Thomas Hackbarth

Entwurf und Implementierung einer Zertifizierungsstelle unter Berücksichtigung des Stellvertreterprobems

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Technische Informatik

am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Ing. Martin Hübner
Zweitgutachter : Prof. Dr.rer.nat. Gunter Klemke

Abgegeben am 19. Februar 2004

Thomas Hackbarth

Thema der Diplomarbeit

Entwurf und Implementierung einer Zertifizierungsstelle unter Berücksichtigung des Stellvertreterproblems

Stichworte

Kryptographie, Zertifikat, PKI, Stellvertreter, Java

Kurzzusammenfassung

In dieser Diplomarbeit geht es um die Lösung des Stellvertreterproblems in einer Public-Key-Infrastruktur. Das Stellvertreterproblem entsteht, sobald ein Zertifikatsinhaber für eine bestimmte Zeit nicht verfügbar ist und für diese Zeit ein Stellvertreter einsetzt. Eine verschlüsselte Nachricht kann von dem Stellvertreter nicht gelesen werden, da dieser nicht über die privaten Schlüssel der zu vertretenden Person verfügt. Das von mir entwickelte System löst dieses Problem, indem es für die Stellvertreter eigene Zertifikate erstellt, in denen neben dem zu zertifizierenden Schlüssel auch Informationen über die zu vertretende Person gespeichert sind. Bei der Verschlüsselung einer Nachricht werden die Stellvertreterzertifikate, die den eigentlichen Empfänger vertreten, gesucht. Werden solche Zertifikate gefunden, wird die Nachricht mit dem Schlüssel aus dem Zertifikat zusätzlich verschlüsselt.

Für dieses System wurde ebenfalls eine Implementierung erstellt, die in der Sprache Java mit Hilfe der Technik RMI programmiert wurde.

Thomas Hackbarth

Title of the paper

concept and implementation of a certification authority in consideration of the deputy problem

Keywords

cryptography, certificate, PKI, deputy, java

Abstract

This diploma thesis deals with the solution of the deputy problem in a public key infrastructure. The deputy problem develops, as soon as a certificate owner is not available for a certain time and appoints for this time a deputy. A coded message cannot be read by the deputy because he doesn't have the private keys of the person who is represented. The system developed by me solves this problem in providing own certificates for the deputies, in which beside the key also information about the person, who is represented, are stored. During the coding of a message it is looked for a deputy certificate, which represents the actual receiver. If such certificates are found, the message will be coded additionally with the key from the certificate.

For this system an implementation was likewise made available, which was provided in the language Java with the help of the technology by RMI.

Inhaltsverzeichnis

1 Einleitung 1
1.1 Motivation 1
1.2 Ziel der Arbeit 1
1.2.1 Aufgabenstellung der fiktiven Firma 1
1.3 Aufbau der Arbeit 2
2 Grundlagen der Kryptographie 3
2.1 Was ist Kryptographie? 3
2.2 Symmetrische Verfahren 3
2.3 Moderne Formen von symmetrischen Verfahren 3
2.3.1 DES 3
2.3.2 IDEA 4
2.3.3 AES 4
2.3.4 Nachteile von symmetrischen Verfahren 4
2.4 Asymmetrische Verfahren 5
2.5 Moderne Formen von asymmetrischen Verfahren 5
2.5.1 Diffie-Hellmann 5
2.5.2 RSA 6
2.5.3 Nachteile von asymmetrischen Verfahren 7
2.6 Kombination der beiden Verfahren (Hybridverfahren) 7
2.7 Signaturen 8
2.7.1 Hashfunktionen 8
2.8 Umgang mit öffentlichen Schlüsseln 9
2.8.1 Direct Trust 9
2.8.2 Web of Trust 9
2.8.3 Hierarchical Trust 9
2.9 Varianten des Modells „Hierachical Trust“ 10
2.9.1 Verwendung mehrerer CAs 10
2.10 Trennung von Schlüsseln 10
3 PKI-Anwendungen und Produkte 12
3.1 Standards 12
3.2 Wichtige Komponenten einer PKI 12
3.3 Ablauf einer Zertifikaterstellung 13
3.4 Verfügbare Produkte 13
3.5 Berücksichtigung des Stellvertreterproblems 14
4 Lösungsansätze für das Stellvertreterproblem 15
4.1 Erweiterung des Key-Escrow-Verfahrens 15
4.1.1 Formale Darstellung 16
4.2 Temporäre Sperrung von Empfängerzertifikaten 16
4.2.1 Formale Darstellung 17
4.3 Erstellung von Stellvertreterzertifikaten 18
4.3.1 Besonderheiten des Stellvertreterzertifikats 19
4.3.2 Formale Darstellung 20

5 Detailentwurf einer Lösung 21
5.1 Wahl einer Lösung 21
5.2 Das Enrollment 22
5.3 Die Schlüsselerzeugung 22
5.4 Zertifikatsantrag 23
5.5 Zertifizierungsstelle 23
5.5.1 Verwendete Standards 24
5.6 Der Zertifikatsserver 25
5.7 Der Verschlüsselungsclient 26
5.8 TimeStampserver 27
5.9 Ablauf einer Verschlüsselung mit Stellvertreterzertifikat 27
6 Implementierung des Detailentwurfs 29
6.1 Sprache der Implementierung 29
6.1.1 Historie von Java 29
6.1.2 Eigenschaften von Java 29
6.1.3 Remote Method Invocation (RMI) 30
6.1.4 zusätzlich verwendete Bibliotheken 31
6.2 Datenstrukturen 31
6.2.1 Die Speicherung der Schlüssel 31
6.2.2 Speicherung der verschlüsselten Nachrichten 32
6.2.3 Speicherung der Stellvertreterregelung 33
6.3 Kommunikation 34
6.4 Implementierung der einzelnen Module 36
6.4.1 Schlüsselerzeugung 36
6.4.2 Der Zertifikatsantrag 39
6.4.3 Die Zertifizierungsstelle 41
6.4.4 Der Zertifikatsserver 43
6.4.5 Der TimeStampServer 45
6.4.6 Der Ver-/Entschlüsselungsclient 46
7 Zusammenfassung und Ausblick 53
7.1 Zusammenfassung 53
7.2 Ausblick 53
8 Glossar und Literaturverzeichnis 54
8.1 Glossar 54
8.2 Literaturquellen 56
Anhang 58

Abbildungsverzeichnis

Abbildung 1 : Prinzip einer Verschlüsselung ... 3
Abbildung 2 : Probleme des Schlüsselaustauschs ... 5
Abbildung 3 : Aufbau einer Hybrid-Verschlüsselung ... 7
Abbildung 4 : Ablauf einer Entschlüsselung mit erweitertem Key-Escrow Verfahren ... 15
Abbildung 5 : Stellvertreterregelung mit temporärer Sperrung ... 17
Abbildung 6 : Stellvertreterlösung mit Stellvertreterzertifikaten ... 19
Abbildung 7 : Darstellung einer OID-Struktur ... 25
Abbildung 8 : Schichtenmodell von RMI ... 30
Abbildung 9 : UML-Darstellung der AsymKey-Klassen ... 32
Abbildung 10 : UML-Darstellung der Klassen zur Speicherung einer Nachricht ... 33
Abbildung 11 : UML-Darstellung der Stellvertreter-Klasse ... 34
Abbildung 12 : Übersicht über RMI Kommunikationen ... 34
Abbildung 13 : Screenshot der Schlüsselerstellung ... 36
Abbildung 14 : Passworteingabe bei der Schlüsselerstellung ... 37
Abbildung 15 : Screenshot der Applikation zum Stellen von Zertifikatsanträgen ... 39
Abbildung 16 : Eingabe eines Passwortes, um einen privaten Schlüssel zu entschlüsseln ... 40
Abbildung 17 : Die Applikation der Zertifizierungsstelle ... 41
Abbildung 18 : Screenshot des Zertifikatsservers ... 44
Abbildung 19 : Screenshot des TimeStampServers ... 45
Abbildung 20 : Screenshot der Verschlüsselungskomponente des Clients ... 46
Abbildung 21 : Dialog zum Suchen und Laden von Zertifikaten ... 47
Abbildung 22 : Screenshot der Entschlüsselungskomponente des Clients ... 51

1 Einleitung

1.1 Motivation

Durch das rasante Wachstum des Internets in den letzten Jahren und die starke kommerzielle Nutzung dieses Mediums ist es essentiell geworden die durch diesen unsicheren Kanal transportierten Daten zu schützen. Dies gilt insbesondere für firmeninterne und sicherheitskritische Daten, wie Passwörter und Betriebsgeheimnisse. Das Internet wirft zusätzlich eine Reihe von weiteren Problemen auf, wie zum Beispiel die Authentifizierung[1] und Überprüfung der Integrität der Daten. Diese Probleme können mit Sicherheitsmechanismen der Kryptographie gelöst werden. Die Sicherung der Daten geschieht durch eine Verschlüsselung, die so aufgebaut ist, dass diese nur vom Empfänger der Nachricht wieder entschlüsselt werden kann. Sollte der Empfänger krank oder im Urlaub sein, ist die gesicherte Nachricht bis zur Wiederkehr des eigentlichen Empfängers nicht zu entschlüsseln. Dieses Problem, auch „Stellvertreterproblem“ genannt, wird in der Praxis häufig durch die Übergabe von Passwörtern an die Kollegen oder ähnlichem gelöst. Eine Lösung durch die Informatik wird bisher kaum angeboten.

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist die Konzeption und Realisierung eines Systems, mit dem Nachrichten gesichert über ein unsicheres Medium verschickt werden können. Hierbei dürfen die Nachrichten nur vom eigentlichen Empfänger und von einem möglichen Stellvertreter gelesen werden. Das Besondere an dieser Lösung ist, dass der Stellvertreter nicht über Passwörter oder ähnliches vom zu vertretenden Benutzer besitzen muss, da das System die Nachricht für den Stellvertreter automatisch lesbar macht. Dieses System wird exemplarisch für eine fiktive Firma entwickelt. Das System ist nur in soweit realisiert, dass Nachrichten verschlüsselt und auch für Stellvertreter lesbar sind. Eine Versendung bzw. den Empfang der Nachricht wird nicht realisiert.

1.2.1 Aufgabenstellung der fiktiven Firma

Eine fiktive Firma hat einen zweiten Standort erschlossen. Vor der Erweiterung hat diese Firma innerhalb ihres ersten Standortes Firmenangelegenheiten in ihrem lokalen Netz per E-Mail verschickt. Sicherheitskritische Informationen, wie Passwörter und ähnliches, wurden nur im 4-Augen Gespräch ausgetauscht. Nun sollen die Firmenangelegenheiten und sicherheitskritische Daten auch mit dem zweiten Standort über das unsichere Medium Internet ausgetauscht werden. Folgende Anforderungen sind an die Lösung gestellt:

- Der Inhalt der Nachricht darf nicht von Außenstehenden oder nicht an der Kommunikation beteiligten Mitarbeitern gelesen werden.

[1] **Authentifizierung:** Während der Authentifizierung wird die Identität eines Benutzers verifiziert. Dies kann unter anderem durch Eingabe eines Passworts, Übertragung von Zertifikaten oder Verwendung von Chip-/Magnetkarten geschehen.

- Die Identität des Absenders muss zweifelsfrei nachweisbar sein.
- Die Nachricht muss vor Manipulationen von Außenstehenden oder nicht an der Kommunikation beteiligten Mitarbeitern geschützt sein.
- Es muss möglich sein einen Stellvertreter zu bestimmen, der die Nachrichten für einen abwesenden Empfänger lesen und beantworten kann.
- Aus der Antwort eines Stellvertreters muss ersichtlich sein, dass dieser wirklich als Stellvertreter eingesetzt wurde.
- Die Stellvertreterregelung muss von einer vertrauenswürdigen Stelle eingesetzt werden.
- Ein Sender muss die Möglichkeit haben, die Stellvertreterlösung beim Verschlüsseln abzuschalten, damit nicht persönliche Daten oder zum Beispiel Beschwerden in die falschen Hände fallen.
- Die Stellvertreterregelung darf nur in dem Zeitraum gültig sein, den die zu stellvertretende Person bestimmt hat.
- An die Implementierung sind einige Anforderungen gestellt. So muss das System möglichst einfach für den Benutzer zu handhaben sein. Auch gelten allgemeine Anforderungen: So sollte die Implementierung vollständig und fehlerfrei sein.

1.3 Aufbau der Arbeit

Diese Arbeit teilt sich in acht Kapitel. Nach dieser Einleitung werden in Kapitel 2 die für diese Diplomarbeit notwendigen Grundlagen der Kryptographie vermittelt. In Kapitel 3 wird dann kurz auf die Produkte eingegangen, die zur Zeit zu diesem Thema auf dem Markt sind. Das 4. Kapitel beschäftigt sich mit der Vorstellung verschiedener Konzepte, die ich zur Lösung des Stellvertreterproblems erstellt habe. Nach einem Vergleich der Konzepte wird im 5. Kapitel das am besten zu den Anforderungen der fiktiven Firma passende Konzept im Detail vorgestellt. In Kapitel 6 folgt dann die Beschreibung der Implementierung des Detailkonzeptes mit der Programmiersprache JAVA. Nach der Implementierung werde ich im 7. Kapitel noch eine kleine Zusammenfassung und einen Ausblick zu den Konzepten und der Implementierung liefern. Als Abschluss finden Sie in Kapitel 8 ein Glossar zu den wichtigsten Begriffen und ein Literaturverzeichnis. An das Kapitel 8 schließt sich ein Anhang an, der ein Inhaltsverzeichnis zu der, zur Arbeit beigelegten, CD-ROM enthält.

2 Grundlagen der Kryptographie

2.1 Was ist Kryptographie?

Unter Kryptographie versteht man die mathematischen Verfahren, um Daten geheim zu halten und vor Veränderung zu schützen. Dies geschieht mit Hilfe von Geheimnissen, den sogenannten Schlüsseln. Das Prinzip einer Sicherung mit Schlüsseln (Verschlüsselung) sieht dann so aus:

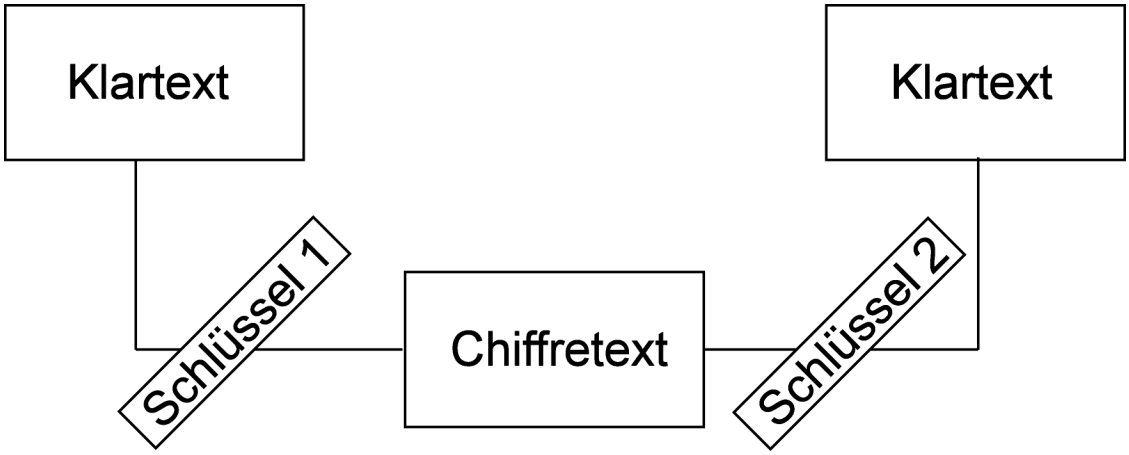

Abbildung 1 : Prinzip einer Verschlüsselung

Ein Klartext wird mit dem ersten Schlüssel und einem bestimmten Verfahren verarbeitet. Dadurch entsteht ein sogenannter „Chiffretext“, der keine Informationen oder Rückschlüsse auf den Klartext zulässt. Dieser Text kann nun durch die Verwendung eines zweiten Schlüssels wieder in den originalen Klartext verwandelt werden.

2.2 Symmetrische Verfahren

Die einfachste und älteste Form der Verschlüsselung ist die symmetrische Verschlüsselung. Bei der symmetrischen Verschlüsselung sind die beiden verwendeten Schlüssel gleich. Die ersten Verschlüsselungen mit diesem Prinzip sind bereits seit den Zeiten von Julius Cäsar bekannt. Ein einfaches Beispiel für eine symmetrische Verschlüsselung sind Substitutionsalgorithmen, die nach festgelegten Regeln Zeichen im Klartext vertauschen.

2.3 Moderne Formen von symmetrischen Verfahren

2.3.1 DES

Das bekannteste aller modernen symmetrischen Verfahren ist der DES. Der „Data Encryption Standard“ (DES) stellt seit den 70er Jahren einen weltweiten Standard dar. Der DES-Algorithmus zeigt zwar schon einige Alterserscheinungen, bietet aber immer noch Schutz vor den meisten Angreifern. DES wurde im Auftrag der amerikanischen Standarisierungsbehörde (NIST früher NBS) von IBM entwickelt. Im Gegensatz zu den vor DES entwickelten Algorithmen kann der DES nur mit dem Computer sinnvoll eingesetzt werden, da er

komplizierter ist als die oben erwähnten Substitutionsalgorithmen. DES kann als Mutter aller modernen symmetrischen Verfahren gesehen werden.

2.3.2 IDEA

Als Alternative zu DES entstand Anfang der 90er Jahre in Zürich der IDEA (International Data Encryption Algorithmen). Dieser Algorithmus arbeitet in einer Softwareimplementierung fast doppelt so schnell wie der DES. Gleichzeitig arbeitet IDEA mit einem 128-Bit Schlüssel, was das „Knacken“ eines Schlüssels mit purem Ausprobieren sehr schwer macht. Obwohl IDEA einer der sichersten Algorithmen ist, hat er auch einen großen Nachteil: IDEA ist patentiert, so dass man für kommerzielle Anwendungen Lizenzgebühren zahlen muss. Für nicht kommerzielle Zwecke kann der IDEA auch ohne Lizenz verwendet werden. Eingesetzt wird der Algorithmus zum Beispiel in dem weit verbreiteten Softwarepaket PGP.

2.3.3 AES

Neben IDEA gibt es heute auch noch den „Advanced Encryption Standard” (AES). Dieses Verfahren wurde als Nachfolger des DES bestimmt. Ermittelt wurde das Verfahren mit Hilfe eines Wettbewerbes, der 1997 begann und an dem viele bekannte Firmen und Personen aus dem Bereich Kryptographie teilgenommen haben. Unter anderem IBM, die schon an dem DES Verfahren mitgearbeitet hatten. Gewonnen hat diesen Wettbewerb der Algorithmus „Rijndael“ (gesprochen „Reindahl“) zweier belgischer Kryptographen. Rijndael und AES sind also zwei Bezeichnungen für ein und den selben Algorithmus. Dieser Algorithmus erfüllt alle Anforderungen, die in dem Wettbewerb gefordert wurden. So ist Rijndael gleichermaßen für Hard- und Softwareimplementierungen geeignet. Auch verwendet Rijndael deutlich größere Schlüssel als der DES, nämlich Schlüssel bis zu einer Länge von 256 Bit. Eine sehr wichtige geforderte Eigenschaft erfüllt Rijndael auch. Es ist vollkommen patentfrei und kann dadurch von jedem verwendet werden.

2.3.4 Nachteile von symmetrischen Verfahren

Obwohl alle beschriebenen symmetrischen Verfahren grundsätzlich sicher sind, haben alle diese Verfahren einen großen Nachteil. Es wird immer ein geheimer Schlüssel gebraucht, der dem Verschlüsselnden und dem Entschlüsselnden bekannt sein muss. Möchte man nun verschlüsselte Nachrichten mit verschiedenen Personen austauschen, entsteht schnell eine gewaltige Anzahl von zu verwaltenden Schlüsseln.

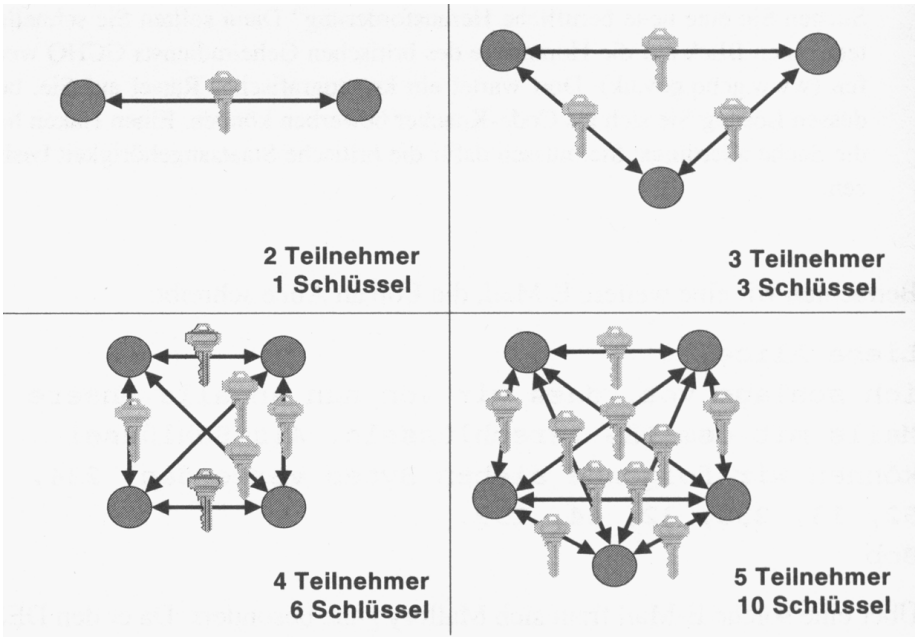

Abbildung 2 : Probleme des Schlüsselaustauschs
Grafik aus: [Schmeh01]

2.4 Asymmetrische Verfahren

Um die Probleme des Schlüsselaustausches zu beheben wurden die asymmetrischen Verschlüsselungsverfahren entwickelt. Bei diesen Verfahren wird kein gemeinsamer geheimer Schlüssel eingesetzt, sondern jeder Kommunikationspartner bekommt ein Schlüsselpaar, das sich in öffentlichen und privaten Schlüssel teilt. Diese Verschlüsselung mit einem asymmetrischen Verfahren läuft eigentlich genauso ab wie bei einem symmetrischen Verfahren. Beim Verschlüsseln wird nun nicht der gemeinsam vereinbarte Schlüssel benutzt, sondern der öffentliche Schlüssel des Empfängers. Der öffentliche Schlüssel ist, wie sein Name schon sagt, öffentlich. Das heißt, dass dieser Schlüssel ohne Bedenken verbreitet werden kann. Eine Nachricht, die mit einem öffentlichen Schlüssel verschlüsselt ist, kann nur mit dem privaten Schlüssel wieder entschlüsselt werden. Eine Entschlüsselung mit dem öffentlichen Schlüssel funktioniert nicht.

2.5 Moderne Formen von asymmetrischen Verfahren

2.5.1 Diffie-Hellmann

Das Verfahren von Diffie und Hellmann war eines der ersten Verfahren zum Schlüsselaustausch. Es ist kein Verschlüsselungsverfahren, aber die mathematischen Theorien dienten allen anderen Verfahren als Grundlage.

Für solche Verfahren werden sogenannte Falltürfunktionen benutzt. Bei diesen Falltürfunktionen handelt es sich um verbesserte Einwegfunktionen. Eine Einwegfunktion ist eine Funktion, die einfach auszuführen, aber schwer – praktisch unmöglich – zu invertieren ist. Bei den Falltürfunktionen gibt es eine Zusatzinformation, die es möglich macht, die eigentlich unmöglich zu lösende Umkehrfunktion zu berechnen. Diese Zusatzinformation

bildet dann den privaten Schlüssel, während die Falltürfunktion den öffentlichen Schlüssel bildet.
Im Fall des Diffie-Hellmann-Verfahrens wird als Falltürfunktion die diskreten Logarithmen verwendet. Ein Ablauf eines Schlüsselaustauschs könnte dann so aussehen:
Alice und Bob wollen einen geheimen Schlüssel austauschen. Dazu vereinbaren beide eine Primzahl p und eine natürliche Zahl g, die kleiner ist als p. Diese beide Informationen bilden den öffentlichen Schlüssel. Als privaten Schlüssel denken sich Alice und Bob jeweils eine natürliche Zahl aus, die kleiner ist als p. Alice denkt sich x aus und Bob y. Nun geschieht folgendes:

- Alice berechnet die Zahl $a = g^x \bmod p$ und schickt diese an Bob
- Bob berechnet die Zahl $b = g^y \bmod p$ und schickt b an Alice
- Alice kann nun folgendes berechnen : $k1 = b^x \bmod p$

Bob berechnet dies : $k2 = a^y \bmod p$
Die Formeln für k1 und k2 können nun wie folgt umgeformt werden:

- $k1 = b^x \bmod p = (g^y \bmod p)^x \bmod p = g^{yx} \bmod p$
- $k2 = a^y \bmod p = (g^x \bmod p)^y \bmod p = g^{xy} \bmod p$

Diese Formeln zeigen, dass die Werte k1 und k2 gleich sind, so dass Alice und Bob nun mit dem selben Schlüssel arbeiten, ohne dass dieser Schlüssel ungesichert versandt wurde.

2.5.2 RSA

Der RSA-Algorithmus wurde von Ron Riverst, Adi Shamir und Leonard Adleman entwickelt. Dieses Verfahren ist das erste wirkliche asymmetrische Verschlüsselungsverfahren. Es beruht wie der Diffie-Hellmann-Algorithmus auf einer Falltürfunktion. Es wurde beim RSA aber nicht der diskrete Logarithmus, sondern die Faktorisierung von großen Zahlen eingesetzt.
Ich möchte hier kurz einige mathematische Hintergründe präsentieren, um die Funktionsweise vom RSA-Algorithmus zu erklären. Die Schlüsselerstellung beginnt mit der Erstellung zweier großer Primzahlen. In der Fachliteratur werden diese Zahlen oft „p“ und „q“ genannt. Aus diesen beiden Zahlen wird das Produkt gebildet, das auch Modul „n“ genannt wird. Als nächstes wird eine Zahl „e“ gewählt, die kleiner als „n“ sein muss. Außerdem dürfen „e“ und das Produkt (p-1)*(q-1) keinen gemeinsamen Teiler außer 1 haben. Es muss jetzt nur noch eine weitere Zahl „d“ erzeugt werden, bei der gelten muss, dass (e*d-1) ohne Rest durch (p-1)*(q-1) teilbar ist.
Aus diesen Zahlen bildet sich das Schlüsselpaar. Der öffentliche Schlüssel besteht aus den Zahlen „n“ und „e“, während sich der private Schlüssel aus den Zahlen „n“ und „d“ zusammensetzt. Die Verschlüsselung berechnet sich dann mit der Formel $c=m^e \bmod n$, wobei „m“ die zu verschlüsselnde Nachricht und „c“ die verschlüsselte Nachricht ist. Der Empfänger kann nun wieder aus „c“ und seinem privaten Schlüssel die Originalnachricht „m“ ermitteln. Dazu wird die oben erwähnte Formel zur Entschlüsselung umgewandelt und lautet dann $m=c^d \bmod n$.

Lange musste die Verwendung des Algorithmus lizenziert werden, da der Algorithmus patentiert war. Doch im Jahre 2000 lief das Patent nach 17 Jahren aus. Das RSA-Verfahren hat mittlerweile den Status eines Verschlüsselungsstandards erreicht.

2.5.3 Nachteile von asymmetrischen Verfahren

Die asymmetrischen Verfahren beheben die Nachteile der symmetrischen Verfahren beim Schlüsselaustausch. Leider haben diese Verfahren auch einige Nachteile. Auf Grund der mathematischen Hintergründe braucht man, um die selbe Sicherheit der symmetrischen Verfahren sicherzustellen, einen deutlich längeren Schlüssel. Bei symmetrischen Verfahren reicht heutzutage eine Schlüssellänge von 256-Bit vollkommen aus. Bei asymmetrischen Verfahren sollte die Länge des Schlüssels aber mindestens 1024 Bit betragen. Dadurch verlängert sich auch die Rechenzeit der einzelnen Verarbeitungsschritte, so dass diese Algorithmen auch deutlich langsamer sind als die symmetrischen Verfahren.

2.6 Kombination der beiden Verfahren (Hybridverfahren)

Beide Verschlüsselungsverfahren habe deutliche Nachteile. Die symmetrischen Verschlüsselungen sind schnell. Aber es gibt Probleme beim Schlüsselaustausch. Die asymmetrischen Verfahren sind langsam, lösen aber das Schlüsseltauschproblem. Um die Vorteile beider Verfahren nutzen zu können, wurde das Hybridverfahren entwickelt. Der Ablauf dieses Verfahrens sieht wie folgt aus:

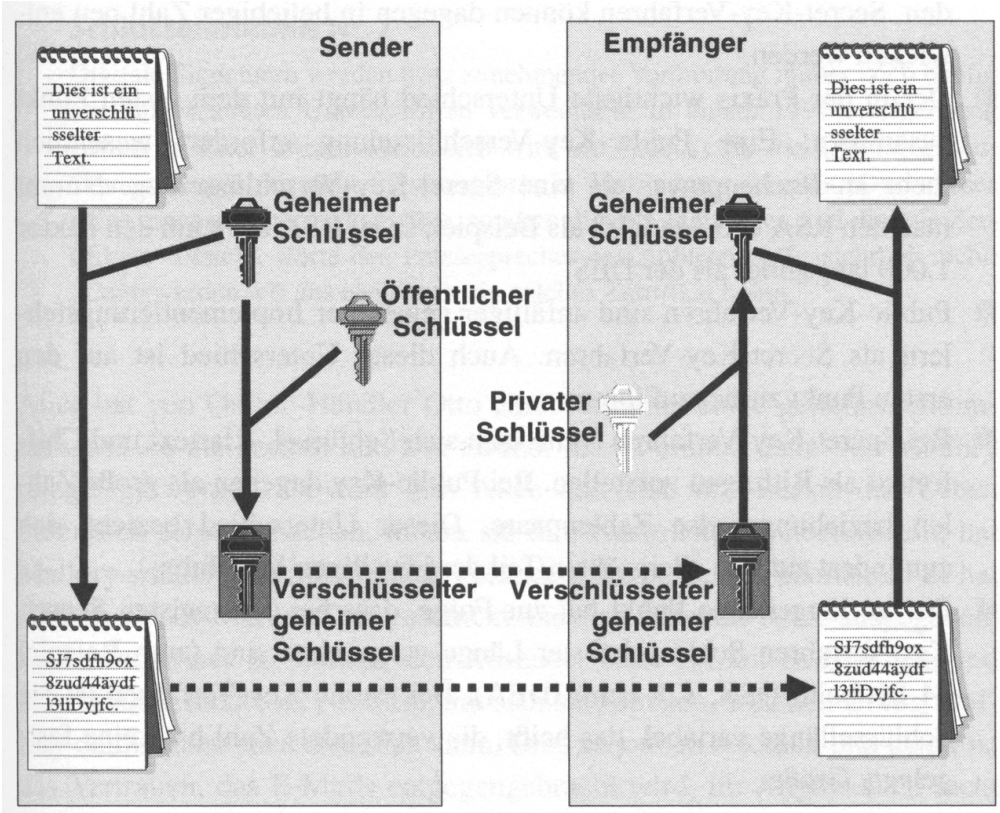

Abbildung 3 : Aufbau einer Hybrid-Verschlüsselung
Grafik aus: [Schmeh01]

Die zu verschlüsselnde Nachricht wird mit einem symmetrischen Verfahren unter zu Hilfenahme eines geheimen zufälligen Schlüssel verschlüsselt. Um dem Empfänger den geheimen Schlüssel mitzuteilen, wird dieser mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Nun wird die verschlüsselte Nachricht und der verschlüsselte Schlüssel an den Empfänger geschickt. Dieser entpackt mit seinem privaten Schlüssel den geheimen Schlüssel und kann damit die verschlüsselte Nachricht lesen.
Praktisch die gesamte im Internet verwendete Kryptographie-Software arbeitet mit dem Hybridverfahren.

2.7 Signaturen

Alle bis jetzt vorgestellten Verfahren stellen sicher, dass eine Nachricht nur vom eigentlichen Empfänger gelesen werden kann. Aber keines dieser Verfahren kann gewährleisten, dass die Nachricht von dem angegebenen Sender stammt. Dafür existieren die sogenannten Signaturen, die mit Hilfe von asymmetrischen Verschlüsselungsverfahren erstellt werden können.
Zur Erstellung einer Signatur wird die zu signierende Nachricht mit dem privaten Schlüssel des Senders verschlüsselt und zusätzlich zur eigentlichen Nachricht an den Empfänger versand. Dieser kann nun den verschlüsselten Anteil der Nachricht mit dem öffentlichen Schlüssel des Senders entschlüsseln und dann die beiden Teile vergleichen. Sind beide Nachrichten gleich, kann sich der Empfänger sicher sein, dass die Nachricht wirklich vom Sender stammt.
Zusätzlich wird mit diesem Verfahren die Nachricht auch vor Veränderungen geschützt. Ein während der Übertragung entstandener Fehler oder eine Manipulation wird aufgedeckt, da der unverschlüsselte und der verschlüsselte Teil der Nachricht verändert werden müssten, um den Vergleich zu bestehen.

2.7.1 Hashfunktionen

Beim Erstellen einer Signatur wird die Nachricht und eine verschlüsselte Kopie der Nachricht verschickt. Dies hat zur Folge, dass die eigentliche Nachrichtengröße verdoppelt wird. Um diesen Nachteil auszugleichen wurden die Signaturen um Hashfunktionen erweitert. Eine Hashfunktion bildet aus einer Nachricht beliebiger Länge einen Wert mit fester Länge. Jede Änderung an der Nachricht liefert einen anderen Wert zurück. Anstatt nun die gesamte Nachricht zu verschlüsseln, wird aus der Nachricht erst ein Hashwert gebildet und dieser dann mit dem privaten Schlüssel verschlüsselt. Der Empfänger der Nachricht kann nun den verschlüsselten Wert entschlüsseln und mit einem aus der unverschlüsselten Nachricht ermittelten Hashwert vergleichen.
Für diese Operationen werden aber nur Hashfunktionen verwendet, die einer Reihe von Anforderungen genügen. Eine der wichtigsten Forderungen an eine solche Funktion ist die Anforderung, dass es für einen möglichen Angreifer möglichst schwierig sein muss, Kollisionen herbeizuführen. Zu einer gegebenen Nachricht darf es also mit realistischem Aufwand nicht möglich sein eine zweite Nachricht mit gleichem Hashwert zu finden. Auch müssen die Funktionen sicherstellen, dass jeder Hashwert gleich oft vorkommt. Sollte ein

Hashwert besonders häufig vorkommen, hat es ein möglicher Angreifer in vielen Fällen einfacher eine falsche Nachricht mit dem gleichen Hashwert wie die Originalnachricht zu finden.
Hashfunktionen, die diese Anforderungen erfüllen, werden auch kryptographische Hashfunktionen genannt. Zu den bekanntesten kryptographischen Hashfunktionen gehören die Funktionen MD5 und SHA-1.

2.8 Umgang mit öffentlichen Schlüsseln

Die beschriebenen Verfahren liefern schon fast alles, was man zur Lösung der Anforderungen braucht. Allerdings muss man noch einige Strukturen einführen, die den Umgang mit den öffentlichen Schlüsseln aller Beteiligten festlegen. So muss sichergestellt werden, dass ein öffentlicher Schlüssel wirklich zu einer realen Person gehört. Auch müssen Verfahren festgelegt werden, die Sperrungen von Schlüsseln für den Fall von Verlust und Diebstahl festlegen. Zusätzlich ist es sinnvoll organisatorische Richtlinien für die Schlüssel, wie Schlüssellänge oder zeitliche Gültigkeit von Schlüsseln zu vereinbaren. Diese Forderungen werden mit geeigneten Infrastrukturen, den sogenannten „Public-Key-Infrastrukturen“ (PKI), gelöst. Es gibt verschiedene Ansätze für die PKI. Die wichtigsten drei Ansätze werde ich hier vorstellen. Diese werden in der Fachliteratur auch „Vertrauensmodelle“ genannt.

2.8.1 Direct Trust

Bei dem Modell „Direct Trust“ werden die Schlüssel persönlich z.B. mit Hilfe von Disketten ausgetauscht. Hierbei ist die angesprochene Authentizität des Schlüsselinhabers gegeben. Die Verbindlichkeit ist leider nur gering, da ein öffentlicher Schlüssel nachträglich abgestritten werden kann. Auch die Sperrung von Schlüsseln ist relativ schwer, da auch die Information über die Sperrung persönlich weiter gegeben werden muss. Dieses Modell kann eigentlich nur im privaten Umfeld verwendet werden, da sich organisatorische Richtlinien nur schwer umsetzen lassen.

2.8.2 Web of Trust

Das Modell „Web of Trust” erweitert den „Direct Trust” um einen Bürgen. So wird der Schlüssel nicht direkt übergeben, sondern über einen Dritten, der den Schlüssel mit einer Signatur seines Schlüssels bestätigt. Dadurch wird die Verbindlichkeit des Schlüssels gesteigert, da auch der Bürge die Zugehörigkeit zwischen Benutzer und Schlüssel kennt. Die Sperrung von Schlüsseln wird durch den Einsatz eines Bürgen nicht erleichtert. Auch diese Form wird im kommerziellen Umfeld kaum eingesetzt. Verwendung findet dieses Modell zum Beispiel in der Kryptographie-Software PGP.

2.8.3 Hierarchical Trust

Dieses Modell setzt an Stelle von vielen Bürgen eine zentrale unabhängige Instanz zur Verteilung und Überprüfung der Schlüssel ein. Diese Instanzen werden „Zertifizierungsstelle“, „Trust-Center“ oder „Certification Authority (CA)“ genannt. Der Begriff „Trust-Center“ ist zwar englisch, aber er wird nicht im englischsprachigen Raum

verwendet. Die Zertifizierungsstellen stellen für jeden öffentlichen Schlüssel ein elektronisches „Zertifikat“ aus. Ein Zertifikat enthält, neben dem öffentlichen Schlüssel und Daten über den Besitzer des Schlüssels und über die ausstellende Zertifizierungsstelle, auch ein Gültigkeitsdatum, das angibt, wie lange ein Zertifikat gültig ist.
Über alle Daten wird von der CA eine Signatur erstellt. Mit dieser Signatur bestätigt die CA die Zusammengehörigkeit zwischen dem Besitzer und dem Schlüssel. Besitzt man den öffentlichen Schlüssel der CA, kann man die Signatur überprüfen und damit die Korrektheit des Zertifikates überprüfen. Eine Sperrung eines Schlüssels ist hier auch sehr einfach umsetzbar, da die Informationen über die Sperrung von der CA zentral verwaltet werden können.
Aus diesen Gründen hat sich dieses Modell in allen kommerziellen Produkten durchgesetzt.

2.9 Varianten des Modells „Hierachical Trust“

Das Modell „Hierachical Trust“ lässt sich in mehreren Varianten vorfinden. Die einfachste und schon beschriebene Variante ist die „Zwei-Stufen-Hierarchie“. Bei dieser Hierarchie gibt es nur eine CA, die die 1. Stufe bildet, und alle Anwender, die die 2. Stufe bilden. Ein Benutzer braucht für diese Variante nur den öffentlichen Schlüssel der einen CA, um alle Zertifikate überprüfen zu können.

2.9.1 Verwendung mehrerer CAs

Häufig gibt es auch die Situation, dass mehrere CAs vorhanden sind, die aber jeweils einen anderen Benutzerkreis haben. Dies tritt zum Beispiel auf, wenn für jeden Standort einer Firma eine Zertifizierungsstelle aufbaut.
Für diesen Fall bietet sich eine sogenannte Cross-Zertifizierung an. Dabei stellen sich die CAs gegenseitig Zertifikate aus. Um ein Zertifikat einer cross-zertifizierten CA zu überprüfen, muss sich ein Benutzer erst das Zertifikat von der cross-zertifizierten CA besorgen und überprüfen. Je mehr CAs eingesetzt werden, desto komplizierter wird die Cross-Zertifizierung, da jede CA für jede andere CA ein Zertifikat erstellen muss.
Um diese Komplexität zu umgehen wird für solche Fälle eine übergeordnete CA eingesetzt. Hierbei stellt die übergeordnete CA für jede untergeordnete CA ein Zertifikat aus. Man spricht in diesem Fall von einer CA-Hierarchie. Die übergeordneten CAs können wiederum von einer weiteren übergeordneten CA zertifiziert werden. Die CA an der Spitze der Hierarchie wird auch Wurzel-CA genannt. Der Benutzer benötigt für diesen Fall nur den öffentlichen Schlüssel der Wurzel-CA, um alle Zertifikate der untergeordnete CAs zu überprüfen.

2.10 Trennung von Schlüsseln

Viele aktuelle Systeme erstellen bei der Schlüsselerstellung 2 Schlüsselpaare. Das eine Paar wird Signierschlüssel oder Sign-Key genannt. Dieses Schlüsselpaar wird nur zum Signieren von Nachrichten benutzt, während das andere Paar zum Verschlüsseln der Nachricht verwendet wird. Dieses Paar wird mit Chiffrierschlüssel oder Encryption-Key bezeichnet. Diese Trennung hat den Vorteil, dass die Kompromittierung eines Schlüssels den anderen

Schlüssel nicht beeinträchtigt. Sollte also der Chiffrierschlüssel „geknackt“ worden sein, ist der Signierschlüssel immer noch ohne Probleme benutzbar. Die Verwendung eines Schlüsselpaares bietet einem möglichen Angreifer einige Möglichkeiten. So ist es möglich, dass ein Benutzer durch das Signieren einer schon verschlüsselten Nachricht diese ohne sein Wissen für den Angreifer entschlüsselt. Der andere Angriffsfall ist der Fall, dass durch eine vom Benutzer durchgeführte Entschlüsselung einer unverschlüsselten Nachricht eine Signatur erstellt werden kann.

3 PKI-Anwendungen und Produkte

3.1 Standards

Mittlerweile haben sich auf dem Markt einige Standards für Zertifikate und Schlüsselspeicherung etabliert. Einer der meist verbreiteten Standards für Zertifikate ist das x509-Format. Das x509-Format wurde von dem Standardisierungsgremium ITU-T (früher CCITT) entwickelt, aber eigentlich nicht als Standard für Zertifikate, sondern als Authentifizierungsstandard für Kommunikationsnetze erarbeitet. Dieses Format liegt mittlerweile in der dritten Version vor.

Eine interessante Standardfamilie ist die Familie der „Public-Key Cryptography Standards" (PKCS). Diese Standards wurden von RSA Data Security Inc. (RSADSI) mit dem Versuch entwickelt, einen Industriestandard für eine Schnittstelle für asymmetrische Verschlüsselungsverfahren zu schaffen. Die PKCS sind nie durch ein Standardisierungsgremium festgelegt worden. Aber durch die Zusammenarbeit mit anderen Firmen hat es RSADSI geschafft eine Reihe von Standards zu entwickeln, die teilweise zu anderen Standards kompatibel sind. So wird zum Beispiel im Standard PKCS#3 ein Verfahren zur Implementierung eines Schlüsselaustauschs mittels Diffie-Hellman beschrieben. Der Standard PKCS#8 beschreibt eine Syntax zur Speicherung von privaten Schlüsseln.

Bruce Schneier fasst die PKCS wie folgt zusammen: „Die PKCS liefern ein Format zur Datenübertragung, das auf asymmetrischen Verschlüsselungsverfahren basiert, sowie eine Infrastruktur zur Unterstützung dieser Übertragung [Zitat Schneier99 Seite 670]."

3.2 Wichtige Komponenten einer PKI

Ein PKI-Produkt besteht aus mehreren Komponenten. Diese Komponenten teilen sich in zwei Gruppen auf. Es existieren die sogenannten zentralen und die dezentralen Komponenten.

Bei den zentralen Komponenten handelt es sich um die Komponenten eines Systems, die zu der CA gehören. Als erstes wäre da eine Instanz zur Überprüfung der Zusammengehörigkeit von realer Person und öffentlichem Schlüssel. Die Arbeit wird von der „Registrierungsinstanz" (RA) durchgeführt. Die Erstellung eines Zertifikats wird dann der Komponente „Zertifizierungsinstanz" (CA) überlassen. Die Verwendung der Abkürzung CA wird ab diesem Punkt etwas schwieriger, da CA sowohl für „Zertifizierungsstelle" als auch für „Zertifizierungsinstanz" innerhalb einer „Zertifizierungsstelle" stehen kann. Daher werde ich für die Zertifizierungsstelle ab jetzt den Begriff „Trust-Center" benutzen. Eine weitere Komponente, die zum Trust-Center gehören kann, ist der Zertifikatsserver. Auf diesem Zertifikatsserver werden dann die erstellten Zertifikate veröffentlicht. Zu einigen Trust-Center-Lösungen gehört auch ein Zeitstempeldienst oder auf englisch „TimeStampServer". Dieser Server hat die Aufgabe beim Verschlüsseln oder Signieren einen aktuellen Zeitstempel zu liefern, der dann ebenfalls verschlüsselt oder signiert wird. Mit diesem Verfahren kann man verschlüsselte oder signierte Dokumente zeitlich einordnen, um zu erkennen, ob zum Beispiel eine Signatur vor oder nach der Sperrung eines Schlüssels erstellt wurde.

Zu den dezentralen Komponenten gehören die Endeinheiten. Hierbei spricht man auch von PKI-Anwendungen. Es handelt sich um die Anwendungen, die die Zertifikate laden und lesen, Dokumente ver- und entschlüsseln und signieren. Eine weitere oft genutzte Komponente ist das „Personal Security Environment“ (PSE). Es handelt sich um die technische Komponente, auf der der private Schlüssel gespeichert wird. Hier werden unter anderem Chipkarten (Hardware-PSE) und besonders gesicherte Dateien auf Festplatten (Software-PSE) benutzt.
Einige PKI-Entwürfe sehen noch weitere dezentrale Komponenten vor. Zusätzlich zur RA innerhalb des Trust-Centers gibt es in einigen Fällen sogenannte „Lokal RA“ (LRA). Diese LRA sind Außenstellen der eigentlichen RA. Diese werden zum Beispiel eingesetzt, wenn ein Unternehmen mehrere Niederlassungen hat. Ein Mitarbeiter hat nun die Möglichkeit an seiner lokalen Niederlassung ein Zertifikat zu beantragen, dass dann weiter gegeben wird.

3.3 Ablauf einer Zertifikaterstellung

Der Ablauf beginnt mit dem Stellen eines Antrages auf ein Zertifikat. Der Antragssteller wird nun einer RA nachweisen müssen, wer er ist und welcher Schlüssel zu ihm gehört. Nach der Überprüfung durch die RA beauftragt diese die CA für den Antragssteller ein Zertifikat auszustellen. Daraufhin wird das Zertifikat wird veröffentlicht. Der Vorgang der Zertifikatserstellung wird Enrollment genannt.
Der beschriebene Ablauf ist sehr allgemein gehalten, da es mehrere Varianten des Enrollment gibt. Die Varianten hängen von einigen Faktoren ab. So ist festzulegen, ob das Schlüsselpaar von einem Trust-Center oder von dem Benutzer selbst erstellt werden soll. Es gibt auch Unterschiede in der Benutzung der RA. Bei einigen Varianten muss ein Antragssteller persönlich bei der RA vorstellig werden, während bei anderen Varianten ein Zertifikat auf rein elektronischem Weg erstellt werden kann.

3.4 Verfügbare Produkte

Einige der bekanntesten kommerziellen Systeme möchte ich hier kurz vorstellen:

- Entrust Authority[1]

  Die kanadische Firma Entrust Technologie bietet mit dem Produkt Entrust/PKI eines der erfolgreichsten PKI-Produkte der Welt an. Entrust/PKI ist vor allem für unternehmensinterne PKIs sehr gut geeignet, da bei öffentlichen PKIs eine von Entrust zusätzlich eingefügte Client-Komponente sehr hinderlich ist. Verfügbar ist dieses System für alle wichtigen kommerziellen Plattformen, wie zum Beispiel Windows 2000 Server, Sun Solaris und IBM AIX.

- UniCert[2]

  UniCert ist eine Entwicklung der Firma Baltimore. Die Konfiguration erfolgt durch eine sehr übersichtliche grafische Oberfläche. Mit dieser Oberfläche lässt sich eine CA-Hierarchie mit den jeweiligen RAs leicht aufbauen. UniCert steht

[1] Im Internet zu finden unter http://www.entrust.com/authority/
[2] Im Internet unter http://www.baltimore.com/unicert/ zu finden

nur auf Windows 2000 bzw. 2003 und NT Plattformen und für Sun Solaris und HP-UX zur Verfügung.

- Windows 2000 bzw. 2003

  Eine weitere PKI-Lösung befindet sich in den weitverbreiteten Windows 2000 bzw. 2003 Servern. Diese PKI-Lösung verfügt zwar nicht über den Funktionsumfang, den die bisher erwähnten Produkte bieten, und ist auch nur für das Windows Betriebssystem verfügbar, aber für Firmen, die sowieso mit diesem Betriebssystem arbeiten, bietet sich dieses System an.

Auch im Open-Source Bereich gibt es einige Projekte zu diesem Thema. Am weitesten entwickelt sind die Projekte „OpenCA“ und „TinyCA“.
OpenCA[1] wird als plattformunabhängiges Projekt mit den Sprachen C, Perl und JavaScript entwickelt. Zur Zeit befindet sich dieses Projekt noch in einer Beta-Phase. Nach dieser Beta-Phase könnte OpenCa eine erstzunehmende Konkurrenz zu den kommerziellen Produkten darstellen.
TinyCA[2] befindet sich ebenfalls noch in einer Beta-Phase. Auch hier ist die Programmiersprache Perl im Einsatz.

3.5 Berücksichtigung des Stellvertreterproblems

Obwohl die angegebenen Produkte schon sehr weit entwickelt sind, wird in keiner Lösung eine Stellvertreterregelung implementiert. Die Hersteller von PKI-Systemen verweisen zu diesem Problem auf Schlüsselrückgewinnungs- (engl. Key-Recovery) oder Schlüsselhinterlegungstechniken (engl. Key-Escrow).
Bei einer Hinterlegungstechnik werden die erstellten privaten Schlüssel nicht nur bei dem Benutzer gespeichert, sondern auch in einer zentralen Instanz, die den Schlüssel nach festgelegten Bedingungen an bestimmte Personen aushändigen kann. Eine der ersten Verwendungen dieser Technik war der EES (Escrowed Encryption Standard), der 1994 den amerikanischen Bürgern die Verwendung von starken Verschlüsselungstechniken erlaubte und andererseits den staatlichen Behörden den Zugriff auf verschlüsselte Daten erlaubte.
Die Weiterentwicklung der Hinterlegungstechnik ist die Schlüsselrückgewinnungstechnik, bei der nicht der private Schlüssel hinterlegt wird, sondern nur ein Rückgewinnungsschlüssel. Mit diesem Schlüssel kann dann eine vom Sender zusätzlich zur Nachricht erzeugte Rückgewinnungsinformation entschlüsselt werden. Die Rückgewinnungsinformation enthält Informationen, mit denen es möglich ist die verschlüsselte Nachricht zu entschlüsseln. Die bekannte Kryptografiesoftware PGP unterstützt ab der Version 5.0 eine Recovery-Technik.

[1] Informationen zu OpenCA unter : http://sourgeforge.net/projects/openca/ oder http://www.openca.org
[2] TinyCA lässt sich im Internet unter der Adresse http://tinyca.sm-zone.net finden.

4 Lösungsansätze für das Stellvertreterproblem

An dieser Stelle werde ich einige eigene Lösungsverfahren für das Stellvertreterproblem vorstellen und bewerten.

4.1 Erweiterung des Key-Escrow-Verfahrens

Dieses Lösungsverfahren beruht auf dem im vorherigen Kapitel erwähnten Key-Escrow-Verfahren. Die privaten Schlüssel werden nicht nur gespeichert, sondern auf einem Server für eine spezielle Applikation verfügbar gemacht. Diese Applikation habe ich mal „ReCrypter“ genannt. Der Ablauf einer Entschlüsselung einer Nachricht durch einen Stellvertreter könnte dann so aussehen:

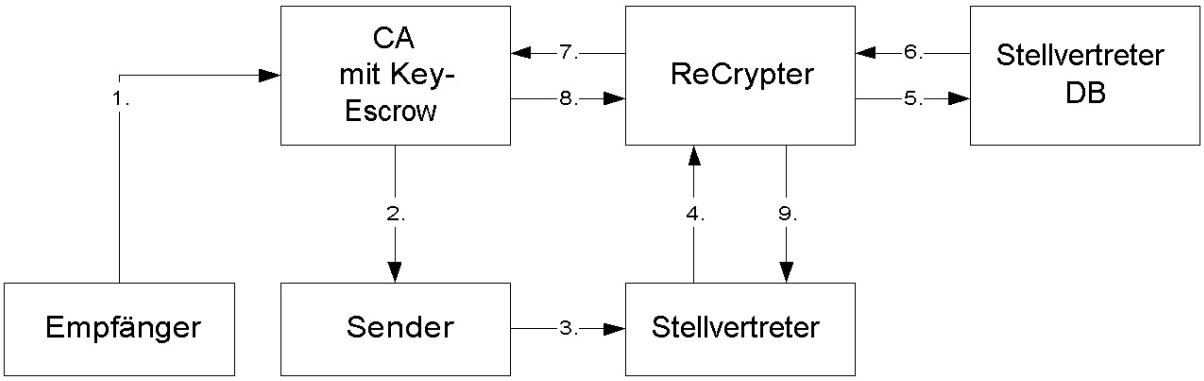

Abbildung 4 : Ablauf einer Entschlüsselung mit erweitertem Key-Escrow Verfahren

Nachdem ein Empfänger sein Zertifikat beantragt und bei der CA erstellt und veröffentlicht wurde (1.), kann ein Sender dieses Zertifikat von der CA abholen (2.) und eine Nachricht damit verschlüsseln. Für den Fall, das der Empfänger einen Stellvertreter eingesetzt hat, geht die Nachricht an den Stellvertreter (3.). Dies kann zum Beispiel durch eine automatische E-Mail-Weiterleitung geschehen. Nun kann dieser Stellvertreter diese Nachricht an den ReCrypter mit dem Auftrag zur Entschlüsselung schicken (4.), da er den Schlüssel des eigentlichen Empfängers nicht hat.. Der Stellvertreter muss diesen Auftrag auch mit seiner Signatur zur Authentifizierung unterschreiben. Der ReCrypter ermittelt mit Hilfe der Signatur und der Originalnachricht den Sender und den eigentlichen Empfänger, an den der Sender die Nachricht geschickt hat. Sind Sender und Originalempfänger bestimmt, muss der ReCrypter entscheiden, ob der angegebene Stellvertreter wirklich als Stellvertreter für den Empfänger eingesetzt wurde. Dazu fragt der ReCrypter eine Stellvertreter-Datenbank an (5.), die von einer dritten unabhängigen und vertrauenswürdigen Stelle, zum Beispiel der Personalstelle, verwaltet wird. Bekommt der ReCrypter die Nachricht, dass der Stellvertreter wirklich eingesetzt wurde (6.), fragt der ReCrypter den privaten Schlüssel des eigentlichen Empfängers aus dem Key-Recovery-Bereich und das öffentliche Zertifikat des Stellvertreters ab (7.). Dann beginnt das Programm mit den erhaltenen Daten (8.) die ursprüngliche Nachricht zu entschlüsseln und danach für den Stellvertreter neu zu verschlüsseln. Neben der erneuten Verschlüsselung muss nun auch die ursprüngliche Signatur des Senders der

Nachricht neu erstellt werden. Die nun erstellte Nachricht wird dann wieder an den Stellvertreter geschickt (9.), der die Nachricht dann lesen kann.
Eine Antwort vom Stellvertreter an den eigentlichen Sender verschlüsselt und signiert der Stellvertreter ohne irgendwelche Eingriffe der beschriebenen speziellen Applikation. Beim Erhalt der Nachricht kann der jetzige Empfänger anhand der Stellvertreterdatenbank prüfen, ob der Sender wirklich ein Stellvertreter der angeschriebenen Person ist.
Das „Key Escrow Verfahren“ speichert nur die privaten Chiffrierschlüssel. Die Signierschlüssel brauchen nicht gespeichert werden, da diese nicht zur Entschlüsselung der Nachricht für den Stellvertreter notwendig sind.

4.1.1 Formale Darstellung

In dieser und weiteren formalen Darstellungen werden einige Formeln verwendet, die ich hier kurz vorstellen möchte :
$\{x\}_{crypt(y),sign(z)}$ steht für eine Information X, die mit dem Schlüssel y verschlüsselt und mit dem Schlüssel z signiert wurde. Die Formeln ***CoK(x)*** und ***CpK(x)*** stehen für den öffentlichen (CoK) und den privaten (CpK) Chiffrierschlüssel von X. Für öffentliche und private Signierschlüssel von X werden die Formel ***SoK(x)*** und ***SpK(x)*** verwendet. ***Cert(x)*** ist eine Formel für ein Zertifikat für den Schlüssel X.

Nachricht :

Von	**An**	**Nachricht**
Person E	CA	Cert(CoK(E)), Cert(SoK(E))
CA	Person S	Cert(CoK(E))
Person S	Person E	$\{Nachricht\}_{crypt(CoK(E)),sign(SpK(S))}$
Person E	Person STE	„Nachricht wird an Person STE weitergeleitet“
Person STE	ReCrypter	$\{\{Nachricht\}_{crypt(CoK(E)),sign(SpK(S))}\}_{crypt(CoK(Recrypter)),sign(SpK(STE))}$
ReCrypter	Stellvertreter-DB	IstXStellvertreterVonY(STE,E,?)
Stellvertreter-DB	ReCrypter	IstXStellvertreterVonY(STE,E,“JA“)
ReCrypter	Person STE	$\{Nachricht\}_{crypt(CoK(STE)),sign(SpK(ReCrypter))}$

Antwort :

Von	**An**	**Nachricht**
Person STE	Person S	$\{Antwort\ auf\ Nachricht\}_{crypt(CoK(S)),sign(SpK(STE))}$
Person S	Stellvertreter DB	IstXStellvertreterVonY(STE,E,?)
Stellvertreter DB	Person S	IstXStellvertreterVonY(STE,E,“JA“)

4.2 Temporäre Sperrung von Empfängerzertifikaten

Das letzte Verfahren hatte den Nachteil, dass alle Schlüssel, auch die privaten Schlüssel, in einem zentralen Punkt gespeichert wurden. Dadurch ist es möglich durch eine Kompromittierung des ReCrypters alle mit diesem System verschlüsselte Nachrichten zu lesen. Um dieses zu verhindern, wird nun die Stellvertreterregelung vor die Verschlüsselung und Versendung gelegt. Das standardmäßige Verfahren zum Testen der Gültigkeit der Zertifikate wird erweitert. Dies sieht dann wie folgt aus :

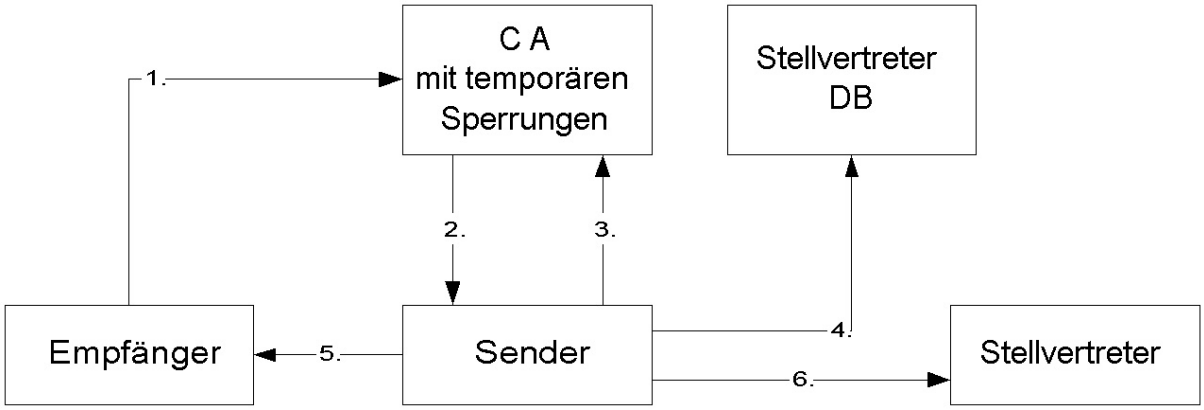

Abbildung 5 : Stellvertreterregelung mit temporärer Sperrung

Auch diese Lösung beginnt mit dem Beantragen eines Zertifikates durch einen Benutzer (1.). Sollte ein Benutzer des Systems ein Stellvertreter benötigen, beantragt dieser bei der Zertifizierungsstelle einen Stellvertreter. Die Zertifizierungsstelle sperrt das Zertifikat für eine bestimmte Zeit und legt in einer zusätzlichen Datenbank einen Stellvertreter für den Benutzer fest. Will nun ein anderer Benutzer eine Nachricht an diesen Benutzer schicken, fordert dieser von der CA erst das Zertifikat an (2.) und überprüft dann, ob es eventuell gesperrt ist(3.). Sollte das Zertifikat gesperrt sein, wird der Sender darüber informiert. Auf Wunsch kann nun der Stellvertreter ermittelt (4.) und das Zertifikat für den Stellvertreter über einen Zertifikatsserver geladen werden. Die Nachricht wird dann mit Hilfe einer Hybridverschlüsselung für den eigentlichen Empfänger (5.) und den Stellvertreter (6.) verschlüsselt. Der Stellvertreter kann die Nachricht, sobald er sie bekommt, mit seinem privaten Schlüssel entschlüsseln. Die Antwort kann der Stellvertreter nur mit seinem privaten Schlüssel signieren, so dass der eigentliche Empfänger wieder die Stellvertreterdatenbank anfragen muss, um zu testen, ob der Stellvertreter wirklich als Stellvertreter eingesetzt wurde. Ist der Stellvertreter nicht mehr nötig, kann die Stellvertreterregelung durch die Aufhebung der temporären Sperrung und Löschung des Eintrages in der Stellvertreterdatenbank gelöscht werden.

4.2.1 Formale Darstellung

Nachricht :

Von	**An**	**Nachricht**
Person E	CA	Cert(CoK(E)), Cert(SoK(E))
CA	Person S	Cert(CoK(E))
Person S	CA	ZertifikatsStatus(Cert(CoK(E)),?)
CA	Person S	ZertifikatsStatus(Cert(CoK(E)),“temporär gesperrt“)
Person S	Stellvertreter-DB	WerIstStellvertreterVon(Cert(CoK(E)),?)
Stellvertreter-DB	Person S	WerIstStellvertreterVon(Cert(CoK(E)),Cert(CoK(STE)))
Person S	Person STE	{Nachricht}$_{\text{crypt(CoK(E),CoK(STE)),sign(SpK(S))}}$

Antwort :

Von	An	Nachricht
Person STE	Person S	{Antwort auf Nachricht}$_{crypt(CoK(S)),sign(SpK(STE))}$
Person S	Stellvertreter-DB	IstXStellvertreterVonY(STE,E,?)
Stellvertreter-DB	Person S	IstXStellvertreterVonY(STE,E,“JA“)

4.3 Erstellung von Stellvertreterzertifikaten

Obwohl das letzte Lösungsverfahren schon die privaten Schlüssel wirklich privat gehalten hat, ist immer noch der Einsatz einer zusätzlichen Stellvertreterdatenbank notwendig, um die Stellvertreterlösung zu realisieren. Dieses Modul wird in diesem Verfahren eingespart. Diese Lösung basiert auf den sogenannten Gruppensignaturen, die David Chaum und Eugène van Heyst 1991 im Rahmen der EuroCrypt vorgestellt haben [Chaum91].

Bei den Gruppensignaturen handelt es sich um einen Typ von Signaturen, die von einer Gruppe von Personen ausgestellt werden können. Jedes Mitglied der Gruppe kann eine gültige Signatur ausstellen. Eine Person, die die Signatur überprüft, erkennt nur, ob ein Mitglied der Gruppe die Signatur erstellt hat. Die genaue Person bleibt dem Überprüfenden verborgen.

Verwendet wurde hier eine stark vereinfachte Version des ersten Realisierungsvorschlages aus der oben erwähnten Veröffentlichung. Dieser Vorschlag sieht für jedes Mitglied einer Gruppe eine gewisse Anzahl von Schlüsselpaaren und Zertifikaten vor. Die Erzeugung und Verteilung geschieht durch eine Person, die sich die Verteilung merkt und so weiß, wer welche Schlüssel besitzt. Alle Zertifikate werden auf denselben Besitzer, nämlich die Gruppe, ausgestellt. Verwendet nun ein Mitglied einen seiner zu einem auf die Gruppe ausgestellten Zertifikat passenden Signierschlüssel, kann der Empfänger feststellen, dass ein Mitglied der Gruppe diese Signatur erstellt hat. Wer die Signatur erstellt hat, kann nur die Person, die die Schlüssel verteilt hat, feststellen.

In meinem Entwurf bekommt jeder Stellvertreter jeweils ein Signier- und Chiffrierschlüsselpaar. Den Zertifikaten für die Schlüssel, die die zu vertretende Person beantragt, wird eine Information mitgegeben, welche Person mit diesen Schlüsseln vertreten wird.

Eine Lösung könnte so aussehen:

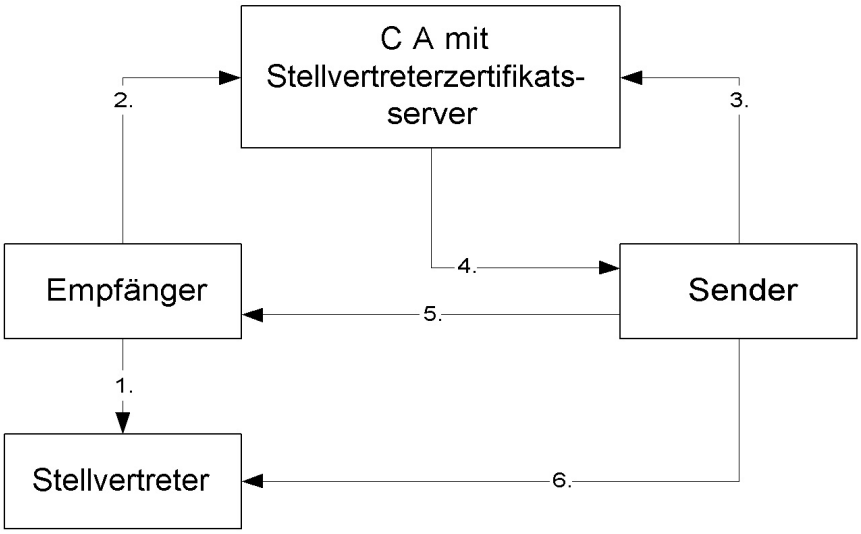

Abbildung 6 : Stellvertreterlösung mit Stellvertreterzertifikaten

Sollte ein Benutzer einen oder mehrere Stellvertreter benötigen, kann dieser Benutzer für jeden Stellvertreter einen neuen Schlüsselsatz (1.) erstellen. Für diesen Schlüsselsatz kann der Benutzer ein besonderes Zertifikat erstellen lassen, in dem die Stellvertreterregelung für sein eigenes Zertifikat mit gespeichert wird. Dieses Zertifikat wird ebenfalls auf dem Zertifikatsserver veröffentlicht (2.). Das Zertifikat ist nur so lange gültig, wie der Stellvertreter benötigt wird. Beim Verschlüsseln der Nachricht wird nun vom Sender zusätzlich zur Überprüfung der Gültigkeit getestet, ob ein gültiges Stellvertreterzertifikat auf dem Server vorhanden ist. Sollte ein Stellvertreterzertifikat vorhanden sein, wird die Nachricht nicht nur mit dem Schlüssel des Empfängers, sondern auch mit dem Schlüssel aus dem Stellvertreterzertifikat, verschlüsselt.
Der Stellvertreter, der den Schlüssel von der zu vertretenden Person bekommen hat, kann nun auch diese Nachricht lesen. Die Antwort kann der Stellvertreter mit dem Stellvertreterschlüssel signieren, wodurch der Empfänger der Antwort weiß, dass diese von einem gültigen Stellvertreter geschrieben wurde.

4.3.1 Besonderheiten des Stellvertreterzertifikats

Das Stellvertreterzertifikat hat einige Besonderheiten gegenüber dem normalen Zertifikat. Es trägt neben den Informationen zu Zertifikatsausteller, Schlüssel, Schlüsselinhaber und Gültigkeit auch Informationen über die zu vertretende Person.
Ein solches Zertifikat hat auch besondere Gültigkeitsregeln. So ist ein solches Zertifikat nur gültig, wenn es, wie normale Zertifikate, nicht abgelaufen oder gesperrt ist, und das Zertifikat, das vertreten wird ebenfalls noch gültig ist. Ist das zu vertretende Zertifikat gesperrt oder abgelaufen, ist auch das Stellvertreterzertifikat ungültig.

4.3.2 Formale Darstellung

Nachricht :

Von	**An**	**Nachricht**
Person E	CA	Cert(CoK(E)), Cert(SoK(E))
CA	Person S	Cert(CoK(E))
Person S	CA	IstStellvertreterVorhanden(Cert(CoK(E)),?)
CA	Person S	IstStellvertreterVorhanden(Cert(CoK(E)),Cert(CoK(STE)))
Person S	Person STE	{Nachricht}$_{\text{crypt(CoK(E),CoK(STE)),sign(SpK(S))}}$

Antwort :

Von	**An**	**Nachricht**
Person STE	Person S	{Antwort auf Nachricht}$_{\text{crypt(CoK(S)),sign(SpK(STE))}}$
Person S	CA	Cert(SoK(STE))
Person S	CA	ZuVertretendePersonVon(Cert(SoK(STE)),?)
CA	Person S	ZuVertretendePersonVon(Cert(SoK(STE)),Cert(SoK(E)))

5 Detailentwurf einer Lösung

5.1 Wahl einer Lösung

Jeder der vorgestellten Lösungen hat Vor- und Nachteile. An dieser Stelle werde ich die verschiedenen Lösungen diskutieren.

Durch die Benutzung von verschlüsselten und signierten Nachrichten erfüllt jeder der vorgestellten Lösungsansätze die ersten drei Forderungen aus dem Kapitel 1.2.1. So ist durch Verschlüsselung sichergestellt, dass nur der Empfänger, der über den passenden privaten Schlüssel verfügt, die Nachricht entschlüsseln und lesen kann. Durch das Signieren der Nachricht wird diese nicht unmittelbar vor Veränderung geschützt, aber jede Veränderung in der Nachricht führt zu einer ungültigen Signatur. Aus der Signatur lässt sich ebenfalls der Absender der Nachricht ermitteln.

Die Erweiterung des Key-Escrow-Verfahrens hat den Nachteil, dass die privaten Chiffrierschlüssel auf einem öffentlich zugänglichen Server gespeichert sind. Dadurch entsteht das Risiko, dass die Schlüssel in falsche Hände geraten. Da die Stellvertreterregelung durch die Stellvertreterdatenbank festgelegt wird, muss auch diese Instanz besonders gesichert werden, da sich sonst ein Stellvertreter selbst einsetzen könnte. Der Aufwand der Pflege des Systems ist auch relativ hoch. So muss neben der CA mit den Zertifikaten auch die Stellvertreterdatenbank und die Zugriffsberechtigungen für den ReCrypter gepflegt werden.

Die Stellvertreterregelungen werden in diesem Lösungsansatz allein durch die Stellvertreter-datenbank festgelegt. Diese Instanz muss forderungsgemäß vertrauenswürdig sein. Eine weitere Forderung verlangt, dass die Stellvertreterregelung nur so lange gültig ist, wie die zu vertretende Person festgelegt hat. Dadurch muss das Ablaufdatum in der Stellvertreter-datenbank gespeichert und abgelaufene Regelungen automatisch gelöscht werden. In der versendeten Nachricht muss zusätzlich einen Eintrag gespeichert werden, der angibt, ob die Nachricht für einen Stellvertreter umkodiert werden darf. Dies ist notwendig, damit der Sender selbst entscheiden kann, ob die Nachricht für einen Stellvertreter lesbar gemacht werden darf. Auch dies ist eine gestellte Forderung. Diese Information muss ebenfalls verschlüsselt und signiert übermittelt werden, da sonst diese Information von jeder Person geändert werden kann.

Bei der Überprüfung einer Antwort durch den Stellvertreter muss die Stellvertreterdatenbank noch einmal befragt werden, ob der Stellvertreter offiziell eingesetzt worden ist. Obwohl diese Lösung die Forderungen erfüllt, hat diese den Nachteil, dass die privaten Schlüssel an einer zentralen Stelle, die öffentlich zugänglich ist, gespeichert sind. Dadurch entsteht ein Risiko. Durch die Kompromittierung dieser Stelle werden alle Schlüssel und damit die entsprechenden Zertifikate kompromittiert. Deshalb habe ich im folgenden Lösungsansatz darauf geachtet, dass die privaten Schlüssel nicht mehr an öffentlichen Stellen liegen.

Dadurch ändert sich der Ablauf der Stellvertreterregelung. Der Stellvertreter kann nun nicht mehr nach der Verschlüsselung die Daten für sich entschlüsseln, sondern die Daten werden bei der Verschlüsselung extra für den Stellvertreter verschlüsselt. Die Stellvertreterdatenbank ist aber immer noch notwendig, um den Stellvertreter ausfindig zu machen. Der Stellvertreter

kann nur durch die direkte Mitarbeit der CA- und der Stellvertreterdatenbank-Mitarbeiter festgelegt werden. Dadurch wird die Forderung nach nur einer für die Stellvertreterregelung zuständigen Instanz nicht befolgt. Dieses Vorgehen ist auch nicht optimal, da die Stellvertreterregelung nicht automatisch nach Ablauf des festgelegten Zeitraums gelöscht wird. Nur wenn die temporäre Sperrung und der Eintrag in der Stellvertreterdatenbank aufgehoben wird, ist die Stellvertreterregelung aufgehoben.
Dieses Problem wird durch die zuletzt vorgestellte Lösung auch behoben. Der Stellvertreter wird von der zu vertretenden Person direkt eingesetzt, in dem diese Person einfach ein Stellvertreterzertifikat erzeugen lässt, veröffentlicht und die privaten Schlüssel an den Stellvertreter weitergibt. Eine zusätzliche Stellvertreterdatenbank ist nicht mehr nötig.
Das im letzen Abschnitt beschriebene Verfahren erfüllt, im Gegensatz zu allen anderen aufgeführten Lösungsmöglichkeiten für das Stellvertreterproblem, alle Anforderungen der fiktiven Firma. So wird die Stellvertreterregelung nur von einer vertrauenswürdigen Stelle, nämlich der CA, geregelt. Auch hat der Sender die Möglichkeit durch die Ablehnung des Stellvertreterzertifikats die Verschlüsselung für den Stellvertreter zu verweigern. Die Gültigkeit der Regelung läuft automatisch nach dem von der zu vertretenden Person bestimmten Zeitraum ab, da dann automatisch das Zertifikat ausläuft. Für diese Lösung ist ein System nötig, das aus mehreren Modulen besteht. Diese werde ich mit deren Funktionen in diesem Kapitel beschreiben.

5.2 Das Enrollment

Bevor ich auf die einzelnen Module eingehe, ist es notwendig das zu benutzende Enrollmentverfahren festzulegen. Im Verfahren wird festgelegt, welche Instanz welche Aufgabe bekommt. Folgendes Enrollment werde ich benutzen:
Die Schlüssel werden dezentral von jedem User selbst erstellt. Dies hat den Vorteil, dass nur er den privaten Schlüssel zu Gesicht bekommt. Ein Abfangen des Schlüssels auf elektronischem oder postalischem Weg ist unmöglich. Dadurch ist es nicht möglich, dass Schlüssel in einem Key-Recovery- oder einem Key-Escrow-System gespeichert werden. Dies kann aber nach einer Erstellung eines Zertifikats nachgeholt werden. Mit diesen Schlüsseln kann ein User nun zu einer lokalen Registrierungsstelle (LRA) gehen und dort einen Antrag auf ein Zertifikat stellen. Dieser Antrag ist auch auf elektronischem Weg möglich. Die Lokale RA überprüft die Angaben und gibt nach der erfolgreichen Überprüfung die Daten an die eigentliche CA weiter. Die CA stellt dann das Zertifikat aus und veröffentlicht es auf einem Zertifikatsserver. Dem User wird das Zertifikat nach der Erstellung ausgehändigt.

5.3 Die Schlüsselerzeugung

Das erste Modul, das ich beschreibe, ist das Modul zur Schlüsselerstellung. Dieses Modul gibt jedem Benutzer die Möglichkeit eigene Schlüsselpaare zu erzeugen. Es werden zwei Schlüsselpaare erzeugt. Ein Signier- und ein Chiffrierschlüsselpaar.
Bei der Erstellung muss der Benutzer seinen Namen und seine E-Mail-Adresse eingeben. Zusätzlich wird eine eindeutige Schlüssel-ID aus dem Verschlüsselungsalgorithmus, der Schlüssellänge, dem Erzeugungszeitpunkt und den Benutzereingaben erzeugt. Diese ID ist

wichtig um zu einem privaten Schlüssel den passenden öffentlichen Schlüssel zu finden. Die erstellten Schlüssel werden zusammen mit den eingegebenen Daten und der Schlüssel-ID in vier Dateien gespeichert. Jede der Dateien enthält die Schlüssel-ID und die Benutzerinformationen. Zusätzlich wird in jeder Datei ein Schlüssel gespeichert. Zwei Dateien enthalten die privaten Chiffrier- und Signierschlüssel, während in den anderen beiden Dateien die passenden öffentlichen Schlüssel gespeichert werden.
Bevor die Schlüssel in den Dateien abgelegt werden, werden diese in einen Standard konvertiert. Die privaten Schlüssel werden in den PKCS8-Standard konvertiert, der speziell für private Schlüssel geschaffen wurde. Für die öffentlichen Schlüssel wird ein X509-Standard verwendet. Die direkte Speicherung in den verwendeten Standards hat den Nachteil, dass die Informationen über den Besitzer des Schlüssels nicht gespeichert werden können.
Es ist sinnvoll die privaten Schlüssel mit einem symmetrischen Verschlüsselungsverfahren zu verschlüsseln damit ein Schlüssel, der gestohlen oder verloren wurde, nicht ohne Wissen des Besitzers benutzt werden kann. Der Schlüssel für das symmetrische Verfahren wird aus einem eingegebenen Passwort gebildet.
Das Modul soll ebenfalls die Möglichkeit bieten, auch nur ein Schlüsselpaar zu erstellen. Damit ist es möglich ein kompromittiertes Schlüsselpaar zu ersetzen.

5.4 Zertifikatsantrag

Mit den erstellten Schlüsseln kann nun ein Zertifikat pro Schlüsselpaar beantragt werden. Diesen Antrag kann der Benutzer bei einer firmeninternen lokalen Registrierungsstelle stellen. Der Benutzer muss für den Antrag die öffentlichen Schlüssel der Schlüsselpaare an die Lokale RA übergeben. Die Registrierungsstelle überprüft anhand eines Personalausweises oder eines ähnlichen Dokumentes, ob der Antragssteller den Benutzerdaten innerhalb der Schlüsseldaten entspricht. Stimmen die Daten überein, werden die Schlüsseldateien mit dem öffentlichen Chiffrier-Schlüssel der CA verschlüsselt und mit dem privaten Signier-Schlüssel der lokalen RA signiert. Die verschlüsselten Daten werden dann als Zertifikatsantrag an die CA weitergeleitet.
Beim Erstellen eines Zertifikatsantrags kann ein Antragssteller ein zu vertretendes Zertifikat angeben. Dadurch entsteht nach der Bearbeitung des Antrags ein Antrag für ein Stellvertreterzertifikat. In diesem muss zusätzlich die Dauer der Gültigkeit der Stellvertreterregelung festgelegt werden. Auch die Informationen über den Stellvertreter werden mit dem Schlüssel der Zertifizierungsstelle verschlüsselt und an diese geleitet.

5.5 Zertifizierungsstelle

Diese Applikation erstellt die eigentlichen Zertifikate, indem sie die Daten aus den Dateien liest, die durch die lokalen RAs erstellt worden sind. Vor dem Entschlüsseln dieser Daten wird zuerst die Signatur überprüft. Die öffentlichen Schlüssel der lokalen RAs zur Überprüfungen liegen in einem speziellen Verzeichnis dieses Tools. Es handelt sich dabei um Zertifikate, die selbst von der CA ausgestellt wurden. Nach der Überprüfung und der Entschlüsselung kann nun mit den Daten aus der Datei ein Zertifikat erstellt werden. In diesem Zertifikat werden neben dem zu zertifizierenden öffentlichen Schlüssel und den

Benutzerdaten auch Informationen über den Aussteller des Zertifikats abgelegt. Dazu gehört neben dem Namen auch die Schlüssel-ID des Signierschlüssels der CA. Eine sehr wichtige Information, die noch im Zertifikat vorhanden ist, ist die Gültigkeit. Jedes Zertifikat wird nur mit einer bestimmten Lebensdauer erstellt. Dies ist eine Sicherheitsmaßnahme, um sicher zu stellen, dass die Schlüssel regelmäßig gewechselt werden. Außerdem wird für jedes Zertifikat eine Seriennummer festgelegt. Dieses Zertifikat wird ebenfalls in einer Datei gespeichert. Da der Zertifikatsantrag und das erstelle Zertifikat als Dateien vorliegen, kann die Zertifikatserstellung auf einem unabhängigen und nicht an ein Netz angeschlossenen Rechner durchgeführt werden. Dies hat den Vorteil, dass die privaten Schlüssel der CA nicht auf einem Rechner vorhanden sein müssen, der sich in einer unsichern Umgebung, wie dem Internet, befindet.
Neben der Erstellung der Zertifikate besitzt die CA die Möglichkeit Stellvertreterzertifikate zu erstellen. Bei der Erstellung werden die Informationen aus einem Stellvertreterantrag geladen und in ein Zertifikat übertragen.

5.5.1 Verwendete Standards

In diesem System werden X509-Zertifikate der dritten Version, die im RFC 3280[1] vorgestellt werden, verwendet. Diese Zertifikate bieten die Möglichkeit neben den Standardattributen, wie dem öffentlichen Schlüssel und den Informationen zum Zertifikatsinhaber, auch eigene Attribute in Form sogenannter Extensions aufzunehmen. Es wird die binäre ASN.1 DER Kodierung verwendetet um diese Daten zu speichern. Die ASN.1 Spezifikation der Extensions sieht wie folgt aus:

```
Extensions   ::=   SEQUENCE SIZE (1..MAX) OF Extension
Extension    ::=   SEQUENCE {
  extnID           OBJECT IDENTIFIER,
  critical         BOOLEAN DEFAULT FALSE,
  extnValue        OCTET STRING }
```

Jede Extension hat einen eindeutigen Bezeichner extnID, ein Flag, das anzeigt, ob diese Extension „kritisch“ ist, und den eigentlichen Wert der Extension (extnValue). Kritische Extensions müssen von Applikationen ausgewertet werden. Falls eine kritische Extension unbekannt ist oder nicht berücksichtigt werden kann, muss die gesamte Verarbeitung abgebrochen werden.
Bei den Object-Identifiers (OID) handelt es sich um eine hierarchisch gegliederte numerische Struktur. Die Gruppe der X509 Zertifikatserweiterungen haben den OID „2.5.29“. Grafisch könnte man diesen Wert wie folgt darstellen:

[1] Das Dokument kann unter der Internet-Adresse http://www.faqs.org/rfcs/rfc3280.html eingesehen werden.

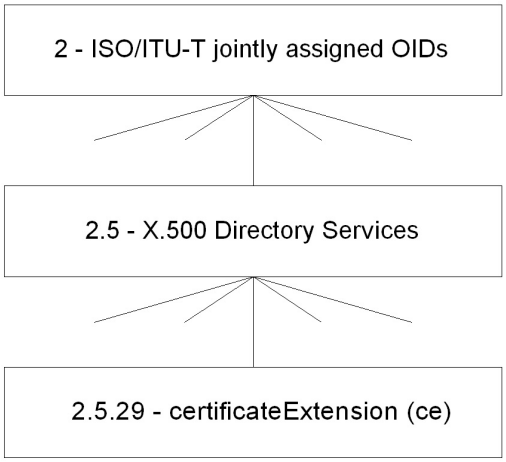

Abbildung 7 : Darstellung einer OID-Struktur

Jede einzelne Extension bekommt einen eigenen OID, der als Knoten an den ce-Knoten gehängt wird. Eine Auflistung aller festgelegten OIDs kann man unter der Internet-Adresse http://asn1.elibel.tm.fr/oid/ finden.
Dieses System verwendet zwei Extensions. In einer Extension, die auch KeyUsage genannt wird, wird der Zweck gespeichert, für den das Zertifikat verwendet werden darf. Diese Extension ist bereits definiert und verfügt über den OID „2.5.29.15“. Als Wert wird ein Boolean-Array gespeichert, bei dem jeder Boolean-Wert für einen Verwendungszweck steht. Aus einem öffentlichen Chiffrierschlüssel werden Zertifikate erzeugt, die nur zum Verschlüsseln von Daten genutzte werden sollen. In dem als KeyUsage gespeicherten Array, ist in diesem Fall nur der erste Boolean-Wert auf „true“ gesetzt, während bei Zertifikaten, die nur zum Überprüfen einer Signatur gedacht sind, der dritte Wert auf „true“ gesetzt ist. Eine Liste der weiteren Möglichkeiten dieser Extension befindet sich ebenfalls in dem angegebenen RFC.
Bei der zweiten Extension handelt es sich um die Stellvertretererweiterung. Für diese Extension existiert kein Standard und kein OID. Aus diesem Grund habe ich den nicht verwendeten OID „2.5.29.60“ genutzt. Um diese Erweiterung in einer kommerziellen Anwendung zu benutzen, wäre es aber sinnvoller einen eigenen OID bei der zuständigen Stelle zu registrieren. Als Wert wird ein DER-kodierter String verwendet, der aus dem später beschriebenen Stellvertreter-Objekt erstellt wird.
Beide Extensions werden als nicht kritisch in dem Zertifikat abgelegt. Dadurch lassen sich die Zertifikate auch in anderen Applikationen verwenden.

5.6 Der Zertifikatsserver

Nach der Erstellung wird das Zertifikat veröffentlicht. Dies geschieht mit Hilfe eines Zertifikatsservers, der den zweiten Teil der Zertifizierungsstelle bildet. Der Verwalter der CA hat die Möglichkeit die Zertifikate zu sperren. Dies ist notwendig, wenn ein Schlüssel kompromittiert wurde und damit vor dem Ablauf der Gültigkeit ungültig wird. Wird ein Zertifikat gesperrt, wird es vom Zertifikatsserver gelöscht und in eine besondere Liste kopiert. Diese List wird auch Certificate Revocation List (CRL) oder Widerrufsliste genannt. Auf

diese List kann der Client zugreifen, um ein Zertifikat auf eine Sperrung zu testen. Dabei wird ein dem „Online Certificate Status Protocol“ (OCSP) verwandtes Protokoll verwendet. Die Anfrage des Clients beantwortet die CA mit „good“, „revoked“ oder „unknown“. „Good“ bedeutet, dass das Zertifikat existiert und nicht gesperrt ist. Antwortet der Server mit „revoked“, ist das Zertifikat gesperrt und sollte vom Client nicht mehr verwendet werden. Die Antwort „unknown“ zeigt an, dass das zu überprüfende Zertifikat unbekannt ist.
Auch Stellvertreterzertifikate werden auf dem Zertifikatsserver veröffentlicht. Dies geschieht aber getrennt von den anderen Zertifikaten. Die Stellvertreterzertifikate können nicht gesperrt werden. Dies ist aber nicht notwendig, da diese Zertifikate nicht auf der Seite des Clients gespeichert werden sollten.

5.7 Der Verschlüsselungsclient

Mit den erstellten Schlüsseln und den Zertifikaten kann ein Benutzer nun Nachrichten an andere Benutzer verschlüsselt verschicken. Dazu wird das Zertifikat des Empfängers benötigt. Dieses Zertifikat wird in einem clientinternen Zertifikatsspeicher gesucht. Ist das Zertifikat nicht im Speicher, kann sich der Client das Zertifikat von dem Zertifikatsserver laden. Ist ein Zertifikat für den Benutzer auf dem Server vorhanden, wird es an den Client gesendet, der dieses dann in dem Zertifikatsspeicher ablegt.
Ist das Zertifikat vorhanden muss überprüft werden, ob es nicht gesperrt ist. Zusätzlich wird getestet, ob ein oder mehrere Stellvertreterzertifikate vorliegen. Für beide Überprüfungen stellt der Client eine Anfrage an den Zertifikatsserver.
Sind alle Zertifikate vorhanden, wird die Nachricht mit einem symmetrischen Verfahren verschlüsselt. Dabei wird der zu verwendende geheime Schlüssel zufällig erzeugt. Der geheime Schlüssel wird dann mit den öffentlichen Schlüsseln aus den Zertifikaten verschlüsselt. Hier wird also ein Hybridverschlüsselungsverfahren eingesetzt. Neben der Verschlüsselung muss der Client auch die Signierung der Daten unterstützen. Dazu muss der Client Zugriff auf die privaten Schlüssel haben. Auch hier ist es wichtig, dass mehr als eine Signatur angelegt werden kann, da ein Sender neben seiner eigenen Signatur auch eine Stellvertretersignatur anlegen können muss.
Die verschlüsselten Schlüssel, die verschlüsselte Nachricht und die Signaturen werden in einer Datei gespeichert, die dann an den Empfänger geschickt werden kann. In diese Datei wird noch die Zeit der Verschlüsselung abgelegt. Die Zeit wird entweder von dem Zeitstempelserver oder von der Systemzeit abgerufen. Sie wird nur vom System gelesen, wenn der Zeitstempelserver nicht verfügbar ist. Die Nachricht wird nicht in einem bestehenden Standard gespeichert. Ich habe darauf in diesem Entwurf verzichtet. Da die verwendete Speicherung für den Ablauf und die Funktionsweise des Entwurfes nicht entscheidend ist, habe ich mich für ein einfaches properitäres Format entschieden.
Der Client beherrscht neben der Verschlüsselung der Daten auch die Entschlüsselung. Der erste Schritt ist das Lesen der Datei und das Zerlegen in die Bestandteile Schlüssel, Nachricht und Signaturen. Im ersten Schritt wird aus den Schlüsselinformationen der geheime Schlüssel geladen, mit dem die Nachricht verschlüsselt ist. Da dieser Schlüssel mit einem

asymmetrischen Verfahren verschlüsselt wurde, wird ein passender privater Schlüssel gesucht, mit dem der geheime Schlüssel entschlüsselt werden kann. Ist ein solcher Schlüssel gefunden, muss dieser noch mit Hilfe eines Passwortes entschlüsselt werden. Mit dem geheimen Schlüssel kann dann die Nachricht entschlüsselt werden. Anschließend werden die Signaturen überprüft. Dazu werden die Zertifikate zu den Schlüsseln benötigt, mit denen die Signaturen überprüft werden können. Diese Zertifikate liegen entweder im Zertifikatsspeicher oder werden auf demselben Weg wie die Zertifikate zur Verschlüsselung vom Zertifikatserver geholt. Nach der Überprüfung der Signaturen ist die Entschlüsselung der Nachricht beendet.

5.8 TimeStampserver

Eine wichtige Komponente des Systems ist der TimeStampServer. Dieser Server hat die Aufgabe die aktuelle CA-Zeit für andere Module zu liefern. Das hat den Vorteil, dass alle Komponenten mit derselben Zeit arbeiten. So ist es nicht möglich die Verschlüsselung oder die Signierung vor- oder nachzudatieren, oder ein schon abgelaufenes Zertifikat zu benutzen.
Wird eine Anfrage an den TimeStampServer gestellt, liest der TimeStampServer die Systemzeit des Rechners, auf dem das Modul läuft, aus und liefert diese an die anfragende Komponente zurück. Durch die Umlaufzeit im Netzwerk empfängt die anfragende Komponente nicht die aktuelle Zeit, sondern eine um einige Millisekunden abweichende Zeit. Dieser Unterschied ist aber zu vernachlässigen, da für die Überprüfung der Gültigkeit der Zertifikate die Abweichung keine große Rolle spielt.

5.9 Ablauf einer Verschlüsselung mit Stellvertreterzertifikat

Unter Verwendung der beschriebenen Komponenten läuft die Verschlüsselung wie in diesem Beispiel ab :

Benutzer S will eine Nachricht an den Benutzer E schicken. Dazu lädt S das Zertifikat von E von dem Zertifikatsserver ZS und prüft es mit der Zeit, die der TimeStampServer TS liefert. Bei der Überprüfung stellt S fest, dass E den Stellvertreter STE eingesetzt hat. S entscheidet, dass die Nachricht auf für STE lesbar sein soll. Dazu lädt S das Stellvertreterzertifikat von STE, dass E beantragt hat, von dem Zertifikatsserver. Darauf wird die Nachricht für E und STE verschlüsselt. Erhält STE die Nachricht kann er diese mit seinem privaten Stellvertreterchiffrierschlüssel entschlüsseln. Eine Antwort an S signiert STE mit seinem Stellvertretersignierschlüssel. S kann dann das Stellvertreterzertifikat testen, um sicher zu stellen, dass STE ein offizieller Stellvertreter von E ist.

Dieser Ablauf ist nur möglich, wenn beide Benutzer den von mir entwickelten Client benutzen, da dieser die nötigen X509 Zertifikatserweiterungen kennt und die verwendeten Protokolle zum Zugriff auf den Zertifikatsserver einsetzen kann. Dieser Client kann am besten in einem internen Netz benutzt werden, da bei jeder Ver- und Entschlüsselung auf den Zertifikatsserver zugegriffen wird. Bei Verwendung eines anderen Clients kann die Stellvertreterregelung nicht automatisch berücksichtigt werden. Da es sich bei den Stellvertreterzertifikaten um X509 Zertifikate handelt, können diese aber wie normale

Zertifikate behandelt werden. So ist es möglich eine Nachricht für einen Stellvertreter zu verschlüsseln. Diese Zertifikate müssen allerdings manuell verteilt oder über ein geeignetes Medium zugänglich gemacht werden. Zur Entschlüsselung der Daten müssen die nötigen privaten Schlüssel aus den Schlüsseldateien extrahiert werden, da diese nicht in einem Standardformat gespeichert sind.

6 Implementierung des Detailentwurfs

6.1 Sprache der Implementierung

Für die Implementierung des Entwurfs habe ich mich für die Programmiersprache Java entschieden. In den folgenden Unterkapiteln möchte ich kurz einen Überblick über Java und die verwendeten Techniken und Bibliotheken geben.

6.1.1 Historie von Java

Die Geschichte von Java begann bei Sun Microsystems im Jahr 1991. Als Bestandteil eines Projektes für interaktives Fernsehen entstand eine Interpretersprache mit dem Namen Oak. Nach der Entwicklung der ersten grafischen Browser, wurde die Weiterentwicklung auf die Internetverbreitung von Oak fokussiert. Im Herbst 1994 wurde die erste Version eines Oak-fähigen Browsers fertiggestellt, die kleine Programme, genannt „Applets“, aus dem Internet laden und ausführen konnte. Wenig später musste die Sprache Oak umbenannt werden, da der Name schon vergeben war. Es wurde der Name Java gewählt. Den Durchbruch hatte die Sprache als der Netscape für die zweite Version des Browsers „Navigator“ Java von Sun lizenzierte. Seit 1996 steht das „Java Development Kit“ zur Verfügung, mit dem jede Person selbst Javaprogramme erstellen kann.

6.1.2 Eigenschaften von Java

Die Entwickler haben Java vollständig neu entwickelt. Die Syntax wurde dabei an die Syntax von C und C++ angelehnt. Java ist sowohl eine objektorientierte als auch eine klassische imperative Programmiersprache nach dem Vorbild von C. Obwohl C++ denselben Anspruch erhebt, unterscheidet sich Java deutlich von C++. Java integriert eine große Anzahl von Features wie Multithreading, strukturiertes Exceptionhandling oder grafische Fähigkeiten.
Java profitiert zusätzlich davon, dass einige Features, die C++ bietet, nicht integriert sind. Zu diesen nicht integrierten Features gehört unter anderem die Mehrfachvererbung, Pointer und die separaten Headerdateien. Obwohl diese Features fehlen, bietet Java genügend Reserven um auch größere Projekte und anspruchsvollere Aufgaben zu realisieren.
Auf einige Features von Java möchte ich hier näher eingehen. So verfügt Java über ein automatisches Speichermanagement. Für das Erzeugen von Objekten ist der explizite Aufruf des new-Operators erforderlich. So bald der Speicher nicht mehr verwendet wird, also ein Objekt nicht mehr referenziert ist, wird das Objekt an den Garbage-Collector weitergegeben, der als niedrigpriorisierter Hintergrundprozess läuft. Der Garbage-Collector gibt den durch die Objekte belegten Speicher an das Laufzeitsystem zurück. Dadurch können viele Fehler, die bei C und C++ dadurch entstehen, dass die Entwickler selbst für das Speichermanagement verantwortlich sind, nicht auftreten.
Wie schon erwähnt bietet Java ein strukturiertes Exceptionhandling. Mit diesem Handling ist es möglich Laufzeitfehler zu erkennen und strukturiert abzuarbeiten. Dadurch ist jede Methode gezwungen jeden Laufzeitfehler abzuarbeiten oder an den Aufrufenden der Methode weiterzugeben, der dann seinerseits verpflichtet ist, den Fehler zu beheben.

6.1.3 Remote Method Invocation (RMI)

Die verschiedenen Module werden mit Hilfe der Remote Method Invocation (RMI) kommunizieren. Mit der Version 1.1 wurde die Programmierschnittstelle der RMI in das JDK aufgenommen. Wie der Name schon sagt, ermöglicht es RMI, Methoden von entfernten, das heißt irgendwo in dem Netzwerk befindlichen, Objekten auszuführen. RMI basiert auf einem Schichtenmodell. Die drei Schichten sind durch ihre Schnittstellen genau spezifiziert.

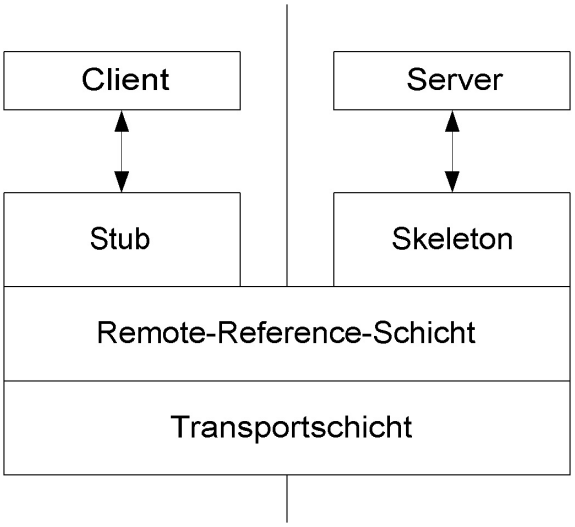

Abbildung 8 : Schichtenmodell von RMI

Die Transportschicht stellt die unterste Ebene dar. Sie stellt Verbindungen zwischen eigener und fremder virtueller Maschine her und überwacht diese Verbindung. Die Kommunikation auf dieser Ebene findet über Sockets statt. Die Verbindungspartner tauschen hier Informationen unter Verwendung zweier Standards aus. Dies ist zum einen Object Serialization und zum anderen das Hypertext Transfer Protocol (HTTP).
Die Transportschicht erhält ihre Anweisungen von der darüber liegenden Remote-Reference-Schicht. In dieser Schicht sind die Regeln für die Weiterleitung von Anfragen auf Remote Objekte implementiert.
Für Anwendungen, die Remote Objekte verwenden, ist weder die Transportschicht noch die Remote-Reference-Schicht sichtbar. Für den Zugriff auf entfernte Objekte dient die Stub/Skeleton-Schicht. Objekte werden mit Hilfe des Object Serialization-Mechanismus in Bitströme umgewandelt und können so über die Adressräume der verschiedenen JVMs hinaus transportiert werden.
Der Zugriff auf ein Remote Object erfolgt über einen sogenannten Stub. Ein Stub stellt einen lokalen Stellvertreter für ein entferntes Objekt dar. Bei einem Methodenaufruf kontaktiert der Stub die Remote-Reference-Schicht und verschickt die serialisierten Parameter. Der Stub wartet nun auf den Abschluss der Transaktion und empfängt den Rückgabewert des Aufrufs. Gegebenenfalls muss dieser in ein Objekt umgewandelt (deserialisiert) werden.
Auf der Serverseite besteht das Gegenstück zum Stub im Skeleton. Wenn ein Methodenaufruf vorliegt, meldet die Remote-Reference-Schicht dies dem entsprechenden Skeleton und der Methodenaufruf wird an das entsprechende Objekt weitergegeben. Die Weitergabe erfolgt

erst, nachdem das Skeleton die Parameter dekodiert hat. Der Rückgabewert des Methodenaufrufs wird wieder serialisiert und der Remote-Reference-Schicht übergeben.
Mit dem JDK wird ein Programm namens rmiregistry mitgeliefert. Mit Hilfe dieses Programms können Remote Objects auf dem Server angemeldet werden, um diese für Clients zugänglich zu machen.

6.1.4 zusätzlich verwendete Bibliotheken

Java verfügt seit der Version 1.1 über die Java Cryptography Architecture (JCA). Die erste Version dieser API stellte Schnittstellen für die Verwendung von digitalen Signaturen und Hashfunktionen zur Verfügung. Nach einer Erweiterung um X509 Zertifizierungsinfrastrukturen, wurde auch die API Java Cryptography Extension (JCE) eingeführt. Diese bot in der ersten Version schon Schnittstellen für symmetrische und asymmetrische Verschlüsselungen, Schlüsselgenerierung und Schlüsselaustausch. Bei der Entwicklung der APIs wurde darauf geachtet, dass zwei Designprinzipien eingehalten wurden. Es wurde gefordert, dass der Benutzer der API unabhängig von der Implementierung und von dem verwendeten Algorithmus ist. Zu diesem Zweck wurden die Implementierungen der einzelnen Algorithmen zu Java Paketen (packages) zusammen gefasst. Diese Pakete werden auch Cryptographic Service Provider (CSP) oder kurz Provider genannt. Der Benutzer hat durch diese Technik die Möglichkeit verschiedene Algorithmen verschiedener Hersteller mit den gleichen Methodenaufrufen zu benutzen. Es ändern sich im Aufruf nur einige Parameter.
Auch für diese Implementierung benötige ich einen Cryptographic Service Provider. Sun hat den neuesten Versionen schon einen relativ guten Provider mitgegeben. Allerdings fehlen in diesem Provider einige Verschlüsselungsalgorithmen, wie zum Beispiel der RSA-Algorithmus zur asymmetrischen Verschlüsselung. Deshalb habe ich mich entschieden einen zusätzlichen Provider zu benutzen. Meine Wahl fiel dabei auf den Provider „BouncyCastle“, der als kostenlose Version unter der Internet-Adresse http://www.bouncycastle.org zu finden ist. Dieser Provider verfügt über alle benötigten Algorithmen. In dieser Arbeit verwende ich die Version 1.20.

6.2 Datenstrukturen

In diesem Kapitel werde ich mich mit den Datenstrukturen beschäftigen, die auf der Festplatte oder ähnlichen Medien gespeichert werden. Dazu gehören die erstellten Schlüssel, die Zertifikate und die verschlüsselten Nachrichten.

6.2.1 Die Speicherung der Schlüssel

Die asymmetrischen Schlüssel werden in den Klassen AsymPrivateKey und AsymPublicKey gespeichert. Beide Klassen sind von der Klasse AsymKey abgeleitet. Diese Klasse enthält Datenfelder um den Namen und die E-Mail-Adresse des Schlüsselinhabers und die eindeutige Schlüssel-ID aufzunehmen. Diese Datenfelder sind vom Typ String. Die Klasse enthält zusätzlich ein Bytearray, in dem ein privater oder ein öffentlicher Schlüssel gespeichert werden kann. In der Klasse wird auch der Algorithmus und die Schlüssellänge gespeichert.

Durch die Speicherung des Algorithmus lässt sich dieser verändern. So ist es möglich nicht nur den in diesem System genutzten RSA-Algorithmus zu benutzen, sondern auch andere Algorithmen zu verwenden. Die Datenfelder können nur über den Konstruktor gesetzt und über Get-Methoden gelesen werden. Zusätzlich beinhaltet die Klasse die Methode „sign“. Mit dieser Methode werden die in einem Objekt dieser Klasse gespeicherten Daten einer Signaturberechnung zur Verfügung gestellt. Während die abgeleitete Klasse für private Schlüssel (AsymPrivKey) keine zusätzliche Datenfelder oder Methoden besitzt, verfügt die Klasse für öffentliche Schlüssel (AsymPubKey) noch über Speichermöglichkeiten für eine Signatur und den für die Signatur verwendeten Algorithmus. Diese erstellte Signatur soll sicherstellen, dass der Inhalt des Objektes nicht verändert wird. Die Signatur wird über alle Datenfelder des Objektes mit dem privaten Schlüssel, der zum gespeicherten öffentlichen Schlüssel passt, erstellt. Die Signatur lässt sich so mit dem im Objekt gespeicherten öffentlichen Schlüssel verifizieren.

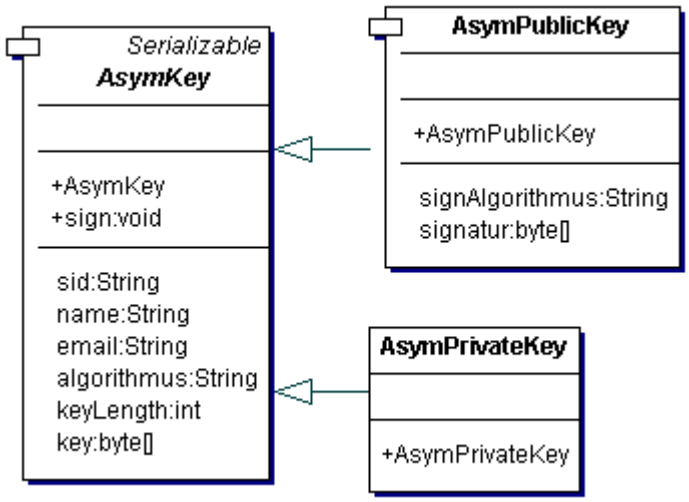

Abbildung 9 : UML-Darstellung der AsymKey-Klassen

6.2.2 Speicherung der verschlüsselten Nachrichten

Um in diesem System eine verschlüsselte Datei zu speichern, sind einige Klassen notwendig. Diese Klasse Message repräsentiert eine hybridverschlüsselte und signierte Nachricht. Dafür besitzt die Klasse drei Arrays, die über set- und get-Methoden gesetzt und gelesen werden können. Eines der Arrays ist ein Array für Objekte der Klasse KeyInfo, in dem für jeden Empfänger der Nachricht der symmetrische Schlüssel verschlüsselt ist. Ein weiteres Array beinhaltet Objekte der Klasse Signatur, in den die erstellten Signaturen gespeicherten werden. Die eigentlichen verschlüsselten Nachrichten werden als Instanz der Klasse SealedObject in einem weitern Array gespeichert.

Beide Klassen, KeyInfo und Signatur, sind abgeleitet von der abstrakten Klasse KryptInfo. Die Klasse KryptInfo verfügt über drei Datenmember in denen die jeweils notwendigen Informationen gespeichert werden können. Gespeichert werden in den Objekten dieser Klasse jeweils eine Schlüsselidentifikation, ein Algorithmus und ein Byte-Array. Je nach abgeleiteter Klasse werden diese Informationen unterschiedlich interpretiert. Da die Klasse KeyInfo zur Speicherung von verschlüsselten symmetrischen Schlüsseln einer Hybridverschlüsselung

dient, wird dieser in dem Byte-Array abgelegt. In dem Feld der Schlüsselidentifikation wird die ID des Schlüssels gespeichert, mit dem der symmetrische Schlüssel wieder entschlüsselt werden kann. Als Algorithmus wird in einem KeyInfo-Objekt der Algorithmus gespeichert, für den der symmetrische Schlüssel erstellt und mit dem die symmetrisch verschlüsselte Nachricht verschlüsselt wurde. In der Klasse Signatur werden dagegen in dem Byte-Array die erstellten Signaturen gespeichert. Das Algorithmus-Feld nimmt in der Klasse Signatur den Algorithmus auf, mit dem die Signatur erstellt worden ist. Im dritten Feld wird dann die Schlüssel-ID des Schlüssels gespeichert, mit dem die Signatur erstellt und überprüft werden kann.

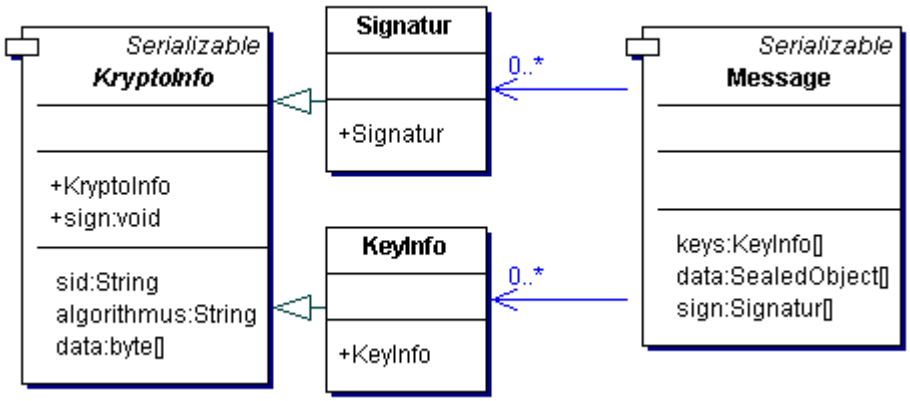

Abbildung 10 : UML-Darstellung der Klassen zur Speicherung einer Nachricht

6.2.3 Speicherung der Stellvertreterregelung

Die dritte Datenstruktur, auf die ich hier genauer eingehen möchte, ist die Klasse Stellvertreter, die Informationen über die Stellvertreterregelung enthält. Dazu verfügt die Klasse über einige Datenfelder und Methoden. Zu den Datenfeldern gehört ein Objekt der Klasse BigInteger, in dem die Seriennummer des zu vertretenden Zertifikats gespeichert werden kann. Ein weiteres Datenfeld nimmt einen aus dem zu vertretenden Zertifikat berechneten Hashwert, der auch als Fingerprint des Zertifikats bezeichnet wird, auf. Neben diesen beiden Datenfeldern existiert noch ein weiteres Feld, in dem die Dauer der Gültigkeit gespeichert werden kann. Diese Information ist nur von Bedeutung, wenn ein Stellvertreter-Objekt für ein Antrag auf ein Stellvertreterzertifikat erzeugt wird. Wird das Stellvertreter-Objekt in einem Zertifikat eingebettet, wird die Dauer der Gültigkeit der Stellvertreterregelung durch die Gültigkeit des Zertifikats vorgegeben.
Die Klasse verfügt zusätzlich über einige Methoden. So besitzt die Klasse eine Methode, die aus einem zu vertretenden X509 Zertifikat ein Stellvertreter-Objekt mit allen wichtigen Informationen erzeugt. Bei der Zertifikatserstellung wird ein Stellvertreter-Objekt in eine standardisierte Kodierung (DER) umgewandelt. Um diese Kodierung durchzuführen, besitzt die Klasse die Methode getDERObject, die die Informationen des Objektes in die gewünschte DER-Kodierung umsetzt. Aus den DER-kodierten Informationen, die in Form eines Byte-Arrays vorliegen, lässt sich mit Hilfe der statischen Methode „getStellvertreter“ wieder ein Stellvertreter-Objekt erzeugen. Auch hier möchte ich die Klasse als UML-Diagramm angeben:

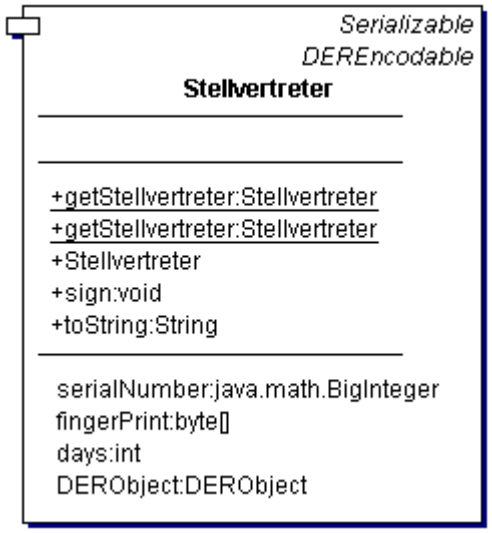

Abbildung 11 : UML-Darstellung der Stellvertreter-Klasse

6.3 Kommunikation

An dieser Stelle möchte ich auf die nötigen Kommunikationen zwischen den Modulen eingehen. Da für die Kommunikation RMI genutzt wird, ist für jede Verbindung ein Java-Interface notwendig. Bevor ich die Interfaces im einzelnen erläutere, folgt hier ein Überblick über alle RMI-Kommunikationen.

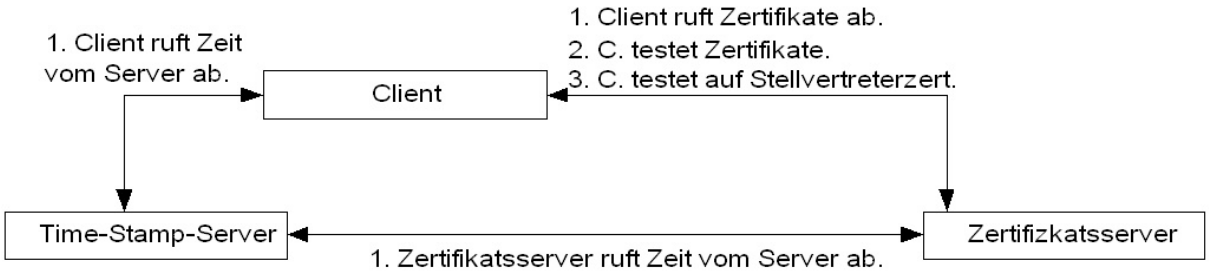

Abbildung 12 : Übersicht über RMI Kommunikationen

In diesem System existieren zwei Module, die über die RMI-Technologie Methoden zur Verfügung stellen. Als erstes werde ich hier den TimeStampServer vorstellen. Dieser Server liefert, wie man aus dem Diagramm erkennen kann, an einen anrufenden Client die aktuelle Zeit zurück. Das nötige Interface sieht dazu wie folgt aus:

```
public interface TimeRequest extends Remote {
  public Date GetServerTime() throws RemoteException;
}
```

Es existiert nur eine einzige Methode, die eine simple Funktionssignatur besitzt. Diese Methode liefert die aktuelle Zeit mit Hilfe eines Date-Objektes zurück. Wichtig für die korrekte Funktion ist das Ableiten des Interfaces von der Klasse Remote. Dadurch wird sichergestellt, das die Methode von einer anderen Applikation aufgerufen werden kann. Sollte bei diesem Aufruf irgendein Fehler auftreten, wird eine RemoteException ausgelöst, die mit dem Schlüsselwort „throws“ an die aufrufende Instanz weiter geleitet wird.

Das zweite Modul, das mit dieser Technik Methoden für andere Applikationen zur Verfügung stellt, ist der Zertifikatsserver, auf dem die erstellten Zertifikate veröffentlicht werden. Dazu veröffentlicht der Server drei Instanzen. Bei der ersten Instanz handelt es sich um eine Instanz einer Klasse, die das CertRequest-Interface implementiert.

```
public interface CertRequest extends Remote {
  public X509Certificate getCertificateByMail(String mail,
     boolean sign) throws RemoteException;
  public X509Certificate getCertificateByMail(String name,
     String mail, boolean sign) throws RemoteException;
  public X509Certificate getCertificateBySID(String sid)
     throws RemoteException;
  public X509Certificate getCertificate(BigInteger sn)
     throws RemoteException;
}
```

Durch dieses Interface ist es anderen Applikationen möglich Zertifikat zu suchen und zu laden. Jede Methode bietet dem Aufrufer andere Suchparameter an. So lässt sich mit der Methode „getCertificate“ ein Zertifikat mit der angegebenen Seriennummer suchen, während „getCertificateBySID“ ein Zertifikat mit einer bestimmten Schlüsselidentifikation zurück liefert. Die Methode „getCertificateByMail“ kann ein Zertifikat für einen bestimmten Besitzer suchen. Dabei kann entweder eine E-Mail-Adresse oder eine Kombination aus Name und E-Mail-Adresse des Besitzers des Zertifikats angegeben werden. In beiden Aufrufen wird noch ein boolean-Flag mitgegeben, das angibt, ob es sich um ein Zertifikat zur Überprüfung einer Signatur oder zum Verschlüsseln von Schlüsseln handeln soll.
Eine weitere Klasse, von der eine Instanz veröffentlicht wird, implementiert das CertTest-Interface. Die Methode „getCertificateStatus“ dient zum Testen von Zertifikaten auf Sperrungen.

```
public interface CertTest extends Remote {
   public String getCertificateStatus(BigInteger sn)
     throws RemoteException;
}
```

Der Antwort-String enthält das Ergebnis des Tests des Zertifikats mit der angegebenen Seriennummer.
Die dritte Instanz ist eine Instanz, deren Klasse zur Veröffentlichung der Stellvertreterzertifikate dient. Dazu implementiert die Klasse das CertStellRequest-Interface.

```
public interface CertStellRequest extends Remote {
  public X509Certificate getCertificate(BigInteger sn)
     throws RemoteException;
  public X509Certificate getCertificate(String sid)
     throws RemoteException;
}
```

Mit der Methode „getCertificate(BigInteger)“ ist es möglich ein Stellvertreterzertifikat zu suchen, das das Zertifikat mit der angegebenen Seriennummer vertritt. Der Aufruf mit dem String-Parameter „sid“ liefert ein Zertifikat zurück, dessen Schlüsselidentifikation mit der angegebenen ID identisch ist.

6.4 Implementierung der einzelnen Module

6.4.1 Schlüsselerzeugung

Auf der grafischen Oberfläche braucht der Benutzer Eingabemöglichkeiten für seine Benutzerdaten, wie Namen und E-Mail-Adresse. Zusätzlich müssen Einstellmöglichkeiten für die Schlüssellänge und das Verzeichnis, in dem die fertigen Schlüssel gespeichert werden sollen, vorhanden sein. Um die privaten Schlüssel vor unbefugter Benutzung zu sichern, werden diese noch verschlüsselt. Es besteht auch die Möglichkeit zu wählen, welche Schlüsselpaare erstellt werden sollen. Eine Schaltfläche startet dann den Erzeugungsvorgang.

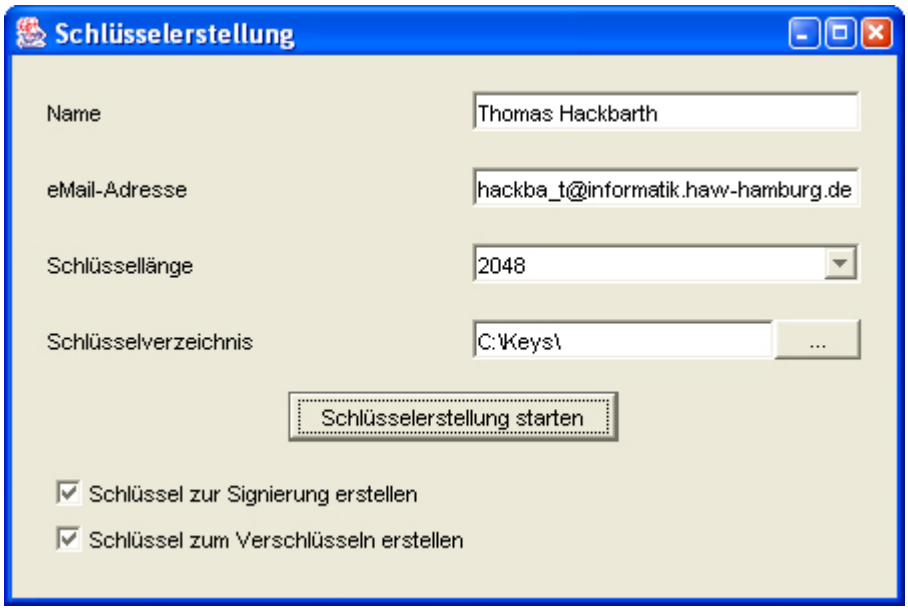

Abbildung 13 : Screenshot der Schlüsselerstellung

Wird der Button betätigt, wird die Schlüsselerstellung gestartet, in dem eine Instanz der Klasse KeyPairGenerator erzeugt wird. Als Parameter wird der Algorithmus, für den die Schlüssel erstellt werden soll, mitgegeben. In diesem System wird das asymmetrische Verfahren RSA eingesetzt. Die Erzeugung sieht so aus:

```
KeyPairGenerator rsaKeyGen = KeyPairGenerator.getInstance("RSA");
```

Ist der Generator erzeugt, wird dieser initialisiert. Dies ist notwendig, um die Länge der Schlüssel, die erzeugt werden, festzulegen. An den Generator wird auch ein Zufallsgenerator übergeben, mit dem der zufällige Schlüssel erzeugt wird. Benutzt wird hier eine besondere Art von Zufallsgeneratoren, sogenannte sichere Zufallsgeneratoren. Der Zufallsgenerator aus dem Java-Paket java.util wird mit der aktuellen Uhrzeit initialisiert. Daher sind die Zufallszahlen reproduzierbar und somit nicht für den Einsatz in kryptografischen Anwendungen geeignet. Als Parameter wird hier die Methode, auf der der Zufallsgenerator basiert, übergeben:

```
SecureRandom zufall = SecureRandom.getInstance("SHA1PRNG");
rsaKeyGen.initialize(keyLaenge,zufall);
```

Der Parameter SHA1PRNG steht für einen auf dem Hashalgorithmus SHA-1 basierenden Zufallsgenerator. Nach der Initialisierung des KeyPairGenerators können neue Schlüsselpaare erzeugt werden.

Um verschiedene Schlüssel des selben Benutzers unterscheiden zu können, wird bei jeder Schlüsselerstellung eine Schlüsselidentifikation erstellt. Gebildet wird die Schlüsselidentifikation aus dem eingegebenen Namen, der E-Mail-Adresse, dem verwendeten Algorithmus und dem öffentlichen Schlüssel, in dem aus allen Daten ein MD5-Hashwert berechnet wird. Für die Hashberechnung wird ein MessageDigest-Objekt erzeugt, an das die einzelnen Daten als Byte-Array übergeben werden. Dieses Objekt berechnet aus den Daten den Hashwert und liefert diesen als Ergebnis zurück. Der md5hash kann dann in ein String umgewandelt werden und als Schlüsselidentifikation verwendet werden.

```
MessageDigest md5 = MessageDigest.getInstance("md5");
md5.update(jtName.getText().getBytes());
md5.update(jtMail.getText().getBytes());
md5.update(keys.getPublic().getAlgorithm().getBytes());
md5.update(keys.getPublic().getEncoded());
byte[] md5hash = md5.digest();
```

Da die Erzeugung jedes Schlüsselpaars gleich abläuft, wird die Erzeugung in eine Methode verlagert. Diese Methode wird für jedes vom Benutzer gewünschtes Schlüsselpaar ausgeführt. Als Parameter werden an die Methode der KeyPairGenerator und ein String übergeben, der die Art des Schlüsselpaars beinhaltet. So steht der Inhalt „sign“ für einen Signierschlüssel und „ciph“ für einen Chiffrierschlüssel.

Wird die Methode aufgerufen wird als erstes mit dem KeyPairGenerator ein Schlüsselpaar erzeugt. Java speichert das Schlüsselpaar in einer Instanz der Klasse KeyPair.

```
KeyPair  keys = rsaKeyGen.generateKeyPair();
```

Aus diesem Schlüsselpaar lassen sich nun mit den Aufrufen keys.getPublic() und keys.getPrivate() die öffentlichen (Klasse PublicKey) und die privaten Schlüssel (Klasse PrivateKey) ermitteln. Sind die Schlüssel ermittelt, werden diese in eine Datei gespeichert. Dazu werden die Schlüssel in einem speziellen Format in ein Bytearray umkodiert. Für die öffentlichen Schlüssel wird das X509-Format benutzt, während für den privaten Schlüssel das PKCS8-Format genutzt wird. Für diese Umkodierung besitzen die beiden Schlüsselklassen die Methode getEncoded().

Bevor die umkodierten Schlüssel in eine Datei geschrieben werden, wird der private Schlüssel noch gegen unbefugte Benutzung geschützt. Dazu wird dieser mit einem symmetrischen Verfahren verschlüsselt. Der zum Verschlüsseln benötigte Schlüssel wird aus einem Passwort gebildet, das von dem Benutzer mit Hilfe dieses Dialoges eingegeben wird.

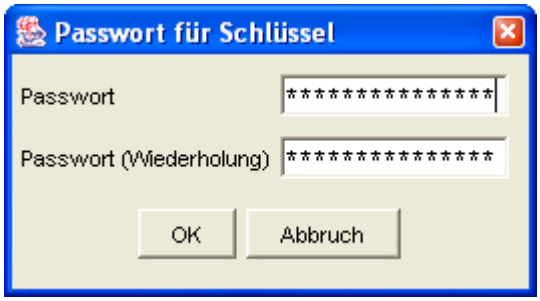

Abbildung 14 : Passworteingabe bei der Schlüsselerstellung

Der Dialog darf nur verlassen werden, wenn in beiden Eingabefeldern das gleiche Passwort steht oder der Abbruch Knopf gedrückt wird. Beim Abbrechen der Passworteingabe wird auch die Erstellung der Schlüsseldateien abgebrochen. Mit dieser Form der Eingabe des Schlüssels ist es möglich für den Signierschlüssel ein anderes Passwort als für den Chiffrierschlüssel festzulegen. Passwörter werden in Java nicht als Strings, sondern als Char-Array dargestellt. Dies hat den Vorteil, dass die Passwörter explizit aus dem Speicher gelöscht bzw. überschrieben werden können. Das Löschen eines Strings führt nur zum Löschen des Verweises auf diesen String, aber nicht zur Löschung der eigentlichen Daten.
Um den privaten Schlüssel mit einem Char-Array zu verschlüsseln, wird in Java die Password Based Encryption (PBE) verwendet. Erstellt werden die benötigten Schlüssel aus dem Array mit Hilfe von Hashfunktionen. Für die Benutzung der PBE sind neben dem Array noch zwei weitere Parameter notwendig. Diese Parameter erhöhen die Sicherheit des Schlüssels. Einer dieser Parameter ist die Rundenzahl, die die Anzahl der zusätzlichen Hashberechnungen angibt. Dieser Parameter wird als Iteration bezeichnet. Der zweite Parameter ist ein Byte-Array aus dem bei jeder Hashberechnung Werte mit in die Berechnung einfliesen. Dieses Array wird auch Salt genannt. Die Parameter Salt und Iteration müssen beim Ver- und Entschlüsseln identisch sein, da sonst aus dem Passwort unterschiedliche Schlüssel erstellt werden würden und damit die korrekte Entschlüsselung nicht funktioniert.
Um eine solche Verschlüsselung in Java zu benutzen, sind einige Schritte auszuführen. Aus dem Char-Array, das das Passwort enthält, wird zunächst in eine interne Datendarstellung umgewandelt.

```
PBEKeySpec keySpec = new PBEKeySpec(passwortarray);
```

Aus dieser Datendarstellung wird mit Hilfe einer Schlüsselfactory ein geheimer Schlüssel erstellt.

```
SecretKeyFactory keyFact =
SecretKeyFactory.getInstance("Algorithmus");
SecretKey secKey = keyFact.generateSecret(keySpec);
```

Die bereits beschriebenen Parameter werden in einen Parameterobjekt zusammengefasst.

```
PBEParameterSpec parameter = new PBEParameterSpec(salt,iteration);
```

Mit diesen Objekten kann nun die Verschlüsselung begonnen werden. Dazu wird ein Verschlüsselungsobjekt für den verwendeten Algorithmus angelegt und initialisiert.

```
Cipher pbe = Cipher.getInstance("Algorithmus");
pbe.init(modus, secKey, parameter);
```

Mit dem Modus lässt sich hier festlegen, ob die Daten ver- oder entschlüsselt werden sollen. Der verwendete Algorithmus setzt sich aus einer Hashfunktion, mit dem der Schlüssel erstellt wird, und einem symmetrischen Verschlüsselungsverfahren, mit dem die Daten verschlüsselt werden. In diesem System benutze ich den Algorithmus „PBEWithSHAAnd128BitRC4“. Das bedeutet, dass der Schlüssel mit der Hashfunktion SHA erstellt und mit dem Verfahren RC4 mit einer Schlüssellänge von 128 Bit verschlüsselt wird.
Mit dem initialisierten Objekt kann nun der private Schlüssel verschlüsselt werden. Dazu wird der als Bytearray kodierte Schlüssel an das Objekt übergeben. Nach der Verschlüsselung wird wieder ein Byte-Array zurückgeliefert.

```
byte[] cryptedKey = pbe.doFinal(privateKey.getEncoded());
```

Bevor die Schlüssel gespeichert werden, wird eine Signatur für den öffentlichen Schlüssel angelegt. Dafür ist es notwendig ein Objekt der Klasse Signatures zu instanzieren. Ähnlich wie das Verschlüsselungsobjekt wird auch dieses Objekt mit Hilfe einer Factorymethode erzeugt. Als Algorithmus lässt sich hier eine Kombination eines asymmetrischen Verschlüsselungsverfahren mit einer kryptografischen Hashfunktion. Mit der Hashfunktion wird ein Hashwert gebildet, der dann verschlüsselt wird. In diesem System verwende ich die Kombination SHA1WithRSA. Dies bedeutet, dass aus den Daten mit der kryptografischen Hashfunktion SHA1 ein Hashwert gebildet wird, der dann mit dem RSA-Algorithmus verschlüsselt wird.
Nach der Verschlüsselung werden die Schlüssel gespeichert. Benutzt werden dazu die schon erwähnten Klassen AsymPrivateKey und AsymPublicKey. Für jedes erstellte Schlüsselpaar werden zwei Dateien angelegt. In einer Datei wird das AsymPrivateKey-Objekt serialisiert, während in die andere Datei das AsymPublicKey-Objekt geschrieben wird. Der Dateiname bildet sich aus der Schlüsselidentifikation und dem als Schlüsselunterscheidung übergebenden String (sign oder ciph). Für den privaten Schlüssel wird die Dateiendung .private benutzt und für den öffentlichen Schlüssel die Dateiendung .public. Sind die Schlüssel gespeichert, ist die Schlüsselerstellung beendet.

6.4.2 Der Zertifikatsantrag

Nach der Erstellung der Schlüsseldateien kann im nächsten Schritt für diese Schlüssel ein Zertifikatsantrag gestellt werden. Dazu werden die Dateien, in denen die öffentlichen Schlüssel gespeichert sind, an die lokale RA weitergeleitet. Der Antrag wird mit folgender Applikation erstellt:

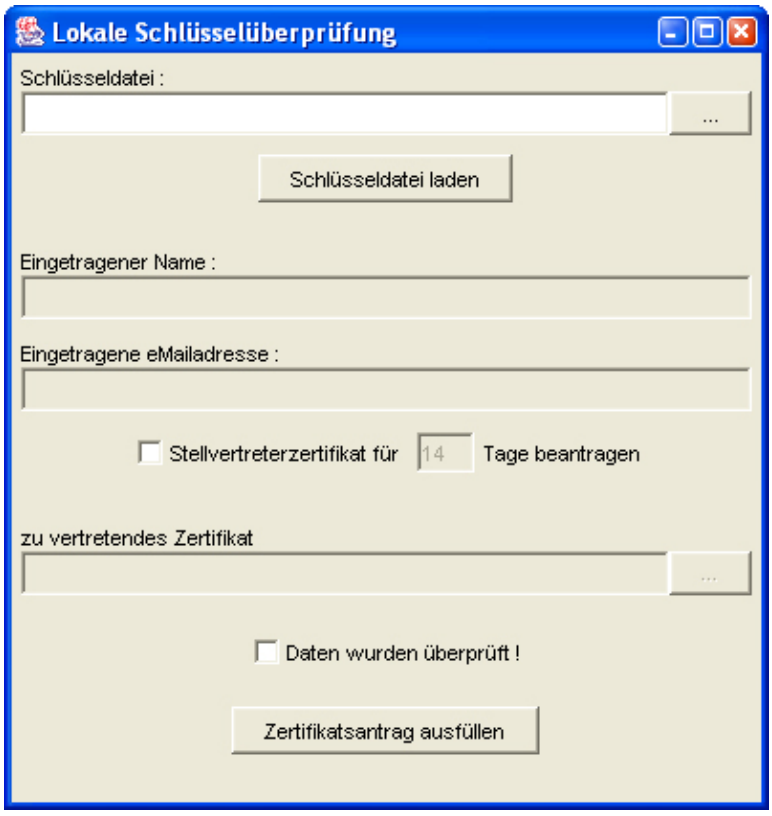

Abbildung 15 : Screenshot der Applikation zum Stellen von Zertifikatsanträgen

Im oberen Teil der Applikation kann der Schlüssel, für den der Antrag erstellt werden soll, ausgewählt und geladen werden. Beim Laden wird das AsymPubKey Objekt, das in die Datei serialisiert wurde, wieder eingelesen. Bevor der Schlüssel verwendet werden kann, wird die im Schlüssel-Objekt gespeicherten Signatur überprüft. Ist die Signatur gültig werden die im Schlüssel gespeicherten Informationen in der Applikation gespeichert und angezeigt. Nach dem Laden ist der Administrator der lokalen RA dazu verpflichtet die angezeigten Angaben zu kontrollieren. Sind alle Angaben korrekt, bestätigt der Administrator die Richtigkeit, in dem dieser das Feld „Daten wurden überprüft !“ aktiviert. Ist dieses Feld aktiviert und der Schlüssel geladen, kann der Antrag gestellt werden. Der Antrag besteht aus einem Objekt der Klasse HaW.Message. Dabei wird das geladene Schlüsselobjekt mit einem symmetrischen Verfahren unter zu Hilfenahme eines Objektes der Klasse SealedObject verschlüsselt. Ich habe mich bei diesem System für die Verwendung des AES-Algorithmus entschieden. Der Schlüssel für die symmetrische Verschlüsselung wird dann mit dem öffentlichen Chiffrierschlüssel der Zertifizierungsstelle, der aus einem Zertifikat geladen wird, verschlüsselt. Aus den entstanden Daten und den Daten aus dem Zertifikat wird ein KeyInfo-Objekt erzeugt, das zusätzlich zu dem verschlüsselten SealedObject in ein Message-Objekt gespeichert wird. Nach dieser Speicherung wird dann eine Signatur über diese Elemente erzeugt. Dazu muss der private Signierschlüssel des Administrators der lokalen RA geladen werden. Beim Ladevorgang wird der Administrator aufgefordert das Passwort, mit dem der private Schlüssel verschlüsselt wurde, einzugeben. Dabei wird folgender Dialog verwendet, der auch bei allen zukünftigen Passworteingaben benutzt wird.

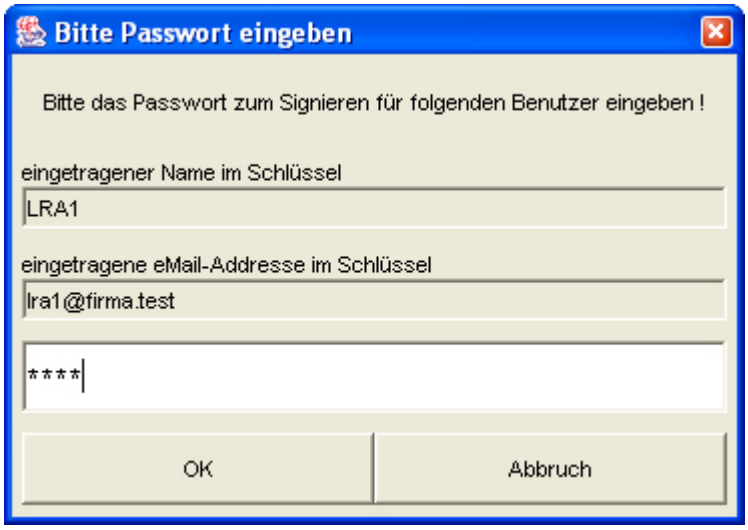

Abbildung 16 : Eingabe eines Passwortes, um einen privaten Schlüssel zu entschlüsseln

Nur wenn das Passwort korrekt eingegeben wurde, kann der private Schlüssel entschlüsselt werden. Ist der Schlüssel geladen, wird die Signatur erzeugt, die anschließend ebenfalls in dem Message-Objekt gespeichert wird. Zum Abschluss wird das Objekt in eine Datei serialisiert.

Neben der Erstellung von Anträgen auf „normale“ Zertifikate, werden in der lokalen RA auch die Anträge für Stellvertreterzertifikate ausgestellt. Dafür aktiviert der Administrator die

Option „Stellvertreterzertifikat für x Tage beantragen“. Darauf hin erhält der Administrator die Möglichkeit ein zu vertretendes Zertifikat und die Dauer der Stellvertreterregelung einzustellen.
Bei der Erstellung des Antrages wird aus dem zu vertretenden Zertifikat ein Stellvertreter-Objekt generiert, das alle wichtigen Informationen enthält. Dieses Objekt wird mit dem selben Verfahren und demselben Schlüssel, mit dem auch der öffentliche Schlüssel, für den das Zertifikat erstellt werden soll, verschlüsselt wurde, verschlüsselt. Das daraus entstehende SealedObject wird ebenfalls in das Message-Objekt gespeichert. Bei der Erstellung der Signatur wird das Stellvertreter-Objekt mit berücksichtigt. Das Message-Objekt kann auch mit dem Stellvertreterzusatz einfach in eine Datei serialisiert. Die in der Applikation verwendeten Parameter, wie der symmetrische Verschlüsselungs-algorithmus und die Dateien, in denen die Zertifikate und privaten Schlüssel gespeichert sind, sind in einer Datei mit der Endung „properties“ gespeichert. Auf die gespeicherten Werte kann über eine Instanz der Klasse ResourceBundle aus dem Java-Paket „java.util“ zugegriffen werden.

6.4.3 Die Zertifizierungsstelle

Nach der Erstellung der Zertifikatsanträge werden diese mit einem frei wählbaren Verfahren an die Zertifizierungsstelle weitergeleitet. Die Zertifizierungsstelle wird durch folgende Applikation dargestellt.

Abbildung 17 : Die Applikation der Zertifizierungsstelle

In oberen Teil lässt sich, wie in der Applikation zur Erstellung des Zertifikatsanträge, eine Datei, in der ein öffentlicher Schlüssel gespeichert ist, auswählen und laden. Der öffentliche Schlüssel kann aus einer Datei, die bei der Schlüsselerstellung erzeugt wurde, geladen werden. Mit dieser Methode ist es möglich für Schlüssel ein Zertifikat zu erzeugen, die nicht durch eine lokale RA überprüft wurden. Notwendig ist diese Methode zur Erstellung der

Zertifikate für lokalen RAs und die Zertifizierungsstelle selbst. Neben diesen Schlüsseln kann diese Applikation die Zertifikatsanträge für „normale“ und Stellvertreter-Zertifikate laden. Beim Laden eines Zertifikatantrags wird im ersten Schritt das, in der Datei gespeicherte, Objekt deserialisiert. Mit Hilfe der in diesem Objekt gespeicherten Informationen wird ein privater Chiffrierschlüssel gesucht, mit dem der verschlüsselte symmetrische Schlüssel entschlüsselt werden kann. Sollte die Zertifizierungsstelle nicht über einen passenden privaten Schlüssel verfügen, wird der Zertifikatsantrag abgelehnt, da der Inhalt nicht gelesen werden kann. Wenn ein passender Schlüssel gefunden wurde, wird auch hier der Benutzer der Applikation dazu aufgefordert, das Passwort zum Entschlüsseln des verschlüsselten asymmetrischen Schlüssel einzugeben. Mit dem entschlüsselten privaten Chiffrierschlüssel wird dann der symmetrische Schlüssel entschlüsselt, mit dem dann die verschlüsselten Nachrichten entschlüsselt werden können. Bei diesen Nachrichten handelt es um ein Objekt des Typs der Klasse AsymPubKey, das den zu zertifizierenden öffentlichen Schlüssel enthält. Bei den Anträgen für Stellvertreterzertifikate existiert in dem Message-Objekt ein zusätzliches Objekt vom Typ Stellvertreter, das die Informationen über das zu vertretende Zertifikat enthält. Im nächsten Schritt werden die im Message-Objekt gespeicherten Signaturen überprüft. Diese Signaturen werden mit öffentlichen Schlüsseln verifiziert, die aus Zertifikaten geladen werden, die sich in einem festgelegten Verzeichnis befinden. Sind alle Signaturen gültig, wird der öffentliche asymmetrische Schlüssel übernommen.
Nach dem Laden des zu zertifizierenden Schlüssels, kann der Administrator die Zertifikatserstellung mit dem Drücken des Buttons „Zertifikat erstellen“ starten. Darauf wird eine Instanz der Klasse X509V3CertificateGenerator erzeugt, die später das eigentliche X509 Zertifikat erzeugt.

```
X509V3CertificateGenerator xgen = new X509V3CertificateGenerator();
```

Diesem Generator werden dann alle Parameter des späteren Zertifikats zugewiesen. Dazu gehört der aus dem AsymPubKey-Objekt gelesene öffentliche Schlüssel und die aus dem selben Objekt ermittelten Informationen zu dem Namen und der E-Mail-Adresse des Besitzers des öffentlichen Schlüssels. Diese Daten werden in einem bestimmten Format an den Generator übergeben. Der Name und die E-Mail-Adresse werden in ein Objekt der Klasse X509Name gespeichert, das diese Daten in das für ein X509 Zertifikat vorgeschriebene DER Format konvertiert.

```
String toSignStr = "CN=" + apubkey.getName() + ",E=" +
apubkey.getEmail();
X509Name toSign = new X509Name(toSignStr);
xgen.setSubjectDN(toSign);
```

Ebenfalls wird dem Generator auch die Dauer der Gültigkeit des zu erzeugenden Zertifikats mitgeteilt. Ein X509 Zertifikat besitzt zwei Datenfelder um die Dauer der Gültigkeit des Zertifikats auszudrücken. In einem Datenfeld wird der Zeitpunkt gespeichert, ab wann das Zertifikat gültig ist. In dieser Applikation entspricht dieser Zeitpunkt dem Zeitpunkt an dem das Zertifikat ausgestellt wird.

```
xgen.setNotBefore(Calendar.getInstance().getTime());
```

Das andere Datenfeld gibt an, ab wann das Zertifikat ungültig wird. Dieser Zeitpunkt wird in diesem System dadurch festgelegt, in dem auf das aktuelle Datum die Dauer der Gültigkeit in Tagen (in diesem Codestück mal 100 Tagen) addiert wird.

```
Calendar oneY = Calendar.getInstance();
oneY.add(Calendar.DAY_OF_YEAR, 100);
xgen.setNotAfter(oneY.getTime());
```

Neben den Informationen über den zu zertifizierenden Benutzer werden dem Zertifikat noch Informationen über den Aussteller des Zertifikats hinzugefügt. Auch hier werden die Informationen in einem Objekt des Typs X509Name abgelegt und an den Generator übergeben.

Zusätzlich werden an den Generator auch die Erweiterungen übergeben, die das Zertifikat später haben soll. Für jedes Zertifikat wird der Verwendungszweck (KeyUsage) gespeichert.

```
int keyusage = 0;
if (keyFile.endsWith("sign")) {
   keyusage = X509KeyUsage.digitalSignature;
} else {
   keyusage = X509KeyUsage.keyEncipherment;
}
```

Dieser Verwendungszweck wird dann an den Zertifikatsgenerator als Extension übergeben.

```
X509KeyUsage keyus = new X509KeyUsage(keyusage);
DERObjectIdentifier keyIdent = new DERObjectIdentifier("2.5.29.15");
xgen.addExtension(keyIdent, false, keyus);
```

Wurde ein Stellvertreter-Objekt (stellver) aus einem Antrag gelesen, wird dieses Objekt auch als Extension verarbeitet.

```
if (stellver != null) {
   DERObjectIdentifier stIdent = new DERObjectIdentifier("2.5.29.60");
   xgen.addExtension(stIdent, false, stellver.getDERObject());
}
```

Nach den Extensions wird noch die eingegebene Seriennummer in Form eines BigIntegers an an den Generator weitergeleitet. Ist diese Information gesetzt, kann das Zertifikat erzeugt werden. Dazu ist es notwendig, das der private Signierschlüssel der Zertifizierungsstelle geladen und entschlüsselt wird. Mit dem entschlüsselten Schlüssel (pk) kann dann das Zertifikat unterschrieben werden.

```
X509Certificate xcert = null;
xcert = xgen.generateX509Certificate(pk);
```

Das entstandene Zertifikat wird dann als Byte-Array kodiert in einer Datei gespeichert. Mit diesem Schritt ist die Zertifikatserzeugung abgeschlossen.

6.4.4 Der Zertifikatsserver

Die erstellten Zertifikate können anschließend auf dem Zertifikatsserver veröffentlicht werden. Dies geschieht durch ein einfaches Kopieren der Zertifikatsdatei in ein bestimmtes Verzeichnis. Dieses Verzeichnis und andere Einstellungen werden dem Zertifikatsserver durch einen Einträge in einer Properties-Datei mitgeteilt. Beim Start der Applikation werden alle verfügbaren Zertifikate aus dem Verzeichnis geladen. Nach dem Laden eines Zertifikats wird dieses getestet. Dabei wird getestet, ob das Zertifikat noch nicht abgelaufen und von der

bekannten Zertifizierungsstelle unterschrieben ist. Fällt ein Test negativ aus, ist das Zertifikat ungültig und wird gelöscht. Im anderen Fall wird das Zertifikat in einer ArrayList gespeichert. Der Inhalt der Liste wird in der Applikation angezeigt. Diese Liste wird ebenfalls an eine Instanz einer Klasse weitergeleitet, die die in der Liste gespeicherten Zertifikate veröffentlicht, weitergeleitet. Bei dieser Klasse handelt es sich um eine Klasse, die das im Abschnitt Kommunikation beschriebene Interface zur Veröffentlichung von Zertifikaten („CertRequest“) implementiert.
Um die Fähigkeiten der Instanz anderen Applikationen zur Verfügung zu stellen, wird diese an einem Namensdienst angemeldet. Dieser Namensdienst liefert zu einer RMI-URL die genaue Speicheradresse der Instanz. Ohne den Namensdienst würde ein Zugriff auf die Instanz nicht möglich sein, da bei jeder Instanzierung ein anderer Speicherbereich genutzt wird. Als Namensdienst existiert im Java-Runtime-Enviroment (JRE) die Applikation „rmi-registry“. Diese muss gestartet werden, bevor dieser Server oder der später beschriebene TimeStampServer gestartet werden.
Die Bedienungsoberfläche des Zertifikatsservers bietet dem Benutzer einige Möglichkeiten:

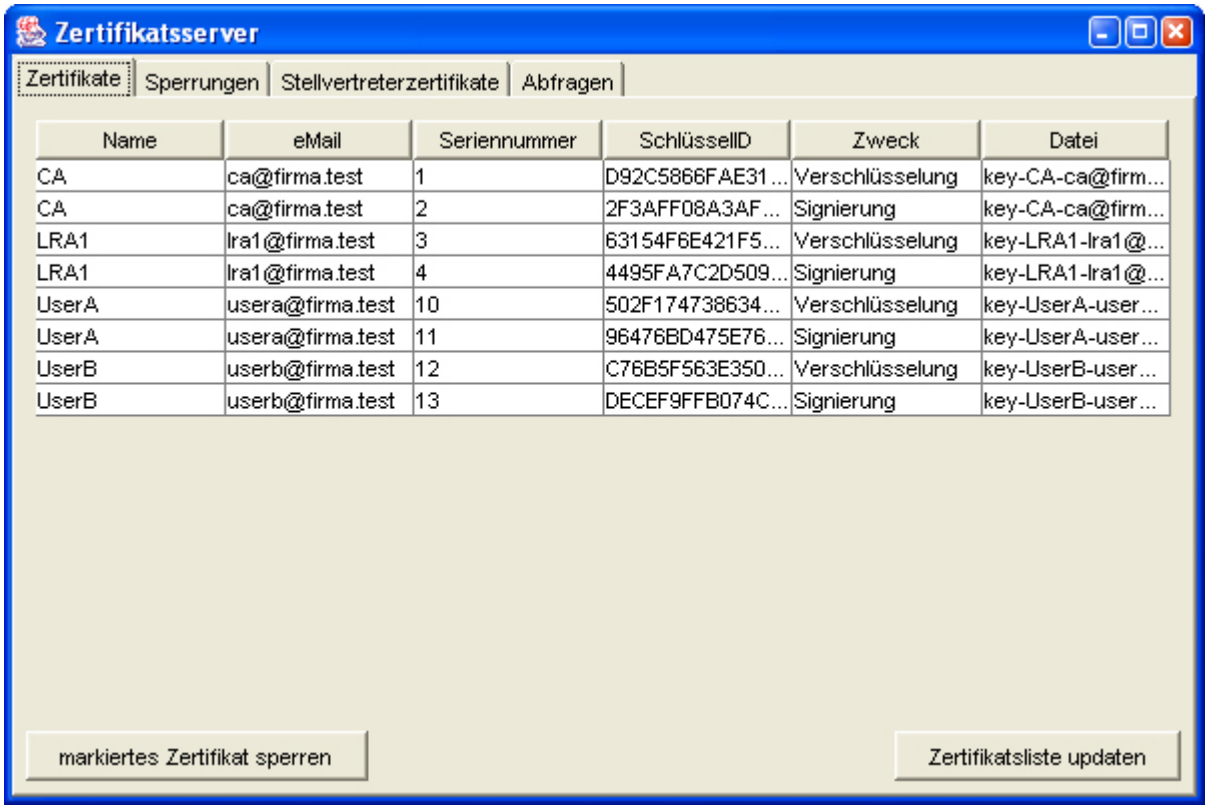

Abbildung 18 : Screenshot des Zertifikatsservers

So lässt sich die Liste der zu veröffentlichen Zertifikate durch einen Druck auf den Button „Zertifikatsliste updaten“ aktualisieren. Bei der Aktualisierung wird dieselbe Funktion ausgeführt, die auch beim Start der Applikation benutzt wird. So muss die Applikation beim Hinzufügen eines neuen Zertifikats nicht neu gestartet werden. Ebenfalls lassen sich mit dieser Applikation Zertifikate sperren. Dabei werden die Dateien der zu sperrenden Zertifikate aus dem Verzeichnis, in dem diese liegen, in ein anderes Verzeichnis verschoben. Auch die gesperrten Zertifikate werden in einer ArrayList gespeichert. Diese Liste wird

zusammen mit der Liste der existierenden, ungesperrten Zertifikate an eine Instanz einer Klasse, die das RMI-Interface „CertTest“ implementiert, übergeben. Auch diese Instanz wird an dem Namensdienst angemeldet. Diese Instanz liefert dann auf Anfragen den Status des Zertifikats zurück. Als Parameter der Anfrage wird die Seriennummer des zu testenden Zertifikats übergeben. Ist ein Zertifikat mit der angegebenen Seriennummer in der Liste mit den ungesperrten Zertifikaten, ist das Zertifikat gültig und es wird ein String mit dem Inhalt „good“ zurück geliefert. Wenn es kein Zertifikat mit der angegebenen Seriennummer in dieser Liste gibt, wird in der Liste der gesperrten Zertifikate gesucht. Existiert in der Liste der gesperrten Zertifikate ein Zertifikat mit der angegebenen Seriennummer, ist das Zertifikat zwar bekannt aber gesperrt und darf deshalb nicht verwendet werden. Die Antwort auf die Anfrage ist dann ein String mit dem Inhalt „revoked“. Wird in keiner der beiden Listen ein passendes Zertifikat gefunden, ist das zu testende Zertifikat unbekannt. Die Antwort enthält dann den String „unknown“.
Neben der Darstellung der veröffentlichten und der gesperrten Zertifikate, werden auch die verfügbaren Stellvertreterzertifikate angezeigt und veröffentlicht. Diese Zertifikate befinden sich ebenfalls in einem eigenen Verzeichnis. Auch das Einlesen dieser Zertifikate geschieht auf dem selben Weg, mit dem auch die „normalen“ Zertifikate eingelesen wurden. Die dadurch entstandene Liste wird an eine Instanz einer Klasse weitergegeben, die das Interface zur Veröffentlichung von Stellvertreterzertifikaten („CertStellRequest“) implementiert.
Der Zertifikatsserver listet zur Kontrolle des Betriebs alle auftretenden Anfragen und die darauf folgenden Antworten in einem Textfeld auf. Um die Übersicht zu verbessern, kann der Benutzer dieses Feld löschen.

6.4.5 Der TimeStampServer

Bevor ich auf den eigentlichen Client eingehe, möchte ich noch den TimeStampServer erwähnen. Beim Start wird eine Instanz einer Klasse erzeugt, die das Interface „TimeRequest“ implementiert. Auch diese Instanz wird an dem RMI-Namensdienst angemeldet.
Bei Anfragen an diesen Server wird dann die aktuelle Systemzeit des Rechners, auf dem diese Applikation läuft, zurückgeliefert.

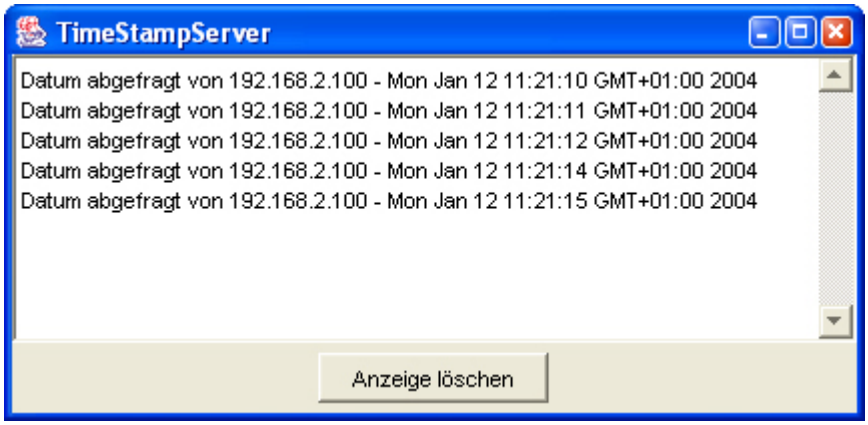

Abbildung 19 : Screenshot des TimeStampServers

Auch bei dieser Applikation werden in einem Textfeld, das auf Wunsch auch gelöscht werden kann, Informationen über jede Anfrage dargestellt.

6.4.6 Der Ver-/Entschlüsselungsclient

Mit dieser Applikation haben die Benutzer des Systems die Möglichkeit Nachrichten zu ver- und entschlüsseln. Dazu teilt sich die Applikation in zwei Teile. Mit Hilfe des ersten Teils lässt sich eine Nachricht verschlüsseln. Dem Benutzer stehen dabei ein paar Eingabefelder zur Verfügung.

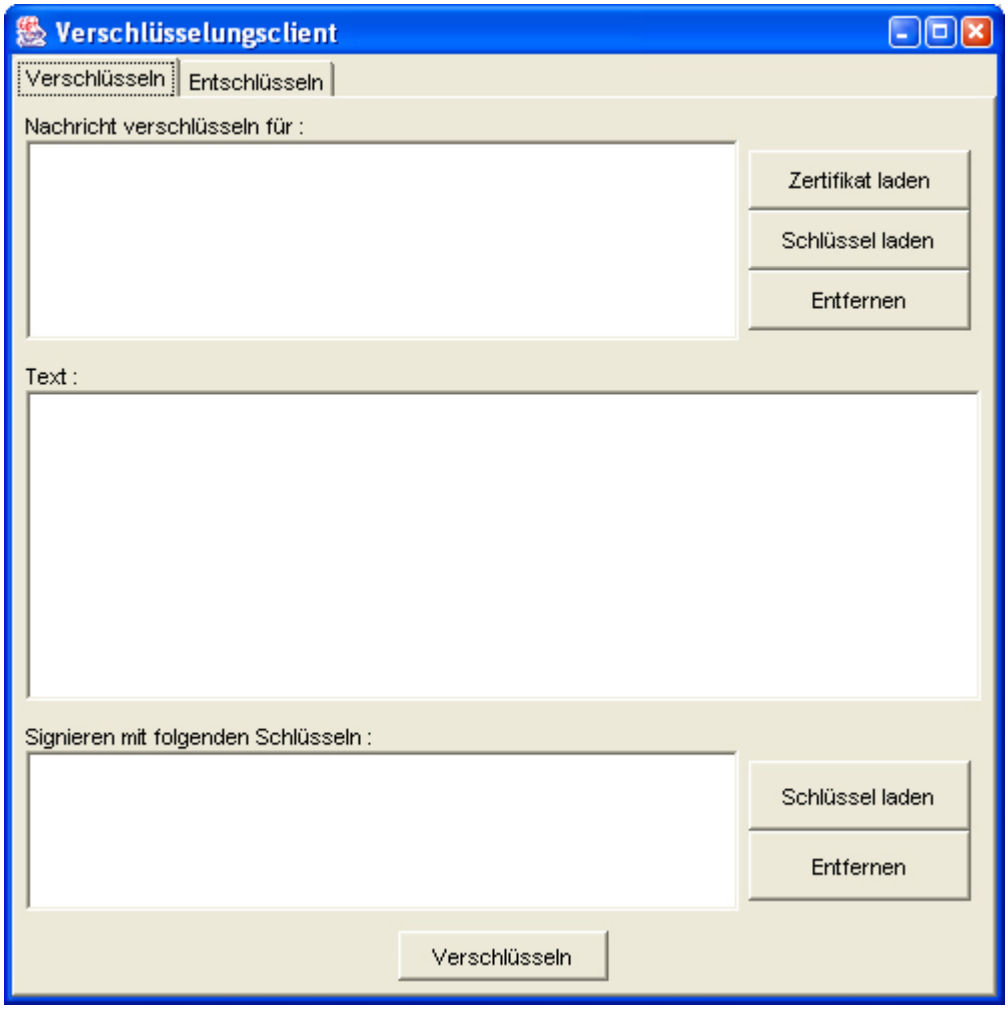

Abbildung 20 : Screenshot der Verschlüsselungskomponente des Clients

Im oberen Teil kann ein Benutzer die Schlüssel wählen, mit denen er die Nachricht verschlüsseln möchte. Dem Benutzer werden dazu zwei Möglichkeiten geboten. Nach einem Druck auf den Button „Schlüssel laden“ öffnet sich ein Dateiauswahldialog, mit dem der Benutzer öffentliche Chiffrierschlüssel-Dateien auswählen kann. Nach der Auswahl der Datei wird aus der Datei das AsymPubKey-Objekt geladen und in die Liste der Empfänger eingetragen. Die in dem Objekt gespeicherten Informationen, wie Name und E-Mail-Adresse, werden dann im dem Feld „Nachricht verschlüsseln für :“ angezeigt. Mit dieser Methode ist es möglich auch unzertifizierte Schlüssel zu verwenden. Dies kann zum Beispiel von der lokalen RA genutzt werden, um zu überprüfen, ob ein öffentliche Schlüssel zu einem

Antragssteller gehört. Über den Button „Zertifikat laden“ hat der Benutzer die Möglichkeit Zertifikate von Empfängern zu laden. Dazu wird folgender Dialog angezeigt:

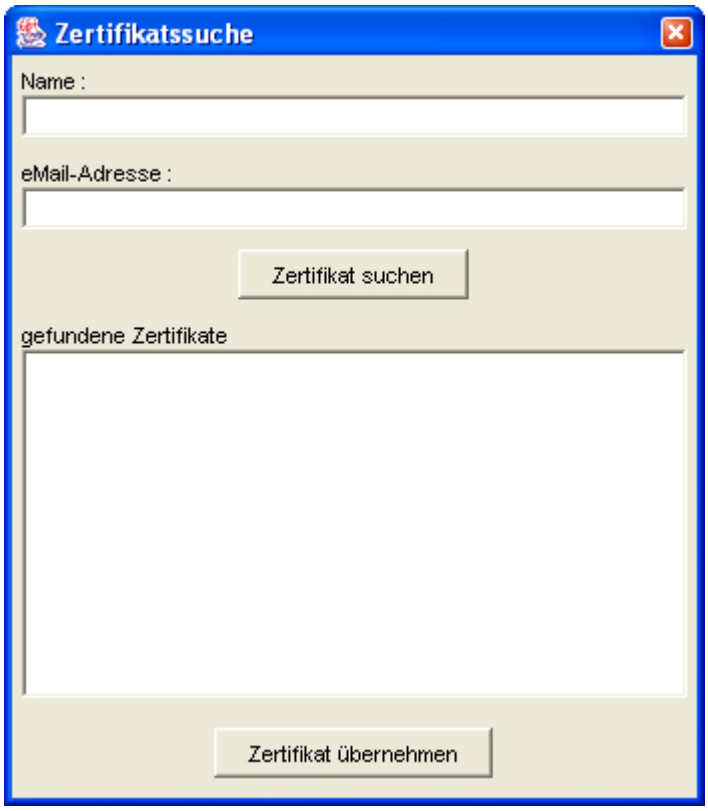

Abbildung 21 : Dialog zum Suchen und Laden von Zertifikaten

Im oberen Teil dieses Dialoges kann der Benutzer eingeben nach welchem Zertifikat gesucht werden soll. Dazu kann der Benutzer den Namen und die E-Mail-Adresse der zu suchenden Person eingeben. Durch einen Druck auf den Button „Zertifikat suchen“ wird dann nach einem zu den eingegebenen Daten passenden Zertifikat gesucht. Der erste Schritt der Suche besteht aus der Suche in einem internen Zertifikatsspeicher. Der Zertifikatsspeicher besteht aus einem Verzeichnis, in dem die bekannten Zertifikatsdateien abgelegt sind. Befindet sich ein passendes Zertifikat in dem Speicher wird es auf seine Gültigkeit getestet. Beim Test auf Gültigkeit werden drei Informationen getestet. Im ersten Schritt wird getestet, ob es sich bei dem Zertifikat um ein Zertifikat handelt, das die bekannte Zertifizierungsstelle ausgestellt hat. Ist es kein Zertifikat der Zertifizierungsstelle, wird es aus dem Speicher gelöscht und die Suche geht weiter. Ist es aber von der richtigen Zertifizierungsstelle ausgestellt, wird das Zertifikat weiter getestet. Im nächsten Test wird getestet, ob das gefundene Zertifikat überhaupt noch zeitlich gültig ist. Dafür wird versucht die aktuelle Zeit vom TimeStampServer zu laden. Sollte eine Anfrage nicht möglich sein, wird die eigene Systemzeit genutzt.

```
// xc ist das zu testende Zertifikat
boolean gueltig = false;
// Laden des CA-Zertifikats
X509Certificate root =
LoadCert.loadCert(resource.getString("rootCA"));
// Zertifikat gefunden - Nun Testen auf Datum und PublicKey !
```

```
if (LoadCert.checkCert(xc, root.getPublicKey(), aktuelleZeit)) {
  // Testen auf Sperrungen
  if (LoadCert.getCertStatus(xc)) {
    // Certificate ist OK !!!
    gueltig = true;
  }
}
```

Wenn das Zertifikat abgelaufen ist, wird es ebenfalls gelöscht. Ist das Zertifikat aber noch gültig, wird der letzte Test gestartet. Dabei wird versucht den Status des Zertifikats vom Zertifikatsserver zu laden. Dazu wird eine Anfrage an den Server gestellt, und die Seriennummer als Parameter übergeben. Ist das Zertifikat gesperrt, wird es gelöscht. Wenn die Anfrage „unknown“ zurück liefert, muss der Benutzer entscheiden, ob er ein unbekanntes Zertifikat benutzen möchte. Diese Abfrage tritt auch auf, wenn der Zertifikatsserver nicht erreichbar ist. Liefert die Anfrage „good“ als Antwort, werden die Daten des Zertifikats in dem Feld „Gefundene Zertifikate“ angezeigt.

Sollte in dem internen Zertifikatsspeicher kein Zertifikat gefunden werden, wird eine Anfrage an den Zertifikatsserver gestellt. Dazu wird die Methode „getCertificateByMail“ aus dem Interface „CertRequest“ benutzt. Als Parameter wird hier der eingegebene Name und die eingegebene E-Mail-Adresse benutzt. Der letzte Parameter vom Typ boolean, gibt an, ob es sich um ein Zertifikat zum Verschlüsseln oder Überprüfen einer Signatur handeln soll. Das angegebene „false“ bedeutet hier die Suche nach einem Zertifikat zum Verschlüsseln. Liefert der Aufruf kein Ergebnis, wird die Methode noch mal aufgerufen. Im diesem Aufruf, wird aber nur die eindeutige E-Mail-Adresse und der boolean-Wert als Parameter übergeben.

```
CertRequest cr = (CertRequest)
  Naming.lookup("rmi://server/CertRequest");
X509Certificate xc = cr.getCertificateByMail(Name, eMail, false);
if (xc == null) {
  xc = cr.getCertificateByMail(eMail, false);
}
```

Liefert auch diese Anfrage kein Ergebnis, ist die Suche beendet. Wurde aber ein Zertifikat bei der Anfrage an den Zertifikatsserver gefunden, wird dieses Zertifikat mit dem selben Ablauf, wie die Zertifikate aus dem Zertifikatsspeicher, getestet. Ist das gefundene Zertifikat gültig, wird es indem internen Zertifikatsspeicher gespeichert und ebenfalls in die Liste der gefundenen Zertifikate eingetragen.

Ist die Suche beendet, wird die Liste der gefundenen Zertifikate in dem Feld „gefundene Zertifikate“ dargestellt. Der Benutzer hat nun die Möglichkeit entweder eine neue Suche zu starten, den Dialog ohne Wahl eines Zertifikats zu beenden oder ein Zertifikat auszuwählen, in dem er dieses durch einen Klick auf den Eintrag markiert und dann auf den Button „Zertifikat übernehmen“ drückt. Nach dem Drücken des Buttons wird der Dialog geschlossen und das gewählte Zertifikat an den Verschlüsselungsclient übergeben. Bevor der Schlüssel, der in dem Zertifikat enthalten ist, zu der Liste der Empfänger hinzugefügt wird, wird noch eine Anfrage an den Zertifikatsserver gestellt. In dieser Anfrage wird für das gewählte Zertifikat ein Stellvertreterzertifikat gesucht. Liefert die Anfrage kein Stellvertreterzertifikat zurück, wird der Schlüssel zu der Liste der Empfänger hinzugefügt. Ist aber ein solches

Zertifikat vorhanden, erhält der Benutzer die Wahl, ob er seine Nachricht auch für den Stellvertreter verschlüsseln möchte. Möchte der Benutzer dies, wird der Schlüssel aus dem Stellvertreterzertifikat ebenfalls in die Liste der Empfänger eingetragen.
Neben dem Hinzufügen von Schlüssel hat der Benutzer auch die Möglichkeit die vorhandenen Schlüssel wieder aus der Liste zu entfernen. Dazu muss der Benutzer einen Schlüssel aus der angezeigten Liste mit einem Klick markieren und dann den Button „Entfernen“ drücken.
Der Benutzer hat auch die Möglichkeit Schlüssel festzulegen, mit denen er seine Nachricht signieren möchte. Um einen privaten Signierschlüssel zu laden, bietet die Applikation im unteren Teil den Button „Schlüssel laden“ an. Bei einem Druck auf diesen Button öffnet sich ähnlich wie beim Laden von öffentlichen Chiffrierschlüsseln ein Dateiauswahldialog, in dem Dateien ausgewählt werden können, die einen privaten Signierschlüssel enthalten. Nach der Auswahl der Datei wird das, in der Datei gespeicherte, AsymPrivKey-Objekt geladen und einer ArrayList hinzugefügt. Auch der Inhalt dieser Liste wird in der Applikation angezeigt. Identisch zum Löschen der Chiffrierschlüssel ist das Löschen der Signierschlüssel. Der Benutzer markiert einfach den zu löschenden Schlüssel und drückt dann den „Entfernen“-Button neben der Liste.
Im mittleren Bereich kann der Benutzer die zu verschlüsselnde Nachricht eingeben. Nach der Eingabe der Nachricht und der Wahl von mindestens je einem Signier- und Chiffrierschlüssels kann die Verschlüsselung durch das Betätigen des Buttons „Verschlüsseln“ gestartet werden. Die Verschlüsselung beginnt mit der Erstellung eines zufälligen symmetrischen AES-Schlüssels.

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
keygen.init(256);
SecretKey aesKey = keygen.generateKey();
```

Mit diesem Schlüssel wird dann die Nachricht mit Hilfe eines Verschlüsselungsobjekts vom Typ Cipher verschlüsselt. Das Ergebnis der Verschlüsselung ist dabei ein verschlüsseltes Objekt von Typ SealedObject.

```
Cipher aesCiph = Cipher.getInstance("AES");
aesCiph.init(Cipher.ENCRYPT_MODE, aesKey);
SealedObject sealed = new SealedObject(nachricht, aesCiph);
```

Mit dem selben Schlüssel und der selben Vorgehensweise wird auch die aktuelle Zeit, die in einem Objekt des Typs Date vorliegt, verschlüsselt. Die Zeit wird, genau wie beim Testen der Zertifikate, von dem TimeStampServer bezogen. Nur wenn der Server nicht erreichbar ist, wird die eigene Systemzeit genutzt. Diese beiden verschlüsselten Objekte werden dann in ein SealedObject-Array gespeichert, welches wiederum in einem Objekt der Klasse Message abgelegt wird.
Der zufällige symmetrische Schlüssel wird anschließend mit jedem ausgewählten öffentlichen Chiffrierschlüssel verschlüsselt. Das Ergebnis der Verschlüsselung wird in Form eines Byte-Arrays in einem KeyInfo-Objekt abgelegt. In diesem Objekt werden zusätzlich die Schlüsselidentifikation des verwendeten öffentlichen Schlüssels und der für die Verschlüsselung der Nachricht verwendete Algorithmus abgelegt. Alle entstandenen KeyInfo-

Objekte werden anschließend zu einem KeyInfo-Array zusammengefasst, welches auch an das schon erwähnte Message-Objekt übergeben wird.
Im nächsten Schritt wird mit jeden ausgewählten Signierschlüssel eine Signatur erzeugt. Um mit den privaten Schlüsseln die Signaturen zu erzeugen, muss der Benutzer dass Passwort für jeden genutzten Schlüssel eingegeben werden. Bei der Berechnung der Signatur werden alle KeyInfo-Objekte und die unverschlüsselten Nachrichten- und Zeit-Objekte verarbeitet. Die erstellte Signatur wird in einem Signatur-Objekt abgelegt. Dort werden ebenfalls der verwendete Signaturalgorithmus und die Schlüsselidentifikation des Schlüssels, mit der die Signatur erstellt wurde, abgelegt. Nach einer Zusammenfassung aller entstandenen Signatur-Objekte zu einem Array, wird dieses Array auch an das Message-Objekt übergeben.
Im letzten Schritt wird ein Dateiauswahldialog geöffnet, mit dem der Benutzer die Datei auswählen kann, in der die verschlüsselte Nachricht gespeichert werden soll. In die Datei, die mit Endung „crypt“ versehen wird, wird das Message-Objekt serialisiert. Sobald die Verschlüsselung abgeschlossen ist, wird der Benutzer darüber informiert. Im Anschluss darauf, werden die grafische Oberfläche und die internen Variablen gelöscht und auf ihren Ausgangswert zurückgesetzt.
Der zweite Teil der Applikation dient zur Entschlüsselung der Nachrichten. Durch einen Klick auf die Schaltfläche „Entschlüsseln“ wird die Anzeige gewechselt. Es kommen dann die Eingabemöglichkeiten zur Entschlüsselung zum Vorschein. Die Applikation sieht dann so aus:

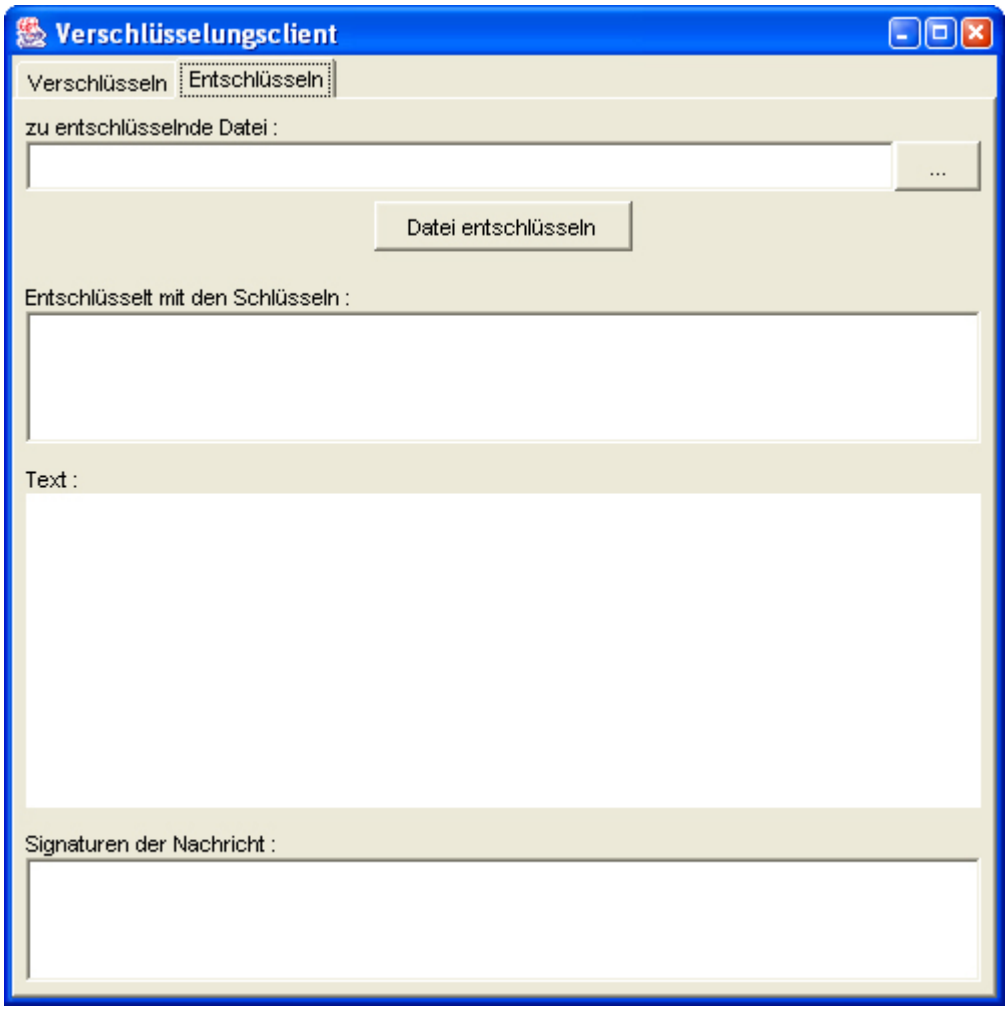

Abbildung 22 : Screenshot der Entschlüsselungskomponente des Clients

Im oberen Teil der Applikation kann eine Datei angegeben werden, die die verschlüsselte Nachricht enthält. Es ist ebenfalls möglich eine solche Datei mit Hilfe eines Dateiauswahldialogs zu wählen, indem der Benutzer den Button mit der Beschriftung „...“ betätigt.

Nach der Auswahl der Datei kann die Entschlüsselung mit dem Button „Datei entschlüsseln“ gestartet werden. Im ersten Schritt der Entschlüsselung wird aus der gewählten Datei das gespeicherte Objekt der Klasse Message geladen. Aus diesem Objekt werden anschließend die verschlüsselten symmetrischen Schlüssel, die in KeyInfo-Objekten gespeichert sind, genommen. Die in den Objekten gespeicherte Schlüsselidentifikation wird dann mit allen, in einem Verzeichnis vorhandenen, privaten Chiffrierschlüsseln verglichen. Wenn ein passender Schlüssel gefunden wird, kann der symmetrische Schlüssel entschlüsselt werden. Der Benutzer muss nach dem Fund des Schlüssels das Passwort eingeben, mit dem der private Schlüssel verschlüsselt ist. Sollten mehrere Schlüssel passen, wird nur für den ersten Schlüssel, der gefunden wurde, das Passwort verlangt, da der symmetrische Schlüssel nur einmal entschlüsselt werden muss. Das aus der Entschlüsselung entstandene Byte-Array wird dann zu einem symmetrischen Schlüssels für den zur Verschlüsselung der Nachricht verwendeten Algorithmus konvertiert.

```
Key secKey = new SecretKeySpec(uncryptKey, keyInfo.getAlgorithmus());
```

Mit dem symmetrischen Schlüssel werden anschließend die verschlüsselte Nachricht und die verschlüsselte Zeit entschlüsselt und angezeigt.

```
Cipher unCryObj = Cipher.getInstance(keyInfo.getAlgorithmus());
unCryObj.init(Cipher.DECRYPT_MODE, secKey);
SealedObject[] sealed = nachricht.getData();
String theText = (String) sealed[0].getObject(secKey);
Date aktuelleZeit = (Date) sealed[1].getObject(secKey);
```

Nach der eigentlichen Entschlüsselung werden die in dem Message-Objekt als Signatur-Objekte gespeicherten Signaturen überprüft. Dazu wird nach einem Zertifikat gesucht, dessen Schlüsselidentifikation mit der Identifikation übereinstimmt, die im Signatur-Objekt gespeichert ist. Diese Zertifikate werden ebenfalls zuerst in dem internen Zertifikatsspeicher gesucht. Wird im Zertifikatsspeicher ein passendes Zertifikat gefunden, wird es nach den selben Regeln getestet, nach denen auch die Zertifikate zur Verschlüsselung getestet werden. Wird kein gültiges Zertifikat gefunden, wird ein „normales“ Zertifikat mit der im Signatur-Objekt gespeicherten Schlüsselidentifikation auf dem Zertifikatsserver gesucht. Auch ein hier gefundenes Zertifikat wird getestet. Ist ein gefundenes Zertifikat gültig, wird mit dem öffentlichen Schlüssel aus dem Zertifikat die Signatur überprüft. Das Ergebnis der Überprüfung wird im unteren Teil der Applikation dargestellt.

Ist weder im Zertifikatsspeicher noch bei der Anfrage an den Zertifikatsserver ein passendes „normales“ Zertifikat gefunden worden, wird auf dem Zertifikatsserver ein Stellvertreterzertifikat mit passender Schlüsselidentifikation gesucht. Beim Testen eines gefundenen Stellvertreterzertifikats wird neben den schon erwähnten Tests auch das Zertifikat gesucht und getestet, welches durch das Stellvertreterzertifikat vertreten wird. Die damit gefundene Stellvertreterregelung wird zusätzlich zum Ergebnis der mit dem Stellvertreterschlüssel durchgeführten Zertifikatsüberprüfung im unteren Teil der Applikation angezeigt. Mit diesem Schritt ist die Entschlüsselung abgeschlossen.

7 Zusammenfassung und Ausblick

7.1 Zusammenfassung

In dieser Diplomarbeit geht es um die Lösung des Stellvertreterproblems in einer Public-Key-Infrastruktur. Das Stellvertreterproblem entsteht, sobald ein Zertifikatsinhaber für eine bestimmte Zeit nicht verfügbar ist und für diese Zeit ein Stellvertreter einsetzt. Eine verschlüsselte Nachricht kann von dem Stellvertreter nicht gelesen werden, da dieser nicht über die privaten Schlüssel der zu vertretenden Person verfügt. Das von mir entwickelte System löst dieses Problem, indem es für die Stellvertreter eigene Zertifikate erstellt, in denen neben dem zu zertifizierenden Schlüssel auch Informationen über die zu vertretende Person gespeichert sind. Bei der Verschlüsselung einer Nachricht werden die Stellvertreterzertifikate, die den eigentlichen Empfänger vertreten, gesucht. Werden solche Zertifikate gefunden, wird die Nachricht mit dem Schlüssel aus dem Zertifikat zusätzlich verschlüsselt.
Für dieses System wurde ebenfalls eine Implementierung erstellt, die in der Sprache Java mit Hilfe der Technik RMI programmiert wurde.

7.2 Ausblick

Obwohl das System funktioniert, bietet es noch einige Verbesserungsmöglichkeiten. Einer der wohl wichtigsten Punkte sind Standards. Die im System erstellten Zertifikate entsprechen bereits dem X509-Standard und lassen sich auch durch andere Applikationen verwenden. Die Nachricht und die Schlüssel sind dagegen in einem properitären Format gespeichert. Dadurch lassen sich verschlüsselte Nachrichten, die mit diesem System erstellt wurden, auch nur mit diesem System entschlüsseln.
Auch das System selbst bietet einige Verbesserungsmöglichkeiten. So könnten die Schlüssel zur Verbesserung der Benutzbarkeit in Schlüsselringen nach dem Vorbild von anderen PKI-Systemen zusammengefasst werden. Die Java-Crypto-API biete für diesen Zweck sogenannte „KeyStores“ an. Ich habe mich bei meiner Implementierung gegen eine Verwendung solcher Speicher entschieden, da dadurch das Verständnis des Aufbaus des Systems und der Verschlüsselung deutlicher wird.
Der Zertifikatsserver lässt sich ebenfalls verbessern. So verfügt die Schnittstelle zum Testen von Zertifikaten auf Sperrungen nur über die Seriennummer als Parameter. Dies reicht aus, solange nur Zertifikate einer CA auf dem Server veröffentlicht werden. Sollten Zertifikate mehrerer CAs veröffentlicht werden, kann es zu Kollisionen kommen, wenn die CAs identische Seriennumern verwenden. In diesem Fall könnte die Abfrage um den Fingerprint des zutestenden Zertifikat erweitert werden.

8 Glossar und Literaturverzeichnis

8.1 Glossar

Asymmetrische Verschlüsselung

Im Gegensatz zur symmetrischen Verschlüsselung wird bei asymmetrischen Verfahren zum Verschlüsseln ein anderer Schlüssel eingesetzt als zum Entschlüsseln. Zum Verschlüsseln und Überprüfen von digitalen Signaturen wird der öffentliche Schlüssel eingesetzt, zum Entschlüsseln und signieren der geheime Schlüssel.

Certification Authority

Zertifizierungsinstanz innerhalb eines hierarchischen Zertifizierungsmodells.

DES

Digital Encryption Standard; Ein Algorithmus zur symmetrischen Verschlüsselung von Informationen mit einer Schlüssellänge von 56 Bit. Der DES-Algorithmus gilt nach heutigen Maßstäben als unsicher und sollte nicht mehr verwendet werden.

Digitale Signatur

Aus dem geheimen Schlüssel und einer Datei wird durch Anwendung einer Hashfunktion eine eindeutige Zeichenfolge gebildet. Mit Hilfe des öffentlichen Schlüssels kann nun jeder überprüfen, ob die Datei tatsächlich von dem angegebenen Urheber stammt.

Fingerprint

Eindeutige Prüfsumme des öffentlichen Schlüssels, die wesentlich kürzer als der Schlüssel selbst ist. Wird zum überprüfen bzw. verifizieren eines öffentlichen Schlüssels herangezogen.

Geheimtext

Die mit Hilfe eines Verschlüsselungsverfahren verschlüsselten Daten.

Hashfunktion

Eine Hashfunktion ist eine Funktion, die aus einer Information eine eindeutige Prüfsumme errechnet.

Hybridverschlüsselung

Verschlüsselungsverfahren, bei dem sowohl symmetrische als auch asymmetrische Verschlüsselungsverfahren eingesetzt werden. Die Daten werden mit einem zufällig erzeugten Sitzungsschlüssel verschlüsselt. Dieser Sitzungsschlüssel wird dann mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Auf der Empfängerseite wird mit Hilfe des privaten Schlüssels der Sitzungsschlüssel entschlüsselt, um damit die Nachricht zu entschlüsseln.

IDEA

International Data Encryption Algorithm; Ein Algorithmus zur symmetrischen Verschlüsselung von Informationen. Der IDEA-Algorithmus ist patentiert. Kommerzieller Einsatz erfordert den Erwerb einer Lizenz.

Key Escrow

Schlüsselhinterlegung; Maßnahme um Schlüssel nach Verlust aus einem Speicher wiederzuerlangen.

Key Recovery

Schlüsselwiederherstellung; Verfahren, um verschlüsselte Daten wieder zugänglich zu machen, ohne den privaten Schlüssel zu besitzen.

Klartext

Allgemein: unverschlüsselte Daten

Kompromittierung

Unbeabsichtigte oder unautorisierte Offenlegung des (Geheim-)Schlüssels bzw. der verschlüsselten Daten.

Kryptografie

Wissenschaft von der Verschlüsselung von Daten und deren Anwendung.

Öffentlicher Schlüssel

Public Key; Bei asymmetrischen und hybriden Verschlüsselungsverfahren der frei zugängliche Schlüssel, der das Überprüfen einer Signatur und das Verschlüsseln von Nachrichten ermöglicht.

Privater Schlüssel

Private Key; Asymmetrischer Schlüssel, der sowohl das Signieren und Entschlüsseln von Nachrichten ermöglicht.

RMI

Remote Method Invocation

Technik um Methoden in einem verteilten System über die Grenzen eines Rechners hinaus auszuführen.

RSA

Ein Algorithmus zum Signieren und asymmetrischen Verschlüsseln von Informationen. Der RSA-Algorithmus – benannt nach seinen Erfindern Rivest, Shamir und Adelman – gilt nach heutigen Maßstäben als sicher.

Schlüssel

Datensequenz, die benutzt wird, um mit einer Verschlüsselungssoftware aus einem Klartext einen Geheimtext zu erzeugen (Verschlüsselung) und um aus einem Geheimtext den Klartext wieder herzustellen (Entschlüsselung).

Signatur / Signieren

Jeder Besitzer eines privaten Schlüssels kann einer Nachricht eine elektronische Signatur („Unterschrift“) anbringen. Mit Hilfe des dazugehörigen öffentlichen Schlüssels lässt sich die Authentizität der Nachricht nachweisen.

Symmetrische Verschlüsselung

Verschlüsselung, bei der die Ver- und Entschlüsselung derselbe Schlüssel erforderlich ist. Bei Einsatz symmetrischer Verschlüsselung für die Kommunikation ist die Übertragung des Schlüssels über einen zusätzlichen, sicheren Kanal notwendig. Für diesen zusätzlichen Kanal wird heute meistens asymmetrische Verschlüsselung verwendet.

Verschlüsselung

Das Verändern eines Textes, eines Bildes bzw. allgemein einer Datei unter Verwendung eines Schlüssels und nach einen festgelegten Verfahren (Verschlüsselungsalgorithmus), mit dem Ziel, die Inhalte für andere unkenntlich zu machen, wobei der Vorgang unter Verwendung des Schlüssels wieder umkehrbar ist.

Verschlüsselungsalgorithmus

Methode nach der aus dem Klartext der Geheimtext erzeugt wird. Man unterscheidet zwischen symmetrischen und asymmetrischen Verschlüsselungsalgorithmen.

8.2 Literaturquellen

[Eckert03] Eckert, Claudia
IT-Sicherheit; Konzepte – Verfahren – Protokolle
2. Auflage; Oldenbourg-Verlag 2003
ISBN : 3-486-27205-5

[Schmeh01] Schmeh, Klaus
Kryptografie und Public-Key-Infrastrukturen im Internet
2. Auflage; dpunkt-Verlag 2001
ISBN : 3-932588-90-8

[Lipp00] Lipp, Peter; Farmer, Johannes; Bratko, Dieter; Platzer, Wolfgang; Sterbenz, Andreas
Sicherheit und Kryptographie in Java
Addison-Wesley-Verlag 2000
ISBN : 3-8273-1567-0

[Schneier96] Schneier, Bruce
Angewandte Kryptographie
Addison-Wesley-Verlag 1996
ISBN : 3-89319-854-7

[Beutelspacher01] Beutelspacher, Albert; Schwenk, Jörg; Wolfenstetter, Klaus-Dieter
Moderne Verfahren der Kryptographie
4. Auflage; Vieweg-Verlag
ISBN : 3-528-36590-0

[Krüger99] Krüger, Guido
GoTo Java 2
Addison-Wesley-Verlag 1999
ISBN : 3-8273-1370-8

[Chaum91] Chaum, D.
„Group Signatures“ – Advances in Cryptology
EUROCRYPT ’91 Proceedings, Springer-Verlag 1991
Seiten 257-265.

[gnupp.de] http://www.gnupp.de
Internetseite zum Thema Verschlüsselungssoftware

Anhang

Als Anhang ist dieser Diplomarbeit eine CD-ROM beigelegt. Hier finden Sie ein Inhaltsverzeichnis dieser CD-ROM.

Im Wurzelverzeichnis der CD-ROM befinden sich die Verzeichnisse „Diplomarbeit“, „Implementierung“ und „Java Security Provider“.

In dem Verzeichnis „Diplomarbeit“ befindet sich die elektronische Version dieser Arbeit. Die Arbeit liegt im Microsoft-Word-Format und im PDF-Format vor. Die Datei mit dem Namen „Diplomarbeit“ enthält die eigentliche Arbeit, während sich in der Datei „Deckblatt“ das offizielle Deckblatt befindet. In den Unterverzeichnissen „Screenshots“ und „Zeichnungen“ sind die von mir erstellten Zeichnungen und Screenshots gespeichert.

Das Verzeichnis „Implementierung“ enthält die eigentliche Implementierung des von mir entwickelten Systems. Da das System mit der Entwicklungsumgebung „JBuilder“ von Borland geschrieben wurde, ist in diesem Verzeichnis die von JBuilder erstellte Projektstruktur abgelegt. Das Verzeichnis enthält neben der Projektdatei „Diplomarbeit.jpx“ auch den Java Security Provider BouncyCastle, dessen Implementierung in der Datei „bcprov-jdk14-120.jar“ gespeichert ist. Zusätzlich beinhaltet dieses Verzeichnis noch einige Konfigurationsdateien für das Projekt.
In dem Unterverzeichnis „classes“ befinden sich die kompilierten Klassen des Systems. Der Quellcode ist in dem Unterverzeichnis „src“ abgelegt. In beiden Verzeichnissen existiert eine Verzeichnisstruktur, die die Klassenstruktur des Systems wiederspiegelt. Im Verzeichnis „doc“ befindet sich eine mit dem Tool „javadoc“ erstellte Hilfe zu den erstellten Klassen.

Das Verzeichnis „Java Security Provider“ enthält eine komplette Version des verwendeten Security Providers BouncyCastle. Es handelt sich um eine Version, die zum Zeitpunkt der Arbeitserstellung unter der URL http://www.bouncycastle.org/download/crypto-120.zip geladen werden konnte.

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(4) bzw. §25(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 19.02.2004