

Technische Universität Darmstadt



Fachbereich Informatik

Fachgebiet Theoretische Informatik

Prof. Dr. Johannes Buchmann

Implementierung des Austausches kryptographischer Komponenten in FlexiPKI mittels Update Management Protocol

Diplomarbeit
von Igor Kalenderian

Betreuer: Michael Hartmann
Sönke Maseberg

Abgabetermin: 6. November 2001

Inhaltsverzeichnis.

<u>Inhaltsverzeichnis.</u>	2
<u>EHRENWÖRTLICHE ERKLÄRUNG</u>	4
<u>Vorwort.</u>	5
<u>1</u>	<u>Einführung</u> 6
<u>1.1</u>	<u>Idee einer Fail-Safe-PKI</u>6
<u>1.2</u>	<u>Funktionsweise einer Fail-Safe-PKI</u>8
<u>1.3</u>	<u>Austausch in Schadensfall</u>8
1.3.1	Phase 0: Schaden tritt ein.....9
1.3.2	Phase 1: Revokation.10
1.3.3	Phase 2: Update Service leitet schadensabhängige Maßnahmen ein.10
1.3.4	Phase 3: Client empfängt UMP.....10
1.3.5	Client führt UMP aus.....10
1.3.6	Folgende Phasen.....11
<u>2</u>	<u>Definition der Datenstruktur</u> 12
<u>2.1</u>	<u>Definition der UMP Datenstruktur</u>12
<u>2.2</u>	<u>Datenstruktur für Komponentenaustausch der Software- PSE</u>15
<u>2.3</u>	<u>Datenstruktur für Komponentenaustausch der ICC</u>16
<u>2.4</u>	<u>Datenstruktur der Zertifikationsanforderung</u>17
<u>2.5</u>	<u>Security Condition</u>17
<u>2.6</u>	<u>Registry</u>18
<u>2.7</u>	<u>UMP und PKCS#7</u>22
<u>3</u>	<u>Implementierung der Datenstruktur</u> 24
<u>3.1</u>	<u>Implementierungswerkzeuge</u>24
<u>3.2</u>	<u>Implementierung der ASN.1 Datenstruktur</u>25
<u>4</u>	<u>Implementierung des Trust Center Administrator Tools</u> 28

	3
<u>4.1</u>	<u>Die Klasse CAAdminDialog.....</u> 28
<u>4.2</u>	<u>Die Klasse Info.</u> 33
<u>4.3</u>	<u>Die Klasse CAStatus.</u> 33
<u>4.4</u>	<u>Die Klasse UMPSend.</u> 34
<u>4.5</u>	<u>Die Klasse MailTransport.</u> 34
<u>5</u>	<u>Austausch der kryptographischen Komponenten.....</u> 36
<u>5.1</u>	<u>Die Klasse Update.</u> 38
<u>5.2</u>	<u>Die Klasse RegistryImpl.</u> 38
<u>5.3</u>	<u>Die Klasse ClientUpdate.</u> 40
<u>5.4</u>	<u>Die Klasse ICCUpdate.</u> 41
<u>5.5</u>	<u>Die Klasse CardUpdateGemPlus.....</u> 42
<u>5.6</u>	<u>Die Klasse Verify.....</u> 43
<u>5.7</u>	<u>Andere Klassen.</u> 43
<u>5.7.1</u>	<u>Die Klasse Dialogs.</u> 43
<u>5.7.2</u>	<u>Die Klasse Stat.</u> 44
<u>5.7.3</u>	<u>Die Klasse UpdateInProgress.</u> 44
<u>5.7.4</u>	<u>Die Klasse Resolver.</u> 45
<u>5.7.5</u>	<u>Transportklassen.....</u> 45
<u>6</u>	<u>Zusätzliche Funktionalität der Zertifikatverwaltung.</u> 46
<u>7</u>	<u>Einbindung des UMP in CDC Plugin für Microsoft Outlook,..</u> 47
<u>7.1</u>	<u>Änderungen in Java Teil.</u> 47
<u>7.2</u>	<u>Änderungen in C++ Teil.</u> 47
<u>7.3</u>	<u>Probleme der Integrierung des Plugins und UMP.</u> 47
<u>7.4</u>	<u>Kompabilität.</u> 48
<u>7.4.1</u>	<u>Anforderungen an Hardware.....</u> 48
<u>7.4.2</u>	<u>Betriebssysteme.....</u> 48
<u>7.4.3</u>	<u>Java und Kartensoftware.....</u> 48
<u>7.4.4</u>	<u>Geschwindigkeitstest.</u> 48
<u>7.4.5</u>	<u>Installation.....</u> 49
<u>Schlusswort.</u>	50
<u>Literaturverzeichnis.</u>	51

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 6.11.2001

Vorwort.

Gegenstand dieser Diplomarbeit war die Entwicklung des Austausches von kryptographischen Komponenten, wie Signaturverfahren oder Zertifikaten, in der FlexiPKI, die am Lehrstuhl von Prof. Buchmann entwickelt wurde. Die Praxistauglichkeit des gewählten Ansatzes wurde durch Integrierung des Komponentenaustausches in früher entwickelte Plugin für Microsoft Outlook nachgewiesen.

Die Arbeit ist in 7 Kapitel gegliedert, die sich wie folgt zusammenfassen lassen:

In Kapitel 1 findet sich eine Einführung in das Thema.

In Kapitel 2 werden die Datenstrukturen definiert, die zur Durchführung von Komponentenaustausch relevant sind.

Kapitel 3 beschäftigt sich mit Implementierung der in Kapitel 2 definierten Strukturen.

In Kapitel 4 wird entwickeltes Trust Center Administrator Tool dargestellt.

Kapitel 5 beschäftigt sich mit Implementierung des Komponentenaustausches in der PKI.

In Kapitel 6 wird die zusätzliche Funktionalität der Zertifikatverwaltung beschrieben.

Kapitel 7 zeigt die Möglichkeit von Integration des Outlook Plugins mit Komponentenaustausch.

Ich danke an dieser Stelle Prof. Buchmann für die Möglichkeit, mich im Rahmen meiner Diplomarbeit mit diesem interessanten Thema zu beschäftigen, sowie meinen Betreuern Michael Hartmann und Sönke Maseberg für ihre Unterstützung. Des weiteren bin ich Markus Tak dankbar, dass er stets für mich Zeit hatte, wenn Probleme auftraten. Außerdem möchte ich allen Mitarbeitern des Fachgebiets Theoretische Informatik am Fachbereich Informatik der Technischen Universität Darmstadt und allen übrigen Personen, die zum Gelingen dieser Diplomarbeit beigetragen haben, danken.

1 Einführung.

Um sichere Nutzung des Internets zu garantieren, ist es unerlässlich, Maßnahmen gegen unbemerkte Manipulation und Lesen von Daten zu treffen. Insbesondere steigt die Rolle der sicheren Kommunikation mit zunehmender Nutzung des Internets für Abwicklung der vertraulichen Geschäftsprozessen zwischen der Unternehmen oder zwischen Kunden und Unternehmen.

Ein Weg, Internet sicher zu machen, ist die Benutzung von *Public-Key-Infrastrukturen (PKI)*. Internet-Nutzer, die Sicherheit wollen, müssen in einer PKI geheime und öffentliche Schlüssel besitzen und sie müssen auf die öffentlichen Schlüssel ihrer Kommunikationspartner zugreifen können. Eine PKI aufzubauen, ist technisch und organisatorisch sehr aufwendig.

Die Sicherheit der Public-Key-Kryptographie hängt mit der Lösung von „schwieriger“ mathematischer Problemen zusammen, wie z.B. der Bestimmung der Primfaktoren einer natürlichen Zahl oder der Berechnung von diskreten Logarithmen in bestimmten mathematischen Körpern. Allerdings ist es nicht bewiesen, dass diese mathematische Probleme wirklich „schwierig“ sind. Gerade in der letzten Zeit wurden große Fortschritte bei den Lösungen solcher Probleme erzielt. Als Beispiel kann man den Zahlkörpersieb Algorithmus von John Pollard nennen, der die Faktorisierungszeit deutlich verbesserte. Als Folge dieser Entwicklung kann z.B. der meist bekannte RSA Algorithmus mit Schlüssellänge von 512 Bit nicht mehr als sicher angesehen.

Eine mögliche Lösung dieser Probleme besteht darin, dass man sich bei der Implementierung einer PKI nicht auf ein bestimmtes Verfahren festlegen soll, sondern die Auswahl diesbezüglich flexible gestaltet. Das heißt, dass in Schadensfällen die Basisverfahren ausgetauscht werden können. Genau dieser Ansatz wurde an dem Lehrstuhl von Prof. Buchmann an der TUD Darmstadt gewählt und im Rahmen mehreren Projekte wurden verschiedene Problemstellungen erfolgreich gelöst. So werden neben den traditionellen Verfahren wie RSA oder DSA auch neuere Kryptographie wie elliptische Kurven oder Zahlkörperkryptographie unterstützt.

In dieser Diplomarbeit geht es um die Lösung eines Problems, und zwar um Austausch der kryptographischen Komponenten einer PKI, wenn der Schadensfall eingetreten ist. Dabei wurde ein Trust Center Service implementiert, das im Falle des Schadens einer Komponente (Signaturverfahren oder Zertifikat) dem Trust Center Administrator erlaubt, den Client darüber zu informieren und die neuen Komponenten zur Verfügung zu stellen. Dem Client wurde seinerseits ein Service zur Verfügung gestellt, das diese Informationen von Trust Center auswertet und automatisch den Austausch der schadhaften durch sichere Komponenten durchführt. Dieses Service wurde als zusätzliche Funktionalität des Plugins für Microsoft Outlook 2000, der am Lehrstuhl von Prof. Buchmann entwickelt wurde, implementiert.

1.1 Idee einer Fail-Safe-PKI.

Diese Diplomarbeit basiert auf dem Artikel „*Fail-Safe-Konzept für Public-Key-Infrastrukturen*“ von M. Hartmann und S. Maseberg [HaMa01]. Unten folgt eine kurze Zusammenfassung dieses Artikels, sowie anschließend eigentliche Aufgabenstellung dieser Arbeit.

Zunächst stellen die Autoren fest, dass zwei Arten des Schadens eintreten könnten:

- kompromittieren einer kryptographischen Komponente, wie Signatur-Algorithmus, Verschlüsselungs-Algorithmus, Hashfunktion oder Formatierung. Dieser Schaden betrifft alle Schlüssel zu diesem Verfahren.
- kompromittieren von einsatzspezifischen Komponenten, wie Systemparametern oder einer Klasse von Schlüsseln zu einem Algorithmus. Die Klasse ist eine Teilmenge aller Schlüssel und kann insbesondere ein einzelner Schlüssel sein.

Diese Schäden betreffen Signaturen und Kryptogrammen, die mit diesen Verfahren erstellt wurde. Außerdem wären die Zertifikate betroffen, die in einer PKI die Verbindung von öffentlichen Schlüsseln und Teilnehmern herstellen.

Die durchgeführte Analyse zeigt, dass der Ausfall einer funktionsfähigen PKI folgende unbefriedigend gelösten Probleme zur Folge hat:

- Die digitalen Signaturen verlieren ihre Beweiskraft.
- Die Revokationsmechanismen sind nicht genug gesichert.
- Kurzfristige Verschlüsselung verliert ihre Vertraulichkeit.
- Verfügbarkeit einer PKI ist nicht mehr gewährleistet.

Als Lösung wurde ein Fail-Safe-Konzept für Public-Key-Infrastrukturen vorgeschlagen, welches eine optimale Konfiguration und Funktionsweise einer PKI beschreibt, die im Schadensfall die weitere Funktionsfähigkeit und den sicheren Austausch kompromittierter Komponenten ermöglicht. Dabei wurde Fail-Safe wie ein Prozess definiert, der bei einem Schadensfall (Fail) in Aktion tritt und versucht, die Funktionsfähigkeit des Systems zu erhalten und den Schaden zu reparieren, mit anderen Worten das System sicher (Safe) weiterzuführen.

Die Lösungsidee besteht darin, dass in einer PKI mehrere voneinander möglichst unabhängige Komponenten (kryptographische Algorithmen und Schlüssel) vorhanden sind. Dadurch kann eine PKI weiterhin sicher funktionieren, sollte eine Komponente nicht mehr sicher sein. Eine andere wichtige Eigenschaft ist der dynamische Austausch einer schadhafte Komponente durch eine sichere zu gewährleisten.

In diesem System werden z.B. die Signaturen ihre Gültigkeit nicht verlieren, wenn die Dokumente mehrfach (genannt auch multiple) signiert werden. Wenn ein Signaturverfahren nicht mehr als sicher gilt, kann man trotzdem mit anderen Verfahren die Signatur sicher verifizieren. Auch die Revokationsmechanismen werden von möglichen Fehlern deutlich unabhängiger. Durch iterative Verschlüsselung bleibt die Vertraulichkeit der Daten erhalten und die Verfügbarkeit des System wird durch Ausfall eines Systemteils wird weniger beeinflusst.

Der Funktionsfähigkeit dieses Systems liegt die Annahme zugrunde, dass es höchst Unwahrscheinlich ist, dass alle (oder gleichzeitig zwei) kryptographischen Verfahren, die auf voneinander unabhängigen mathematischen Problemen beruhen, gleichzeitig unsicher werden. Das heißt, dass nicht für alle mathematische Probleme auf einmal ein effizienter Algorithmus entdeckt werden kann oder die Rechnerleistung so stark ansteigt, dass alle diese Probleme effizient gelöst werden können.

Wir bezeichnen dieses Konzept weiterhin als *Fail-Safe-PKI*.

Die folgende Modell wird der Fail-Safe-PKI zugrundegelegt, welcher wie folgt verallgemeinert werden kann:

- In System gibt es nur zwei Stufen: Zertifizierungsinstanz und von ihr zertifizierten Zertifikatinhabern.

- Die multiplen kryptographischen Verfahren beschränken sich auf zwei.
- Den Schlüsseln in Zertifikaten wird keine Funktion zugeordnet.

Die Fail-Safe-PKI enthält zwei unabhängige Komponenten und deswegen kann als zwei quasi unabhängige PKI betrachtet werden. Diese werden als PKI^A und PKI^B bezeichnet. Jede PKI hat seine eigene Schlüssel, Zertifikate, Signatur- und Verschlüsselungsverfahren.

Die wichtigen Anwendungsprotokolle, wie 'Secure/Multipurpose Internet Mail Extensions (S/MIME)' für den sicheren Daten-Transfer via e-Mail oder das 'Transport Layer Security (TLS)' / 'Secure Socket Layer (SSL)' Protokoll für Authentisierungen, werden von der Fail-Safe-PKI unterstützt. Die in PKIX diskutierte 'Cryptographic Message Syntax (CMS)' findet sich - leicht abgewandelt - in dem von RSA Security Inc. propagierten 'Standard Public Key Cryptographic Standard #7 (PKCS#7)' wieder, der sich zunehmend durchsetzt und deshalb in der Fail-Safe-PKI genutzt wird. [SMIME], [TLS], [CMS], [PKCS7] PKCS#7 bietet bereits die Möglichkeit multipler digitaler Signaturen und iterativer Verschlüsselungen [HaMa01].

Da einige Elemente solcher Fail-Safe-PKI noch nicht existieren, müssen sie noch definiert und implementiert werden. Diese Komponenten sind unter anderen:

- Ein Update Service muss geschaffen werden, das den Austausch einer PKI Komponenten vorbereitet und anstößt. Der Austausch betrifft kryptographische Verfahren und Zertifikate.
- Für die Durchführung des Austausches ist es notwendig, ein spezielles Protokoll, das Update Management Protokoll (UMP) genannt wird, zu definieren. Dieser Datentyp wird über einen Object Identifier (OID), der unterhalb der OID der Arbeitsgruppe von Prof. Buchmann angeordnet ist, weltweit eindeutig zuzuordnen sein. UMP enthält Informationen über den Schaden und einen Link zum Download des Codes. Der Vorteil liegt darin, dass das Trust-Center so die Verteilung der Datenmenge zeitlich strecken und die Kapazitäten von Speichern und Netzen optimal nutzen kann [HaMa01]. Die Austauschinformation in UMP kann sowohl Software-PSE (Personal Security Environment - Persönliche Sicherheitsumgebung¹) als auch Chipkarte (auch Smart Karte oder ICC - Integrated Circuit Card genannt) betreffen.

1.2 Funktionsweise einer Fail-Safe-PKI.

Eine Fail-Safe-PKI kann in zwei Modi funktionieren:

Im einfachen Funktionsmodus wird nur eine PKI benutzt. Die Daten kann man signieren, verschlüsseln, Zertifikate verwalten, abfragen etc. Die zweite PKI wird nicht benutzt, ist aber für den Schadensfall an Verfahren oder Zertifikaten der erster PKI einsatzbereit.

Im erweiterten Funktionsmodus werden beide PKI für die Erstellung von multiplen Signaturen oder iterativen Verschlüsselung genutzt.

1.3 Austausch in Schadensfall.

Nehmen wir an, dass ein Schadensfall tritt ein und dem Trust Center wird davon bekannt. Dann leitet das Update Service des Trust Centers die notwendigen Maßnahmen ein, in denen die unsicheren

¹ Unter „persönlicher sicherer Umgebung“ versteht man ein Gebiet, wo privaten Schlüssel oder andere sensible Information sicher abgelegt werden kann. Dies kann entweder in einem Computerspeicherbereich organisiert werden oder auf einer Chipkarte.

Komponenten einer PKI durch neue unkompromittierte ersetzt werden sollen. Die Abb. 1 gibt grobe Überblick über Ablauf des Austausches, der im folgenden ausführlicher beschrieben wird [HaMa01].

1.3.1 Phase 0: Schaden tritt ein.

Die Autoren haben drei Typen von Fehlern in einer PKI definiert:

Schaden S1: Signatur-Verfahren sign^A oder Verschlüsselungs-Verfahren encrypt^A kompromittiert oder fehlerhaft implementiert.

Schaden S2: CA-Schlüssel² prK_{CA}^A als Sicherheitsanker kompromittiert.

Schaden S3: CH-Schlüssel³ prK_{CH}^A kompromittiert.

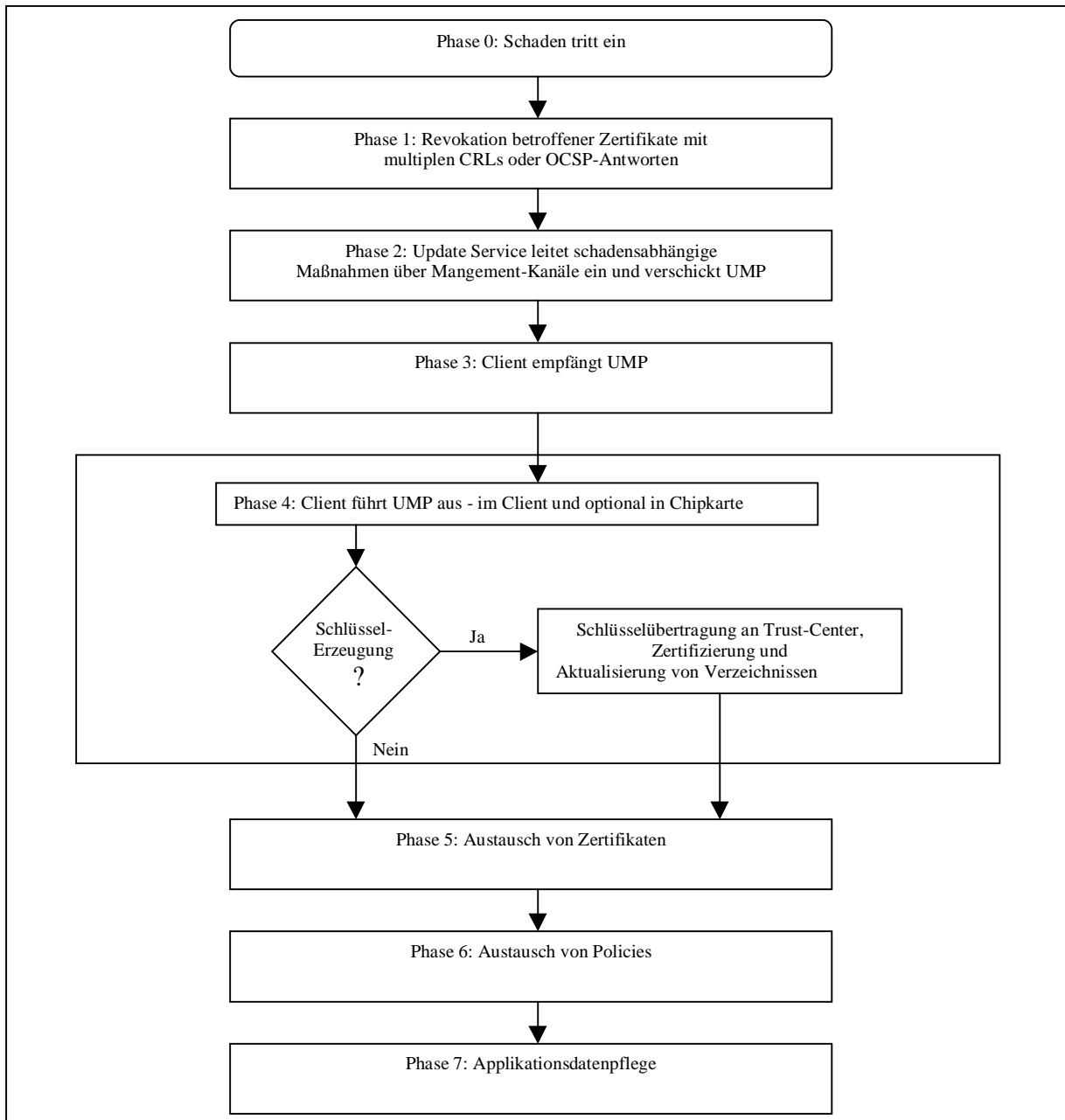


Abb. 1: Grob skizzierter Ablauf im Schadensfall

² CA Certificate Authority (Zertifizierungsinstanz), ein Bestandteil von Trust Center, das Zertifikaten herausgibt und revoziert.

³ CH Certificate Holder (Zertifikatinhaber) – bekommt Zertifikaten von CA, erstellt Signaturen und Kryptogrammen.

1.3.2 Phase 1: Revokation.

Nachdem das Trust Center über Schaden erfährt (als Quellen kommen Bundesamt für Sicherheit in Informationstechnik (BSI), CH, Presse etc. in Frage), wird es die betroffenen Zertifikate von PKI^A revozieren und Sperrlisten aktualisieren.

Die PKI^B funktioniert währenddessen ohne Einschränkungen weiter [HaMa01].

1.3.3 Phase 2: Update Service leitet schadensabhängige Maßnahmen ein.

Zu jedem Fehlertyp, der in Phase 0 eingetreten ist, ist eine entsprechende Maßname zu definieren:

- Maßname M1: Austausch von kryptographischen Verfahren.
- Maßname M2: Austausch von CA Schlüssel oder CA Zertifikaten.
- Maßname M3: Austausch von CH Schlüssel.

Um diese Maßnahmen durchzuführen, definiert das Trust Center ein Update Management Protokoll (UMP) und verschickt es an alle betroffene Clients. Das UMP muss dabei multiple signiert werden: Je einmal mit unsicherem und mit sicherem Signaturverfahren.

1.3.4 Phase 3: Client empfängt UMP.

Das UMP kommt in einen Mail Client eingebettet in PKCS#7 Objekt⁴. Der Client kann den Komponentenaustausch zu dem Zeitpunkt veranlassen, wenn er das für richtig hält. So kann es z.B. vernünftig sein, Austausch dann auszuführen, wenn die Auslastung des Netzes minimal ist, da die neuen Komponenten über Internet geladen werden müssen. Auch ist die Verschiebung denkbar, wenn der Client noch die alten Komponenten noch benutzen will oder muss, z.B. um verschlüsselte Texte mit einem unsicheren Verfahren zu entschlüsseln.

1.3.5 Client führt UMP aus.

Der Client hat ein Kommando, das UMP Ausführung startet. Wenn er dies macht, wird durch eine eindeutige Kennzeichnung erkannt, dass es sich um ein UMP handelt. Hierdurch wird der Austausch veranlasst. Der eigentliche Komponentenaustausch wird nur dann stattfinden, wenn folgende Bedingungen erfüllt sind:

- Der Client kann UMP sicher authentifizieren. Dafür wird UMP multiple mit mindestens zwei unabhängigen Signaturverfahren signiert. Der Client benutzt speziell dafür entwickelte Registry⁵ um sicherzustellen, dass die zwei Signaturverfahren voneinander unabhängig sind.
- Der Client stellt fest, dass ankommendes UMP nicht aus der Vergangenheit stammt und wiederholt von einem Angreifer geschickt wird⁶. Dafür wird in UMP ein Zähler eingebaut, der

⁴ Siehe Abschnitt 2.7

⁵ Siehe Abschnitt 2.6

⁶ Dieser Angriff ist als Replay Attacke bekannt.

mit lokal gespeichertem Zähler beim Client verglichen wird. Nur wenn der aktuelle UMP Zähler größer als der lokale Zähler ist, wird der Austausch fortgesetzt.

Der Austausch kann entweder Software PSE oder Smart Karte oder beides betreffen. Es werden:

- Verfahren entfernt/ neuinstalliert.
- Trust Center Zertifikaten entfernt/installiert.
- Benutzerzertifikaten entfernt/installiert.
- Die Schlüssel zu neuen Verfahren generiert.
- Kartenkomponenten entfernt/installiert.

Alle oben aufgeführten Schritte sind optional und können unabhängig voneinander und in beliebiger Kombination ausgeführt werden.

Das Ergebnis des Austausches kann eine Anfrage an Trust Center sein, ein neues Benutzerzertifikat zu dem neugenerierten Schlüssel zu erstellen, falls die Generierung des Schlüssels von Trust Center veranlasst wurde. Dafür muss ein spezielles Protokoll entwickelt werden⁷.

1.3.6 *Folgende Phasen.*

Die folgenden Phasen, die in Abb. 1 dargestellt sind, werden hier nur kurz vorgestellt, da sie nicht der Gegenstand dieser Diplomarbeit sind.

- In Phase 5 werden die neuen Zertifikaten an Clients verteilt.
- In Phase 6 werden die neuen Policies verteilt, falls dies nötig ist. Die Verfahren dazu existieren bereits.
- Die Phase 7 beschäftigt sich mit der Datenpflege. So müssen z.B. die Daten neuverschlüsselt und resigniert werden.

⁷ Siehe Abschnitt 2.1

2 Definition der Datenstruktur.

Als erster Schritt zur Implementierung des Komponentenaustausches ist es notwendig, noch nicht existierende Datenstrukturen von Update Management Protocol (UMP) festzulegen, da es sich um ein ganz neues kryptographisches Protokoll handelt.

Wie es bei der Definition von kryptographischen Protokollen oft geschieht, wird zur Definition der Datenstruktur von UMP die plattformunabhängige ASN.1 Datenbeschreibung herangezogen. Die Benutzung der ASN.1 hat folgende Vorteile:

- Normierung durch ITU.
- Universelle Definitionssyntax für Datenstrukturen.
- Plattformenunabhängigkeit.
- Encodings definieren binäre Darstellung (z.B. DER).

2.1 Definition der UMP Datenstruktur.

Es sind folgenden ASN.1 Datenstrukturen spezifiziert:

UMP hat eine weltweiten eindeutige Bezeichnung, und zwar:

```
CDC-UMP OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) dod(6) internet(1) private(4)
enterprises (1) TUD(8301) cdc(3) management(3) ump(2) }
```

In folgenden werden ASN.1 Datenstrukturen definiert, die UMP repräsentiert.

```
UpdateComponent ::= SEQUENCE {
    version      INTEGER      DEFAULT 1,
    freeText     EXPLICIT     UTF8String      OPTIONAL,
    identity     EXPLICIT     Identity        OPTIONAL,
    clientRelevant EXPLICIT   ClientRelevant  OPTIONAL,
    iccRelevant  EXPLICIT     Link           OPTIONAL,
    protection   SEQUENCE OF Protection      -- zwei gefordert
}
```

Das `version` spezifiziert UpdateComponent Protokollversion (z.Z. 1), `freeText` spezifiziert die Informationen über einen Austausch, die einem Client vor dem Austauschbeginn präsentiert wird, `clientRelevant` enthält Informationen über den Austausch der Software PSE, `iccRelevant` gibt an, wo die neue Kartenkomponente zu finden ist. Das Feld `protection` enthält Signaturen über Inhalt des UpdateComponent.

Die Identity wird durch folgende ASN.1 Datenstruktur spezifiziert:

```
Identity ::= Sequence {
    subjectName [0] EXPLICIT SubjectName OPTIONAL -- Identität des Clients aus
Zertifikat.
    iccsn       [1] EXPLICIT BIT STRING OPTIONAL -- ICCSN von Chipkarte.
}
```

Dabei spezifiziert `subjectName` in X501 Format (z.B. CN=kalender, OU=CDC, O=TUD) die Identität des Besitzers von Software PSE und `iccsn` spezifiziert die Identität der Karte (z.Z. als ATR Antwort von der Karte).

Die `ClientRelevant` wird durch folgende ASN.1 Datenstruktur repräsentiert:

```
ClientRelevant ::= SEQUENCE {
    messageClientID  [0]  INTEGER      OPTIONAL,
    deletions        [1]  Deletions    OPTIONAL,
    installations    [2]  Installations OPTIONAL
}
```

Das `messageClientID` repräsentiert den schon oben beschriebenen UMP Zähler, der zur Vermeidung der replay Attacken benutzt werden kann, `deletion` spezifiziert die Komponenten von Software PSE, die entfernt werden sollen und `installations` gibt an, welche neuen Komponenten in Software PSE installiert werden sollen. Alle Felder sind nicht Pflichtfelder (als `OPTIONAL` bezeichnet) und können in beliebiger Kombination auftreten.

Die `Deletions` Datenstruktur wird folgendermaßen definiert:

```
Deletions ::= SEQUENCE {
    algorithm  [0]          AlgorithmIdentifier      OPTIONAL,
    trustAnchor [1]        SEQUENCE OF KeyOrCertToDelete OPTIONAL,
    chKey      [2]        SEQUENCE OF KeyOrCertToDelete OPTIONAL
}
```

In `algorithm` ist der Algorithmus (seinen Object Identifier) angegeben, der aus PSE gelöscht werden soll.

Soweit möglich, sind die kryptographischen Primitiven anzugeben. Der Client findet dann über Registry alle zusammengesetzten Verfahren und löscht sie⁸. Im Falle einer Java Plattform, werden alle JCA Provider entfernt, die diese Algorithmen implementieren.

In `trustAnchor` bzw. in `chKey` sind die Trust Center Zertifikate (Schlüssel) bzw. Benutzerzertifikaten spezifiziert, die aus dem Software - PSE (Zertifikatverwaltung) gelöscht werden sollen.

Die `KeyOrCertToDelete` wird durch folgende ASN.1 Struktur definiert:

```
KeyOrCertToDelete ::= SEQUENCE {
    algorithm  AlgorithmIdentifier,
    usage      KeyUsage            OPTIONAL
}
```

In `algorithm` ist das Signaturverfahren definiert, zu dem ein Zertifikat aus der Zertifikatverwaltung entfernt werden soll. Zusätzlich kann in `usage` noch andere Informationen über Zertifikat spezifiziert werden.

⁸ Siehe Abschnitt 2.6

Die KeyUsage definiert die mögliche Benutzung von einem bestimmten Zertifikat und wird in "Internet X.509 Public Key Infrastructure Certificate and CRL Profile"[X509] beschrieben. Die folgende ASN.1 Datenstruktur repräsentiert KeyUsage:

```
KeyUsage ::= BIT STRING {
    digitalSignature (0), -- security mechanisms other than nonRepudiat.
    nonRepudiation (1), -- digital signature of a dokument
    keyEncipherment (2), -- public key is used for key transport
    dataEncipherment (3), -- for enciphering user data
    keyAgreement (4), -- Diffie-Hellman key management
    keyCertSign (5), -- for certificate signing
    cRLSign (6), -- for revocation information signing
    encipherOnly (7), -- gehört zu keyEncipherment: for enciphering
    decipherOnly (8) -- gehört zu keyEncipherment: for deciphering
}
```

Die Installations wird durch folgende ASN.1 Datenstruktur definiert:

```
Installations ::= SEQUENCE {
    provider [0] Link OPTIONAL,
    registry [1] Registry OPTIONAL,
    trustAnchor [2] SEQUENCE OF TrustAnchorToInstall OPTIONAL,
    chKey [3] SEQUENCE OF CHKeyToInstall OPTIONAL,
}
```

In provider wird ein Link angegeben, welche die Internet Adresse für die neue Komponente enthält, in registry ist die neue Registry enthalten, falls eine neue benötigt wird⁹, trustAnchor und chKey enthalten entsprechend das neue Trust Center Zertifikat (oder den neuen öffentlichen Schlüssel) und die Anforderung, ein neues Schlüsselpaar zu generieren. Alle Felder sind als OPTIONAL bezeichnet und sind beliebig miteinander kombinierbar.

Link wird durch folgende ASN.1 Datenstruktur definiert:

```
Link ::= UTF8String
```

Dabei liegt es in der Verantwortung des Trust Center Administrators die syntaktisch (und natürlich semantisch) richtige URL's zu spezifizieren. So wird es von Anwendung verlangt, dass die URL mit Protokollnamen anfängt und mit anschließenden Doppelpunkt und Doppelpunkt folgt, wie z.B.:

```
"http:" "://" host oder
"ftp:" "://" host | "ftp:" Username ":" passwort "@host
```

Der TrustAnchorToInstall wird durch folgende ASN.1 Datenstruktur beschrieben:

```
TrustAnchorToInstall ::= SEQUENCE {
    algorithm AlgorithmIdentifier,
    usage KeyUsage, OPTIONAL
    trustAnchor TrustAnchor
}
```

⁹ siehe Abschnitt 2.6

Der neue trustAnchor soll zu dem Verfahren, der in algorithm angegeben wurde, in Zertifikatverwaltung des Clients installiert werden. Optional kann dabei usage (siehe oben) spezifiziert werden.

Unter TrustAnchor ist folgende ASN.1 Datenstruktur zu finden:

```
TrustAnchor ::= CHOICE {  
    certificate Certificate,  
    key          BIT STRING  
}
```

Das heißt, das entweder ein Zertifikat oder ein öffentlicher Schlüssel als Trust Center Anker dienen kann. Die Definition von Certificate ist in [X509] zu finden.

Die CHKeyToInstall aus Installations ist definiert durch folgende ASN.1 Struktur:

```
CHKeyToInstall ::= SEQUENCE {  
    algorithm AlgorithmIdentifier,  
    usage      KeyUsage OPTIONAL  
    keylength  INTEGER  
}
```

Der Client soll einen neuen Schlüssel(-paar) zu dem Verfahren, das in algorithm spezifiziert ist, generieren. Der Schlüssel soll die geforderte in keylength Länge haben. Darüber hinaus kann in usage die spezifische Benutzung des Schlüssels angegeben werden (siehe oben).

Es ist gefordert, dass der Inhalt der UMP authentisch, d.h. signiert sein soll. Aus oben beschriebenen Überlegungen¹⁰, ist es auch gefordert, dass der Inhalt mehrfach (multiple) signiert werden muss.

Die folgende ASN.1 Struktur wird für den Inhaltsschutz des UMP benutzt:

```
Protection ::= SEQUENCE {  
    signatureAlg AlgorithmIdentifier,  
    signature     BIT STRING,  
    certificates  Certificate OPTIONAL  
}
```

Das Feld signatureAlg spezifiziert Signaturalgorithmus, der für Signieren eingesetzt wird und das Feld signature enthält die berechnete Signatur selbst. Optional kann ein Zertifikat angegeben werden, das dem Client die Signaturverifikation erleichtern kann.

2.2 Datenstruktur für Komponentenaustausch der Software-PSE.

Wie oben schon beschrieben wurde, holt sich der Client eine neue kryptographische Komponente aus dem Internet, indem er die in UMP spezifizierte URL folgt.

¹⁰ siehe Abschnitt 1

In Falle einer Java Plattform und eines Austausches der Software PSE Komponenten wird der Client die folgende Datenstruktur vorfinden, die über das Internet geladen und lokal gespeichert wird:

```
UpdateComponentCode ::= SEQUENCE {
    provider SEQUENCE OF Provider,
    protection SEQUENCE OF Protection
}
```

Das Feld provider enthält JCA Provider, die die alten Provider ersetzen sollen, und das Feld protection enthält die Schutzsignaturen (mindestens zwei voneinander unabhängige Signaturverfahren gefordert) für diese Struktur.

Der Provider wird durch folgende ASN.1 Struktur definiert:

```
Provider ::= SEQUENCE {
    providerName UTF8String,
    code BIT STRING
}
```

Das Feld providerName enthält den vollständigen Namen des Providers, z.B. cdc.standard.CDCStandardProvider oder cdc.ec.CDCECProvider. Das Feld code enthält den gesamten binären Providercode.

2.3 Datenstruktur für Komponentenaustausch der ICC.

In Falle des Kartenkomponentenaustausches wird die folgende ASN.1 Struktur aus dem Internet geladen:

```
CardUpdateComponent ::= SEQUENCE {
    packageAIDToDel [0] BitString OPTIONAL
    appletAIDToDel [1] BitString OPTIONAL
    name UTF8String,
    packageName UTF8String,
    appletName UTF8String,
    packageAID BitString
    appletAID BitString
    content BitString
    protection SEQUENCE OF Protection
}
```

Diese enthält alle notwendigen Informationen über die Kartenkomponenten, die gelöscht werden sollen und die Informationen über neuinstallierte Komponente. Das Feld content enthält den gesamten binären Code der neuen Komponente. Die CardUpdateComponent Struktur ist Proprietär und definiert für ein speziellen Kartentyp. In diesem Fall handelt es sich um GemXPRESSO 211 IS Javacard von der Fa. GemPlus. Unter Umständen, kann diese Struktur für die anderen Kartentypen auch anders aussehen.

2.4 Datenstruktur der Zertifikationsanforderung.

Wenn das Trust Center in UMP die Generierung von neuem Schlüssel anfordert, soll der Client den neuen öffentlichen Schlüssel an Trust Center in authentischer Weise zurücksenden. Nach der Überprüfung des neuen Schlüssels hat der Trust Center ein neues Benutzerzertifikat auszustellen und durch existierende Wege, die hier nicht weiter festgelegt werden sollen, an den Client weiterzuleiten. Für die Zertifikatsanforderung ist die folgende ASN.1 Datenstruktur vorgesehen:

```
UpdateComponentResponse ::= SEQUENCE {
    version                Integer DEFAULT 1
    subjectName            Name
    messageClientID       [0] EXPLICIT Integer OPTIONAL
    subjectPKInfo         [1] EXPLICIT SubjectPublicKeyInfo OPTIONAL
    protection            SEQUENCE OF Protection
}
```

Das Feld `subjectName` enthält den Namen des Client in X501 Format und wird von Trust Center für die Zuordnung zwischen entsprechendem UMP und `UpdateComponentResponse` benutzt. Auch der `messageClientID` kann dabei helfen. Das `SubjectPublicKeyInfo` ist ausführlich in RFC 2459 beschrieben [X509]. Auch hier wird der Inhalt durch multiple Signaturen (mind. 2) in `protection` geschützt. Ein Signaturverfahren basiert dabei auf neugeneriertem Schlüssel (z.B. MD5withRSA Signaturverfahren für einen neuen RSA Schlüssel). Das zweite soll von dem Ersten unabhängig sein (z.B. SHA1withDSA).

2.5 Security Condition.

Ein wichtiger Punkt ist, dass der Client (oder die Chipkarte) entscheiden muss, ob UMP authentisch, richtig adressiert und aktuell ist. Adressierung und Aktualität wird über in UMP integrierte Felder geregelt. Was aber ist mit Authentizität und Integrität unter dem Gesichtspunkt, dass eine Signatur ungültig ist. Wie in Kapitel 1 gezeigt, muss UMP mit zwei verschiedenen Verfahren signiert sein. Die Frage ist, wie der Client entscheidet, ob dies die richtigen sind. Der Client muss dazu Security Conditions verwalten. Es gibt zwei Arten:

- Statische Security Conditions - Die Signaturen aus UMP müssen zu den Zertifikaten des Trust Centers passen. Falls das Trust Center das UMP nicht signiert, sondern ein von dem Trust Center zu diesem Zweck autorisierter Dienstleister, muss ein Attributszertifikat mit kurzer Gültigkeitsdauer vorhanden sein.
- Dynamische Security Conditions legen mögliche Paare von voneinander unabhängigen Verfahren fest¹¹.

Darüber hinaus muss der Client separat seinen `MessageClientID` abspeichern. Das ist allerdings nicht sicherheitskritisch, sondern nur für Denial-of-Service (DoS)-Attacks gut und nützlich.

Den folgenden Ablauf kann man dabei wie folgt festlegen:

¹¹ Siehe Abschnitt 2.6

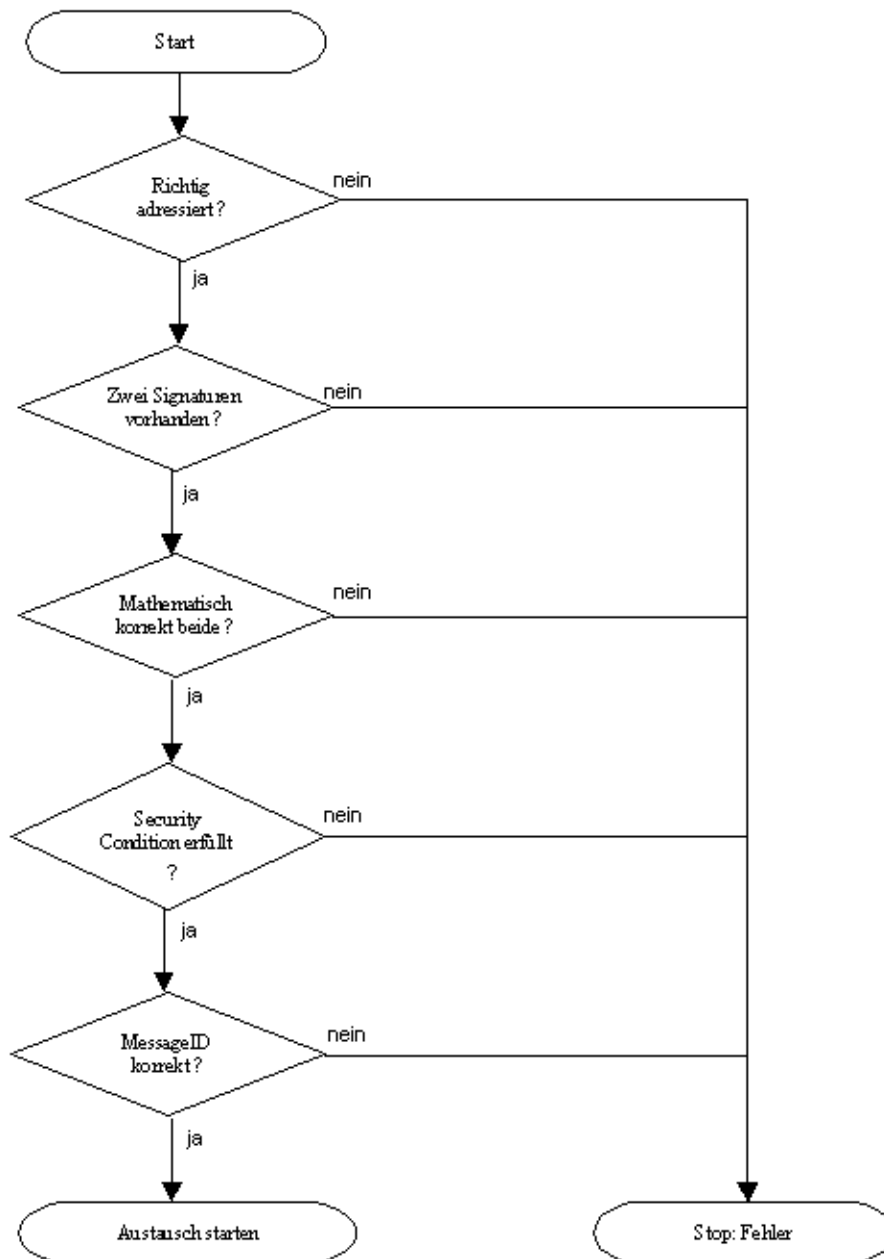


Abb. 2

2.6 Registry.

Registry definiert für alle kryptographischen Primitiven¹² und Verfahren die zum Austausch nötigen Verfahren, die präsentiert sein müssen, damit ein Austausch möglich ist. Die Festlegung, welche Security Condition¹³ pro Primitive oder zusammengesetztem Verfahren gut ist, ist von Personen im Trust-Center festzulegen.

¹² Unter kryptographischen Primitiven verstehen wir die Verfahren, die als Grundlage für andere kryptographische Verfahren dienen. Z.B. Signaturverfahren MD5withRSA (solche Verfahren nennen wir zusammengesetzt) besteht aus zwei Primitiven: Hashfunktion MD5 und RSA Verfahren.

¹³ Siehe Abschnitt 2.5

Die Registry ist sicherheitskritisch, deshalb:

- Die Registry ist multipel signiert, falls der Client nicht für ihre Unversehrtheit garantieren kann.
- Oder die Registry hat selber folgende Security Conditions: man kann die sie immer lesen, Schreibzugriff ist nur intern möglich und nicht von außen.

Es wird eine Policy festgelegt, die besagt, dass von den n vorhandenen Signaturen mindestens m mathematisch korrekt sein müssen ($m \leq n$). Außerdem muss beachtet werden, dass das zu deaktivierende Verfahren unter diesen m Signaturen zu finden ist¹⁴. Wir treffen für unseren Fall die Annahme, dass $n=m=2$ ist.

Von der Idee her, ist die Registry wie folgt als Tabelle vorstellbar (eine Auswahl):

ID	Primitive	enthalten in	Security Conditions
1	OID von RSAEncryption		((11 OR 12 OR 13 OR 14) AND (15 OR 16)) OR (15 AND 16)
2	OID von RSASignature	11, 12, 13, 14	(11 OR 12 OR 13 OR 14) AND (15 OR 16)
3	OID von DSA	15	15 AND (11 OR 12 OR 13 OR 14 OR 16)
4	OID von ECDSA	16	16 AND (11 OR 12 OR 13 OR 14 OR 15)
5	OID von DES with CBC		((11 OR 12 OR 13 OR 14) AND (15 OR 16)) OR (15 AND 16)
6	OID von MD5	12	12 AND (11 OR 13 OR 14 OR 15 OR 16)
7	OID von SHA1	11, 15, 16	(11 OR 15 OR 16) AND (12 OR 13 OR 14)
8	OID von RIPEMD160	13, 14	(13 OR 14) AND (11 OR 12 OR 14 OR 16)
9	OID von PKCS#1 padding	13	13 AND (11 OR 12 OR 14 OR 15 OR 16)
10	OID von ISO9796-2rnd padding	11, 14	(11 OR 14) AND (12 OR 13 OR 15 OR 16)

¹⁴ Sonst kann unter Umständen ein Angreifer ein sicheres Verfahren mit Hilfe eines unsicheren Verfahren entfernen.

ID	Zusammengesetzte Verfahren	Security Condition
11	OID von RSASignature with SHA1 according to ISO/IEC 9796-2	k.A.
12	OID von RSASignature with MD5	12 AND (15 OR 16)
13	OID von RSASignatur with RIPEMD160 and PKCS#1 padding	13 AND (15 OR 16)
14	OID von RSASignature with RIPEMD160 according to ISO/IEC 9796-2	14 AND (15 OR 16)
15	OID von DSA With SHA1	15 AND (12 OR 13 OR 14)
16	OID von ECDSA With SHA1	16 AND (12 OR 13 OR 14)

Abb. 3

Die Tabelle in Abb. 3 besteht aus zwei Teilen: zum einen der Teil mit kryptographischen Primitiven mit dazu gehörenden Hinweisen auf zusammengesetzte Verfahren, wo diese Primitive benutzt wird und zu zweiten Security Conditions für diese Primitive. Der zweite Teil der Tabelle besteht aus zusammengesetzten Verfahren und entsprechenden Security Conditions. Die beiden Teile können nicht separat voneinander betrachtet und eingesetzt werden.

Bemerkung: Nicht alle Verfahren müssen bei jedem Client vorhanden sein. Aber das Trust-Center hält nur eine (!) Registry für alle Clients bereit. Damit wird versucht zu verhindern, dass jeder Client eine andere Registry bekommt und das Trust-Center alle die dafür nötigen Informationen verwalten muss, was vom Aufwand her wahrscheinlich nicht realisierbar wäre.

Ein Beispiel für die Registry Tabelle:

Stellen wir uns vor, dass ein Algorithmus entdeckt wurde, der für die großen Zahlen schnell Primfaktoren berechnen kann. Damit wären alle kryptographischen Algorithmen, die auf dieses Problem aufsetzen, automatisch unsicher, so auch RSA. Über Registry Tabelle wird der Client feststellen, dass die Signaturverfahren SHA1withRSA, MD5withRSA und RIPEMD160withRSA aus PSE zu entfernen sind. Das UMP muss in diesem Fall entweder mit SHA1withRSA (mögliche Varianten MD5withRSA oder RIPEMD160withRSA) und SHA1withDSA (mögliche Variante SHA1withECDSA) signiert werden.

Für die Beschreibung der Registry wird wiederum ASN.1 Datendefinition benutzt.

```
Registry ::= SEQUENCE {
    entries SEQUENCE OF Entry,
    protection SEQUENCE OF Protection OPTIONAL
}
```

Jedes Entry beschreibt eine Zeile in der Registry Tabelle. Das Feld protection enthält multiple Signaturen über alle Zeilen der Tabelle.

Das Entry wird durch folgende ASN.1 Datenstruktur definiert:

```
Entry ::= SEQUENCE {
    identifier INTEGER,
```

```

    primitiveOrConstructed PrimitiveOrConstructed
}

```

In identifier wird die Ordnungsnummer der Zeile in der Registry Tabelle gespeichert und primitiveOrConstructed gibt an, um welches Teil der Tabelle es sich handelt.

PrimitiveOrConstructed wird wie folgt beschrieben:

```

PrimitiveOrConstructed ::= CHOICE {
    primitive    [0] Primitive,
    constructed  [1] Constructed
}

```

Das heißt, eine Zeile kann entweder von Typ Primitive oder von Constructed sein:

```

Primitive ::= SEQUENCE {
    oid      ObjectIdentifier,
    containIn SequenceOf Integer OPTIONAL
    securityConditions SequenceOf ORCondition
}

```

Das Feld oid enthält weltweit eindeutige Bezeichnung eines Algorithmus, containIn enthält die Liste mit zusammengesetzten Verfahren, wo diese Primitive vorkommt, und in securityCondition wird eine Liste mit Paaren von Signaturalgorithmen gespeichert, die miteinander kombinierbar sind, d.h. mathematisch voneinander unabhängig sind.

```

Constructed ::= SEQUENCE {
    oid      ObjectIdentifier,
    securityConditions SequenceOf ORCondition OPTIONAL
}

```

Diese Struktur gleicht bis auf die containIn Liste der Primitive Datenstruktur.

Die ORCondition ist durch folgende ASN.1 Datenstruktur beschrieben:

```

ORCondition ::= SEQUENCE {
    entry_one INTEGER,
    entry_two INTEGER
}

```

Diese Struktur definiert, dass wenn das UMP mit einem Signaturalgorithmus mit Ordnungsnummer entry_one in der Registry Tabelle unterschrieben wird, kann er mit Signaturalgorithmus mit Ordnungsnummer entry_two kombiniert werden und umgekehrt.

Ein Beispiel, wie die gesamte Registry Tabelle aussehen kann:

```

Registry {
    entries
    {
        {identifier '1',
        primitiveOrConstructed : primitive {

```

```

oid      'OID OF RSAEncryption'
securityConditions {
  {11,15},{12,15},{13,15},{14,15},{11,16},
  {12,16},{13,16},{14,16},{15,16}
}
},
...
{identifier '16',
primitiveOrConstructed : 'OID von ECDSA with SHA1'
securityCondition {
  {12,16}
}
},
protection Protection      OPTIONAL -- falls notwendig
}

```

Es gibt nur einen Fall, wenn es nötig sein wird, die eine Registry durch die andere zu ersetzen. Dieser Fall tritt dann ein, wenn ein völlig neuer Algorithmus (Primitive oder Zusammengesetzt) entdeckt wurde und die alte Registry davon keine Kenntnisse hat. Dann wird die neue Registry von Trust Center Administrator neu erstellt und an alle Clients mit dem nächsten UMP versendet. Jeder Client speichert dann die aktuelle Version der Registry lokal und benutzt sie für Konsultationen bei Verarbeitung aller nachfolgenden UMP's.

2.7 UMP und PKCS#7.

Public Key Cryptographic Standard #7 (PKCS#7) von RSA Inc. stellt ein Standard-Format für die Übermittlung von Daten bereit. Es können u.a. Klartexte (data), signierte Texte (signedData), verschlüsselte Texte (envelopedData) und signierte Kryptogramme (signedAndEnvelopedData) über ASN.1 codiert werden.

Ein PKCS#7-Objekt baut sich generell wie folgt auf: Ein ContentInfo beinhaltet einen ContentType, der angibt, was nun folgt und die Daten selber. Bereits definiert sind Data, SignedData, EnvelopedData und so weiter. Dazu sind die OIDs im PKCS#7-Standard angegeben:

```
data OBJECT IDENTIFIER ::= { pkcs-7 1 }
```

```
ContentInfo ::= SEQUENCE {
  contentType ContentType,
  content [0] EXPLICIT ANY DEFINED BY contentType OPTIONAL }

```

```
ContentType ::= OBJECT IDENTIFIER
```

```
signedData OBJECT IDENTIFIER ::= { pkcs-7 2 }
```

```
SignedData ::= SEQUENCE {
  version      Version,
  digestAlgorithms DigestAlgorithmIdentifiers,
  contentInfo  ContentInfo,
}

```

```
certificates [0] IMPLICIT ExtendedCertificatesAndCertificates OPTIONAL,  
crls [1] IMPLICIT CertificateRevocationLists OPTIONAL,  
signerInfos SignerInfos  
}
```

```
.....  
.....
```

Diese Liste wollen wir um unser UpdateComponent mit unserem OID ergänzen.

```
Contents CONTENTS ::= {  
{Data IDENTIFIED BY data}  
// andere contents  
{UpdateComponent IDENTIFIED BY updateComponent} –neu hinzugefügt.  
}
```

```
UpdateComponent OBJECT IDENTIFIER ::= {iso(1) identified-organization(3) dod(6) internet(1)  
private(4) enterprises (1) TUD(8301) cdc(3) management(3) ump(2)}
```

```
UpdateComponent ::= SEQUENCE {...}15
```

¹⁵ Siehe Abschnitt 2.1

3 Implementierung der Datenstruktur.

3.1 Implementierungswerkzeuge.

Da alle anderen Komponenten des Outlook Plugins in Java implementiert sind, ist Java die ausgewählte Sprache für die Austauschimplementierung. Die Java bietet folgende Vorteile:

- Heutige Anwendungen sind so komplex, dass eine effiziente und fehlerfreie Softwareentwicklung die Verwendung einer objektorientierten Programmiersprache gebietet. Java nimmt in dieser Hinsicht eine Sonderstellung ein, denn Java ist eine moderne Programmiersprache, die besonders im Hinblick auf Sicherheit entwickelt wurde.
- Java besitzt keine Hintertüren - Damit zählt Java zu den wenigen Programmiersprachen, in denen die Korrektheit vom Programmcode einfach überprüft werden kann. Dies ist eine wichtige Eigenschaft einer Programmiersprache, in der Sicherheitsmechanismen implementiert werden sollen.
- Java ist portabel - Java Programme funktionieren ohne Änderung auf fast jedem modernen Betriebssystem und bieten wohldefinierte Schnittstellen zu allen gängigen Standards. Diese Plattformunabhängigkeit ermöglicht den Einsatz von Produkten auch in heterogenen Umgebungen. [FLEXIPKI]

Weiterhin bietet Java mit *Java-Cryptographic-Architecture (JCA)* und *Java Cryptography Extension (JCE)* die vordefinierten und flexiblen Sicherheitinfrastrukturen.

Für die Implementierung mit der ASN.1 definierten Datenstrukturen benötigt man speziellen Werkzeuge, die diese Strukturen auf Java-Datenstrukturen abbildet. Dafür wurde an der Fraunhofer Gesellschaft von Dr. Volker Roth entwickelte Paket CODEC ausgewählt.

Der Name "CODEC" ist eine Abkürzung für Coder/DECoder, solche, die das Hauptteil von CODEC darstellen: Klassen für Kodierung und Dekodierung verschiedener Datenformaten (Base64, DER, RFC1779, PKCS). Alle diese Formate und allgemeine Regeln für Kodierung und Dekodierung sind Standards, die auf Teilmengen von ASN.1 Standard basieren. In CODEC sind alle einfachen ASN.1 Typen durch spezielle Java-Objekte repräsentiert. Auch mehrere komplexere Strukturen wurden implementiert.

Dieses Paket entstand in FHG im Jahre 1999 im Laufe des Projektes SeMoA (Secure Mobile Agents). Bei den Versuchen, die mobilen Agenten mit kryptographischen Funktionen zu entwickeln, hat man auf das Problem gestoßen, dass die Implementierungen der kryptographischen Funktionen auf den Servern von verschiedenen Providern sehr unterschiedlich ist. Aus diesem Grund hatten die mobilen Agenten sehr großen Datenoverhead, um möglichst mit vielen Providern kompatibel zu sein. Um den Datenumfang zu reduzieren, sollte jeder Provider die Entwicklung seiner kryptographischen Funktionen auf Basis von einem einheitlichen Paket stellen. Da kein geeignetes freiverfügbares Paket zur Verfügung stand, wurde von der FHG in Zusammenarbeit mit der Technischen Universität Darmstadt (Fachgebiet Theoretische Informatik) ein CODEC - Paket entwickelt, welches genau die gestellten Anforderungen erfüllt.

Seit seiner Entwicklung ist CODEC zu einem unverzichtbaren Werkzeug bei der Implementierung der kryptographischen Protokolle geworden und wird bei der FlexiPKI Entwicklung breit eingesetzt.

3.2 Implementierung der ASN.1 Datenstruktur.

Wie schon im letzten Abschnitt beschrieben wurde, stellt das CODEC - Paket alle ASN.1 Grunddatentypen zur Verfügung. Ausgehend von diesen Datentypen wurden die anderen Datenstrukturen implementiert, so auch kryptographische Standards PKCS#7, PKCS#10, PKCS#11, PKCS#12 etc.

Die Implementierung von PKCS#7 ist von besonderer Bedeutung, da UMP mit diesem Standard zusammenhängt¹⁶.

Die PKCS#7 Klassen befinden sich in `codec.pkcs7` Package. Die Wichtigste für uns ist die Klasse `ContentInfo`, die gleichnamige ASN.1 Datenstruktur implementiert. Diese Klasse stellt den Konstruktor zur Verfügung, dem als Parameter eine Registry übergeben wird. Eine Registry für PKCS#7 enthält eine Tabelle, die die Zuordnung zwischen Content-Type und content herstellt. Die Standardregistry für die Klasse `ContentInfo` wird durch die Klasse `PKCSRegistry` in dem Package `codec.pkcs` implementiert. Dort sind folgende PKCS#7 Content-Typen definiert:

- `Data`
- `SignedData`
- `EnvelopedData`
- `SignedAndEnvelopedData`
- `EncryptedData`

Für das UMP muss diese Registry erweitert werden. Die neue Registry ist durch die Klasse `UMPRegistry`¹⁷ im Package `smime.ump` implementiert. Sie definiert zusätzlich zu obengenannten Typen auch neueingeführten Typen `UpdateComponent`. Aus technischem Grund wird sich die Klasse `UpdateComponent`, welche die entsprechende ASN.1 Datenstruktur implementiert, auch in Package `codec.pkcs7` befinden. Wenn die `ContentInfo` Klasse mit dieser Registry erzeugt wird, wird sie über den neuen Typen informiert.

Wie schon oben gesagt wurde, implementiert die Klasse `UpdateComponent` die gleichnamige ASN.1 Datenstruktur. Üblich ist, dass die Klassennamen mit den Namen der ASN.1 Datenstrukturen übereinstimmen¹⁸. Außer der Klasse `UpdateComponent` befinden sich alle Klassen, die UMP Datenstrukturen implementieren, im Package `smime.ump`. Die Implementierung der ASN.1 mit CODEC verläuft üblicherweise nach gleichem Muster, das unten schematisch gezeigt wird.

Nehmen wir an, dass folgende ASN.1 Struktur mit CODEC zu implementieren ist.

```
A ::= SEQUENCE {
    B ASN1DatenTyp1
    C ASN1DatenTyp2
}
```

Die folgende Java Klasse zeigt eine mögliche Lösung:

¹⁶ Siehe Abschnitt 2.7

¹⁷ Für die Erweiterung der Registry wird die statische Methode `addOIDRegistry` der Klasse `codec.asn1.OIDRegistry` benutzt, die Zugriffsberechtigung (`AccessControl`) überprüft. Die Zugriffsberechtigung muss deswegen in der Datei `java.policy` entsprechend gesetzt werden.

¹⁸ Ausnahme aus dieser Praxis ist die Registry ASN.1 Datenstruktur, die durch die Klasse `RegistryASN1` implementiert wird.

```
public A extends ASN1Sequence {
    private ASN1DatenTyp1 B;
    private ASN1DatenTyp2 C;

    // Öffentliches Strukturkonstruktor ist fast immer nötig.
    // Erzeugt leere Struktur
    public A {
        Initialisierung von Instanzvariablen.
        B = new ASN1DatenTyp1();
        add(B);
        C = new ASN1DatenTyp1();
        add(C);
    }

    // Definition von anderen Konstruktoren nach Bedarf
    .....
    // Üblicherweise werden auch Methoden für Schreib-/ Lesezugriffen definiert. Je nach Situation,
    // können sie entweder öffentlich (public), geschützt (protected) oder privat (private) sein.
    setB{ }
    getB{ }

    setC{ }
    getC{ }

} // Ende der Klassendefinition.
```

Nach diesem Muster sind alle ASN.1 Datenstrukturen in Package `smime.ump` implementiert. Für Einzelheiten wird an die Dokumentation und evtl. an Quellcode verwiesen.

4 Implementierung des Trust Center Administrator Tools.

Eine Teilaufgabe, die in Rahmen dieser Diplomarbeit gestellt wurde, war die Entwicklung des Beispieltools, mit dessen Hilfe ein verantwortlicher Mitarbeiter (Administrator) des Trust Centers die UpdateComponent- Strukturen aufbauen und anschließend an seine Clients versenden kann.

Diese Aufgabe implementieren folgende Klassen:

- CAAdminDialog
- CAStatus
- Info
- UMPSend
- MailTransport

Der Zusammenhang und die Beziehungen zwischen den Klassen zeigt folgende Abb. 4

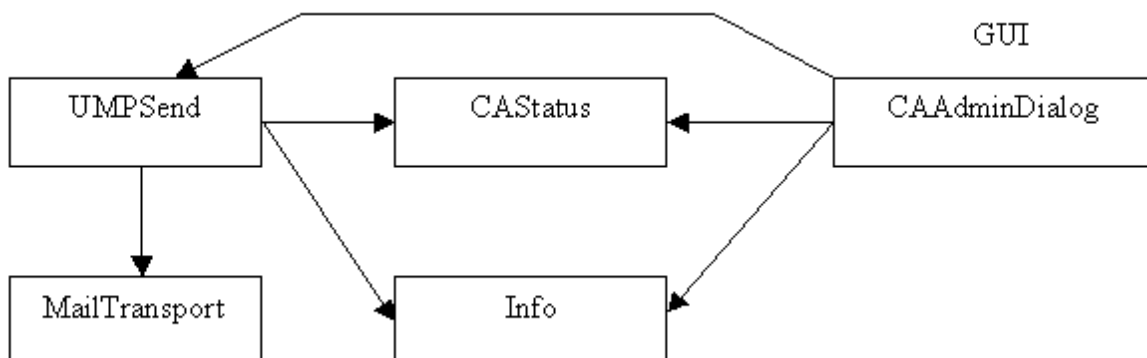


Abb. 4

4.1 Die Klasse CAAdminDialog.

Die Klasse CAAdminDialog implementiert die graphische Oberfläche für Trust Center Administrator. In mehreren graphischen Dialogen kann der Administrator alle notwendigen Informationen für den Aufbau von UpdateComponent spezifizieren.

Als erster wird jener Dialog angezeigt, der dem Trust Center Administrator die kryptographische Algorithmen zu spezifizieren erlaubt, die bei dem Client entfernt werden müssen. Er kann die Signaturalgorithmen spezifizieren, deren Trust Center und (oder) Benutzerzertifikate beim Client gelöscht werden müssen.

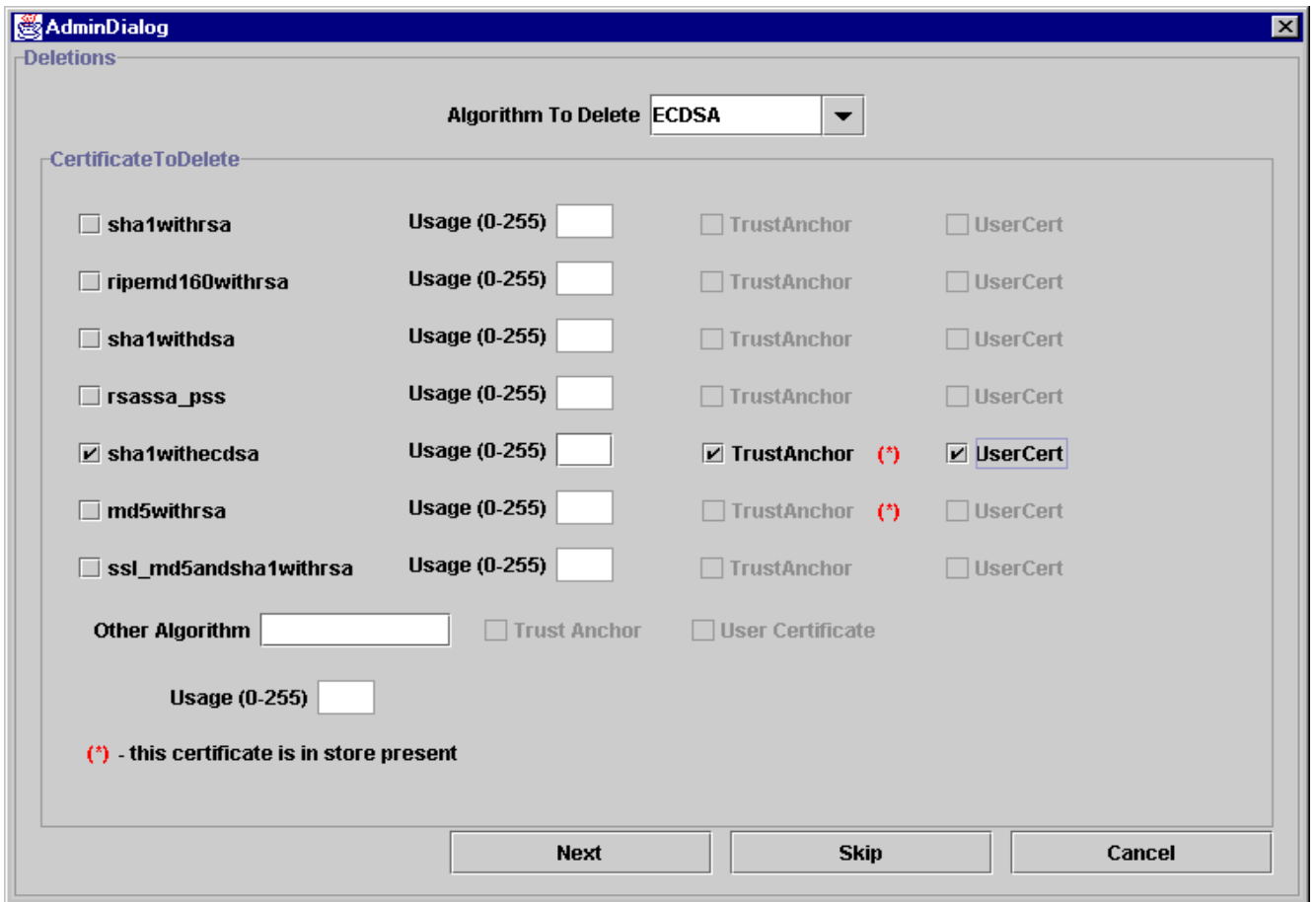


Abb. 5

Danach wird der nächste Dialog angezeigt. Dieser erlaubt dem Administrator, die Internet-Adressen für neuen Software PSE und (oder) Smart Card Komponenten anzugeben und die Signaturalgorithmus für das neue Trust Center Zertifikat, das beim Client installiert werden muss und den kryptographischen Algorithmus für das neue Schlüsselpaar, welches beim Client generiert werden muss, zu spezifizieren.

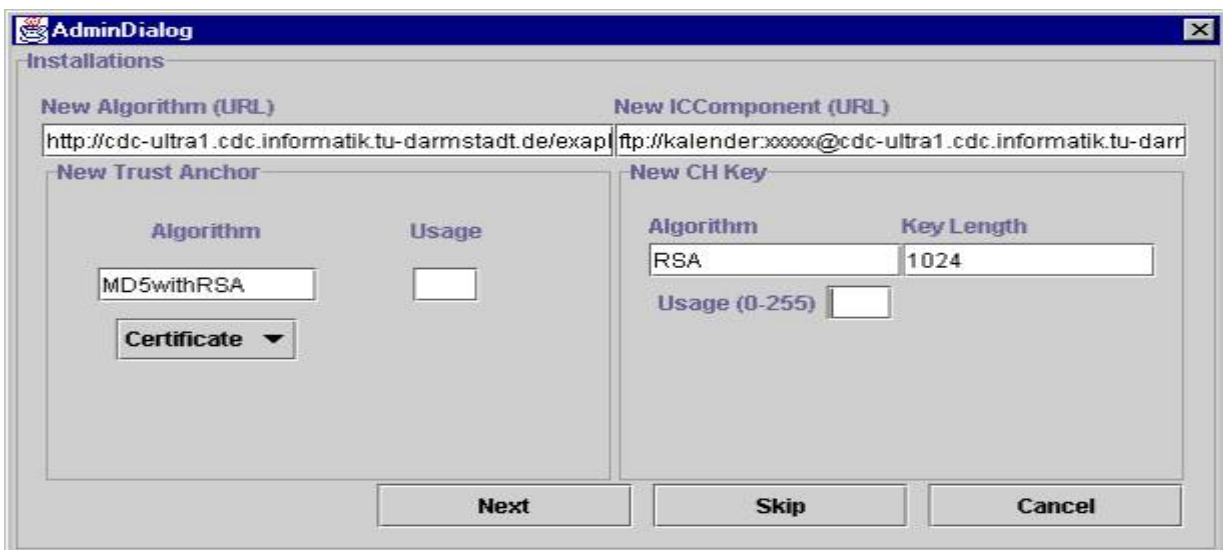


Abb. 6

Außer der Einrichtung der neuen kryptographischen Komponenten kann es dazu kommen, dass der Administrator die neue Registry¹⁹ spezifizieren soll. Dafür stellt das Administrator-Tool folgende drei Dialoge zur Verfügung. In erstem Dialog auf der Abb. 7 wird angegeben, wie viele kryptographische Primitive und (oder) zusammengesetzte Algorithmen die neue Registry enthält.

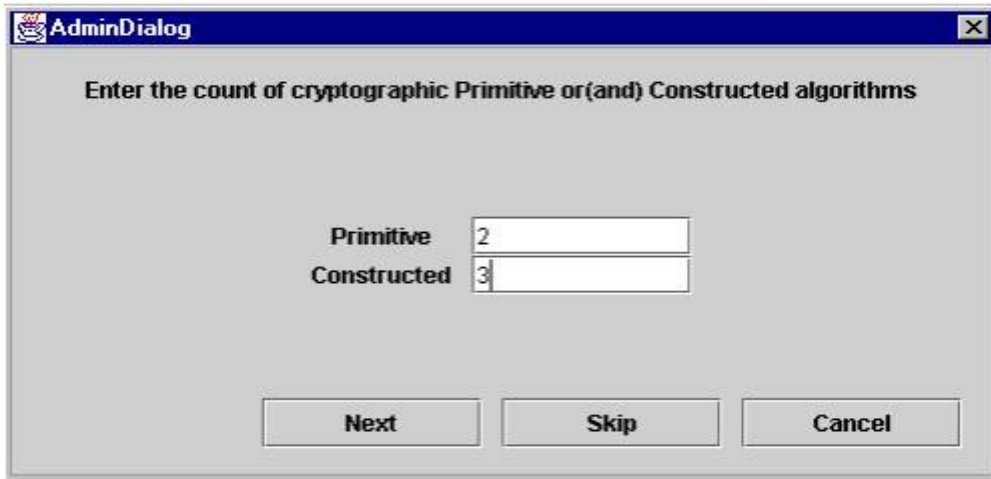


Abb. 7

Wenn der auf der Abb. 7 dargestellter Dialogfeld den Knopf „Next“ gedrückt wird, erscheint folgende Tabellenvorlage für kryptographische Primitive der neuen Registry: Die erste Spalte wird automatisch durchnummeriert und nicht mehr änderbar. Die zweite Spalte ist für Algorithmus OID vorgesehen. In der dritten Spalte kann man die ID's von Algorithmen angeben, die diese kryptographische Primitive beinhaltet²⁰. Die einzelne ID's müssen durch Semikolon getrennt werden. In der vierten Spalte muss der Administrator die Security Conditions²¹ angeben: eine Security Condition stellt dabei ein Paar von ID's in der Registry Tabelle, die durch Doppelpunkt getrennt sind. Die einzelne Security Conditions sind durch Semikolon voneinander zu trennen. Die Pflichtfelder in dieser Tabelle sind mit Sternchen markiert.

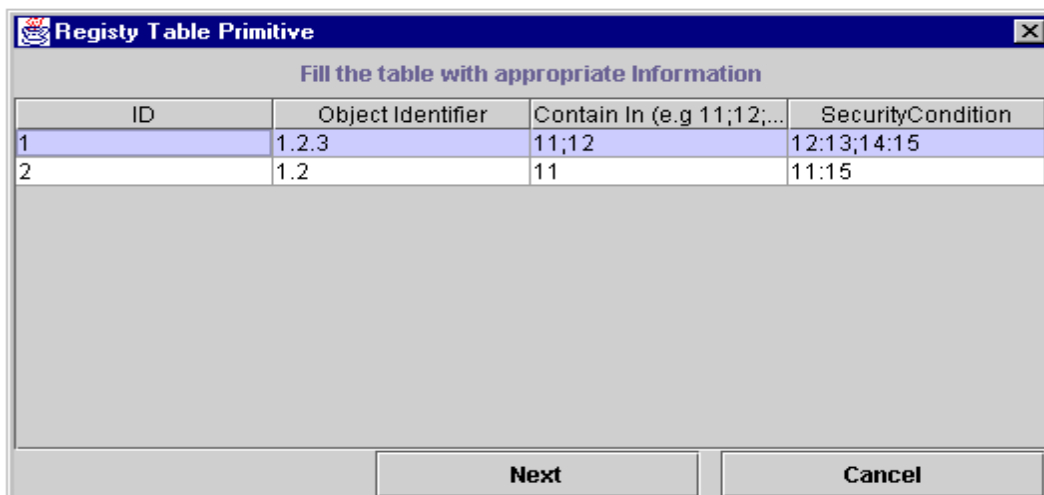


Abb. 8

¹⁹ Siehe Abschnitt 2.6

²⁰ Siehe Abschnitt 2.6

²¹ Siehe Abschnitt 2.6

Wenn in dem letzten Dialogfeld den „Next“ Knopf gedrückt wird, wird die Vorlage für den zweiten Teil der Registry angezeigt. Dieser Teil repräsentiert die zusammengesetzten Algorithmen. Bis auf ContainIn Spalte ist diese Tabelle der oben beschriebenen Tabelle für kryptographische Primitive gleich. Die Nummerierung der ID Spalte ist durchgehend für beide Tabellen.

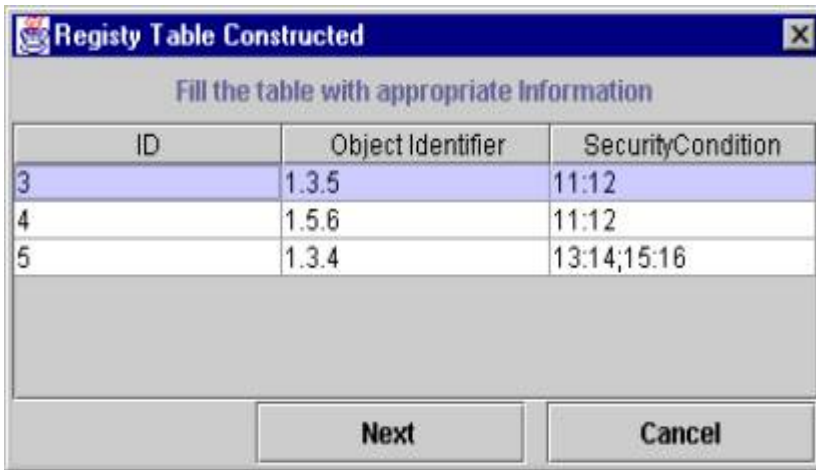


Abb. 9

Als nächstes wird ein Dialog angeboten, in dem die Signaturalgorithmen spezifiziert werden können, mit denen UpdateComponent signiert wird. Dieser Dialog wird dynamisch aufgebaut. Während der Initialisierung der Klasse Info²² werden alle angemeldeten JCA Provider untersucht und die von ihnen implementierten Signaturalgorithmen gespeichert. Diese Information wird dann beim Aufbau dieses Dialoges benutzt. Auch der Provider, der ein Signaturalgorithmus implementiert, wird angezeigt. So kann es vorkommen, dass der gleiche Signaturalgorithmus mehr als einmal auftaucht: z.B. SHA1withDSA aus SUN Provider und SHA1withDSA aus CDCStandard Provider. Für jeden vorhandenen Signaturalgorithmus wird darüber hinaus Zertifikatverwaltung durchsucht. Falls ein entsprechendes Zertifikat gefunden wurde, wird das Kästchen „Attach Certificate“ mit einem roten Stern markiert und dieses Kästchen ist für die Selektion freigegeben.

²² Siehe Abschnitt 4.2

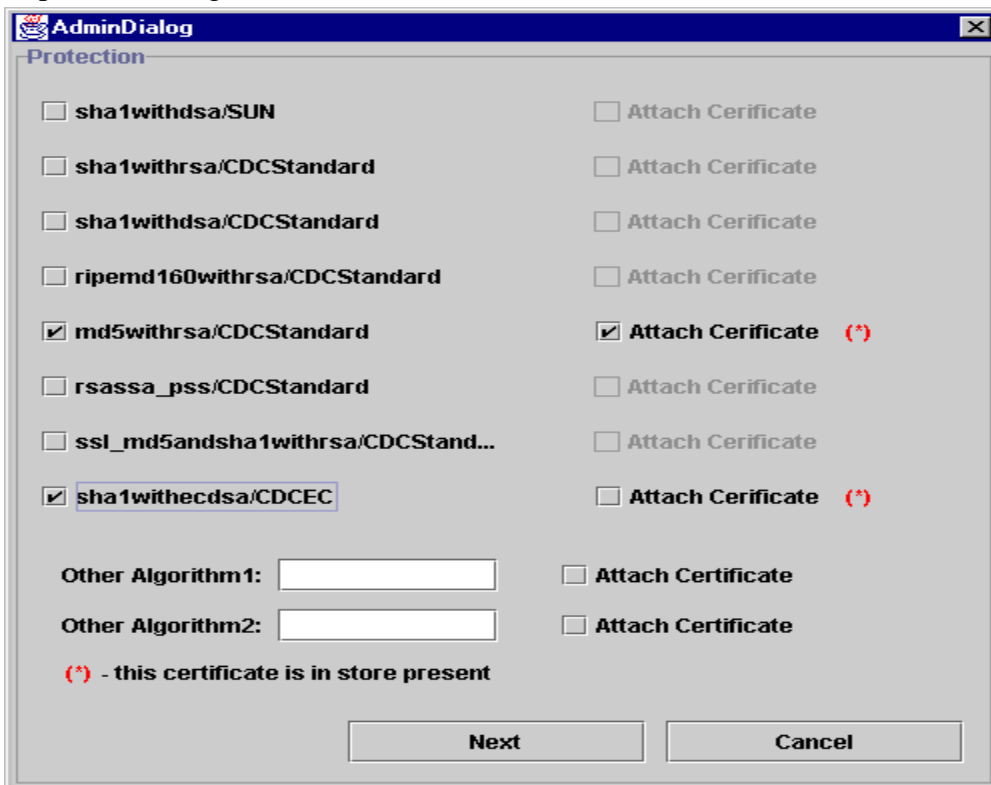


Abb. 10

Als letztes wird ein Dialogfeld angezeigt, welches erlaubt, entweder diese UpdateComponent an alle bekannte Clients zu versenden oder an einen Client mit einer bestimmten Identität. Als Identität von Smart Card gilt dabei hexadezimale Answer-To-Reset (ATR) Sequenz (z.B. 00 01 02). Als Identity gilt ein SubjectName aus einem Benutzerzertifikat.

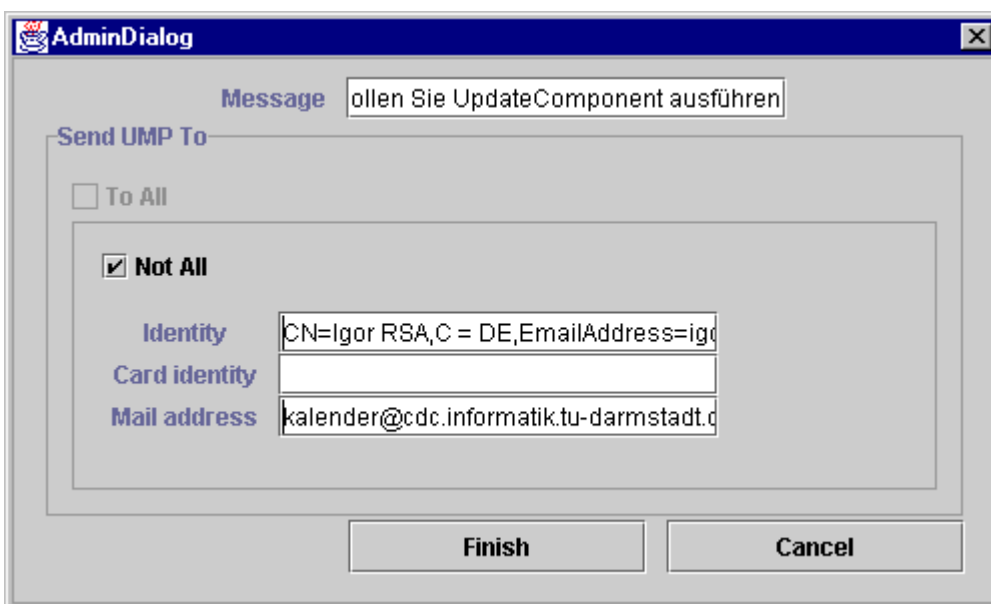


Abb. 12

4.2 Die Klasse *Info*.

Dabei benutzt die Klasse `CAAdminDialog` die Klasse `Info`, wo verschiedene Konfigurationsinformationen über das Trust Center gespeichert sind und zwar:

- Homeverzeichnis vom Trust Center Administrator
- Internet-Adresse vom Mailserver
- Mailadresse vom Trust Center
- Die Namen vom Trust Center in X501 Format
- Name von der Konfigurationsdatei, in der alle Informationen dauerhaft gespeichert werden.
- Die Liste mit allen kryptographischen Primitiven und konstruierten Signaturalgorithmen²³.
- Die Liste mit allen Signaturalgorithmen. Diese wird während der Initialisierung der Klasse `Info` dynamisch aufgebaut.
- Die Liste mit allen Signaturalgorithmen und zugehörigen Providern. Damit kann der Benutzer ein Signaturalgorithmus aus einem bestimmten Provider aussuchen.

Außerdem stellt diese Klasse die `set-` und `get-` Methoden für Setzen obengenannten Felder zur Verfügung.

4.3 Die Klasse *CAStatus*.

Die Informationen, die in GUI gesammelt werden, werden in Instanz der Klasse `CAStatus` gespeichert. Die folgende Informationen werden im `CAStatus` Objekt geführt:

- Algorithmus, die aus dem Client entfernt werden müssen.
- Trust Center-Zertifikaten, die aus dem Client entfernt werden müssen.
- Benutzerzertifikaten, die aus dem Client entfernt werden müssen.
- Internet-Adresse, wo sich die neue kryptographische Komponente für PSE befindet.
- Internet-Adresse, wo sich die neue kryptographische Komponente für Smart Card befindet.
- Signaturalgorithmus, für das ein neues Trust Center-Zertifikat installiert werden muss.
- Kryptographischer Algorithmus, für den ein neuer Schlüssel beim Client generiert werden muss.
- Die Länge des neu generierten Schlüssels.

²³ Siehe Abschnitt 2.6

- Information, ob die Trust Center-Zertifikaten mit Signaturen mitgeschickt werden müssen, um Verifikation von Signaturen beim Client zu erleichtern.
- Text, der beim Client erscheint, bevor der Austausch von seinen kryptographischen Komponenten durchgeführt wird.
- Information, ob die UpdateComponent zu allen Clients geschickt wird oder nicht.
- Information über Identitäten von Client und Smart Card, bei denen Update durchgeführt werden muss.
- Informationen über kryptographischen Primitiven und zusammengesetzten Verfahren für die neue Registry.
- Neue Registry selbst.

Genauso wie in Klasse `Info` stellt diese Klasse alle notwendige Methoden für Zugriffe auf diese Informationen.

4.4 Die Klasse `UMPSend`.

Der eigentliche Aufbau des UMP findet in der Klasse `UMPSend` statt. Die Instanz der `UMPSend` Klasse wird in der Klasse `CAAdminDialog` als Reaktion auf Druck des Knopfes „Finish“ des letzten Dialoges gegründet und die aktuellen Instanzen von `Info` und `CAStatus` Klassen an diese Instanz übergeben. In `UMPSend` Klasse spielt die `send`-Methode die zentrale Rolle. Die Informationen aus `CAStatus` und `Info` Klassen werden mit Hilfe der Zugriffsmethoden ausgelesen und auf die ASN.1 Strukturen von `UpdateComponent` abgebildet.

Nachdem `UpdateComponent` aufgebaut wurde, werden die Signaturen über DER kodierte Felder von `UpdateComponent` berechnet und in diesem UMP gesetzt. Anschließend wird `UpdateComponent` in PKCS#7 Objekt eingebunden und mit Hilfe der Klasse `MailTransport` an Client(s) verschickt.

4.5 Die Klasse `MailTransport`.

Die Klasse `MailTransport` ist die Implementierung von `Transport` Interface, die die Übertragung von `UpdateComponent` von Trust Center zu Client per Mail gewährleistet. Für die Implementierung werden die `JavaMail` [JAVAMAIL] und `Java Activation Framework` [JAF] Klassen von Sun herangezogen. Diese stellen bequeme API's zum SMTP (Simple Mail Transport Protocol) Protokoll bereit und übernehmen damit den größten Teil der Implementierung. Die zentrale Stelle der Klasse `MailTransport` ist die Methode `send`, die als Parameter eine Instanz von PKCS#7 Objekt erhält. Sie versendet das UMP an Adressen, die im `CAAdminDialog` angegeben wurde. Die speziell dafür eingeführte Content-type heißt `application/pkcs7-ump` oder `application/x-pkcs7-ump`.

Um die richtige Behandlung von diesem Content-type zu gewährleisten, wurde die Klasse `application_pkcs7_ump` entwickelt, die eine Implementierung von `DataContentHandler` Interfaces ist.

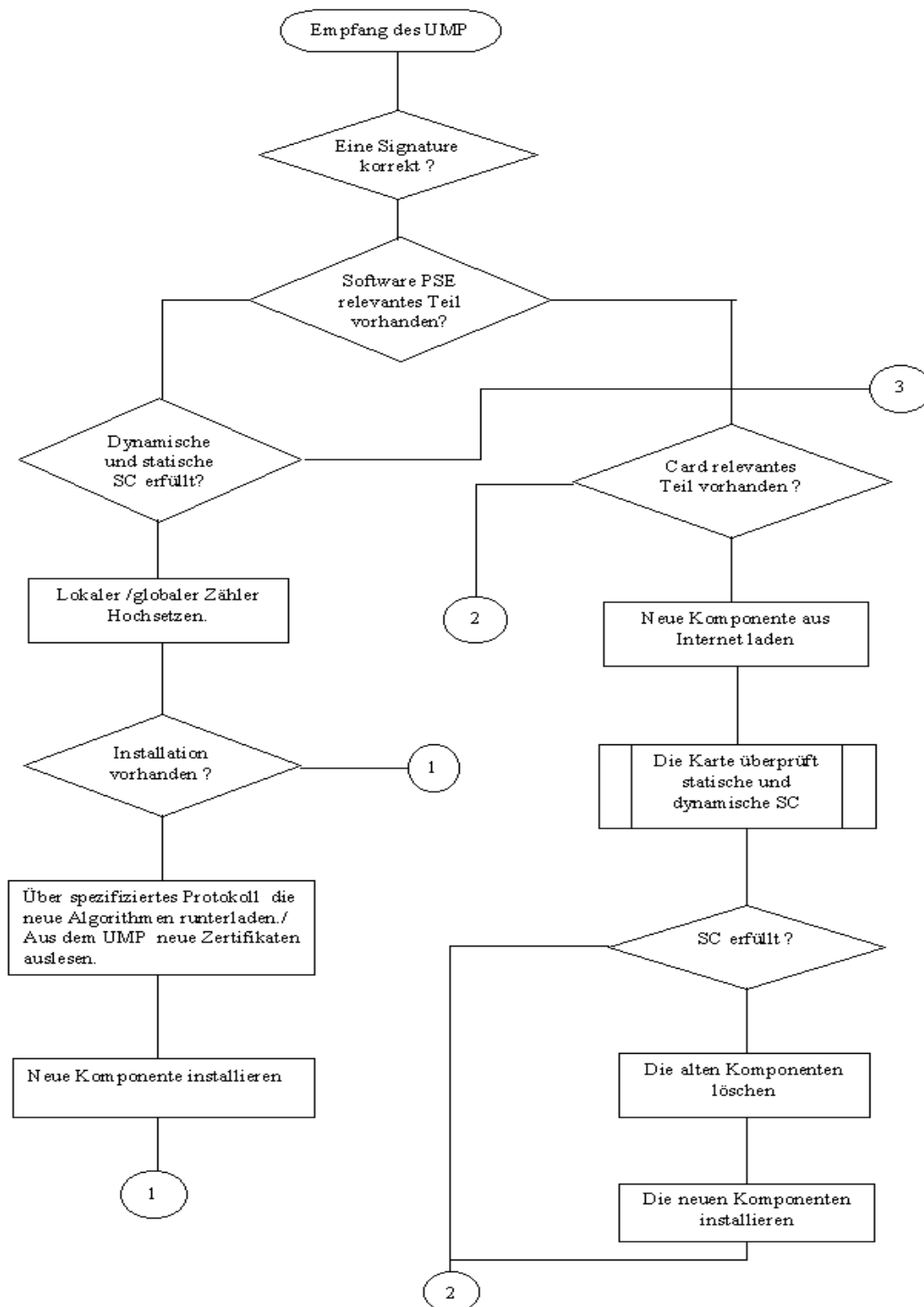
Zwei Methoden sind dabei zu implementieren:

- `getContent()`, die aus einem `InputStream` ein entsprechendes Objekt erzeugen.
- `writeTo()`, die Inhalt eines Objektes in `OutputStream` schreibt.

Alle `ContentHandler` sind in der Methode `registerContentHandle()` registriert.

5 Austausch der kryptographischen Komponenten.

Die Abb. 12 zeigt Ablauf des Austausches von Komponenten einer PKI bei einem Client:



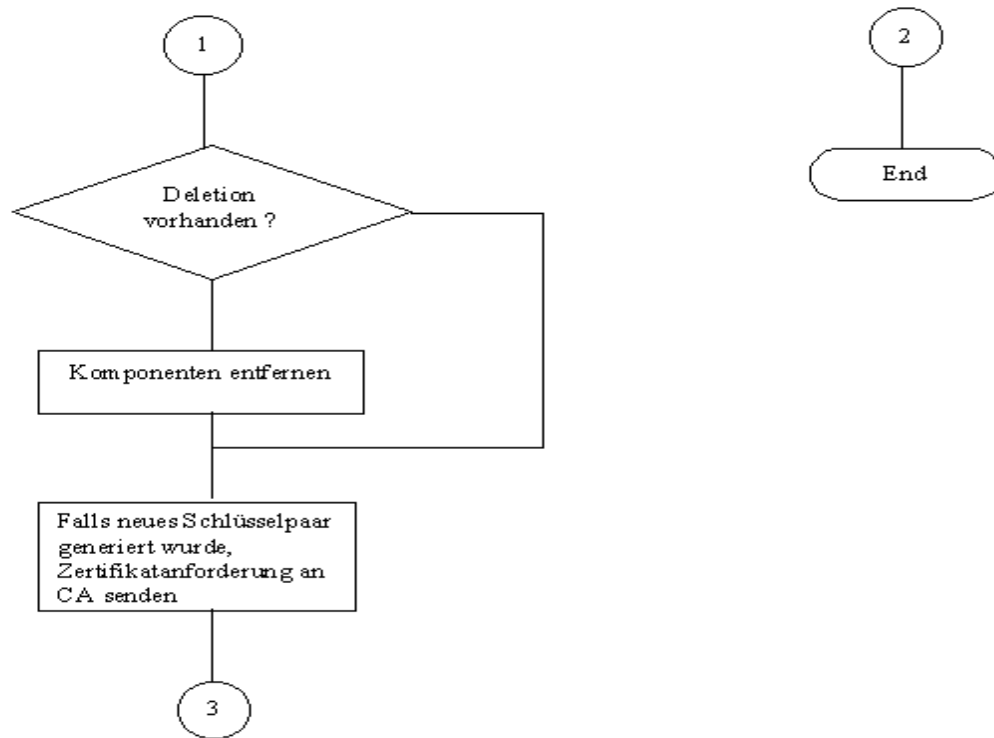


Abb. 12

Die folgenden Klassen implementieren update der kryptographischen Komponenten des Clients:
Update

- Verify
- ClientUpdate
- ICCUpdate
- CardUpdate
- RegistryImpl

Alle diese Klassen befinden sich in smime.ump.update Package.

Die Zusammenhang der Klassen ist aus der Abb. 13 zu entnehmen:

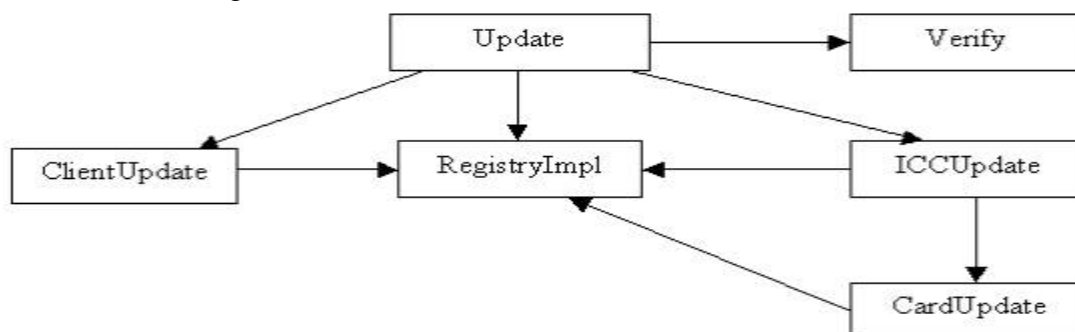


Abb. 13

5.1 Die Klasse Update.

Nachdem Outlook Plugin erkennt, dass die ankommende Mail Instanz der Klasse `UpdateComponent` enthält, wird diese Instanz zur Verarbeitung an die Klasse `Update` weitergeleitet. Die Instanz von `UpdateComponent` wird in dieser Klasse analysiert. Folgendes wird überprüft:

- Anzahl von Signaturen, die in `UpdateComponent` vorhanden sind. Es sind mindestens zwei Signaturen gefordert. Wenn diese Bedingung verletzt ist, wird der ganze Vorgang abgebrochen.
- An dieser Stelle ist auch gefordert, dass mindestens eine Signatur mathematisch korrekt ist und verifizierbar. Wenn dies nicht erfüllt ist, wird der Vorgang abgebrochen.
- Wenn in `UpdateComponent` die Identität von Empfänger spezifiziert ist, wird die eigene Identität mit der Identität in `UpdateComponent` verglichen. Nur bei Übereinstimmung wird der Austauschvorgang weitergeführt.
- Wenn in `UpdateComponent` die Card Identität spezifiziert ist, wird die Identitäten der eigenen Smart Cards mit der Identität in `UpdateComponent` verglichen. Nur bei Übereinstimmung wird Austauschvorgang weitergeführt.
- Es wird überprüft, ob ein Software PSE- relevantes Teil und (oder) ein Card – relevantes Teil in `UpdateComponent` spezifiziert ist. Wenn beide nicht vorhanden sind, wird der Vorgang beendet. Wenn ein Software PSE – relevantes Teil in `UpdateComponent` spezifiziert ist, wird eine Instanz der Klasse `ClientUpdate` gegründet und die aktuelle `UpdateComponent` Instanz an diese Instanz übergeben. Wenn ein Card – relevantes Teil spezifiziert ist, wird eine Instanz der Klasse `ICCUUpdate` gegründet und die aktuelle `UpdateComponent` Instanz an diese Instanz weitergeleitet.
- Falls die informative Mitteilung für Benutzer in `UpdateComponent` vom Trust Center angegeben wurde, wird diese in einem Dialog angezeigt. Der Austauschvorgang beginnt nur dann, wenn der Benutzer die Mitteilung positiv bestätigt.

Die Information über den gesamten Ablauf wird am Ende in einem Dialogfenster angezeigt.

5.2 Die Klasse RegistryImpl.

Diese Klasse spielt eine zentrale Rolle bei der Durchführung des Komponentenaustausches. Alle anderen Klassen benutzen diese Klasse, die alle wichtigen Konfigurationsinformationen enthält, verwaltet und außerdem die notwendigen Schnittstellen für die Zertifikatverwaltung zur Verfügung stellt.

Folgende Konfigurationsinformationen werden von dieser Klasse verwaltet:

- Pfad für die Konfigurationsdatei des Clients.

- Name der Konfigurationsdatei ist conf.ini und lässt sich nicht ändern.
- Anzahl der Signaturen in UpdateComponent, die mathematisch korrekt verifiziert werden müssen.
- Alle bekannten Namen des Trust Centers (kodiert in X501 Format) .
- Alle bekannten eigenen Namen (kodiert in X501 Format).
- Anzahl der bisher ausgeführten globalen Updates.
- Anzahl der bisher ausgeführten speziellen Updates für diesen Client.
- Alle Signatur- und Haschalgorithmus, die diesem Client bekannt sind.
- Lokalgespeicherte Instanz der globalen Tabelle mit Security Conditions²⁴.
- Pfad, wo die neue kryptographische Komponente installiert wird.
- Pfad des Homeverzeichnis des Clients.
- Pfade, die für Smart Card Software und Card Update relevant sind.
- Daten, die für Software PSE relevant sind, z.B. das benutzte Betriebssystem.
- Daten, die für Smart Card relevant sind, z.B. benutzte Karte (GemPlus etc).

Alle notwendigen Informationen werden aus Konfigurationsdateien während der Klasseninitialisierung ausgelesen. Außerdem sind in der Klasse Zugriffsmethoden für das Lesen dieser Informationen implementiert.

Die wichtigen Methoden in dieser Klasse sind:

- `getSecurityCondition` - diese Methode analysiert mit Hilfe der lokalgespeicherten Tabelle `RegistryASN1`²⁵ den Algorithmus, der entfernt werden sollte und die Signaturalgorithmen, mit denen UpdateComponent vom Trust Center signiert wurde. Diese Methode liefert `true` zurück, wenn die SecurityCondition, die in `RegistryASN1` spezifiziert wurde, erfüllt sind.
- `getProvider` - diese Methode liefert Namen aller JCA Provider zurück, die einen bestimmten Algorithmus enthalten.
- `getSignAlgOID` - diese Methode liefert den Signaturalgorithmus zurück, der vom entfernten Algorithmus unabhängig ist und der einen gemeinsamen Grundalgorithmus (z.B. RSA) mit neuinstalliertem Schlüsselpaar hat. Ein Beispiel dafür ist der Signaturalgorithmus `MD5withRSA`, wenn der entfernte Algorithmus `SHA1` heißt und das neuinstallierte Schlüsselpaar ein RSA – Schlüsselpaar ist.

²⁴ Siehe Abschnitt 2.5

²⁵ Siehe Abschnitt 2.6

- `getOtherAlgorithm` – diese Methode liefert den Signaturalgorithmus zurück, der vom entfernten Algorithmus mathematisch gesehen völlig unabhängig ist. Ein Beispiel dafür ist der Signaturalgorithmus `MD5withRSA`, wenn der entfernte Signaturalgorithmus `SHA1withECDSA` war.

Wie oben schon erwähnt wurde, implementiert diese Klasse auch die Schnittstelle zur Zertifikatverwaltung, indem man die Aufrufe von Methoden der Klasse `CertificateManager` kapselt. Dafür hält die Klasse `RegistryImp` eine Referenz auf die Klasse `CertificateManager`.

Die hier relevanten Methoden sind:

- `deleteCertificate` und `deleteCACertificate` – diese Methoden löschen entsprechend das Benutzerzertifikat oder das Trust Center-Zertifikat zu einem bestimmten Signaturalgorithmus.
- `createCHKeys` – diese Methode erzeugt ein neues Schlüsselpaar zu einem angegebenen Algorithmus, wie `RSA` oder `ECDSA`
- `addCertificate` – fügt ein neues Zertifikat zum PSE des Clients.
- `getPrivateKey` – gibt das geheime Teil des neugenerierten Schlüsselpaares zurück.
- `getPrivateKey` – überladene Methode. Liefert ein schon existierende geheimer Schlüssel aus Zertifikatverwaltung zu einem Signaturalgorithmus zurück.

Die Instanz der Klasse `RegistryImpl` wird in der Klasse `Update` erzeugt. Alle Klassen, die an dem Austauschvorgang teilnehmen, halten eine Referenz auf diese Instanz.

5.3 Die Klasse `ClientUpdate`.

Diese Klasse ist für Durchführung des Software PSE- Komponenteaustausches verantwortlich. Eine Instanz dieser Klasse wird nur dann in der Klasse `Update` erzeugt, wenn in der `UpdateComponent` Struktur die `ClientRelevant` Struktur nicht leer ist²⁶. Falls diese Bedingung erfüllt ist, wird eine Instanz der `ClientUpdate` Klasse erzeugt und eine aktuelle Instanz der Klasse `RegistryImpl` und `UpdateComponent` selbst an diese Instanz übergeben.

Wie oben schon erwähnt wurde, sind alle Änderungen, die in Software PSE des Clients passieren müssen, in der `ClientRelevant` Struktur spezifiziert. Gerade diese Struktur wird in dieser Klasse bearbeitet.

Als erstes wird der `UpdateComponent` Zähler aus der aktuellen `UpdateComponent` Struktur (falls spezifiziert), mit lokal gespeichertem Zähler verglichen. Der lokale Wert muss kleiner sein, als in der neuen `UpdateComponent`. Falls diese Bedingung verletzt ist, wird der Austauschvorgang für die Software PSE beendet. Andersfalls wird der neue Wert lokal dauerhaft gespeichert.

²⁶ Siehe Abschnitt 2.1

Wenn der Algorithmus, der zur Entfernung aus der PSE bestimmt ist, spezifiziert ist, werden als erste alle JCA Providers ermittelt, die diesen Algorithmus enthalten. Später werden diese Providers aus dem System entfernt.

Anschließend wird versucht, Trust Center-Zertifikat und (oder) Benutzerzertifikat zum angegebenen Algorithmus zu löschen, wenn ein entsprechender Eintrag in UpdateComponent gesetzt ist.

Nachdem alle spezifizierten Komponenten entfernt sind, wird das Installationsteil von UpdateComponent betrachtet, falls dieses dort vorhanden ist. Falls ein neuer Algorithmus installiert werden muss, wird das angegebene Link (URL) ausgewertet. In Abhängigkeit davon, welches Protokoll in dem Link spezifiziert wurde, wird entsprechender Mechanismus zum Runterladen benutzt. Zur Zeit sind zwei Möglichkeiten implementiert: HTTP und FTP Protokolle. Die heruntergeladene Datei enthält eine DER – kodierte CodeUpdateComponent ASN.1 Struktur. In Falle einer Java basierten Plattform werden in heruntergeladener Komponente Code für die neuen JCA Providers enthalten. Dieser Code wird in Datei rausgeschrieben und die neuen Providers werden dann beim JCA Framework angemeldet.

Wenn UpdateComponent das neue Trust Center Zertifikat für einen Signaturalgorithmus enthält, wird es in der Software PSE installiert.

Darüber hinaus kann in UpdateComponent eine Anforderung spezifiziert werden, ein neues Schlüsselpaar für ein Algorithmus, wie z.B. RSA oder DSA zu erzeugen. Der öffentliche Schlüssel bleibt dann beim Client und der Privatschlüssel wird mit UpdateComponentResponse²⁷ an das Trust Center zurückgeschickt. Die Bedingung dabei ist, dass UpdateComponentResponse mit multiplen Signaturen geschützt werden muss. Die Signaturalgorithmen, die dafür geeignet sind, werden hier aufgrund der vorhandenen Daten ermittelt.

5.4 Die Klasse ICCUpdate.

Diese Klasse ist für die Durchführung des Komponentenaustausches auf der Smart Card verantwortlich. Eine Instanz dieser Klasse wird nur dann in der Klasse Update erzeugt, wenn die ICCRelevant Teil in UpdateComponent gesetzt ist. Bei einer UpdateComponent, die für eine spezielle Smart Card bestimmt ist, wird auch überprüft, ob die Identität der Smart Card, die ein Client besitzt, mit der Identität in UpdateComponent übereinstimmt. Wenn diese Bedingungen erfüllt sind, wird eine Instanz der Klasse ICCUpdate erzeugt und die aktuelle Instanz der Klasse RegistryImpl und das Link (URL) an diese Instanz von ICCUpdate übergeben. Dieses Link ist nichts anderes als Inhalt der ICCRelevant ASN.1 Struktur.

Als Erstes wird die neue Kartenkomponente aus dem Internet heruntergeladen. Auch hier sind zwei Wege möglich: über HTTP oder FTP Protokolle. Inhalt der heruntergeladenen Datei besteht aus DER - kodierter ASN.1- Struktur CardUpdateComponent²⁸. Die dort enthaltenen Signaturen werden verifiziert. Wenn die Signaturen korrekt sind, werden die Informationen aus der CardUpdateComponent – Struktur an der Instanz der Klasse CardUpdateGemPlus übergeben. Diese Klasse implementiert den eigentlichen Austausch auf der speziellen Karte (siehe den nächsten Abschnitt).

²⁷ Siehe Abschnitt 2.1

²⁸ Siehe Abschnitt 2.1

5.5 Die Klasse *CardUpdateGemPlus*.

Diese Klasse implementiert den Komponentenaustausch auf der GemPlus Java Card. Als solche implementiert diese Klasse das Java Interface *CardUpdate*. Dieses Interface enthält drei Methoden:

initialize() – initialisiert das Kartensystem.

delete() – löscht die in *CardUpdateComponent* angegebenen alten Kartenkomponenten.

install() – installiert die neuen Kartenkomponenten, die in *CardUpdateComponent* spezifiziert sind.

GemPlus 2.4 Java Card Software ist völlig mit Java Card 2.1 und Open Platform 2.0 Standards kompatibel und benutzt Open Card Framework (OCF) für Kommunikation mit Kartenleser oder Kartenlesersimulator. Die Initialisierung des Kartensystems sieht dann folgendermaßen aus:

```
// start the OCF layer
SmartCard.start();
.....
// select the target specific CardTerminal
terminal = CardTerminalRegistry.getRegistry().cardTerminalForName(target);

// create a new card request object
CardRequest cr = new CardRequest();

// set a specific card terminal to the request
cr.setCardTerminal(terminal);
.....
// wait for card insertion
SmartCard sc = SmartCard.waitForCard(cr);
// get the OP/VOP specific CardService
serv = (CardServiceVOP211)
sc.getCardService(CardServiceVOP211.class, true);
.....
// service authentication object creation
authenticationInput = new VOPAuthenticationInput();
.....
// process mutual authentication
// initialize/update and external/authenticate are done
Result result = serv.openSecureChanel(authenticationInput);
.....
.....
```

Für die Implementierung von *delete* und *install* Methoden werden spezifische Klassen der GemPlus benutzt. Diese Methodenimplementierungen sind für verschiedene Karten unterschiedlich.

Für *delete()* Methode müssen in *CardUpdateComponent* Applet Bezeichnung (Englisch Applet Identifier oder kurz AID) und Packet Bezeichnung (Englisch Package Identifier oder kurz PID) angegeben werden. Dann werden die beiden Elemente von der Karte gelöscht.

Für *install()* Methode werden in *CarUpdateComponent* neben den Informationen über die neue Kartenkomponente wie der Name der Datei für die neue Komponente, ihre PID und AID, auch binäre Code der neuen Komponente enthalten. Dieser Code wird durch GemPlus Software geeignet interpretiert und blockweise auf die Karte geladen.

5.6 Die Klasse *Verify*.

Diese Klasse stellt die Methoden zur Verfügung, die Verifikation der Signaturen in UpdateComponent durchführen.

- `verifyUpdateComponentSign` Methode liefert logischen Wert „wahr“ (Englisch „true“) zurück, wenn die Anzahl von mathematisch korrekt verifizierten Signaturen größer oder gleich dem festgelegten in Konfigurationsdatei Wert ist. Üblicherweise ist dieser Wert auf zwei gesetzt.
- `verify` Methode liefert „true“ zurück, wenn jede einzelne Signatur mathematisch korrekt ist. Diese Methode wird in `verifyUpdateComponentSign()` Methode aufgerufen.

5.7 Andere Klassen.

Einige wichtige Klassen sind in Package `smime.ump.update.util` zusammengefasst.

Dazu gehören folgende Klassen:

- `Dialogs`
- `Stat`
- `UpdateInProgress`
- `Ret`
- `Resolver`

5.7.1 Die Klasse *Dialogs*.

Diese Klasse implementiert allgemeine graphische Dialoge, die von vielen anderen Klassen benutzt werden. Die wichtigen Methoden in dieser Klasse sind:

`infoDialog` – implementiert einen modalen Dialog, der dem Client bestimmte Informationen präsentiert.

`errorDialog` – implementiert einen modalen Dialog, der beim Auftreten von unerwarteten Ausnahmezustände aufgerufen wird.

`showSummeryInfo` – implementiert eine graphische Informationsdarstellung für Benutzer über einen durchgeführten Komponentenaustausch. Diese Methode wird zusammen mit der Instanz der Klasse `Stat` benutzt²⁹.

`certRequestDialog` – implementiert eine graphische Schnittstelle für Benutzer, der zwischen sofortigen Versendung der ASN.1 Struktur `UpdateComponentResponse` an das Trust Center oder lokalem Zwischenspeichern dieser Struktur in einer Datei wählen kann.

`fileNameDialog` – implementiert einen modalen Dialog, der eine lokale Datei auswählen lässt. Wird in Methode `certRequestDialog` benutzt.

²⁹ Siehe Abschnitt 5.7.2

Die Klasse `Dialogs` hängt eng mit der Klasse `Status` zusammen. In dieser Klasse wird z.B. der Zustand der gedrückten Tasten gespeichert. Dieser kann von anderen abgefragt werden. Damit wird der bekannte Model-View-Controller (MVC) Ansatz implementiert.

5.7.2 Die Klasse `Stat`.

Alle relevanten (wie z.B. Löschen oder Hinzufügen von Zertifikaten oder Providern) Handlungen, die im Laufe des Komponentenaustausches stattfinden, werden als eine Zeichenkette in einer Instanz dieser Klasse gespeichert. Am Ende des Austausches wird diese Instanz an die Methode `showSummeryInfo` der Klasse `Dialogs` übergeben. Diese Methode zeigt die relevanten Informationen an.

5.7.3 Die Klasse `UpdateInProgress`.

Diese Klasse dient zur Visualisierung des Austauschprozesses. Die Klasse ist als ein Thread mit niedrigster Priorität realisiert. Die Klassen `Update`, `ClientUpdate`, `ICCUpdate`, die an dem Komponentenaustausch teilnehmen, halten gleiche Referenz auf Instanz dieser Klasse. Die Informationen, die dem Benutzer angezeigt werden sollen, werden als Parameter an die Methode `outputTF` übergeben, die diese Informationen in einem separaten Fenster anzeigt.

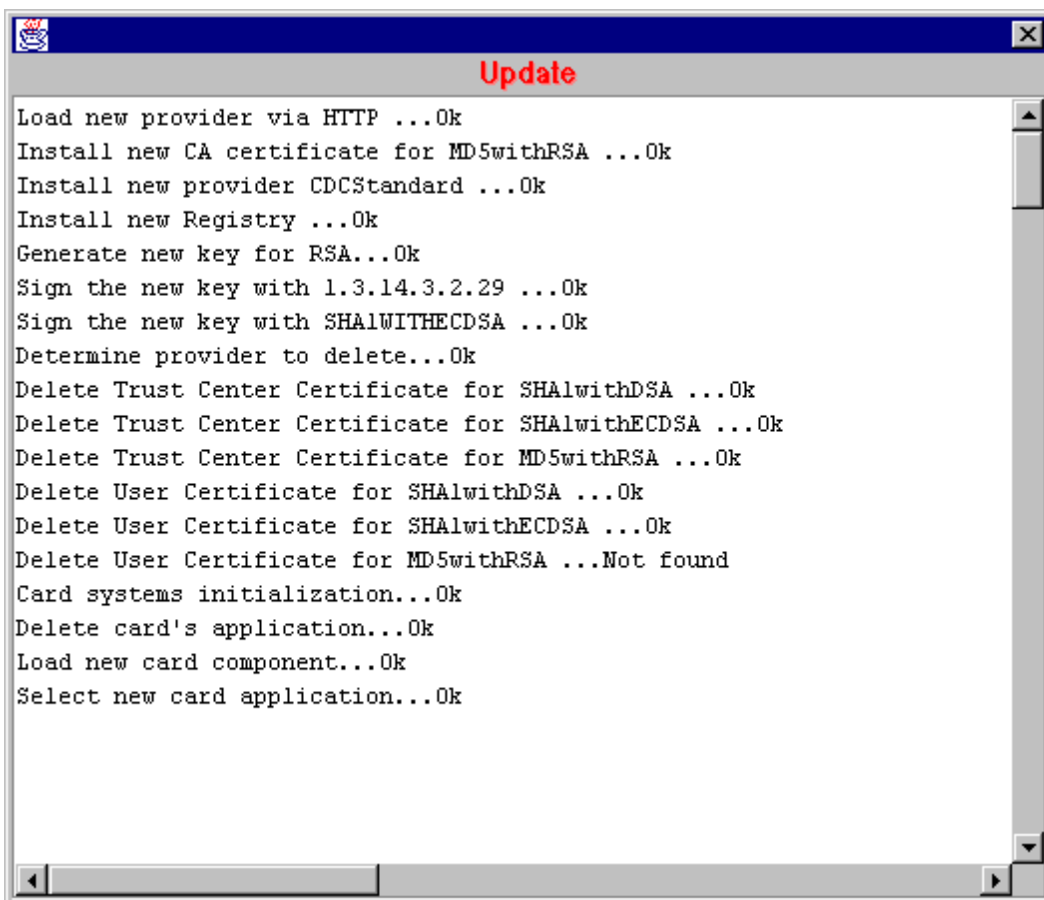


Abb. 14

5.7.4 Die Klasse Resolver.

Diese Klasse führt Umwandlung von Algorithmusnamen zu Algorithmus OID und umgekehrt. Damit stellt sie die gleiche Funktionalität wie die Klasse `codec.util.JCA` zur Verfügung. Allerdings funktioniert die letzte nur dann, wenn die JCA Provider mit entsprechenden Algorithmen installiert sind. Andersfalls werden Ausnahmen ausgelöst. Im Gegenteil, die Klasse Resolver funktioniert unabhängig von installierten Providern.

5.7.5 Transportklassen.

Wie oben schon erwähnt wurde, spezifiziert das Trust Center die Links (URL's) mit Internet Adressen, wo sich die neuen kryptographische Komponenten befinden. Der Transportmechanismus (oder einfach Internet Protokoll) ist aus Flexibilitätsgründen nicht weiter spezifiziert. Bei konkreter Implementierung des Komponentenaustausches, sind zwei Möglichkeit implementiert:

- HTTP und
- FTP Protokolle.

Die Klassen, die diese Aufgaben implementieren, befinden sich in Java – Package `smime.transport`. Zu den Klassen gehören:

- Interface Load
- Die Klasse LoadHTTP und
- Die Klasse LoadFTP

Die Interface `Load` enthält zwei Methoden:

- `setRequestParameter`, die Anfrageparameter geeignet für spezifische Protokolle vorbereitet. Z.B. für HTTP Anfrage werden die Parameter PSE und WinNT in dieser Methode zu Zeile `?source=PSE&os=WinNT` gewandelt und an das spezifizierte in UpdateComponent Link angehängt.
- `get` lädt die neue kryptographische Komponente über Internet.

Die beiden anderen Klassen sind die Implementierungen von dieser Interface für HTTP bzw. FTP Protokolle.

Für die aktive HTTP- Anbindung wurde außerdem ein Servlet entwickelt , das die Anfrageparameter auswerten kann und in Abhängigkeit davon eine neue passende Komponente auswählen kann.

Um Servlet ausführen zu können, braucht man einen servletfähigen Web – Server. Für diese Zwecke wurde Tomcat-Jakarta 3.2. Servletengine eingesetzt.

6 Zusätzliche Funktionalität der Zertifikatverwaltung.

Um die gestellte Aufgabe zu erfüllen, wurde es notwendig, die Implementierung von Zertifikatverwaltung zu ändern und an neue Anforderungen anzupassen.

Es sind folgende Änderungen vorgenommen.

In der Klasse `CertificateSearchCriteria` sind zwei neue Suchkriterien eingeführt:

- `algorithmName` gibt die Möglichkeit, ein Zertifikat nach Signaturalgorithmus Name zu suchen und
- `algorithmOID` gibt die Möglichkeit ein Zertifikat nach seinem Signaturalgorithmus Objekt Identifikator zu suchen

In Interface `CertificateManagerSpi` sind zwei zusätzliche Methoden definiert:

- `createKeyPair` erzeugt das Schlüsselpaar zu dem neuinstallierten Algorithmus
- `removeCertificate` entfernt Zertifikate mit definierten Kriterien.
- `isKeyPresent` liefert `true` zurück, wenn ein Schlüssel zu dem angegebenen Signaturalgorithmus existiert.

Die Methoden sind in der Klasse `CMTestProvider` implementiert.

7 Einbindung des UMP in CDC Plugin für Microsoft Outlook.

Wie bereits erwähnt wurde, bildete eine andere Diplomarbeit, die am Lehrstuhl von Prof. Buchmann in Januar 2001 von Dirk Schramm [SCHRAMM] abgeschlossen wurde, die Grundlage für diese Diplomarbeit. Um Komponentenaustausch mittels UpdateComponent realisieren zu können, wurde es nötig, einige Änderungen im Quellcode von CDC Erweiterung für Outlook 2000 vorzunehmen. Allerdings dank dem gewählten Format von UpdateComponent Struktur als Teil von PKCS#7 Standard, waren diese Änderungen nicht besonders umfangreich.

7.1 Änderungen in Java Teil.

Grundsätzlich ging es hier darum, dem Plugin die neue UpdateComponent Struktur bekannt zu machen. Dafür wurde der neue Typ für Mailinhalt (s.g. Content Type) eingeführt. Die Mails mit UpdateComponent können entweder vom Typ `application/x-pkcs7-ump` oder vom Typ `application/pkcs7-ump` sein. Diese Typdefinition findet sich in der Klasse `SMime`. In der gleichen Klasse werden in der Methode `registerDataContentHandlers` der zusätzliche `DataContentHandler` für diesen Typ registriert. Dieser `DataContentHandler` ist durch die Klasse `application_pkcs7_ump` in Java Package `smime.handlers` implementiert. Auch die Methoden `getContent`, `writeTo` und `decode` der Klasse `SMimePkcs7` wurden so geändert, dass sie der neue Typ erkennen und bearbeiten können.

Der eigentliche Aufruf von Austauschprozesses (d.h. Update Klasse) findet in der Klasse `SmimeReadMessageContext` statt. Wenn erkannt wird, dass die Mail eine Instanz der Klasse `UpdateComponent` beinhaltet, wird die Instanz der Update Klasse erzeugt. Die aktuelle Instanz von `UpdateComponent` wird dabei an Instanz von Update übergeben.

7.2 Änderungen in C++ Teil.

Die einzige Möglichkeit `UpdateComponent` in Outlook auszuführen, besteht über Menü `FlexiSMime`, und zwar über den Menüpunkt „UpdateComponent ausführen“. Um dies zu ermöglichen, wurde die Implementierung der Funktion `doCommand` der Schnittstelle `IexchExtCommands` entsprechend geändert und die zusätzliche Funktion `executeUpdateComponent` entwickelt. Außerdem wurden die Ressourcendateien entsprechend angepasst.

7.3 Probleme der Integrierung des Plugins und UMP.

Es ist leider nicht gelungen, die `UpdateComponent` direkt (d. h. durch Doppelklick auf die entsprechende Mail oder durch Drücken des Knopfes „Öffnen“ in der Werkzeugleiste) auszuführen. Die Ursache dafür ist, dass das UMP eine an sich nicht signierte Mail ist. Die interne Bearbeitung der Nachrichten in der Outlook basiert auf MAPI (Messaging Application Programming Interface) Nachrichten. 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 beispielweise den Typ „IPM.Note“ und eine signierte E-Mail den Typ „IPM.Note.SMime“. Der Plugin greift automatisch nur dann ein, wenn er die signierte oder verschlüsselte Nachricht

erkennt. Dann werden die Nachrichtenklassen in Plugininterne konvertiert (beispielweise „IPM.Note.SMime“ wird in „IPM.Note.FlexiSMime“ umgewandelt) und von Plugin bearbeitet. Alle anderen Nachrichten werden von Plugin nicht direkt bearbeitet. Da die UMP E-Mail von Nachrichtenklasse „IPM.Note“ ist, wird sie von Plugin nicht beachtet.

7.4 Kompabilität.

7.4.1 Anforderungen an Hardware.

Die beschriebene Erweiterung wurde auf verschiedenen Rechnern mit Intel Pentium Prozessor entwickelt und getestet. Da keine spezielle Funktionen der Hardware benutzt werden, sollte ein Betrieb mit jedem Pentium-kompatiblen Prozessor möglich sein. Für einen effizienten Betrieb ist jedoch eine Taktrate von 300 MHz, sowie ein Arbeitsspeicher von 64 MB zu empfehlen.

Als Kartenleser diene GCR410 mit entsprechender GemXPRESSO 211 IS Karte. Diese Karte enthält einen 8 Bit Prozessor 32 K von ROM, 32 K von EEPROM und 2 K RAM. Zusätzlich enthält diese Karte Java Card Virtual Machine (JCVM) , Java Card API und andere Komponente.

7.4.2 Betriebssysteme.

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

7.4.3 Java und Kartensoftware.

Die Grundlage für das Ausführen des Java-Codes bildet die Java Virtual Machine von Sun zusammen mit den dazu gehörenden Standardklassen, sowie verschiedenen zusätzlichen Paketen. Die folgenden Komponenten wurden benutzt:

Komponenten	Version
Sun JDK	1.2.2
JavaBeans Activation Framework	1.0.1
JavaMail	1.2

Als Kartensoftware wurde die GemXPRESSO RAD 2.4 der Firma GemPlus eingesetzt. Diese Software unterstützt komplett die Java Card 2.1 und Open Platform 2.0 Standards. Für die Kommunikation mit der Karte wird OpenCardFramework und Sun Kommunikation API's eingesetzt. Das ganze ist noch einmal in der folgender Tabelle zusammengefasst.

Komponenten	Version
GemPlus GemXPRESSO	2.4 oder 2.11
OpenCardFramework	1.1
Sun ComAPI	2.0

7.4.4 Geschwindigkeitstest.

Die Geschwindigkeit der Ausführung des UMP beim Client ist im allgemeinen schwer zu bestimmen, da das UMP selbst verschieden gestalten werden kann. Für den Test wurden folgender Fall ausgesucht:

- Lösche alle Signaturverfahren, die mit ECDSA zusammenhängen.
- Lösche Zertifikaten zum SHA1withECDSA Verfahren (CA und Benutzerzertifikat).
- Installiere neues Signaturverfahren auf Basis von RSA (CDCStandard Provider).
- Installiere neues CA Zertifikat zum MD5withRSA Verfahren.
- Erzeuge neues Schlüsselpaar zum RSA mit der Länge 1024 Bit.
- Installiere neue Registry.
- Lade neue Komponente auf die Karte.

Die Tests waren auf einem PC mit Pentium 2 Prozessor mit 500 MHz Taktfrequenz und 128 MB RAM unter Windows NT 4.0 durchgeführt.

Diese Tests haben ergeben, dass die Gesamtausführungszeit (ausgenommen Zeit für interaktive Benutzereingaben) zwischen 80 und 100 Sec. beträgt. Davon beträgt das Laden des neuen JCA Providers über Internet zwischen 25 und 30 Sec., Generieren des neuen Schlüsselpaares zwischen 40 und 50 Sec. und Laden der neuen Komponente auf die Karte zwischen 15 und 20 Sec.

7.4.5 Installation.

Die Installationsvorgänge für das Trust Center Administrator Tool und Microsoft Outlook Plugin mit zusätzlicher Austauschfunktionalität sind in entsprechenden Verzeichnissen in README Dateien schrittweise beschrieben. Dabei wurde vorausgesetzt, dass die oben beschriebenen Systemanforderungen erfüllt sind.

Schlusswort.

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. Zur Zeit funktioniert der Austausch mit der modifizierten Zertifikatsverwaltung, die für den Outlook Plugin entwickelt wurde und nur als Basis für eine Weiterentwicklung angesehen kann.

Das andere Problem betrifft das Funktionieren des Komponentenaustausches für die JCA Infrastruktur. Die JCA Provider, die in JCA die Implementierung der Algorithmen und Strukturen wie KeyStores und Zertifikaten übernehmen, sind nicht abgeschlossen, d.h. es gibt Abhängigkeiten zwischen Providern. Beispielsweise, benutzt CDCECProvider die SHA1 Hashfunktion aus einem anderen Provider wie SUN oder CDCStandard. Auch der Zufallsgenerator SHA1PRNG wird nicht von allen Providern implementiert. Dieses praktische Problem erschwert zur Zeit den Austausch, aber mit der Weiterentwicklung verschiedenen Provider (insbesondere CDCStandard und CDCEC) wird dieser Nachteil behoben.

Literaturverzeichnis.

- [CMS] Network Working Group: *Cryptographic Message Syntax*. RFC 2630, 1999.
- [FLEXIPKI] Technische Universität Darmstadt, Fachbereich Informatik, Fachgebiet Kryptographie, Computeralgebra, Lehrstuhl Prof. Dr. Buchmann: *FlexiPKI*. <http://www.informatik.tu-darmstadt.de/TI/Forschung/FlexiPKI/Welcome.html>
- [HaMa01] Michael Hartmann, Sönke Maseberg: Fail-Safe Konzept für Public-Key-Infrastrukturen.
- [JAF] JavaBeans Activation Framework Documentation, <http://java.sun.com/products/javabeans/glasgow/jaf.html>
- [JAVAMAIL] JavaMail API Documentation, <http://java.sun.com/products/javamail/>
- [JDK] Java 2 SDK Documentation, Version 1.2.2, <http://java.sun.com/j2se/>
- [PKCS7] RSA Laboratories: Public-Key Cryptography Standards #7: Cryptographic Message Syntax Standard. Version 1.5. 1993. <http://www.rsalabs.com/pkcs/pkcs-7>
- [SCHRAMM] Entwicklung einer flexibler Java-basierter S/MIME Erweiterung für Microsoft Outlook, Diplomarbeit TUD, Fachbereich Informatik, Fachgebiet Theoretische Informatik, Januar 2001
- [S/MIME] IETF Working Group: *S/MIME Mail Security (smime)*. <http://www.ietf.org/html.charters/smime-charter.html>
- [TLS] Network Working Group: *The TLS Protocol - Version 1.0*. RFC 2246, 1999.
- [X509] RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile.