüsseln der Nachricht aktiviert, ist vor dem Absenden eine entsprechende Verarbeitung erforderlich. Die im folgenden beschriebenen Klassen übernehmen diese Aufgaben.

5.5.1 Die Klassen `SMimeMessageOptions` und `SMimeMessageOptionsDialog`

Die Klasse `SMimeMessageOptions` enthält die S/MIME-Einstellungen einer Nachricht oder im Rahmen der S/MIME-Grundkonfiguration¹⁸ die entsprechenden Voreinstellungen. Dies sind im einzelnen:

- Nachricht vor dem Absenden signieren (ja / nein)
- Liste von Zertifikaten und Signaturalgorithmen (siehe Anmerkung)
- gewählte Zertifikate zur Nachricht hinzufügen (ja / nein)
- signierte Nachrichten mit Content-Type "multipart/signed" statt "application/pkcs7-mime"¹⁹ verschicken (ja / nein)
- Nachricht vor dem Absenden verschlüsseln (ja / nein)
- zu verwendendes Verschlüsselungsverfahren
- eigenes Zertifikat bei der Verschlüsselung als Empfänger mit einschließen, damit man selbst die Nachricht bei Bedarf wieder entschlüsseln kann (ja / nein)
- zum Verschlüsseln für sich selbst zu verwendendes eigenes Zertifikat

Anmerkung zur Liste von Zertifikaten und Signaturalgorithmen:

Hier werden die Zertifikate, mit denen die Nachricht signiert werden soll, zusammen mit dem jeweils gewünschten Signaturverfahren gewählt. Dabei können beliebig viele Zertifikate und auch ein Zertifikat mehrmals mit unterschiedlichen Signaturverfahren angegeben werden, um die Sicherheit zu erhöhen oder beim Einsatz neuer Verfahren die Abwärtskompatibilität zu gewährleisten. So wäre es beispielsweise bei der Verwendung eines auf elliptischen Kurven basierenden Public-Key-Kryptoalgorithmus ratsam, zusätzlich auch RSA einzusetzen, damit jedes S/MIME-fähige Mailprogramm die Signatur überprüfen kann.

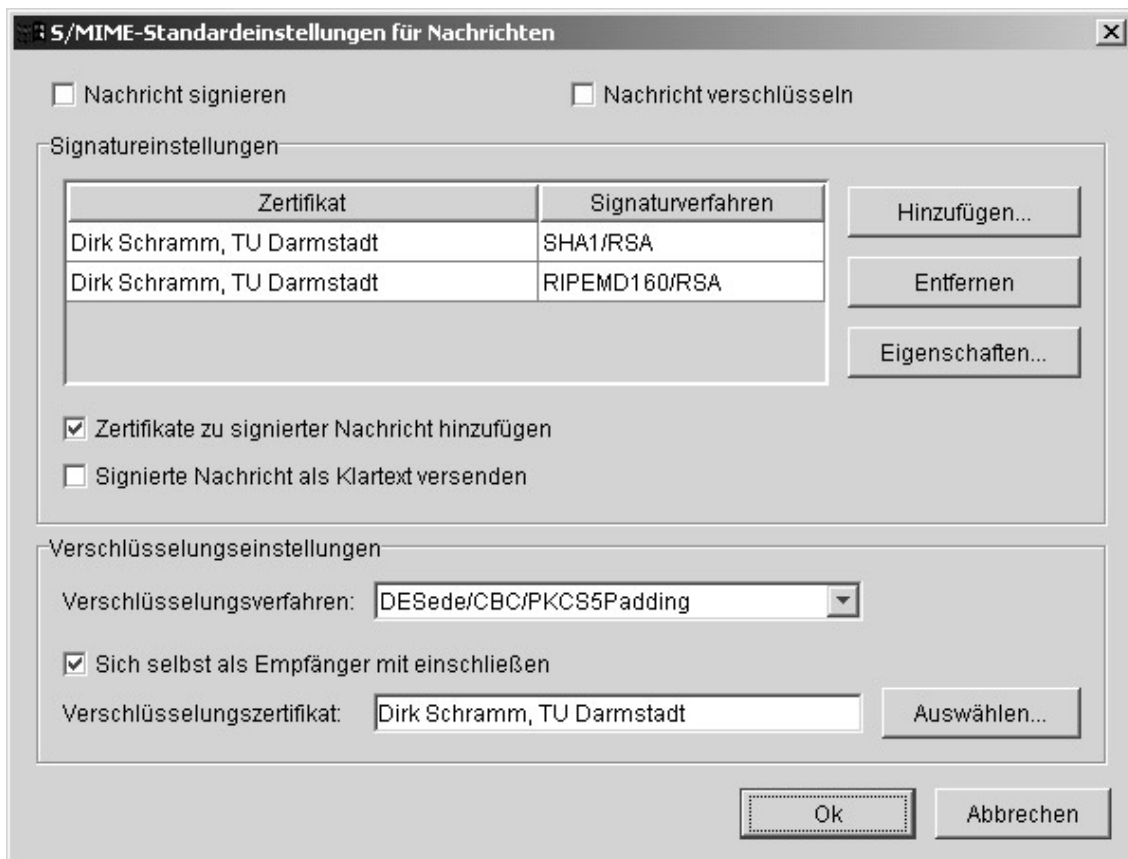
Zusätzlich besteht die Möglichkeit, beliebige weitere Zertifikate in die Liste aufzunehmen ohne für sie ein Signaturverfahren zu wählen. Diese werden dann lediglich zur Nachricht

¹⁷ siehe Abschnitt 5.5.1

¹⁸ siehe Abschnitt 5.3

hinzugefügt. Auf diesem Wege können unter anderem Root-Zertifikate oder solche, deren Einsatz auf das Ver- und Entschlüsseln von Nachrichten beschränkt ist, dem Empfänger übermittelt werden.

Die Klasse `SMimeMessageOptionsDialog` implementiert den folgenden Swing-Dialog zum Bearbeiten der S/MIME-Einstellungen einer Nachricht auf der Grundlage eines `SMimeMessageOptions`-Objekts.



5.5.2 Die Klasse `SMimeWriteMessageContext`

Eine Instanz der Klasse `SMimeWriteMessageContext` hält die S/MIME-Einstellungen²⁰ einer Nachricht, die der Benutzer gerade verfaßt. Auf dessen Wunsch hin kann diese Konfiguration über ein Dialogfenster eingesehen und bearbeitet werden. Die Voreinstellungen liefert dabei die S/MIME-Grundkonfiguration²¹. Zusätzlich übernimmt die jeweilige Instanz der Klasse das Signieren und/oder Verschlüsseln der Nachricht entsprechend der gewählten Einstellungen vor dem Absenden.

Im einfachsten Fall ist nur der Aufruf von zwei Methoden dieser Klasse nötig:

¹⁹ siehe Abschnitt 1.2.2

²⁰ siehe Abschnitt 5.5.1

²¹ siehe Abschnitt 5.3

1. *showMessagePropertiesDialog*

Diese Methode öffnet ein Dialogfenster, über das der Benutzer die S/MIME-Einstellungen²² der Nachricht verändern kann.

2. *processMessage*

Diese Methode verarbeitet eine gegebene Nachricht vor dem Absenden. Hat der Benutzer innerhalb der S/MIME-Einstellungen das Signieren aktiviert, werden mit den dafür gewählten Zertifikaten Signaturen über den gesamten Nachrichteninhalte erstellt und entsprechend den Einstellungen in einer S/MIME-Struktur mit dem Content-Type "multipart/signed" oder "application/pkcs7-mime"²³ kodiert.

Bei aktivierter Verschlüsselung werden zunächst die Zertifikate aller Empfänger der Nachricht anhand der E-Mail-Adressen gesucht. Ist diese Suche in einzelnen Fällen nicht erfolgreich, kann der Benutzer jeweils ein Zertifikat manuell auswählen, den Empfänger überspringen oder den Vorgang abbrechen. Danach erfolgt die Verschlüsselung des gesamten Mailinhalts mit dem in den S/MIME-Einstellungen gewählten Verfahren, wobei für jedes der zuvor eingelesenen Zertifikate, sowie optional auch für das des Absenders, je eine RecipientInfo-Struktur²⁴ erstellt und der Nachricht hinzugefügt wird.

Entsprechend der S/MIME-Grundkonfiguration²⁵ zeigt diese Methode zudem optional vor der Verarbeitung der Nachricht die aktuellen Einstellungen an, so daß der Benutzer sie bei Bedarf noch verändern kann.

5.6 Mehrsprachigkeit

Um eine einfache Übersetzung der Benutzeroberfläche in andere Sprachen zu ermöglichen, sind sämtliche Texte in ListResourceBundle-Klassen ausgelagert. Diese befinden sich jeweils in einem "Subpackage" mit dem Namen "resources" und haben den gleichen Namen wie die Klasse, die die Ressourcen verwendet, mit angehängter Sprachkennung.

Beispielsweise besitzt die Klasse `smime.ui.SMimeOptionsDialog` derzeit zwei Ressourcenklassen:

Klasse	Sprache
<code>smime.ui.resources.SMimeOptionsDialog</code>	Standard (Englisch)
<code>smime.ui.resources.SMimeOptionsDialog_de</code>	Deutsch

²² siehe Abschnitt 5.5.1

²³ siehe Abschnitt 1.2.2

²⁴ siehe Abschnitt 1.2.3

²⁵ siehe Abschnitt 5.3

Die Klasse `SMimeUIResources` enthält diverse statische Hilfsfunktionen für einen besonders einfachen Zugriff auf die Ressourcen. An dieser Stelle sei vor allem die Methode `load` erwähnt, die für einen gegebenen Swing-Dialog automatisch sämtliche in der zugehörigen Ressourcen-Klasse spezifizierten Texte lädt. Dabei muß lediglich über die von der Klasse `JComponent` implementierte Methode `setName` jeder Komponente des Dialogs, die Text enthält, ein Name zugeordnet werden. Die Schlüssel der Ressourcen setzen sich dann aus diesem Namen und verschiedenen Suffixes wie ".Text", ".Title", ".ToolTipText", ".Mnemonic" etc. zusammen.

Einfache Hinweis- oder Abfragedialoge, die die Klasse `JOptionPane` bereitstellt, lassen sich über die Methoden `showMessageDialog` und `showOptionDialog` anzeigen. Auch hier werden alle benötigten Ressourcen automatisch auf der Grundlage eines gegebenen Dialognamens geladen.

Zudem ermöglicht die Klasse `SMimeUIResources` die Verwendung von Platzhaltern innerhalb von Ressourcen-Strings, die zur Laufzeit durch dynamische Daten ersetzt werden können.

5.7 Entwicklung eines S/MIME-Plugins

Die in diesem Kapitel beschriebenen Klassen bieten alle wichtigen Funktionen, die zur Entwicklung eines S/MIME-Plugins notwendig sind. Dieser Abschnitt liefert hierzu einen zusammenfassenden Überblick.

1. Initialisierung

Das Plugin sollte nach dem Laden zunächst die Methode `initializeDefault` der Klasse `SMimeUI`²⁶ aufrufen, um die notwendige Initialisierung durchzuführen.

2. S/MIME-Menü

Um dem Benutzer eine einfache Bedienung zu ermöglichen, sollten das Hauptmenü des Mailprogramms und/oder entsprechende Werkzeuggesten mindestens um die folgenden S/MIME-Funktionen erweitert werden, die auch direkt von der Klasse `SMimeUI` bereitgestellt werden:

Funktion	Methode der Klasse <code>SMimeUI</code>
Zertifikatsverwaltung starten	<code>showCertificateManagerDialog</code>
S/MIME-Grundeinstellungen bearbeiten	<code>showOptionsDialog</code>
Allgemeine Informationen anzeigen	<code>showAboutDialog</code>

²⁶ siehe Abschnitt 5.2

3. Nachricht lesen

Beim Öffnen einer S/MIME-Nachricht durch den Benutzer muß das Plugin für die Verarbeitung vor dem Anzeigen sorgen. Dazu wird zunächst eine Instanz der Klasse `SMimeReadMessageContext`²⁷ erstellt und die Nachricht an die Methode `processMessage` übergeben. Wenn diese einen Fehler zurückmeldet, sollte der gesamte Vorgang abgebrochen werden. Anderenfalls enthält die Nachricht keine S/MIME-Strukturen mehr und kann vom Mailprogramm angezeigt werden.

Während dessen sollte der Benutzer über das Menü oder ein Werkzeugleistensymbol den Status der Nachricht abfragen können, der sich durch den Aufruf der Methode `showMessageProperties` des `SMimeReadMessageContext`-Objekts ausgeben läßt.

4. Nachricht schreiben

Beginnt der Benutzer mit dem Schreiben einer Nachricht, sollte das Plugin eine Instanz der Klasse `SMimeWriteMessageContext` erstellen und es dem Benutzer über einen entsprechenden Menüpunkt oder ein Werkzeugleistensymbol ermöglichen, die S/MIME-Einstellungen der Nachricht zu bearbeiten. Dies wird intern durch einen Aufruf der Methode `showMessageProperties` des `SMimeWriteMessageContext`-Objekts realisiert.

Vor dem Absenden muß das Plugin die Nachricht, falls nötig, signieren oder verschlüsseln. Durch einen Aufruf der Funktion `messageProcessingRequired` des `SMimeWriteMessageContext`-Objekts kann zunächst überprüft werden, ob der Benutzer Signatur oder Verschlüsselung aktiviert hat. Falls ja, muß die Nachricht an die Methode `processMessage` übergeben werden, die die entsprechende Verarbeitung übernimmt. Wenn diese einen Fehler zurückmeldet, sollte der Vorgang abgebrochen werden und der Benutzer das Bearbeiten der Nachricht fortsetzen können. Anderenfalls enthält die Nachricht die benötigten S/MIME-Strukturen und kann vom Mailprogramm abgeschickt werden.

Ausführlichere Informationen zu den in diesem Kapitel vorgestellten Klassen finden sich in der mit JavaDoc erzeugten Dokumentation.

²⁷ siehe Abschnitt 5.4.1

6 JAVA NATIVE INTERFACE FRAMEWORK FÜR MICROSOFT VISUAL C++

Alle bisher vorgestellten Klassen sind in Java implementiert, wodurch die Plattformunabhängigkeit und damit auch eine gute Wiederverwendbarkeit sichergestellt werden. Für die Erweiterung von Microsoft Outlook ist jedoch die Entwicklung einer DLL (Dynamic Link Library) in C++ erforderlich (siehe Kapitel 7). Dieses Kapitel beschreibt, welche Voraussetzungen für die Nutzung von Java-Klassen in C++ zu schaffen sind.

6.1 Grundlagen

Das Java Native Interface (JNI) ermöglicht die Verwendung der Java Virtual Machine (JVM) in eigenen Anwendungen, die in C oder C++ geschrieben sind. Im Falle von Microsoft Windows existiert hierfür eine DLL, die Funktionen bereitstellt, um die JVM zu laden, Java-Objekte zu erstellen, Methoden aufzurufen, etc. Die Programmierung erfolgt dabei jedoch auf einer recht niedrigen Ebene, auf der der Entwickler selbst dafür Sorge zu tragen hat, daß beispielsweise Objekt-Referenzen wieder freigegeben werden und beim Aufruf von Java-Methoden die richtige Anzahl von Parametern mit den korrekten Typen vorliegt. Zudem prüft die JVM aus Effizienzgründen bei den meisten Funktionsaufrufen die Parameter nicht auf Gültigkeit, so daß Fehler häufig zu Schutzverletzungen und damit Programmabbrüchen führen.

Es erscheint deshalb sinnvoll, zwischen der C++-Anwendung und dem Java Native Interface eine weitere Ebene einzuführen, die weitgehend von den technischen Detailproblemen des JNI abstrahiert und einen einfachen und weniger fehleranfälligen Umgang mit Java-Klassen ermöglicht. Hierfür bietet das objektorientierte Konzept von C++ ideale Voraussetzungen. Das Ziel dabei ist die Verwendung von C++-Klassen, die jeweils eine Java-Klasse repräsentieren und sämtliche Methodenaufrufe an diese weiterleiten.

Es existieren bereits Lösungen für ein solches "Framework" wie z.B. JACE²⁸, die jedoch keine ausreichende Integration in das Betriebssystem Windows bieten. Dies gilt insbesondere beim Umgang mit String-Objekten, sowie für die Anzeige modaler Swing-Dialoge²⁹. Vor allem letzteres ist für eine S/MIME-Erweiterung aber zwingend erforderlich. Daher wurde im Rahmen dieser Diplomarbeit ein eigenes JNI Framework für Microsoft Windows, basierend auf Visual C++ 6.0 entwickelt, dessen wichtigste Konzepte im folgenden vorgestellt werden.

²⁸ siehe <http://www.reyelts.com/JACE/>

²⁹ siehe Abschnitt 6.7

6.2 Die Klasse JObject

Die C++-Klasse `JObject` repräsentiert die Java-Klasse `java.lang.Object` und stellt somit die Spitze der Klassenhierarchie dar. Ihre wichtigste Aufgabe besteht in der automatischen Verwaltung der Referenz des zugrundeliegenden Java-Objekts. Die Idee dabei ist die Verwendung von Autovariablen dieses Typs (oder einer der Unterklassen), die sich durch ein entsprechendes Überladen der Zuweisungs- und Vergleichsoperatoren wie Objektvariablen in Java verhalten. Beispiel:

```
JObject o1 = JNULL;
JObject o2 = JNULL;

o1 = getSomeObject();
if (o1 != JNULL) {
    o2 = o1.toString();
}
```

Sobald der Gültigkeitsbereich der Variablen `o1` und `o2` verlassen wird (auch im Falle einer Exception) gibt der Destruktor der Klasse `JObject` die jeweilige Referenz des Java-Objekts automatisch frei. So werden "Memory Leaks" wirksam vermieden.

Bei der Zuweisung einer Objektvariable an eine andere erfolgt zur Laufzeit zunächst eine Typprüfung auf der Grundlage des `JClass`-Objekts³⁰ der Zielvariable. Ist der Typ des zugewiesenen Objekts nicht kompatibel, wird eine Ausnahme vom Typ `JniClassCastException` ausgeworfen. Anderenfalls erfolgt nach der Freigabe der bisherigen Objektreferenz der Zielvariable eine Duplikation der zugewiesenen Referenz.

6.3 Die Klasse JClass

Die C++-Klasse `JClass` stellt das Pendant zur Java-Klasse `java.lang.Class` dar. Instanzen dieses Typs repräsentieren beliebige Java-Klassen und ermöglichen es, deren statische Methoden aufzurufen, sowie die statischen Felder zu lesen oder zu verändern. Zudem lassen sich Objekte der jeweiligen Java-Klasse erstellen und es kann auch auf deren Methoden und Felder zugegriffen werden.

Für diese Aufgaben stellt die Klasse `JClass` alle notwendigen Funktionen zur Verfügung. Sie bildet damit die Grundlage für die Implementierung der Schnittstelle einer Java-Klasse durch eine C++-Klasse. Dementsprechend enthält `JObject` eine statische Funktion `getClass`, die in jeder Unterklasse reimplementiert wird, und das jeweilige `JClass`-Objekt zurückgibt. Um die in Abschnitt 6.2 angesprochene Typprüfung bei Zuweisungen vornehmen zu können, wird zusätzlich in jeder Unterklasse von `JObject` die virtuelle Funktion `getObjectClass` überschrieben, so daß sich der Typ einer Objektvariable zur Laufzeit feststellen läßt.

³⁰ siehe Abschnitt 6.3

6.4 Die Klasse `JString`

Die C++-Klasse `JString` repräsentiert die Java-Klasse `java.lang.String` und überträgt den von Java bekannten bequemen Umgang mit Zeichenketten durch das weitere Überladen von Operatoren auch auf C++. Zusätzlich läßt sich ein `JString`-Objekt durch implizite Typumwandlungen wie eine Variable des Typs `char*` (ASCII) oder `wchar_t*` (Unicode) verwenden und somit direkt als Parameter an entsprechende C++-Funktionen übergeben. Das folgende Beispiel verdeutlicht dies:

```
JString s1 = "Hello";
char    *s2 = " ";
JString s3 = JNULL;

s3 = "world";
s1 += s2 + s3 + "!";
cout << s1 << endl << flush;
```

(Ausgabe: "Hello world!")

6.5 Die Klassen `JArray` und `JObjectArray`

Für den Umgang mit Java-Arrays stehen zwei Template-Klassen zur Verfügung. `JArray` kann dabei mit jedem einfachen Java-Datentyp verwendet werden (`jint`, `jchar`, etc.) und erlaubt neben dem elementweisen Zugriff auch das effiziente Kopieren ganzer Indexbereiche zwischen dem Array und einem gegebenen Speicherblock. Alle gängigen Array-Typen sind bereits definiert (`JByteArray`, `JCharArray`, etc.).

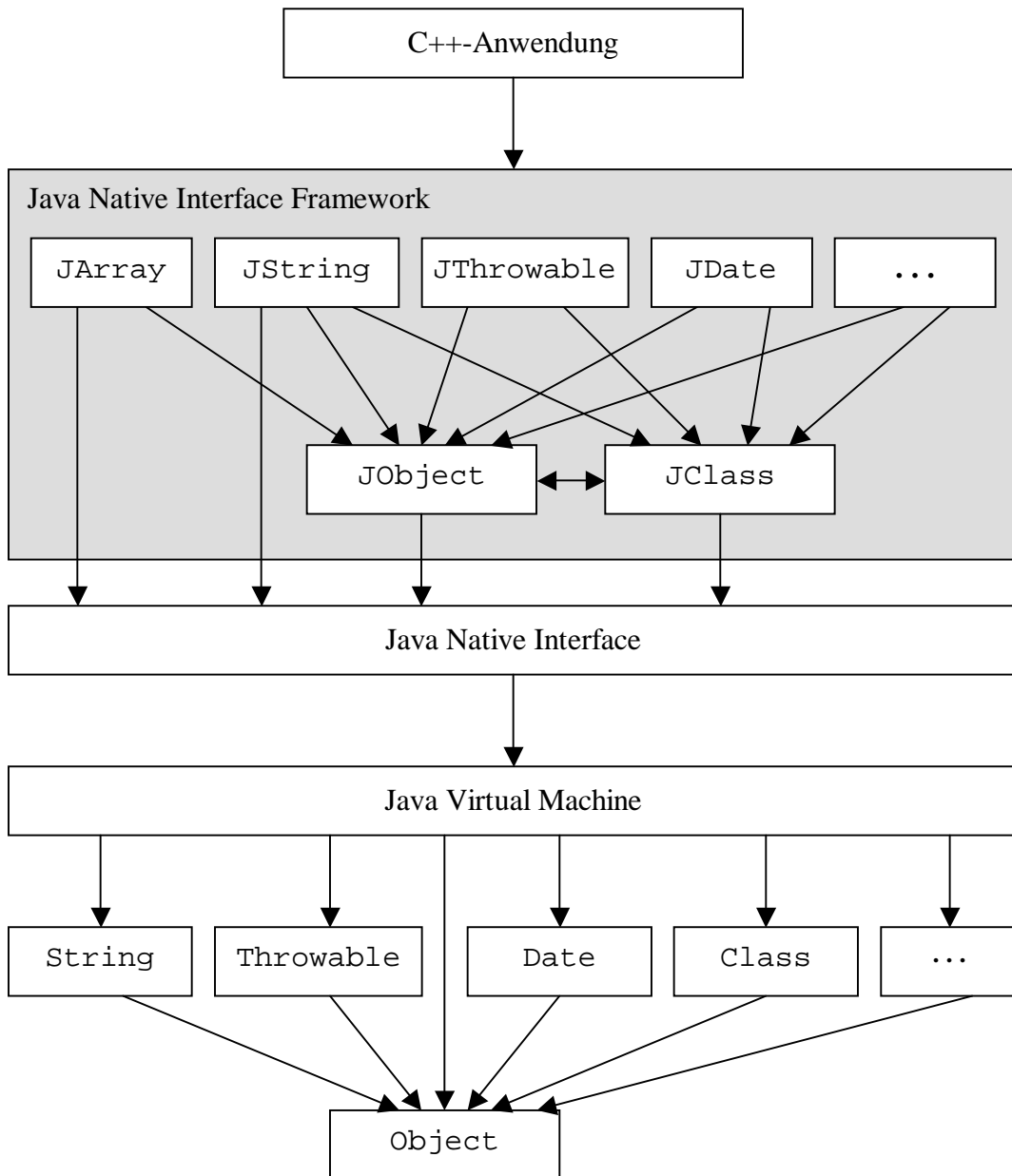
Bei der Verwendung von Objekt-Arrays kommt die zweite Klasse `JObjectArray` zum Einsatz. Sie erlaubt zwar nur einen elementweisen Zugriff, dafür werden jedoch beim Auslesen automatisch die dem Elementtyp entsprechenden C++-Objekte erstellt.

6.6 Bisher implementierte Wrapper-Klassen

Die folgende Tabelle liefert eine Übersicht der bisher implementierten C++-"Wrapper"-Klassen und ihrer jeweiligen Java-Pendants.

C++-Klasse	Java-Klasse
JObject	<code>java.lang.Object</code>
JClass	<code>java.lang.Class</code>
JString	<code>java.lang.Class</code>
JThrowable	<code>java.lang.Throwable</code>
JException	<code>java.lang.Exception</code>
JDate	<code>java.util.Date</code>
JMailHeader	<code>javax.mail.Header</code>
JMailInternetAddress	<code>javax.mail.internet.InternetAddress</code>
SMime	<code>smime.SMime</code>
SimpleMimeMessage	<code>smime.SimpleMimeMessage</code>
SimpleMimeAttachment	<code>smime.SimpleMimeAttachment</code>
SMimeUI	<code>smime.ui.SMimeUI</code>
SMimeMessageContext	<code>smime.ui.SMimeMessageContext</code>

Die folgende Darstellung verdeutlicht noch einmal die bisher beschriebene Struktur des Java Native Interface Frameworks:



6.7 Anzeigen von modalen Swing-Dialogen

Das Anzeigen modaler Dialogfenster, die in Java mit Swing implementiert sind, aus einer normalen Windows-Anwendung heraus ist mit verschiedenen Problemen verbunden. Das erste besteht darin, daß bei der Verwendung der Methode `show` eines modalen Swing-Dialogs der aufrufende Thread (falls es sich nicht um den Event-Dispatching Thread handelt) so lange blockiert, bis das angezeigte Fenster wieder geschlossen wird. Somit darf ein solcher Aufruf nicht von einem Thread einer Windows-Anwendung aus erfolgen, der eigene Fenster geöffnet hat, weil sonst die Verarbeitung der Window-Messages unterbrochen würde. Dadurch wiederum kann der Fensterinhalt nicht mehr neu gezeichnet werden, wenn beispielsweise ein anderes darüberliegendes Fenster verschoben oder geschlossen wird, und es entsteht für den Benutzer der Eindruck, als seien die entsprechenden Fenster "tot".

Dieses Problem läßt sich lösen, indem zunächst ein Worker-Thread gestartet wird, der dann den modalen Swing-Dialog anzeigt. Der ursprüngliche Thread wartet währenddessen auf das Ende des Worker-Threads, verarbeitet dabei aber weiter die Window-Messages. Um zu vermeiden, daß dadurch auch Benutzereingaben über Maus oder Tastatur für andere Fenster akzeptiert werden, was dem Prinzip eines modalen Dialogs widersprechen würde, müssen vorher alle sichtbaren Fenster für Eingaben deaktiviert werden.

Ein weiteres Problem besteht in der eigentlich üblichen Verbindung eines Dialogs zu seinem "Owner-Window". Dieses wäre hier ein Fenster der Windows-Anwendung, das sich beim Erzeugen des Swing-Dialogs jedoch nicht in Form eines `Window`-Objekts spezifizieren läßt. Dadurch wird ein Anwählen des Owner-Windows z.B. über die Task-Leiste nicht automatisch an das Dialogfenster weitergeleitet, sondern das Owner-Window selber kommt in den Vordergrund und verdeckt dabei möglicherweise das Dialogfenster. Dieses läßt sich in einem solchen Fall nur noch über die Tastenkombination Alt-Tabulator zurückholen, weil es in der Task-Leiste nicht sichtbar ist und das Owner-Window bedingt durch die vorhergehende Deaktivierung der Eingabe nicht minimiert oder verschoben werden kann.

Um dieses Problem zu vermeiden, wird zunächst über die Windows-API ein echtes Dialogfenster für das Hauptfenster der Anwendung erzeugt, das jedoch für den Benutzer nicht sichtbar ist. Sobald dieses Fenster aktiviert wird (was auch nach einer vorhergehenden Aktivierung des Hauptfensters passiert), holt es den Swing-Dialog in den Vordergrund. Auch das ist noch mit Problemen verbunden, weil über die Swing-Klassen kein Zugriff auf die vom Betriebssystem benutzten Window-Handles möglich ist. Deshalb wird nach einem sichtbaren und für Eingabe aktivierten Fenster des aktuellen Prozesses gesucht, das nicht zum aufrufenden Thread gehört, um dieses dann schließlich zu aktivieren.

Die im Modul `jni_util` implementierte Funktion `jniRunModal` übernimmt all die in diesem Abschnitt genannten Aufgaben. Als Parameter wird dabei ein Verweis auf die

Instanz einer Unterklasse von `JniRunnable` übergeben, die die virtuelle Funktion `run` implementiert. Mit Hilfe dieser Funktion läßt sich beliebiger Code in dem bereits beschriebenen Worker-Thread ausführen. Dabei auftretende Ausnahmen werden gespeichert und im aufrufenden Thread erneut ausgeworfen, so daß die Anwendung sie auf einfache Weise verarbeiten kann.

6.8 Ausnahmen

Treten beim Ausführen einer Funktion des JNI Frameworks Fehler auf, wird eine Ausnahme des Typs `JniException` oder einer Unterklasse ausgeworfen. Der Zugriff auf die Fehlermeldung in Textform ist dabei unter anderem durch eine Typumwandlung des jeweiligen Objekts in `char*` oder `wchar_t*` möglich.

Besonders hervorgehoben sei an dieser Stelle die Unterklasse `JThrowable`, die die Java-Klasse `java.lang.Throwable` repräsentiert und somit die Basisklasse für alle innerhalb der Java Virtual Machine oder des Java-Codes ausgeworfenen Ausnahmen darstellt. Die Ausnahmebehandlung in C++ läßt sich folglich durch Abfangen von `JThrowable` oder einer der Unterklassen wie `JException` leicht auf Java-Ausnahmen erweitern.

Ausführlichere Informationen zu den in diesem Kapitel vorgestellten Klassen finden sich in den Kommentaren der jeweiligen Quelltextdateien.

7 S/MIME-ERWEITERUNG FÜR MICROSOFT OUTLOOK

Nachdem in den bisherigen Kapiteln alle Grundlagen für die Entwicklung von S/MIME-Plugins beschrieben wurden, befaßt sich dieses Kapitel nun mit einer konkreten Erweiterung für Microsoft Outlook.

7.1 Grundlagen

Microsoft Outlook unterstützt im Sinne der Abwärtskompatibilität sogenannte "Exchange Client Extensions", die ursprünglich für Microsoft Exchange entwickelt wurden. Hierbei handelt es sich um DLLs (Dynamic Link Libraries), die eine Funktion exportieren, die bei jedem Aufruf ein COM-Objekt erzeugt und einen Verweis darauf zurückgibt. Diese Objekte können diverse Schnittstellen implementieren, deren Funktionen ähnlich wie Event-Handler von Outlook beim Auftreten bestimmter Ereignisse aufgerufen werden, und dabei eigene Aktionen ausführen können.

So enthält die Schnittstelle `IExchExtMessageEvents` beispielsweise die Funktion `OnRead`, die jeweils beim Einlesen einer Nachricht vor dem Anzeigen ausgeführt wird. Sie kann dabei selber auf die Nachricht zugreifen und diese falls nötig verändern oder auch den ganzen Vorgang abbrechen.

Auf diesem Wege läßt sich das Verhalten von Microsoft Outlook in weiten Bereichen beeinflussen. Eine Alternative hierzu stellen die ebenfalls unterstützten COM-Add-Ins dar. Diese setzen jedoch auf einer etwas abstrakteren Ebene an und bieten nicht alle für die Entwicklung einer S/MIME-Erweiterung notwendigen Funktionen, insbesondere beim Zugriff auf S/MIME-Nachrichten und angehängte Dateien. Sie werden deshalb hier nicht weiter betrachtet.

Die zentrale Komponente des in den folgenden Abschnitten beschriebenen S/MIME-Plugins für Microsoft Outlook stellt die Klasse `OutlookSMIME` dar, die einige der verfügbaren Erweiterungsschnittstellen implementiert.

7.2 Erweiterung des Outlook Menüs und der Werkzeugleiste

Damit der Benutzer alle S/MIME-Funktionen bequem aus Outlook heraus verwenden kann, wird zur Menüleiste ein neues Menü mit dem Namen "FlexiSMIME" hinzugefügt.

Dieses ermöglicht im einzelnen

- die Anzeige des S/MIME-Status der ausgewählten oder geöffneten Nachricht, bzw. das Bearbeiten der S/MIME-Einstellungen beim Schreiben einer Nachricht,
- das Öffnen des Hauptdialogs der Zertifikatsverwaltung,
- das Bearbeiten der S/MIME-Grundeinstellungen, sowie
- die Anzeige von allgemeinen Informationen über Flexi S/MIME

Außerdem wird die Werkzeugleiste um ein Symbol erweitert, das mit einem Mausklick die Anzeige des S/MIME-Status bzw. der S/MIME-Einstellungen einer Nachricht ermöglicht.

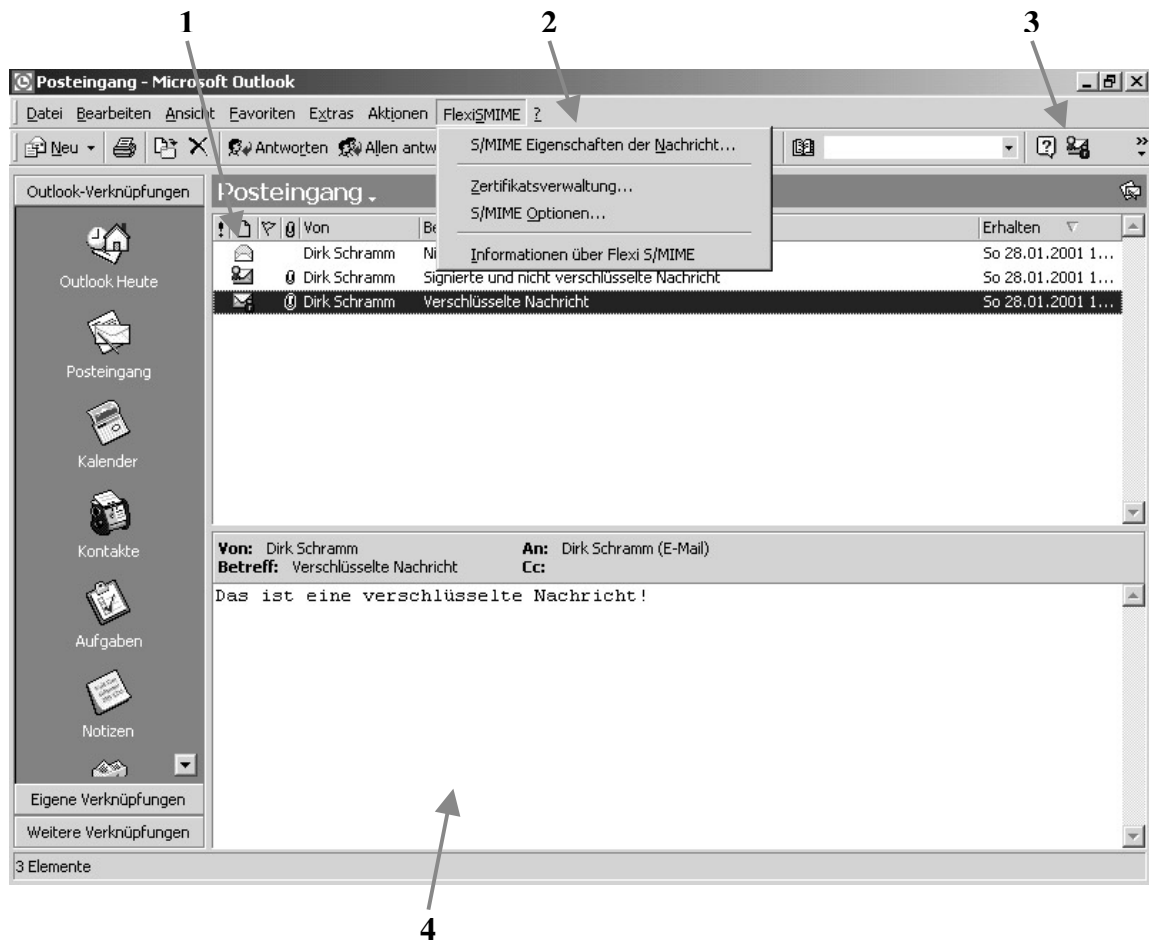
Um dies zu realisieren, implementiert die Klasse OutlookSMIME die Schnittstelle `IExchExtCommands`. Hierüber erfolgt beim Öffnen eines neuen Outlook-Fensters zunächst das Hinzufügen des Menüs und des Werkzeugleistensymbols. Die Funktion `DoCommand` führt dann die verschiedenen Kommandos aus, wenn sie vom Benutzer angewählt werden. Im Falle der S/MIME-Grundeinstellungen beispielsweise ruft sie die statische Methode `showOptionsDialogModal` der Klasse `SMimeUI` auf, die ihrerseits mit Hilfe des Java Native Interface Frameworks³¹ die Kontrolle an die Methode `showOptionsDialog` der Java-Klasse `smime.ui.SMimeUI`³² übergibt.

Zur einfachen Übersetzung in andere Sprachen werden sämtliche Texte über Ressourcen geladen. Im Visual Studio Projekt zum Erstellen der DLL für die Outlook-Erweiterung existieren derzeit zwei Ressourcen-Skripte für Englisch und Deutsch. Über entsprechende Projekt-Konfigurationen läßt sich damit für jede Sprache eine eigene Version der Datei OutlookSMIME.dll kompilieren. Bei der späteren Verwendung wird auch automatisch die Sprache der S/MIME-Dialoge entsprechend gewählt.

³¹ siehe Kapitel 6

³² siehe Abschnitt 5.2

Das folgende Bild zeigt die Benutzeroberfläche von Microsoft Outlook 2000 nach der Aktivierung der S/MIME-Erweiterung:



1. Die Icons zur Kennzeichnung von signierten oder verschlüsselten Nachrichten³³
2. Das neues Menü mit den S/MIME-Funktionen
3. Das Werkzeugleistensymbol zum Anzeigen der S/MIME-Eigenschaften einer Nachricht
4. Im Gegensatz zu Outlook ohne Erweiterung können hier auch verschlüsselte Nachrichten im Vorschauenfenster angezeigt werden.

³³ siehe Abschnitt 7.3.1

7.3 Verarbeitung der Nachrichten

Die Verarbeitung von Nachrichten innerhalb von Outlook basiert auf der MAPI (Messaging Application Programming Interface), die hierfür eine Standardschnittstelle bietet. Eine MAPI-Nachricht wird durch ein Objekt, das die Schnittstelle `IMessage` unterstützt, repräsentiert und ist dabei völlig unabhängig von einem beim Transport verwendeten Format wie MIME. Diese Abstraktion ist bei der Verwendung unterschiedlicher Nachrichtenformate zwar von Vorteil, bedeutet aber im Falle der S/MIME-Erweiterung, die ja auf MIME aufbaut, einen erheblichen Mehraufwand, weil eine entsprechende Konvertierung notwendig wird³⁴.

7.3.1 Anpassen der Nachrichtenklasse

Microsoft Outlook unterstützt bereits von sich aus S/MIME und verarbeitet Signaturen und Verschlüsselungen beim Anzeigen einer Nachricht normalerweise intern, bevor ein Plugin Einfluß darauf nehmen kann. Dieses Problem läßt sich umgehen, indem man dafür sorgt, daß Outlook S/MIME-Nachrichten nicht als solche erkennt.

Hier spielt die Nachrichtenklasse eine entscheidende Rolle. Dabei handelt es sich um ein Attribut, das in Form einer Zeichenkette den Typ jeder Nachricht kennzeichnet. Eine normale E-Mail hat beispielsweise die Klasse "IPM.Note", eine S/MIME-Nachricht dagegen "IPM.Note.SMIME" oder "IPM.Note.SMIME.MultipartSigned". Wird nun in den letzteren beiden Fällen die Nachrichtenklasse durch eine eigene ersetzt, die Outlook nicht kennt, erfolgt keine automatische interne Verarbeitung der S/MIME-Strukturen mehr, und die S/MIME-Erweiterung kann diese Aufgabe übernehmen.

Die Anpassung der Nachrichtenklasse wird mit Hilfe der Funktion `convertInboundMapiMessageClass`, die im Modul `MapiMimeConvert` implementiert ist, vorgenommen. Diese überprüft, ob es sich bei einem gegebenen `IMessage`-Objekt um eine S/MIME-Nachricht handelt und setzt in diesem Fall eine der folgenden Nachrichtenklassen:

Nachrichtenklasse	Bedeutung
IPM.Note.FlexiSMIME.Encrypted	die Nachricht ist verschlüsselt
IPM.Note.FlexiSMIME.Signed	die Nachricht ist signiert und nicht verschlüsselt
IPM.Note.FlexiSMIME	bei der Verarbeitung der S/MIME-Nachricht traten Fehler auf

Falls die Nachricht ihrerseits weitere Nachrichten als Anhang enthält, wird für diese der Vorgang rekursiv wiederholt.

³⁴ siehe Abschnitt 7.3.2

Die beschriebene Überprüfung und Anpassung der Nachrichtenklasse erfolgt in drei Fällen:

1. Eintreffen neuer Nachrichten

Die Klasse `OutlookSMIME` implementiert die Schnittstelle `IExchExtSessionEvents`, deren Funktion `OnDelivery` von Outlook beim Empfang einer neuen Nachricht aufgerufen wird und deren Bearbeitung ermöglicht.

2. Auswählen einer Nachricht

Wenn der Benutzer den Auswahlbalken in der Nachrichtenübersicht auf eine Nachricht setzt, deren Klasse mit "IPM.Note.SMIME" beginnt und die nicht als bisher unversandt markiert ist, erfolgt ebenfalls eine Umwandlung. Damit werden auch Nachrichten erfaßt, die bereits vor der Installation der S/MIME-Erweiterung vorhanden waren, oder vom Benutzer verschickt wurden. Um dies zu realisieren implementiert die Klasse `OutlookSMIME` zusätzlich die Schnittstelle `IExchExtUserEvents`, deren Funktion `OnSelectionChange` bei jedem Verschieben des Auswahlbalkens aufgerufen wird.

3. Lesen einer Nachricht

Wird eine Nachricht zur Anzeige im Vorschaufenster oder beim Öffnen eingelesen, erfolgt eine letzte Überprüfung der Klasse innerhalb der Funktion `OnRead` der Schnittstelle `IExchExtMessageEvents`. Falls eine Anpassung notwendig ist, wird diese durchgeführt und der Einlesevorgang dann abgebrochen, weil in dieser Situation die Verarbeitung der S/MIME-Strukturen von Outlook bereits durchgeführt wurde und nicht mehr rückgängig gemacht werden kann. Diese Variante kommt dann zum Tragen, wenn eine noch nicht versandte Nachricht im Postausgang geöffnet werden soll.

Bei der Anpassung der Nachrichtenklasse gibt es eine wichtige Ausnahme, die bereits angedeutet wurde. Beim Absenden einer vom Benutzer verfaßten S/MIME-Nachricht muß diese mit einer der von Outlook verwendeten S/MIME-Klassen gekennzeichnet werden, um eine korrekte Verarbeitung durch den Transport-Provider sicherzustellen. Dieses übernimmt die ebenfalls im Modul `MapiMimeConvert` implementierte Funktion `convertOutboundMapiMessageClass`.

Für die neuen Nachrichtenklassen `IPM.Note.FlexiSMIME.*` werden zusätzliche MAPI-Formulare registriert, die den jeweiligen Nachrichten ein passendes Icon zuordnen. So läßt sich bereits in der Übersicht erkennen, welche Nachricht verschlüsselt oder signiert ist. Bei der Anzeige kommt dann das original Outlook-Formular für die Klasse `IPM.Note` zum Einsatz.

7.3.2 Konvertierung einer Nachricht zwischen MAPI und JavaMail

Wie in der Einleitung dieses Kapitels bereits erläutert wurde, arbeitet Outlook intern mit MAPI-Nachrichten, auf die nicht direkt von JavaMail aus zugegriffen werden kann. Daher ist eine entsprechende Konvertierung von MAPI nach JavaMail und zurück notwendig, die vor und nach der S/MIME-Verarbeitung durchgeführt wird. Die dafür notwendigen Funktionen `MapiToMime` und `MimeToMapi` sind im Modul `MapiMimeConvert` implementiert.

Bei der Konvertierung kommen die Klassen `SimpleMimeMessage` und `SimpleMimeAttachment` (siehe Abschnitt 3.5) zum Einsatz, die eine ähnlich abstrakte Sichtweise auf eine Nachricht und deren Anhänge bieten, wie die MAPI-Schnittstellen `IMessage` und `IAttachment`. Hierbei werden die Eigenschaftswerte (Properties) der MAPI-Objekte so gut wie möglich auf die `get`- und `set`-Methoden der Java-Klassen abgebildet.

Für einen einfachen Zugriff auf die Eigenschaftswerte eines MAPI-Objekts, das die Schnittstelle `IMAPIProp` implementiert (dies ist bei Nachrichten und deren Anhängen der Fall), wurde die Klasse `MapiSPropValue` entwickelt. Diese übernimmt automatisch die Speicherverwaltung für die benötigte `SPropValue`-Struktur und ermöglicht zudem die Verwendung der Funktion `OpenProperty` und der Schnittstelle `IStream` zum Auslesen oder Verändern großer Eigenschaftswerte, wie dem Inhalt einer angehängten Datei.

In ähnlicher Weise vereinfacht die Klasse `MapiSRowSet` den Umgang mit `SRowSet`-Strukturen, die beim Zugriff auf die Liste der Empfänger einer MAPI-Nachricht, sowie die Liste der Anhänge benötigt werden, und verwaltet den Speicher eigenständig.

Einschränkungen:

Derzeit bestehen noch verschiedene Einschränkungen bei der Konvertierung des Nachrichtentextes. Rich-Text wird überhaupt nicht unterstützt, weil dies mit einem deutlichen Mehraufwand verbunden wäre und es sich hierbei nur um ein proprietäres Format von Microsoft handelt, das innerhalb des Internets kaum eine Bedeutung hat. Beim Umwandeln einer Rich-Text-Nachricht wird deshalb nur der ASCII-Text übernommen.

Ein weiteres Problem besteht beim Versenden von HTML-Nachrichten. Wenn Microsoft Outlook mit der Option "Unternehmen oder Arbeitsgruppe" installiert wird, bildet Rich-Text die Grundlage für HTML, und erst der Transport-Provider nimmt beim Erzeugen der MIME-Nachricht eine entsprechende Konvertierung vor. Dadurch ist kein Zugriff auf den HTML-Code über MAPI möglich. Die Umwandlung von Rich-Text nach HTML ist mit einem erheblichen Aufwand verbunden und wird daher im Moment beim Versenden von S/MIME-Nachrichten nicht unterstützt. In diesem Fall würde ebenfalls nur der ASCII-Text übernommen. Davon nicht betroffen ist der Empfang von S/MIME-Nachrichten mit HTML-Text. Hier ist eine Konvertierung problemlos möglich.

Bei einer Installation von Outlook mit der Option "Nur Internet-Mail" ergeben sich auch beim Versenden von S/MIME-Nachrichten mit HTML-Text keine Probleme, weil in diesem Fall der HTML-Code direkt in der MAPI-Nachricht gespeichert wird.

7.3.3 Verarbeitung einer Nachricht vor dem Anzeigen

Während der Anzeige einer Nachricht erstellt Outlook ein Objekt der Klasse `OutlookSMIME` mit dem Kontexttyp `EECONTEXT_READNOTEMESSAGE`. Dieses implementiert die Schnittstelle `IExchExtMessageEvents`, deren Funktion `OnRead` von Outlook nach dem Einlesen der MAPI-Nachricht aber vor dem Anzeigen aufgerufen wird.

Diese Funktion überprüft anhand der Nachrichtenklasse³⁵ ob es sich um eine S/MIME-Nachricht handelt. In diesem Fall wird daraus zunächst über die Funktion `MapiToMime`³⁶ ein `SimpleMimeMessage`-Objekt erstellt, dessen weitere Verarbeitung die Methode `processMessage` der Klasse `SMimeReadMessageContext`³⁷ übernimmt. Treten dabei keine Probleme auf, erfolgt anschließend eine Aktualisierung der MAPI-Nachricht auf der Grundlage des veränderten `SimpleMimeMessage`-Objekts über die Funktion `MimeToMapi`³⁸. Anderenfalls wird der gesamte Vorgang abgebrochen.

Das `OutlookSMIME`-Objekt hält nach der Verarbeitung weiter eine Referenz auf das verwendete `SMimeReadMessageContext`-Objekt, um dessen Methode `showMessageProperties` aufrufen zu können, sobald der Benutzer die Anzeige des S/MIME-Status der Nachricht über das Menü oder die Werkzeugleiste wählt.

Der beschriebene Vorgang läuft sowohl beim Öffnen einer Nachricht, also auch bei deren Anzeige im Vorschauenfenster ab. Um hier bei verschlüsselten Nachrichten die zweimalige Eingabe eines Paßwortes zu vermeiden und um zusätzlich die Verarbeitungsgeschwindigkeit zu erhöhen, überprüft die Funktion `OnRead` zunächst, ob bereits ein anderes `OutlookSMIME`-Objekt für die Anzeige der gleichen Nachricht existiert. In diesem Fall werden dessen `SMimeMessageContext`-Objekt, sowie die bereits verarbeitete Nachricht in Form des `SimpleMimeMessage`-Objekts einfach übernommen.

7.3.4 Verarbeitung einer Nachricht vor dem Absenden

Wenn der Benutzer beginnt, eine neue Nachricht zu schreiben, erstellt Outlook ein Objekt der Klasse `OutlookSMIME` mit dem Kontexttyp `EECONTEXT_WRITENOTEMESSAGE`. Dieses erzeugt intern ein `SMimeWriteMessageContext`-Objekt³⁹, um die Kontrolle an dessen Methode

³⁵ siehe Abschnitt 7.3.1

³⁶ siehe Abschnitt 7.3.2

³⁷ siehe Abschnitt 5.4.1

³⁸ siehe Abschnitt 7.3.2

³⁹ siehe Abschnitt 5.5.2

`showMessageProperties` übergeben zu können, sobald der Benutzer die Anzeige der S/MIME-Einstellungen über das Menü oder die Werkzeugleiste wählt.

Wie bereits angesprochen, implementiert die Klasse `OutlookSMIME` die Schnittstelle `IExchExtMessageEvents`. Vor dem Absenden der Nachricht ruft Outlook zunächst deren Funktion `OnSubmit` auf und nach dem vollständigen Speichern `OnWriteComplete`. Die letztere Funktion überprüft als erstes anhand eines innerhalb von `OnSubmit` gesetzten Flags, ob die Nachricht tatsächlich abgesendet und nicht nur gespeichert werden soll.

In diesem Fall und wenn der Benutzer das Signieren und/oder Verschlüsseln aktiviert hat, wird aus der MAPI-Nachricht zunächst über die Funktion `MapiToMime`⁴⁰ ein `SimpleMimeMessage`-Objekt erstellt, dessen weitere Verarbeitung die Methode `processMessage` des `SMimeWriteMessageContext`-Objekts übernimmt. Treten dabei keine Probleme auf, erfolgt anschließend eine Aktualisierung der MAPI-Nachricht auf der Grundlage des nun signierten und/oder verschlüsselten `SimpleMimeMessage`-Objekts über die Funktion `MimeToMapi`⁴¹. Anderenfalls wird der gesamte Vorgang abgebrochen, und der Benutzer kann das Bearbeiten der Nachricht fortsetzen.

7.4 Fehlerbehandlung

Fast alle im Rahmen dieser Arbeit entwickelten Funktionen melden Fehler, die sie nicht selbst behandeln können, durch das Auswerfen von Ausnahmen. Auf C++-Ebene kommen dabei folgenden Typen zum Einsatz:

Ausnahme	Bedeutung
<code>JThrowable</code>	Ausnahmen der JVM und der Java-Methoden
<code>JniException</code>	Fehler innerhalb des JNI Frameworks
<code>WinApiException</code>	Fehler beim Ausführen einer Funktion der Windows API
<code>MapiException</code>	Fehler beim Ausführen einer MAPI-Funktion
<code>_com_error</code>	Fehlermeldungen der Klassen <code>_bstr_t</code> und <code>_com_ptr_t</code>

Das Auswerfen der Ausnahmen erfolgt dabei größtenteils mit Hilfe von Makros wie `throwWinApiException`, die über die Präprozessorvariablen `__FILE__` und `__LINE__` automatisch den Namen der Quelldatei und die Zeilennummer zur Fehlermeldung hinzufügen und somit die Fehlersuche erheblich erleichtern. Zudem werden

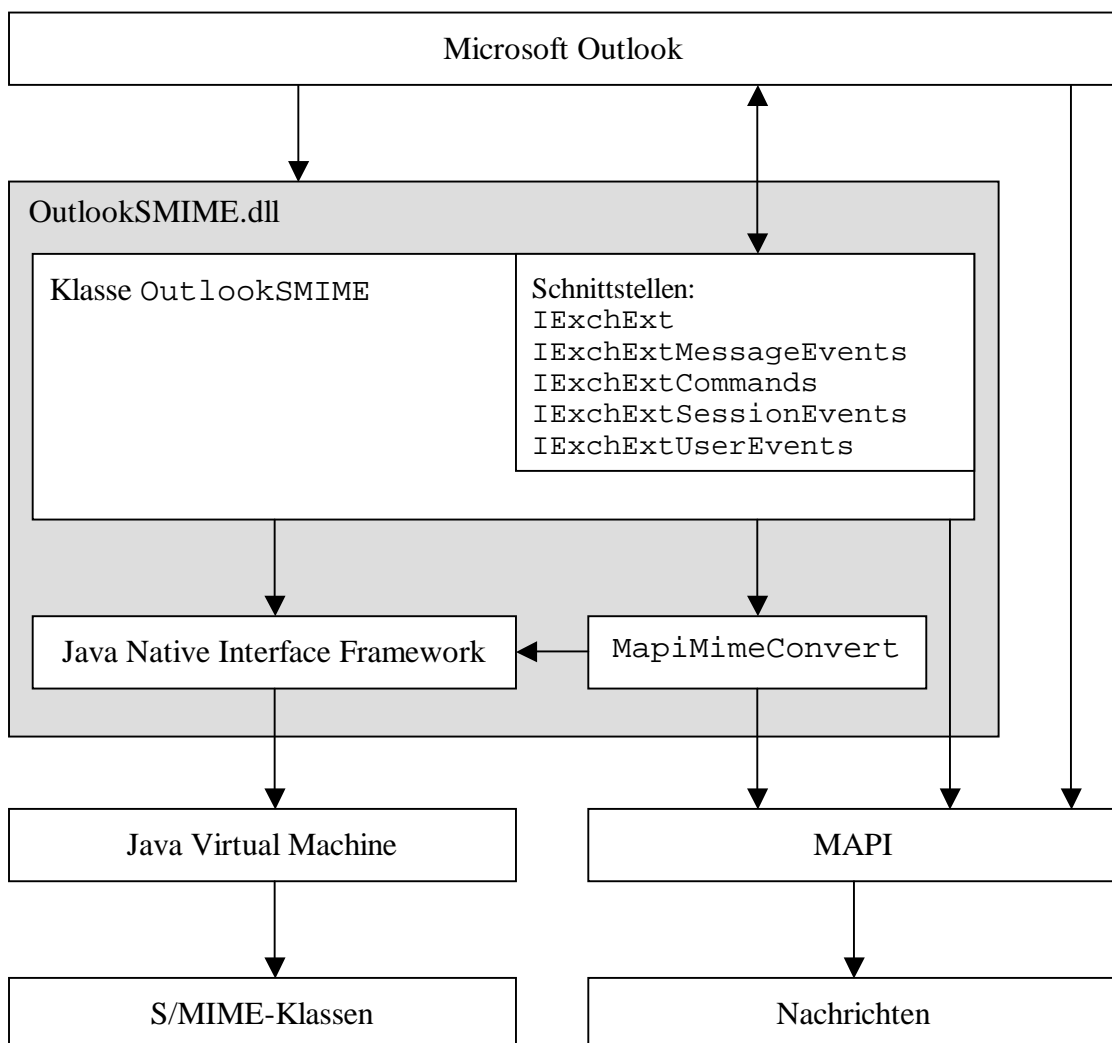
⁴⁰ siehe Abschnitt 7.3.2

⁴¹ siehe Abschnitt 7.3.2

Fehlercodes, die unter anderem bei MAPI und Windows API Verwendung finden, soweit wie möglich durch aussagekräftigere Texte ergänzt.

Das Abfangen unbehandelter Ausnahmen geschieht spätestens in den Schnittstellen-Funktionen der Klasse `OutlookSMIME`. In diesem Fall wird der Benutzer durch ein Dialogfenster über den aufgetretenen Fehler informiert und der jeweilige Vorgang abgebrochen. Bei einer Ausnahme des Typs `Throwable` oder einer Unterklasse kommt hier die Methode `showExceptionDialog` der Klasse `SMimeUI`⁴² zum Einsatz, die auch einen Zugriff auf den Stack-Trace ermöglicht.

Die folgende Abbildung zeigt die Struktur der beschriebenen S/MIME-Erweiterung für Microsoft Outlook im Überblick:



⁴² siehe Abschnitt 5.2

Ausführlichere Informationen zu den in diesem Kapitel vorgestellten Klassen und Schnittstellen finden sich in den Kommentaren der jeweiligen Quelltextdateien, sowie in [EXCHEXT].

7.5 Installation

Zur Aktivierung der entwickelten S/MIME-Erweiterung in Microsoft Outlook sind zwei Schritte erforderlich. Zum einem muß unter dem Registrierungsschlüssel "HKEY_LOCAL_MACHINE\Software\Microsoft\Exchange\Client\Extensions" ein neuer Eintrag erstellt werden, der den Namen der DLL, die die Erweiterung enthält, spezifiziert. Im Falle von Flexi S/MIME ist zusätzlich das Anlegen eines "Extension Configuration Files" im Directory "Addins" innerhalb des Installationsverzeichnisses von Microsoft Office erforderlich. Mit Hilfe dieser Konfigurationsdatei läßt sich unter anderem auf einfache Weise die Anzeige des S/MIME-Menüs und des Werkzeugleistensymbols auf die Ordner beschränken, die E-Mails enthalten.

Um eine benutzerfreundliche Installation zu gewährleisten, exportiert die Datei OutlookSMIME.dll zwei zusätzliche Funktionen `DllRegisterServer` und `DllUnregisterServer`, die die beschriebene Registrierung vornehmen bzw. beim Deinstallieren wieder rückgängig machen. Diese finden normalerweise bei DLLs Verwendung, die als COM-Server die von ihnen zur Verfügung gestellten Klassen selbst in der Windows-Registry eintragen. Deshalb bieten Installationsprogramme wie InstallShield oder der Windows Installer die Möglichkeit, diese Funktionen automatisch aufzurufen, wenn die Datei vorher als "selbst-registrierend" gekennzeichnet wurde.

Im Verzeichnis "Install" auf der beiliegenden CD befindet sich zu Demonstrationszwecken ein mit dem Visual Studio Installer erzeugtes Installationspaket, das alle für Flexi S/MIME benötigten Dateien enthält und diese eigenständig einrichtet.

Auch ein automatisches Deinstallieren über den Punkt "Software" innerhalb der Systemsteuerung ist möglich. Dabei wird jedoch die in Abschnitt 7.3.1 beschriebene Anpassung der Klassen von S/MIME-Nachrichten nicht rückgängig gemacht. Diese können dadurch ohne Flexi S/MIME nicht mehr normal angezeigt werden, und der Benutzer sieht statt des Inhalts nur noch eine angehängte Datei mit dem Namen "smime.p7m".

7.6 Kompatibilität

Ein wichtiger Punkt bei der Entwicklung einer Softwarekomponente, die von vielen Benutzern in unterschiedlichsten Umgebungen eingesetzt werden soll, ist die Kompatibilität zu anderen Hard- und Softwareprodukten. Dieser Abschnitt liefert hierzu einen Überblick.

7.6.1 Hardwareplattform

Die beschriebene S/MIME-Erweiterung für Microsoft Outlook wurde auf verschiedenen Rechnern mit Intel Pentium-Prozessor getestet. Da keinerlei spezielle Funktionen der Hardware genutzt werden, sollte ein Betrieb auf jedem Rechner mit einem Pentium-kompatiblen Prozessor möglich sein. Für ein effizientes Arbeiten sind jedoch eine Takt-rate von mindestens 300 MHz, sowie 64 MB Arbeitsspeicher zu empfehlen.

7.6.2 Microsoft Windows

Alle 32-Bit-Versionen von Microsoft Windows, also Windows 95, 98, 98 SE, ME, NT 4.0 und 2000 werden grundsätzlich unterstützt. Entsprechende Tests fanden jedoch bisher nur mit Windows NT 4.0 und Windows 2000 statt.

7.6.3 Java

Die Grundlage für das Ausführen des Java-Codes bildet die Java Virtual Machine von Sun zusammen mit den dazugehörenden Standardklassen, sowie verschiedenen zusätzlichen Paketen. Tests erfolgten mit den im Anschluß genannten Versionen, jedoch sollten auch neuere, sofern sie abwärtskompatibel sind, direkt unterstützt werden.

Komponente	Version
Sun JDK	1.2.2 und 1.3.0
JavaBeans Activation Framework	1.01
JavaMail	1.1.3 und 1.2

7.6.4 Microsoft Outlook

Die S/MIME-Erweiterung wurde mit Outlook 98 und Outlook 2000 erfolgreich getestet. Auch eine Verwendung mit Outlook 97 sollte möglich sein.

Es sind jedoch verschiedene Einschränkungen zu beachten. In Abschnitt 7.3.2 wurden bereits die bestehenden Probleme beim Versenden von Nachrichten mit HTML-Text angesprochen.

Zudem ist bei der Verwendung der aktuellen Version kein korrekter Versand von signierten Nachrichten im Klartext (Content-Type "multipart/signed"⁴³) möglich. Die Schwierigkeit hierbei besteht darin, daß der Transport-Provider beim Verarbeiten der Nachricht sämtliche Kopfzeilen umformatiert. Davon sind auch die des signierten Nachrichtenteils betroffen, wodurch der zuvor berechnete Hashwert nicht mehr stimmt. Dieses

⁴³ siehe Abschnitt 1.2.2

Problem läßt sich lösen, indem die entsprechenden Kopfzeilen vor dem Signieren bereits exakt so formatiert werden, wie es der Transport-Provider macht. Das genaue Format ist jedoch nicht offiziell dokumentiert und muß daher erst durch Ausprobieren, auch für sämtliche Spezialfälle, ermittelt werden. Dies ist mit einem hohen Aufwand verbunden und war deshalb in der zur Verfügung stehenden Zeit nicht realisierbar.

7.6.5 Andere Mailprogramme

Ein ganz entscheidendes Qualitätskriterium für eine S/MIME-Lösung stellt die Kompatibilität zu anderen Mailclients mit S/MIME-Unterstützung dar. Um diese zu überprüfen, wurden signierte und/oder verschlüsselte Nachrichten zwischen Outlook 2000 mit S/MIME-Erweiterung und den folgenden Programmen ausgetauscht:

- Outlook 2000 (ohne Erweiterungen)
- Outlook 2000 mit Flexi S/MIME
- Outlook Express 5.0
- Netscape Communicator 4.76

Dabei kamen RSA-Zertifikate mit einer Schlüssellänge von 1024 Bit zum Einsatz, sowie SHA-1 als Hashverfahren und DESede für die symmetrische Verschlüsselung. Im Rahmen dieser Tests traten keine Probleme auf.

7.7 Geschwindigkeit

Ein weitere wichtige Rolle im Hinblick auf die Praxistauglichkeit der entwickelten Softwarekomponenten spielt die Geschwindigkeit bei der Verarbeitung von S/MIME-Nachrichten. Die im folgenden genannten Werte wurden auf einem Computer mit 350 MHz Pentium-II CPU und 256 MB Hauptspeicher unter Windows 2000 mit Outlook 2000 und JDK 1.2.2 ermittelt.

Der Start von Outlook wird durch das Laden der Java Virtual Machine um etwa 3 Sekunden verzögert. Bei der ersten Anzeige eines S/MIME-Dialogs ergibt sich durch das Nachladen diverser Java-Klassen und deren Übersetzung durch den JIT-Compiler eine weitere Verzögerung von rund 2 Sekunden. Die Initialisierung des Zufallszahlengenerators bei der ersten Verschlüsselung benötigt ungefähr 12 Sekunden. Danach liegt die Verarbeitungszeit beim Öffnen oder Absenden einer signierten oder verschlüsselten Nachricht mit einer Größe von 10 KB bei unter einer Sekunde (abgesehen von der möglichen Paßworteingabe).

Beim Verarbeiten einer 1 MB großen Nachricht ergeben sich folgende Zeiten:

Vorgang	Dauer
Signieren mit SHA-1/RSA (1024 Bit Schlüssellänge)	3 Sekunden
Überprüfen der Signatur	2 Sekunden
Verschlüsseln mit DESede	4 Sekunden
Entschlüsseln	4 Sekunden

Es zeigt sich somit, daß die Verarbeitungsgeschwindigkeit dieser S/MIME-Implementierung selbst auf einem bereits 3 Jahre alten Rechner für den praktischen Einsatz völlig ausreicht.

8 AUSBLICK

Nach Abschluß dieser Arbeit stellt sich die Frage, wie eine mögliche Weiterentwicklung aussehen könnte.

Eine der wichtigsten anstehenden Aufgaben ist sicher die Realisierung einer leistungsfähigen Zertifikatsverwaltung, als eine der zentralen Komponenten für jede Anwendung, die mit Public-Key-Kryptographie arbeitet. Die in Kapitel 4 vorgestellte Schnittstelle bildet hierfür eine gute Grundlage.

Außerdem sollten die bei der S/MIME-Erweiterung für Outlook derzeit noch bestehenden Einschränkungen beim Versenden von signierten Nachrichten im Klartext⁴⁴ und beim Deinstallieren⁴⁵ beseitigt werden.

Auch eine Entwicklung zusätzlicher S/MIME-Plugins für andere Mailprogramme mit Hilfe der in den Kapiteln 3 und 5 beschriebenen Klassen, sowie optional dem in Kapitel 6 erläuterten JNI Framework erscheint sinnvoll, damit eine möglichst große Gruppe von Anwendern die Vorteile von Flexi S/MIME nutzen kann. In diesem Zusammenhang wäre zudem die Umsetzung des in Abschnitt 2.1 vorgestellten Konzepts einer universell einsetzbaren Lösung in Form eines Proxy-Servers denkbar.

⁴⁴ siehe Abschnitt 7.6.4

⁴⁵ siehe Abschnitt 7.5

LITERATUR

- [RFC822] RFC 822: Standard for the Format of ARPA Internet Text Messages
- [RFC1939] RFC 1939: Post Office Protocol - Version 3
- [RFC2045] RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies
- [RFC2046] RFC 2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types
- [RFC2047] RFC 2047: MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text
- [RFC2048] RFC 2048: Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures
- [RFC2049] RFC 2049: Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples
- [RFC2630] RFC 2630: Cryptographic Message Syntax
- [RFC2632] RFC 2632: S/MIME Version 3 Certificate Handling
- [RFC2633] RFC 2633: S/MIME Version 3 Message Specification
- [RFC2634] RFC 2634: Enhanced Security Services for S/MIME
- [JDK] Java 2 SDK Documentation, Version 1.2.2, <http://java.sun.com/j2se/>
- [JAF] JavaBeans Activation Framework Documentation, <http://java.sun.com/products/javabeans/glasgow/jaf.html>
- [JAVAMAIL] JavaMail API Documentation, <http://java.sun.com/products/javamail/>
- [JAVASEC] Java Security, Scott Oaks, O'Reilly 1998
- [MSDN] Microsoft Developer Network Library, July 2000
- [EXCHEXT] Extending the Microsoft Exchange Client, Platform SDK Documentation, Microsoft Developer Network Library, July 2000