Diplomarbeit zur Erlangung des Grades eines Diplom-Informatikers über das Thema

Aufbau einer PKI mit S/MIME-fähigen Email-Programmen

Christoph Hannnebauer

18. Dezember 2007

Betreuer: Prof. Dr. Gunter Schlageter

FernUniversität in Hagen

Inhaltsverzeichnis

Einleitung

Mit der wachsenden Verbreitung des Internets sind Emails zu einem wichtigen Kommunikationsmedium geworden. Sie beschleunigen und erleichtern den Nachrichtenaustausch zwischen den Benutzern. Auch sensible Informationen wie Lebensläufe oder Bankverbindungen werden hierbei übermittelt. Leider hat der Datenschutz gegenüber dem klassischen Brief mit der Einführung der Emails gelitten, denn zumindest alle an der Übertragung einer Email beteiligten Rechner haben uneingeschränkten Zugriff auf die übertragenen Inhalte. Mit einer automatischen Auswertung der Emails können missbräuchlich Profile der Versender und Empfänger angelegt werden. Außerdem können nicht öffentliche Firmeninformationen in die falschen Hände geraten.

Die mittlerweile ausgereifte Verschlüsselungstechnologie oder Kryptographie soll hierbei Abhilfe schaffen: Emails können von Unberechtigten nicht mehr gelesen oder unbemerkt verändert werden. Während Kryptographie in der Kommunikation innerhalb von Unternehmen und zwischen verschiedenen Unternehmen immer häufiger eingesetzt wird, nutzen Privatanwender Verschlüsselung bisher nur selten.

Diese Abhandlung beschreibt die bisher für Privatanwender verfügbaren Möglichkeiten, Kryptographie zu nutzen. Aufbauend auf einer Analyse der Nachteile der bisherigen Systeme wird eine weitere Möglichkeit vorgeschlagen, mit der Privatpersonen oder kleine Unternehmen, für die der Aufbau einer professionellen Public Key-Infrastruktur (PKI) zu aufwändig ist, ihre Emails kryptographisch sichern können.

Viele Benutzer haben auf ihren Rechnern bereits die Software installiert, um Emails zu ver- und entschlüsseln: Ihr Email-Programm selbst. Viele gängige Email-Programme besitzen heute schon kryptographische Fähigkeiten, trotzdem werden diese Funktionen selten genutzt, weil sie zu kompliziert sind. In dieser Arbeit wird daher untersucht, wie ein System entwickelt werden kann, das ohne Veränderung des Email-Programms oder Installation von Plugins die Nutzung der kryptographischen Fähigkeiten des Email-Programms erleichtert. Konkret betrachtet werden die Email-Programme Microsoft Outlook, Microsoft Outlook Express und Mozilla Thunderbird.

Gliederung

Diese Arbeit gliedert sich in drei Teile. Der aus den Kapiteln 1-3 bestehende Teil I beschreibt die grundsätzliche Funktionsweise von Kryptographie, die vorhandenen Standards und bisher nutzbare Systeme, mit denen Emails verschlüsselt werden können.

Teil II und III enthalten den in dieser Arbeit entwickelten Ansatz, die vorhandenen Email-Programme auch direkt für die kryptographischen Operationen zu nutzen. Der aus den Kapiteln 4-7 bestehende zweite Teil beschreibt die entwickelten oder konfigurierten Server-seitigen Komponenten, die nötig sind, um eine PKI aufzubauen. Kapitel

4 schafft hierbei einen Überblick über den Gesamtablauf und begründet, welche Komponenten nötig sind. Die restlichen Kapitel des zweiten Teils beschreiben die programmierten oder benutzten Komponenten im Detail.

In den Kapiteln 8 und 9 beschreibt der dritte Teil schließlich die Entwicklung eines Programms, das auf den Rechnern der Benutzer ausgeführt wird und installierte Email-Programme für die verschlüsselte Email-Kommunikation konfiguriert. Dieses Konfigurationsprogramm übernimmt die bisher notwendige, komplizierte Einstellung der Email-Programme.

Die in dieser Arbeit gewonnenen Erkenntnisse werden in Kapitel 10 zusammengefasst.

Teil I

Theoretische Grundlagen

1 Kryptographie

In der Literatur werden vier Ziele der Kryptographie unterschieden [Sch96, AL99]:

- Vertraulichkeit - Unbefugte sollen die Nachricht nicht einsehen können (bisweilen auch Geheimhaltung genannt)
- Integrität - Die Nachricht darf während ihrer Übertragung nicht verändert werden können
- Authentizität - Die Nachricht darf nur von dem Absender stammen, der sich als Absender ausgibt
- Verbindlichkeit - Der Absender einer Nachricht darf nicht abstreiten können, dass die Nachricht tatsächlich von ihm gesendet wurde

Die Wichtigkeit dieser Ziele unterscheidet sich je nach Anwendung. Auch die Schwierigkeit der Umsetzung unterscheidet sich je nach Ziel. Verbindlichkeit muss beispielsweise nur bei Verträgen erreicht werden. Bei Urkunden ist oft wichtig, dass der Inhalt nicht verändert wurde, Vertraulichkeit spielt keine Rolle. Im Folgenden wird beschrieben, auf welche Weise diese vier Ziele erreicht werden können.

1.1 Verschlüsselung

Um das Ziel der Vertraulichkeit zu erreichen, benutzt man Verschlüsselung. Man stelle sich zur Veranschaulichung vor, dass eine Nachricht von einem Absender Alice an einen Empfänger Bob übertragen werden soll. Für die Übertragung steht aber nur ein unsicherer Kanal zur Verfügung, das heißt, dass alle zwischen Alice und Bob übertragenen Daten von einem so genannten Angreifer Eve mitgelesen werden können. Ziel der Verschlüsselung ist es nun, die Nachricht von Alice zu Bob zu übertragen und dabei trotzdem zu verhindern, dass Eve den Inhalt der Nachricht versteht. Die Nachricht wird dazu vor ihrer Übertragung in eine Form gebracht, mit der man ohne die Kenntnis eines speziellen Geheimnisses nicht den Inhalt der Nachricht ermitteln kann. Dieses Geheimnis kann beispielsweise ein Passwort sein oder ein so genannter geheimer Schlüssel, dessen Funktion später noch genauer erklärt wird. In Folge muss Bob als Empfänger der Nachricht dieses Geheimnis kennen, denn er soll die Nachricht lesen können. Wenn Eve das Geheimnis nicht kennt, sieht sie nur die Übertragung der verformten und unlesbaren Nachricht über den unsicheren Kanal.

Hieraus ergeben sich mehrere Schwierigkeiten.

- Es muss ein Verfahren gefunden werden, mit dem eine Nachricht unkenntlich gemacht und nur mit Hilfe eines Geheimnisses wieder gelesen werden kann.
- Bob muss ein Geheimnis haben, das Eve nicht kennt.

- Alice muss wissen, dass Bob dieses Geheimnis hat, durch spezielle Algorithmen muss sie aber nicht unbedingt das Geheimnis selbst kennen.

Im Folgenden werden die mathematischen Ansätze umrissen, die diese Schwierigkeiten angehen.

1.1.1 Kryptosysteme

Ein Verschlüsselungsverfahren oder Kryptosystem ist allgemein definiert als Fünf-Tupel (P, C, K, E, D) ([Buc99], Abschnitt 4.1). Hierbei ist

- P die Menge aller Klartexte (unverschlüsselte Nachrichten),
- C die Menge aller Chiffretexte (verschlüsselte Nachrichten),
- K die Menge aller Schlüssel,
- E eine Menge von Verschlüsselungsfunktionen und
- D eine Menge von Entschlüsselungsfunktionen.

Für jeden Schlüssel $e \in K$ gibt es genau eine Verschlüsselungsfunktion $\eta_e : P \to C$ in E, die Klartexte eineindeutig auf Chiffretexte abbildet. Umgekehrt gibt es zu jedem Schlüssel $d \in K$ auch eine Entschlüsselungsfunktion $\delta_d : C \to P$ in D, die Chiffretexte auf Klartexte abbildet. Wenn der Verschlüsselungsschlüssel e und der Entschlüsselungsschlüssel d zusammen passen, entschlüsselt δ_d aus dem Chiffretext $c \in C$ den Klartext $p \in P$, wenn η_e den Klartext p in den Chiffretext c verschlüsselt. Als Formel ausgedrückt bedeutet das

$$\delta_d(\eta_e(p)) = p$$

Wenn Alice eine vertrauliche Nachricht p an Bob schicken möchte, müssen daher einerseits Alice und Bob ein gemeinsames Kryptosystem haben und andererseits Alice einen Schlüssel e haben, der zu einem Schlüssel d von Bob passt. Alice kann dann den Chiffretext $c = \eta_e(p)$ bilden und über einen unsicheren Kanal zu Bob schicken. Bob kann aus dem Chiffretext c wieder den Klartext $p = \delta_d(c)$ bilden. Wenn Eve sich für die Nachricht p interessiert und beim Abhören des unsicheren Kanals den Chiffretext c erkennt, braucht sie immer noch den Schlüssel d. In der Kryptographie geht man im Allgemeinen davon aus, dass eventuelle Angreifer das verwendete Kryptosystem kennen, das heißt, dass die Sicherheit der Verschlüsselung nur von der Kenntnis der verwendeten Schlüssel abhängen darf.

Ein wichtiges, aber nicht hinreichendes Kriterium für die Sicherheit eines Kryptosystems ist daher die Größe der Menge aller möglichen Schlüssel. Wenn die Menge zu klein ist, kann ein Angreifer alle Schlüssel ausprobieren, um den verwendeten Schlüssel zu ermitteln. Da die Schlüssel bei den tatsächlich verwendeten Kryptosystemen Zahlen sind oder zumindest Zahlen enthalten, wird die Sicherheit eines Schlüssels maßgeblich durch seine „Länge“ bestimmt, das heißt durch die Zahl der Binärziffern, die zu seiner Speicherung verwendet werden. Diese Schlüssellänge bestimmt die Zahl der Schlüssel, die bei reinem Ausprobieren (dem Brute-Force-Angriff) durchschnittlich getestet werden müssen, bevor der Angreifer den verwendeten Schlüssel findet. Die Anzahl der Zahlen, die sich mit n Ziffern darstellen lassen, vergrößert sich exponentiell bei einem linearen Anstieg von n. Wenn allen Zahlen mögliche Schlüssel entsprechen und daher bei einem Brute-Force-Angriff ausprobiert werden müssen und wenn jeder Test

eine gewisse Zeit braucht, steigt die Dauer des Brute-Force-Angriffs exponentiell mit steigender Schlüssellänge. Folglich können mit einer hinreichend großen Schlüssellänge Brute-Force-Angriffe effektiv verhindert werden.

Man kann Verschlüsselungsverfahren nach verschiedenen Kriterien einteilen, zum Beispiel danach, ob immer ein Nachrichtenblock verschlüsselt wird oder eine Nachricht unmittelbar während ihrer Übertragung. In dieser Arbeit wird nur eine der möglichen Einteilungen vorgestellt, da die anderen Einteilungen im Kontext dieser Arbeit nicht relevant sind.

1.1.2 Symmetrische und asymmetrische Verschlüsselungsverfahren

Die Verschlüsselungsverfahren, bei denen die beiden zusammenpassenden Schlüssel $e \in K$ zur Verschlüsselung und $d \in K$ zur Entschlüsselung identisch sind, nennt man symmetrische Verschlüsselungsverfahren ([Buc99], Abschnitt 4.2). Umgekehrt sind bei den asymmetrischen Verschlüsselungsverfahren die beiden Schlüssel e und d verschieden.

Die heute bekannten symmetrischen Verschlüsselungsverfahren arbeiten deutlich schneller als die heute bekannten asymmetrischen Verschlüsselungsverfahren. Wenn jedoch eine größere Menge von Personen nur mit symmetrischen Verschlüsselungsverfahren verschlüsselt kommunizieren möchte, muss sich jede Person mit jeder anderen einen gemeinsamen Ver- und Entschlüsselungsschlüssel teilen. Bei n Personen existieren im gesamten System

$$\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$$

Schlüssel. Abbildung 1.1 veranschaulicht die Situation für $n = 5$.

Da die Anzahl der Schlüssel quadratisch mit n anwächst, existieren bei großen n im System mehr Schlüssel als verwaltet werden können. Dieses Problem kann man mit Hilfe der Public-Key-Kryptographie vermeiden. Dies ist ein Verfahren auf Basis asymmetrischer Verschlüsselungsverfahren: Jede teilnehmende Person hat einen öffentlichen Verschlüsselungsschlüssel $e \in K$ und einen dazu passenden geheimen Entschlüsselungsschlüssel $d \in K$ (auch privater Schlüssel genannt). Die beiden Schlüssel bezeichnet man zusammen als Schlüsselpaar. Der Verschlüsselungsschlüssel e lässt sich aus dem Entschlüsselungsschlüssel d berechnen, umgekehrt geht dies jedoch nicht. Daher kann man den öffentlichen Verschlüsselungsschlüssel e allen anderen Personen bekannt geben, den Entschlüsselungsschlüssel d muss man jedoch für sich behalten. Da jeder Zugriff auf Bobs Verschlüsselungsschlüssel e hat, kann auch jeder eine Nachricht an Bob verschlüsseln. Die Nachricht kann aber nur von Bob selbst entschlüsselt werden, da er als einziger im Besitz des geheimen Entschlüsselungsschlüssels d ist. Bei diesem System gibt es bei n teilnehmenden Personen auch nur n Schlüsselpaare, für jede Person genau eines, wie auch in Abbildung 1.2 dargestellt ist.

Der Grundstein der asymmetrischen Verschlüsselungsverfahren wurde 1976 von Diffie und Hellman gelegt [DH76].

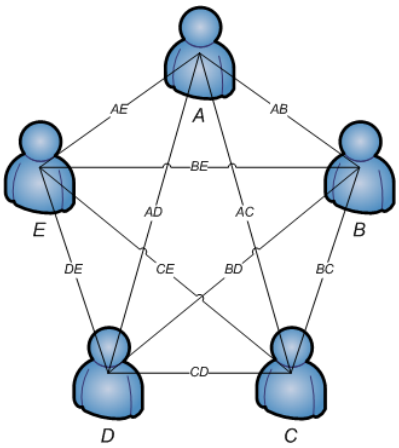

Abbildung 1.1: Bei fünf Personen A, B, C, D und E muss es zehn symmetrische Schlüssel AB, AC, AD, AE, BC, BD, BE, CD, CE und DE geben, wenn jeder mit jedem anderen verschlüsselt kommunizieren will

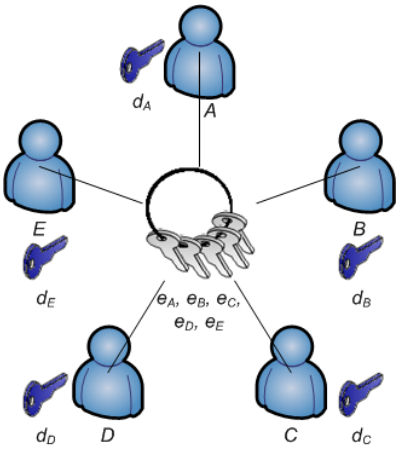

Abbildung 1.2: Bei asymmetrischer Verschlüsselung unter fünf Personen gibt es fünf öffentliche Schlüssel e_A, e_B, e_C, e_D, e_E und fünf private Schlüssel d_A, d_B, d_C, d_D, d_E

1.1.3 Anwendungen

In der Praxis werden symmetrische und asymmetrische Verfahren kombiniert: Will Alice eine verschlüsselte Nachricht an Bob schreiben, so benutzt sie Bobs Verschlüsselungsschlüssel e, um einen symmetrischen Schlüssel x zu verschlüsseln. Die eigentlichen Daten verschlüsselt sie mit dem symmetrischen Schlüssel x. Danach schickt sie Bob die mit x verschlüsselte Nachricht und den mit e verschlüsselten Schlüssel x. Bob entschlüsselt nun x mit seinem zu e passenden Entschlüsselungsschlüssel d. Danach benutzt er x, um die eigentlichen Daten zu entschlüsseln. So erreicht man, dass nur noch der verhältnismäßig kleine Schlüssel x asymmetrisch verschlüsselt werden muss, die großen Datenmengen können schnell und symmetrisch verschlüsselt werden. Die mit x verschlüsselten Daten und das asymmetrisch verschlüsselte x werden immer gemeinsam gespeichert, so dass sich das gesamte Paket so verhält, als wäre es allein asymmetrisch verschlüsselt.

In professionellen Programmen werden überwiegend die Verfahren ElGamal [Elg85] und RSA[1] [RSA78] eingesetzt. Teil des ElGamal-Verfahrens ist das mathematische Konzept einer so genannten Gruppe, welche, auch um Sicherheit zu gewährleisten, bestimmten Voraussetzungen genügen muss [Buc99]. In der Praxis ist diese Gruppe momentan normalerweise die Einheitengruppe eines endlichen Körpers. Vielversprechend, aber bisher mit wenig Verbreitung, ist die Verwendung einer diskreten Gruppe Elliptischer Kurven.

RSA-Schlüssel bis 663 Bits sind bereits faktorisiert und damit unsicher [RSA]. Mit

[1] Die Buchstaben RSA sind die Anfangsbuchstaben der Erfinder des RSA-Kryptosystems R. L. Rivest, A. Shamir und L. Adleman

zunehmender Rechenleistung können immer längere Schlüssel faktorisiert werden. Damit ein verschlüsselter Text auch einige Zeit unlesbar bleibt, sollte man eine deutlich größere Schlüssellänge als 663 Bits wählen. Das US-amerikanische National Institute of Standards and Technology (NIST) empfiehlt für heutige Anwendungen eine Mindestschlüssellänge von 1024 Bits für Verfahren wie RSA und ElGamal und eine Mindestschlüssellänge von 160-223 Bits für auf Elliptischen Kurven beruhende Verfahren [BBB+07].

1.2 Signaturen

Die drei nicht durch Verschlüsselung abgedeckten Ziele der Kryptographie, nämlich Integrität, Authentizität und Verbindlichkeit, kann man mit digitalen Signaturen erreichen. Digitale Signaturen sind ein Werkzeug der Public-Key-Kryptographie. Der Absender einer Nachricht habe daher bereits ein Schlüsselpaar aus öffentlichem und geheimem Schlüssel. Mit dem geheimen Schlüssel kann der Absender (und nur er) für eine Nachricht eine digitale Signatur erzeugen. Diese Signatur ist dann eindeutig der signierten Nachricht und dem Schlüsselpaar des Signierers zugeordnet. Der Empfänger der Nachricht kann mit der digitalen Signatur und dem öffentlichen Schlüssel des Absenders verifizieren, dass die Nachricht seit Erzeugung der Signatur nicht mehr verändert wurde, womit die Integrität gegeben ist. Außerdem ist der Erzeuger der Signatur im Besitz des zugehörigen geheimen Schlüssels. Wenn der Empfänger weiß, dass nur der Absender im Besitz dieses geheimen Schlüssels ist, ist das Ziel der Authentizität ebenfalls erreicht. Wenn er dies zusätzlich noch anderen beweisen kann, erfüllt die Signatur auch das Ziel der Verbindlichkeit.

1.2.1 Hashfunktionen

In allen gebräuchlichen Signaturverfahren werden nicht die Nachrichten selbst signiert, sondern ein so genannter Hashwert der Nachricht. Hashfunktionen sind daher ein zentraler Bestandteil aller Signatur-Algorithmen. Eine Hashfunktion ist eine mathematische Abbildung, die ein beliebiges Datum (zum Beispiel einen Text) auf einen Zahlenwert fester Größenordnung (in der Praxis eine Zahl fester Bytegröße) abbildet. Eine in der Kryptographie verwendbare Hashfunktion sollte auch eindeutig sein: Zwei verschiedene Texte sollten auch verschiedene Hashwerte haben. Ein Gegenbeispiel für Eindeutigkeit, dass also zwei Texte den gleichen Hashwert haben, nennt man eine Kollision. Eine kollisionsfreie Hashfunktion ist zwar mathematisch überhaupt nicht möglich, weil es mehr Texte als Hashwerte gibt, aber diese Kollisionen dürfen wenigstens nicht mit den heutigen Mitteln errechenbar sein [Sch96]. Diese Eigenschaft nennt man dann *kollisionsresistent* [Buc99]. Eine in der Kryptographie einsetzbare Hashfunktion sollte zudem eine „Einwegfunktion“ sein, das bedeutet, dass der Hashwert einer Nachricht leicht errechnet werden kann, zu einem gegebenen Hashwert sollte aber keine Nachricht in vertretbarem Aufwand ermittelbar sein.

1.2.2 Allgemeiner Algorithmus

Bei den heutzutage angewendeten Signaturverfahren bildet der Absender zunächst einen Hashwert der zu übermittelnden Nachricht. Zu diesem Hashwert wird mit Hilfe

des geheimen Schlüssels eine digitale Signatur errechnet. Unter Zuhilfenahme des zugehörigen öffentlichen Schlüssels kann nun jeder, der im Besitz des Hashwerts ist, feststellen, ob die Signatur wirklich zu genau diesem Hashwert passt. Der Empfänger der Nachricht erhält deswegen drei Dinge: Die Nachricht selbst, den vom Absender errechneten Hashwert und die vom Absender erzeugte Signatur. Mit dem öffentlichen Schlüssel des Absenders kann der Empfänger überprüfen, ob die Signatur gültig ist und auch zu dem mitgeschickten Hashwert passt. Dann errechnet der Empfänger den Hashwert erneut aus der Nachricht und vergleicht ihn mit dem Hashwert, der mitgeschickt wurde, und dessen Signatur er eben überprüft hat.

Ist der errechnete Dokument-Hashwert mit dem mitgeschickten identisch und ist die Validierung der Signatur mit dem öffentlichen Schlüssel erfolgreich, dann weiß der Empfänger, dass der wirkliche Absender im Besitz des privaten Schlüssels ist, mit dem die Signatur erstellt wurde. Wenn der Empfänger feststellen kann, dass auch nur der vorgebliche Absender im Besitz des privaten Schlüssels ist, kann der Empfänger davon ausgehen, dass die Nachricht tatsächlich vom angegebenen Absender unverändert abgeschickt wurde. Das letztgenannte Problem, die Zuordnung eines Schlüsselpaares zu einer Person, wird in Kapitel 2 beschrieben.

1.2.3 Anwendungen

Bestimmte Schlüssel kann man sowohl zur Verschlüsselung als auch für Signaturen einsetzen, andere Schlüssel sind jeweils nur für eines der beiden Gebiete verwendbar. Außerdem gibt es Signaturschlüssel, die nur für bestimmte Ziele eingesetzt werden. Dies hat nicht nur technische, sondern auch rechtliche Gründe, wie in Abschnitt 1.2.4 gezeigt wird.

Technisch lassen sich alle zur Verschlüsselung eingesetzten RSA-Schlüssel [RSA78] auch zur Erzeugung von Signaturen einsetzen. Dazu wird eine spezielle Eigenschaft des RSA-Algorithmus ausgenutzt: Man kann die Entschlüsselung einer Nachricht vor ihrer Verschlüsselung durchführen, das heißt Entschlüsselung und Verschlüsselung sind vertauschbar. Um eine Signatur zu erzeugen, „entschlüsselt“ der Absender den Hashwert einer Nachricht mit seinem privaten Schlüssel. Der Empfänger kann diesen entschlüsselten Hashwert mit dem öffentlichen Schlüssel des Absenders wieder verschlüsseln und damit die erste Entschlüsselung aufheben. Das Ergebnis ist der ursprüngliche Hashwert. Mit gewissen Einschränkungen kann man auf diese Weise aus allen Verschlüsselungsalgorithmen, bei denen Ver- und Entschlüsselung vertauschbar sind, Signaturalgorithmen bilden [Buc99].

Beim ElGamal-Verschlüsselungsverfahren lassen sich Ver- und Entschlüsselung nicht vertauschen. Während also aus dem RSA-Verschlüsselungsverfahren ein RSA-Signaturverfahren abgeleitet werden kann, ist es nicht möglich, auf die gleiche Weise aus dem ElGamal-Verschlüsselungsverfahren ein ElGamal-Signaturverfahren abzuleiten. Es gibt jedoch ein Signaturverfahren, welches das gleiche Schlüsselmaterial wie das ElGamal-Verschlüsselungsverfahren benutzt und ebenfalls ElGamal genannt wird [Buc99]. Trotzdem wird dieses Verfahren nicht häufig eingesetzt, sondern nur der eng verwandte Algorithmus Digital Signature Algorithm (DSA). DSA wurde von der NIST [NIS94] standardisiert und arbeitet deutlich schneller als das ursprüngliche ElGamal-Signaturverfahren. Dafür kann es aber auch nur auf den Einheitengruppen endlicher Körper

und nur bis zu einer Schlüssellänge von 1024 Bits angewendet werden. Das ElGamal-Signaturverfahren lässt sich auch auf Elliptischen Kurven anwenden. Außerdem gibt es eine Variante von DSA mit dem Namen Elliptic Curve DSA (ECDSA), die sich ebenfalls auf Elliptischen Kurven anwenden lässt.

1.2.4 Verwendete Schlüssel

Obwohl es also oftmals technisch möglich ist, denselben Schlüssel für Verschlüsselung und Signaturen einzusetzen, ist das unter Umständen aus Gründen der Sicherheit nicht sinnvoll und je nach Anwendung auch rechtlich nicht erlaubt. Es ist nämlich notwendig, den geheimen Entschlüsselungsschlüssel zu sichern, um bei einem Verlust dieses Schlüssels nicht auch die verschlüsselten Daten zu verlieren. In öffentlichen Unternehmen muss gegebenenfalls auch Wirtschaftsprüfern Zugriff auf verschlüsselte Daten gewährt werden. Dies ist praktisch nur möglich, wenn eine zentrale Instanz Zugriff auf alle Entschlüsselungsschlüssel hat.

Es ist dagegen nicht notwendig, einen Signaturschlüssel zu sichern, da man bei Verlust des geheimen Schlüssels ohne Nachteile einen neuen erstellen kann. Wenn ein Signaturschlüssel an einer zentralen Stelle gesichert ist, auf die etwa ein Administrator Zugriff hat, kann schlecht sichergestellt werden, dass eine Signatur auch tatsächlich vom eigentlichen Eigentümer des geheimen Schlüssels ausgestellt ist und nicht von einem böswilligen Administrator. Daher darf ein geheimer Schlüssel nach deutschem Recht nicht gesichert sein, wenn eine mit ihm ausgestellte digitale Signatur eine Unterschrift ersetzen soll ([BGB06] §126a, Absatz 1 und [Sig07]).

Unter Umständen kann es aber auch sinnvoll sein, verschiedene Schlüsselpaare für unterschiedliche Signaturen zu verwenden. Signaturen werden zum Beispiel häufig zum alleinigen Ziel der Authentizitätsprüfung ausgestellt. Dabei wird vom Überprüfenden ein zufällig ausgewählter Wert dem zu Authentifizierenden geschickt, den letzterer signiert und die Signatur zurückschickt. Wenn die Signatur gültig ist, muss der nun Authentifizierte im Besitz des passenden geheimen Schlüssels sein. Bei dieser Prozedur könnte aber ein böswilliger Überprüfender statt eines zufälligen Wertes den Hashwert eines realen Dokumentes schicken. Die zur Authentifizierung erzeugte Signatur wäre dann ohne das Wissen des Schlüsselinhabers eine Signatur zu dem Dokument, dessen Hashwert der Überprüfende geschickt hat. Um diesen Missbrauch zu verhindern, kann man Signaturschlüssel explizit als Schlüssel zur Authentifizierung oder Schlüssel zur Verbindlichkeit kennzeichnen.

2 Public Key-Infrastrukturen

Im Kapitel 1 wurden einige Fragen aufgeworfen, zu deren Beantwortung die konkreten Implementierungen statt der reinen mathematischen Modelle betrachtet werden müssen. Obwohl es zum Beispiel von zentraler Bedeutung ist, zu wissen, ob ein bestimmter öffentlicher Schlüssel zu einer bestimmten realen Person gehört, wurden noch keine Methoden vorgestellt, um diese Zuordnung zu verifizieren. Das Vertrauen in bestimmte Schlüssel zu schaffen, ist eine Aufgabe jeder PKI.

Dieser Vorgang ist jedoch nur zum Teil ein mathematischer, sondern hauptsächlich ein organisatorischer. Es gibt mehrere miteinander konkurrierende Standards zur Speicherung von digitalen Schlüsseln und zur Zuordnung dieser zu realen Personen. Leider sind die vorhandenen Standards in den meisten Fällen kaum zueinander kompatibel.

Adams und Lloyd [AL99] definieren eine PKI als universell einsetzbar, also nicht auf spezielle Anwendungsbereiche eingeschränkt. Für solch eine universelle PKI legen sie eine Reihe von notwendigen Komponenten fest. Sie räumen allerdings ein, dass in einem konkreten Anwendungsfall, wie die sichere Email-Kommunikation in Abschnitt 2.1.4, durchaus auch Komponenten weggelassen werden können, wobei man dann eben keine definitionsgemäße PKI mehr erhält. Im Folgenden werden nur die Komponenten beschrieben, die für den Bereich der Email-Kommunikation relevant sind, da dies der Fokus der vorliegenden Arbeit ist.

2.1 X.509 und S/MIME

Der Standard X.509 [X.505b] des ITU Telecommunication Standardization Sector (ITU-T), einer Sparte der International Telecommunications Union (ITU), beschreibt unter anderem, wie ein öffentlicher Schlüssel zusammen mit Daten über den Besitzer gespeichert werden kann. Die normierte Struktur dieses Paketes ist mit weiteren Attributen erweiterbar. Wegen seiner hohen Verbreitung hat die Internet Engineering Taskforce (IETF) in Request for Comments (RFC) 3280 [HPFS02] ein Profil veröffentlicht, in dem nützliche Erweiterungen des X.509-Standards zusammengefasst sind. Obwohl die Erweiterungen einen eigenen Namen, PKI (X.509) (PKIX), haben, wird häufig trotzdem die Bezeichung X.509 für PKIen des PKIX-Standards verwendet.

Im X.509-Standard ist die Verwendung des RSA-Algorithmus beschrieben; andere Algorithmen werden durch die Erweiterbarkeit von X.509 zwar prinzipiell unterstützt, die genaue Verwendung ist aber nicht standardisiert.

Die Norm Secure Multipurpose Internet Mail Extensions (S/MIME) [Ram04] beschreibt eine Methode, verschlüsselte und/oder signierte Email-Nachrichten zu kodieren und zu dekodieren. Sie ist als Zusatz zur Norm Multipurpose Internet Mail Extensions (MIME) zu betrachten.

2.1.1 Zertifikate und die Vertrauenskette

Im X.509-Standard ist mit dem Aufbau eines Datenpakets aus öffentlichem Schlüssel und persönlichen Daten auch eine Methode definiert, mit der man überprüfen kann, ob die angegebenen persönlichen Daten auch tatsächlich zum Schlüssel gehören. Dazu erhält jedes Datenpaket eine digitale Signatur mit dem öffentlichen Schlüssel einer so genannten Certification Authority (CA). Den öffentlichen Schlüssel, die persönlichen Daten und die zugehörige Signatur nennt man im X.509-Kontext ein Zertifikat, und man sagt, die CA habe das Zertifikat ausgestellt. Um diesen Kontext deutlich zu machen, sagt man auch X.509-Zertifikat.

Zum öffentlichen Schlüssel einer CA gehört ebenfalls ein Zertifikat, das CA-Zertifikat genannt wird. Ein Zertifikat einer Person heißt Benutzer-Zertifikat, ein Zertifikat für einen Rechner nennt man Maschinen-Zertifikat. Die Unterschiede zwischen Maschinen-Zertifikaten und Benutzer-Zertifikaten sind aus der Perspektive dieser Arbeit nicht von Bedeutung, da sie von Zertifikaten zur Sicherung der Email-Kommunikation handelt. Deswegen wird im Text nicht mehr zwischen Benutzer- und Maschinen-Zertifikaten unterschieden.

Wenn bekannt ist, dass ein geheimer Schlüssel nur im Besitz der im Benutzer-Zertifikat angegebenen Person ist, das Zertifikat also ohne Bedenken verwendet werden kann, wird das Benutzer-Zertifikat vertrauenswürdig genannt. Um das Vertrauen in ein Zertifikat zu überprüfen, ist im RFC zu X.509 [HPFS02] ein Algorithmus angegeben, der vereinfacht dargestellt zuerst die Gültigkeit der Zertifikats-Signatur feststellt und dann das Vertrauen in das Zertifikat der ausstellenden CA überprüft. Dadurch verschiebt sich das Problem natürlich nur vom zu überprüfenden Zertifikat auf das CA-Zertifikat. Man wiederholt die Überprüfungsprozedur, wobei man jeweils das CA-Zertifikat des gerade zu überprüfenden Zertifikats ermittelt. Irgendwann findet sich ein Zertifikat, dessen Signatur vom eigenen öffentlichen Schlüssel signiert wurde, also ein so genanntes selbst signiertes (engl. self signed) Zertifikat. Dieses Zertifikat beziehungsweise seinen Besitzer bezeichnet man als Root CA[1]; die anderen CAs nennt man Subordinate CAs oder kurz Sub CAs, weil sie einer anderen CA untergeordnet sind. Üblicherweise speichert man eine Liste von vertrauenswürdigen Root CA-Zertifikaten, um die Gültigkeit einer Root CA zu überprüfen. Wenn man ein zu überprüfendes Zertifikat über die Signaturen zu einer vertrauenswürdigen Root CA zurückverfolgen kann, dann wird das überprüfte Zertifikat als gültig betrachtet, ansonsten nicht. Die Folge von Zertifikaten, vom zu überprüfenden zur Root CA, nennt man auch Vertrauenskette oder englisch Chain of Trust. Ein Beispiel für eine solche Vertrauenskette ist in Abbildung 2.1 zu sehen.

In der Praxis haben die einzelnen Programme eine eigene oder mit anderen Programmen geteilte Datenbank mit vertrauenswürdigen Root CA-Zertifikaten. Der Benutzer kann dann noch Zertifikate hinzufügen oder löschen. Windows XP hält in seiner Datenbank nach Installation etwa 110 solcher Root CA-Zertifikate, die Programme der Mozilla-Suite enthalten gut 100.

[1] selten wird auch der deutsche Begriff Wurzelzertifizierungsstelle verwendet

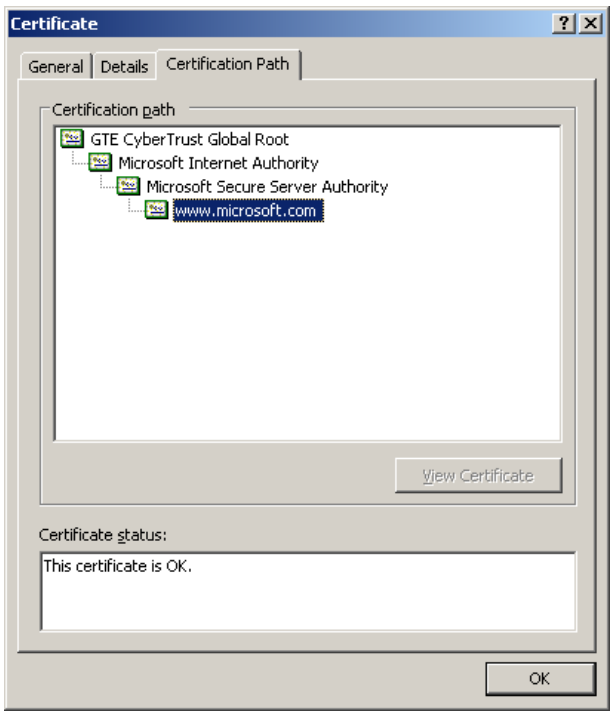

Abbildung 2.1: Die im Microsoft Internet Explorer dargestellte Vertrauenskette des Maschinen-Zertifikats der Internet-Seite `www.microsoft.com` . Die Kette besteht aus den beiden Sub CAs Microsoft Secure Server Authority und Microsoft Internet Authority und der Root CA GTE CyberTrust Global Root. Da die Root CA als vertrauenswürdige Root CA gespeichert ist, wird auch dem Maschinen-Zertifikat vertraut.

2.1.2 Widerruf von Zertifikaten

Es gibt eine Reihe von Situationen, in denen das zu einem Zertifikat gehörende Schlüsselpaar nicht mehr benutzt werden sollte oder kann. Das ist dann der Fall, wenn

- ein Dritter Zugriff auf den privaten Schlüssel hatte und der Schlüssel nicht mehr als geheim gelten kann,
- der private Schlüssel gelöscht worden ist, so dass niemand mehr Zugriff auf ihn hat,
- sich die im Zertifikat hinterlegten Daten geändert haben, wie der Name oder die Email-Adresse, oder
- der Besitzer des Zertifikats aus irgendeinem Grund keinen Anspruch mehr auf die durch das Zertifikat verliehenen Rechte hat.

In diesen Fällen muss das existierende Zertifikat widerrufen werden, das heißt es muss gekennzeichnet werden, dass es nicht mehr verwendet werden darf. Je nach Art des Schlüssels und Grund des Widerrufs ist dieses Verwendungsverbot mehr oder weniger vollständig: Wenn der private Schlüssel nur verloren ist, sind alte Signaturen weiterhin gültig, wurde er jedoch gestohlen, kann keiner Signatur mehr vertraut werden - selbst eine scheinbar alte Signatur könnte der Angreifer nach dem Diebstahl des Schlüssels erstellt und zurückdatiert haben.

Eine Beschreibung verschiedener Widerrufsmethoden findet man in „Understanding PKI“ [AL99]. Im Folgenden werden die beiden gebräuchlichsten Methoden beschrieben.

Certificate Revocation Lists

Eine Möglichkeit festzustellen, ob ein Zertifikat widerrufen wurde, ist eine Widerrufsliste (englisch Certificate Revocation List (CRL)). In diesen Listen stehen die Seriennummern aller von einer CA ausgestellten und danach widerrufenen Zertifikate. Die CA signiert diese Liste, dann kann sie veröffentlicht werden. CRLs haben oft eine eingeschränkte Gültigkeitsdauer - typisch sind zwei Wochen - so dass die Liste nicht ständig abgerufen werden muss, man sich aber auch nicht zu lange auf eine inaktuelle Liste verlassen kann.

Bei stark ausgelasteten CAs können die CRLs sehr groß werden. Zum Beispiel sind einige CRLs der von dem Unternehmen VeriSign betriebenen CA größer als 10 Megabyte. Eine kryptographische Anwendung kann eine so große CRL unter Performance-Gesichtspunkten nicht bei Bedarf über das Internet herunterladen, weswegen Zertifikate oft nicht auf Widerruf geprüft werden, wenn CRLs die einzige Überprüfungsmöglichkeit sind (im Internet Explorer ist die Überprüfung der CRLs zum Beispiel in der Voreinstellung ausgeschaltet).

Wenn man die Anzahl der in einer CRL vorkommenden Seriennummern zählt, kann man die Anzahl der ausgestellten Zertifikate einer CA insgesamt abschätzen. Diese ist ein nicht immer erwünschter Nebeneffekt von CRLs.

Online Certificate Status Protocol

Um das Performance-Problem der CRLs zu vermeiden, wurde das Online Certificate Status Protocol (OCSP) entwickelt. Die Details werden in RFC 2560 [MAM+99] beschrieben. Beim OCSP wird bei der Überprüfung eines Zertifikats dessen Seriennummer an einen Server im Internet übertragen, dieser antwortet dann, ob das angefragte Zertifikat gültig ist oder nicht.

Das aufkommende Datenvolumen ist dadurch deutlich geringer als bei CRLs. Weitere Vorteile sind die höhere Aktualität von OCSP-Servern gegenüber CRLs mit ihrer Gültigkeitsdauer und man kann mittels OCSP keine Aussagen darüber machen, wie viele Zertifikate die CA ausgestellt hat.

2.1.3 Formate im X.509-Kontext

Um Zertifikate, CRLs, private Schlüssel und ähnliches zu speichern, wurden im Rahmen des X.509-Standards verschiedene Formate für diese Daten spezifiziert.

Zertifikate

Der Aufbau von X.509-Zertifikaten ist in der Sprache Abstract Syntax Notation One (ASN.1) spezifiziert [X.505b, HPFS02]. In ASN.1 wird aber nicht festgelegt, wie eine bestimmte Struktur binär kodiert werden kann. Hierfür legt der Standard X.690 der ITU-T [X.602] verschiedene Methoden fest, mit denen man solche in ASN.1 gegebenen Strukturen binär kodieren kann. Für X.509-Zertifikate werden immer die Distinguished Encoding Rules (DER) verwendet. Im Gegensatz zu anderen Kodierungen erlauben die DER nur genau eine Darstellung einer ASN.1-Struktur. Dies ist erforderlich, da nur Binärdaten digital signiert werden können, nicht aber allgemeine Strukturen, und daher die Signatur des Zertifikats nur für eine bestimmte Kodierung gültig ist. Die Kodierung eines gegebenen Zertifikats muss daher eindeutig sein.

Da das DER-kodierte Zertifikat aus Binärdaten besteht, lässt es sich in bestimmten Situationen schwer handhaben: Wenn das Zertifikat beispielsweise per Mail verschickt oder auf einer Webseite angeboten werden soll, eignet sich eventuell ein Format besser, das nur aus lesbaren Zeichen besteht. Hierfür gibt es keine Regel in den entsprechenden Standards, üblich ist es aber, die DER-kodierten binären Daten nochmals mit dem Base64-Verfahren [Jos06] zu kodieren. Die kryptographischen Erweiterungen in Microsoft Windows (CryptoAPI) unterstützen diese Darstellung beispielsweise, weitere mir bekannte Programme sind Pretty Good Privacy (PGP) Desktop der PGP Corporation [PGP] und GNU Privacy Guard (GPG) des GNU is not Unix (GNU)-Projekts [GPG]. Zusätzlich wird - in Anlehnung an die Privacy Enhanced Mail (PEM) [Lin93] - vor dem Base64-kodierten Block eine Kopfzeile

```
-----BEGIN CERTIFICATE-----
```

und nach dem Block eine Fußzeile

```
-----END CERTIFICATE-----
```

eingefügt. Auf diese Weise lassen sich Zertifikate wie Texte verarbeiten, programmatisch als Strings oder in Windows über die Zwischenablage.

Alle Speicherformate für Zertifikate existieren auch für CRLs, da auch diese veröffentlicht werden. Im Base64-kodierten Format wird allerdings die Kopfzeile

```
-----BEGIN CERTIFICATE-----
```

und die Fußzeile weggelassen. Das CryptoAPI erkennt CRLs allerdings sogar mit jeder Kopf- und Fußzeile.

Typische Dateiendungen für Zertifikatsdateien sind .der (binär), .cer und .pem (Base64-kodiert). CRLs haben normalerweise die Dateiendung .crl.

Zertifikatsanträge

Auch vor der Zertifizierung durch eine CA kann es notwendig sein, einen öffentlichen Schlüssel zu speichern. Dies wird vor allem im Beantragungsprozess genutzt, da dieser öffentliche Schlüssel an die CA weitergegeben werden muss, damit diese auf Grundlage des Antrags ein Zertifikat erstellen kann. In diesem Antrag können die Metainformationen vermerkt werden, die später Teil des Zertifikats sein sollen. Die Firma RSA Data Security, Inc. hat die Public Key Cryptography Standards (PKCS) als eine

Reihe von Normen für kryptographische Anwendungen entwickelt. Eine Möglichkeit zur Speicherung von Zertifikatsanträgen wird im PKCS #10 [RSA00] geregelt. Ein Antrag lässt sich aber auch in der Cryptographic Message Syntax nach RFC 2630 speichern [Hou04, Hou07].

Wie bei den Zertifikaten selbst kann der Zertifikatsantrag binär gespeichert werden, für die Übertragung mit bestimmten Medien ist aber ein Base64-kodiertes Format sinnvoller. Zertifikatsanträge im PKCS #10-Format haben die Kopfzeile

```
-----BEGIN CERTIFICATE REQUEST-----
```

Private Schlüssel

Die Anforderungen an ein Speicherformat für private (also geheime) Schlüssel unterscheiden sich von einem für Zertifikate, welche nur öffentliche Informationen enthalten. Man kann zwar auch ausschließlich den privaten Schlüssel speichern, häufig ist der private Schlüssel jedoch nur sinnvoll verwendbar, wenn man weiß, zu welchem Zertifikat und öffentlichen Schlüssel er gehört. Aus diesem Grund kann das gesamte Schlüsselpaar mit Zertifikat in einer Datei des Formats PKCS #12 [RSA99] gespeichert werden. PKCS #12 ermöglicht zusätzlich zur bloßen Speicherung von Daten die Verschlüsselung des Dateiinhalts oder der Teile desselben. Diese symmetrische Verschlüsselung benutzt ein Passwort als Schlüssel. Damit ist der private Schlüssel weniger gefährdet, falls die Datei in die falschen Hände gerät. Die Verschlüsselung und das Passwort sind allerdings optional, so dass nicht in jedem PKCS #12-Paket die privaten Schlüssel gesichert sind. Dateiendungen für Dateien mit PKCS #12-Inhalt sind .pfx oder .p12.

Verschlüsselte und signierte Nachrichten

Der Standard PKCS #7 [RSA93] bietet eine Syntax zur Speicherung, Signierung und Verschlüsselung binärer Daten [Sch96]. Die enthaltenen Daten können jeweils signiert und/oder verschlüsselt sein, müssen aber nicht. PKCS #7-Pakete sind ein bedeutender Teil des S/MIME-Protokolls (siehe Abschnitt 2.1.4).

2.1.4 Verschlüsselte Emails

Unter Führung der RSA Data Security, Inc. wurde 1995 mit der Entwicklung eines auf MIME aufbauenden Emailsicherheits-Standards namens S/MIME begonnen [AL99]. Die neueste Version 3.1 wird in RFC 3851 [Ram04] beschrieben. Der S/MIME-Standard erlaubt die Erstellung signierter und/oder verschlüsselter Emails, indem in die Email ein PKCS #7-Paket eingebettet wird. S/MIME baut auf dem X.509-Zertifikatskonzept auf.

Viele Email-Programme besitzen schon ohne zusätzliche Plugins S/MIME-Unterstützung und können Emails ver- und entschlüsseln sowie signieren und Signaturen überprüfen. Die drei in Kapitel 8 näher betrachteten Email-Programme Microsoft Outlook, Microsoft Outlook Express und Mozilla Thunderbird unterstützten S/MIME.

2.2 PGP

Phil Zimmermann veröffentlichte 1991 die erste Version des Programms PGP [Gar95]. Das Programm speicherte kryptographisches Schlüsselmaterial und verschlüsselte Nachrichten in einem eigenen Format. Mit neueren Versionen gab es immer wieder Änderungen dieses Formats. Obwohl das Format zuerst proprietär war, wurde es 1996 in RFC 1991 [ASZ96] beschrieben, aber nicht standardisiert. Nachdem die Weiterentwicklung des Programms PGP 1997 von Network Associates übernommen wurde [PGP], verabschiedete die IETF 1998 den OpenPGP-Standard in RFC 2440 [CDFT98] als Alternative zum proprietären Format des Programms PGP. Eine von der IETF-Arbeitsgruppe überarbeitete Version erschien im November 2007 als RFC 4880 [CDF+07].

Auch in PGP werden öffentliche Schlüssel zusammen mit Metainformationen abgespeichert. Dies entspricht dem Zertifikat im X.509-Kontext, bei PGP spricht man jedoch einfach von PGP-Schlüsseln, selbst wenn zusätzlich Metainformationen oder sogar ein Paket aus mehreren kryptographischen Schlüsseln gemeint ist. Der Begriff „Zertifikat“ wird im PGP-Umfeld seltener verwendet und hat hier eine andere Bedeutung, auf die in Abschnitt 2.2.2 eingegangen wird.

2.2.1 PGP-Schlüssel

Ein PGP-Schlüssel enthält immer mindestens einen kryptographischen Schlüssel, den Primärschlüssel, welcher Signaturen erzeugen kann. Signierfähigkeit ist erforderlich, da die meisten weiteren Teile des PGP-Schlüssels mit dem Primärschlüssel signiert werden. Dadurch ist kryptographisch gesichert, dass der Besitzer des PGP-Schlüssels diese Informationen beabsichtigt hat und sie nicht erst nachträglich von einem Angreifer in den PGP-Schlüssel eingefügt wurden. Dem Primärschlüssel können weitere Schlüssel, so genannte Subkeys, untergeordnet sein. Diese müssen nicht zwangsläufig signierfähig sein. Eine häufige Kombination ist ein Primärschlüssel zum Signieren mit dem DSA- und ein untergeordneter Schlüssel zum Verschlüsseln mit dem ElGamal-Algorithmus (siehe auch Kapitel 1).

Einem PGP-Schlüssel sind ein oder mehrere so genannte User IDs zugeordnet. Diese sollen einen Namen und eine Email-Adresse des Schlüsselbesitzers enthalten. Ein zwingendes Format ist nicht vorgegeben, ein String bestehend aus dem vollständigen Namen und der in < und > eingefassten Email-Adresse wird aber empfohlen [CDF+07, Res01].

Jeder User ID können beliebig viele Signaturen zugeordnet werden, wobei der Begriff Signatur nicht nur mathematisch gesehen werden darf. Es wird nämlich nicht nur die User ID selbst signiert, zusätzlich wird ein Gültigkeitsdatum und die Art der Signatur festgelegt. Jeder Besitzer eines PGP-Schlüssels, also derjenige, der den zugehörigen geheimen Schlüssel kennt, kann jede User ID eines beliebigen PGP-Schlüssels signieren. Eine solche Signatur bestätigt, dass der Besitzer des signierenden Schlüssels glaubt, dass die User ID tatsächlich zu dem kryptographischen Schlüssel gehört. Eine solche Signatur wird auch Zertifikat genannt, das Signieren einer User ID nennt man entsprechend Zertifizieren.

2.2.2 Vertrauensnetz (Web of Trust)

Da ein PGP-Schlüssel keinen klar definierten Aussteller hat, lässt sich auch nicht jeder PGP-Schlüssel auf eine zentrale Instanz wie die Root CA in X.509 zurückführen. Deswegen bildet sich auch keine gerade Vertrauenskette, sondern über die Signaturen der User IDs ein ganzes Vertrauensnetz, das Web of Trust.

In einfachsten Fall vertraut man nur auf die Zugehörigkeit einer User ID zu einem PGP-Schlüssel, wenn man die User ID selbst zertifiziert hat. Man kann einem PGP-Schlüssel aber noch weitergehendes Vertrauen schenken, indem man ihn als so genannten „Trusted Introducer“ zertifiziert. In diesem Fall behandelt man die Signaturen des Trusted Introducers so, als wären es die eigenen. Das heißt man würde der Zugehörigkeit einer User ID zu einem PGP-Schlüssel auch dann vertrauen, wenn man die User ID selbst nicht zertifiziert hat, dafür aber ein Zertifikat des Trusted Introducers für diese User ID existiert. Eine Ausnahme besteht darin, dass Trusted Introducer keine weiteren Trusted Introducer erzeugen können, sondern immer nur Zertifikate niedrigeren Vertrauensgrades ausstellen dürfen. Eine Grad-0-Signatur entspricht einem herkömmlichen Vertrauen, dass die User ID zum PGP-Schlüssel gehört. Eine Grad-1-Signatur kennzeichnet einen Trusted Introducer, der nur Grad-0-Signaturen ausstellen darf. In Fortsetzung dieses Konzeptes kann ein PGP-Schlüssel mit Grad-n-Signatur entsprechend Grad-(n-1)-Signaturen erzeugen. Obwohl der OpenPGP-Standard [CDF+07] n erst auf maximal 60 beschränkt, werden in der Praxis gewöhnlich höchstens Grad-2-Signaturen ausgestellt. PGP-Schlüssel mit einer solchen Signatur nennt man „Meta Introducer“ oder ebenfalls Trusted Introducer, wenn der Grad des Vertrauens nicht besonders herausgestellt werden soll.

Trusted Introducer entsprechen damit den CAs der X.509-Welt, allerdings muss nicht jeder PGP-Schlüssel auf genau einen Trusted Introducer zurückgeführt werden können, sondern auf beliebig viele (wobei auch gar keiner möglich ist). In Unternehmensnetzen werden auch mit PGP strenge Schlüsselhierarchien wie in X.509 gebildet, so dass der Trusted Introducer auch Subordinate CA und der Meta Introducer auch Root CA genannt wird.

2.2.3 Widerruf

Da es keine zentrale Instanz gibt, die PGP-Schlüssel ausstellt, kann auch niemand Widerrufslisten veröffentlichen. Wurde ein PGP-Schlüssel A kompromittiert, so muss der Besitzer des PGP-Schlüssels A selbst festlegen, wann zu widerrufen ist. Dazu erhält der PGP-Schlüssel eine spezielle Widerrufssignatur, die kennzeichnet, dass er nicht mehr gültig ist. Auf die gleiche Weise kann auch ein einzelner Subkey widerrufen werden. Die Widerrufssignatur kann nur vom Besitzer des geheimen Teils des PGP-Schlüssels A selbst erzeugt werden oder von einem so genannten „Designated Revoker“ (im Standard „Revocation key“ genannt [CDF+07]). Dieser Designated Revoker erhält eine Signatur von A, die ihn als Designated Revoker des PGP-Schlüssels A kennzeichnet.

Statt eines kryptographischen Schlüssels kann man aber auch nur ein einzelnes Zertifikat widerrufen. Das geht wiederum nur mit dem geheimen Schlüssel, mit dem das Zertifikat ursprünglich erzeugt wurde. Ein widerrufenes Zertifikat wird ignoriert, wenn das Vertrauen in eine User ID eines PGP-Schlüssels überprüft wird.

2.2.4 Dateiformate

Schlüsselmaterial, Signaturen, User IDs und verschlüsselte Nachrichten werden in PGP paketweise abgespeichert. Eine Datei kann dabei beliebige Pakete enthalten, so dass nur ein grundsätzliches Dateiformat für verschiedene Arten von Daten existiert. Da viele Kombinationen aber keinen Sinn ergeben, sind nur bestimmte Kombinationen üblich. So könnte man in einer Datei nur den öffentlichen Teil eines PGP-Schlüssels speichern, also das Schlüsselmaterial des Primär- und der untergeordneten Schlüssel, sowie die zugehörigen User IDs und Zertifikate. Eine verschlüsselte Nachricht wäre eine andere sinnvolle PGP-Datei.

Die Pakete können binär kodiert werden oder ähnlich wie in X.509 und PEM nach dem Base64-Verfahren kodiert und mit Kopf- und Fußzeile versehen abgespeichert werden. Bei PGP-Dateien kommen noch ein paar zusätzliche Daten wie eine Prüfsumme hinzu. Kopf- und Fußzeile ergeben sich aus der Art der gespeicherten Daten; der öffentliche Teil eines PGP-Schlüssels wird beispielsweise mit der Kopfzeile

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
```

abgespeichert.

Es ist bei Programmen für Endbenutzer üblich, den öffentlichen Teil der PGP-Schlüssel in einem Public Keyring (mit der Dateiendung .pkr) und den geheimen Teil der PGP-Schlüssel in einem Secret Keyring (mit der Dateiendung .skr) zu speichern. Die beiden Dateien dienen zusammen als Datenbank für lokal gespeicherte Schlüssel.

2.3 Sonstige Schlüssel-Standards

Eine im Jahr 2006 erschienene Sicherheitsstudie der Zeitschrift <kes> [kes06] gibt zwar an, dass die Zahl der S/MIME-Anwender in deutschen Unternehmen in den letzten beiden Jahren von 34 % auf 57 % gestiegen ist, die Zahl der Anwender mit OpenPGP-Schlüsseln liegt mit 66 % aber immer noch höher. Zumindest scheinbar widersprechen diese Daten der Einschätzung Adams' und Lloyds [AL99], Seite 273:

> To date, corporate demand has overwhelmingly been in favor of X.509 Version 3 public-key certificates.

Die unterschiedliche Einschätzung mag daher rühren, dass X.509-Zertifikate noch zahlreiche weitere Anwendungszwecke neben S/MIME haben, OpenPGP hat dagegen einen stärkeren Fokus auf Email-Verschlüsselung. Außerdem berücksichtigt die zitierte Sicherheitsstudie nur deutsche Unternehmen. Unumstritten ist jedoch, dass die Standards neben X.509 und OpenPGP nur eine untergeordnete Rolle spielen. Aus diesem Grund wird über einige weitere Standards nur ein kurzer Überblick gegeben.

2.3.1 Simple PKI

Die Simple PKI (SPKI) der IETF wurde 1996 als einfache Alternative zur komplizierten X.509-Welt entworfen. Das Schlüsselpaar sollte nicht mehr an eine Entität gebunden werden, sondern der Schlüssel soll Grundlage eines Berechtigungssystems werden. Die Kompliziertheit dieses Berechtigungssystems nahm SPKI allerdings den

entscheidenden Vorteil gegenüber X.509, so dass dieses Format sich nicht durchgesetzt hat [AL99].

2.3.2 XML Key Management Specification

Die XML Key Management Specification (XKMS) ist eine vom World Wide Web Consortium (W3C) entworfene Extended Markup Language (XML)-Sprache. Mit ihr können Schlüssel und kryptographisch gesicherte Nachrichten beschrieben werden. Zusätzlich sind Sprachelemente zur Steuerung der PKI definiert, beispielsweise um einen neuen Schlüssel bei der zuständigen Stelle zu beantragen. Da XKMS vorsieht, mit X.509- und PGP-Schlüsseln zusammenzuarbeiten, wird sich dieses Format vielleicht noch stärker verbreiten. [FHBF+01]

2.4 Schlüsselverteilung

Die obigen PKI-Standards regeln, ob man einem gegebenen Schlüssel vertraut oder nicht. Dazu muss man den Schlüssel aber erst einmal haben. Für dieses Problem haben sich verschiedene Lösungen etabliert.

2.4.1 LDAP

Das Lightweight Directory Access Protocol (LDAP) ist ein Nachfolger des X.500 Directory Access Protocol (DAP). Beides sind Protokolle zum Zugriff auf so genannte Verzeichnisdienste. In diesen Verzeichnisdiensten können Informationen über bestimmte Objekte veröffentlicht werden, beispielsweise Personen, Organisationen und Rechner. DAP wurde im ITU-T Standard X.500 spezifiziert [X.505a]. Da DAP ein komplexes und dadurch schwierig zu implementierendes Protokoll ist, hat die IETF versucht, bei LDAP nur so viel festzulegen, wie nötig ist. So setzt LDAP im Gegensatz zu DAP auf dem Transmission Control Protocol/Internet Protocol (TCP/IP) Protokoll-Stapel auf. Inzwischen ist LDAP in verschiedenen Anwendungsbereichen ein bedeutendes Protokoll. Die neueste LDAP-Version 3 hat die IETF in RFC 2251 spezifiziert [WHK97].

Obwohl LDAP nur der Name eines Übertragungsprotokolls ist, werden auch Verzeichnisse, die das LDAP-Protokoll unterstützen, als LDAP-Verzeichnisse bezeichnet. Analog werden Server, die ein LDAP-Verzeichnis anbieten, als LDAP-Server bezeichnet.

Ein LDAP-Verzeichnis hat eine baumartige Struktur. Es gibt also einen oder mehrere Wurzelbehälter, die andere Objekte enthalten. Diese Objekte können Blätter sein, sie können aber auch weitere Tochterobjekte enthalten. Jedes Objekt hat einen eindeutigen Namen, den Distinguished Name (DN). Dieser DN besteht aus einem objektspezifischen (und innerhalb aller Töchter eines Behälters eindeutigen) Relative Distinguished Name (RDN) und dem DN des enthaltenden Mutterbehälters. Dadurch und durch die Eindeutigkeit des RDN innerhalb der Töchter eines Behälters wird der DN wieder eindeutig. Außerdem enthält er auch den „Pfad“ des benannten Objekts, da man aus dem DN die DNs der Mütter bis zur Wurzel zurückverfolgen kann. Die Objekte können verschiedene Attribute haben, die weitere Informationen über das Objekt enthalten. Im LDAP-Standard ist festgelegt, dass alle Objekte ein Attribut objectClass haben müssen,

das die Art des Objekts angibt. Es wird damit also beispielweise festgelegt, ob es sich um einen Behälter, eine Person oder einen Rechner handelt. Welche Werte das Attribut objectClass oder andere Attribute enthalten können, wird im so genannten Schema festgelegt. Dieses Schema legt auch fest, welche objectClass ein Objekt haben muss, um weitere Tochterobjekte haben zu können, und welche objectClass die Tochterobjekte haben können. Das Schema selbst wird auch wieder durch Objekte in einem speziellen Behälter des LDAP festgelegt, so dass ein LDAP-Client bei Verbindung zu einem LDAP-Server überprüfen kann, wie er mit dem LDAP-Verzeichnis umzugehen hat. Das Schema kann dadurch, dass es selbst innerhalb des LDAP-Verzeichnisses liegt, verändert werden, um das Verzeichnis geänderten Bedürfnissen anzupassen.

Im RFC 4523 der IETF [Zei06] ist eine Schemaerweiterung zur Speicherung von X.509-Zertifikaten und verwandten Daten spezifiziert. Jedes LDAP-Verzeichnis mit dieser Schemaerweiterung kann von allen zum RFC 4523 kompatiblen LDAP-Clients genutzt werden, um

- die X.509-Zertifikate eines bestimmten Benutzers an Hand beliebiger Kriterien (zum Beispiel der Email-Adresse) zu finden,
- die Vertrauenskette wie in Abschnitt 2.1.1 beschrieben aufzubauen, das heißt die benötigten CA-Zertifikate zu finden und
- CRLs zu finden.

Das Schema wird Server-seitig beispielsweise vom Microsoft Active Directory und von OpenLDAP unterstützt. Die Email-Programme Microsoft Outlook, Microsoft Outlook Express und Mozilla Thunderbird unterstützen das Schema Client-seitig. Wenn man eine Email mit einem dieser Programme schreibt und das Programm anweist, die Email mit S/MIME verschlüsselt zu versenden, sucht es in bekannten LDAP-Verzeichnissen gemäß dem Schema in RFC 4523 nach X.509-Zertifikaten, die zu den Email-Adressen der Empfänger passen, und benutzt (nach einer Verifikation über eine Vertrauenskette) deren öffentlichen Schlüssel zur Verschlüsselung der Emails.

Für die Speicherung von OpenPGP-Schlüsseln gibt es kein standardisiertes Schema, es ist mit einem proprietären Schema aber trotzdem möglich.

Eine Speicherung von Zertifikaten in einem X.500-Verzeichnis ist ebenfalls möglich, dies wird aber weit weniger häufig eingesetzt als LDAP. LDAP wird zumindest im X.509-Umfeld häufiger eingesetzt als andere Schlüsselverteilungsdienste [AL99].

2.4.2 Email

PGP-Schlüssel in so genannten Key Servern können über Email verwaltet werden, indem eine Email mit einem Befehl in der Betreffzeile an den Server geschickt wird. Mit dem Befehl *add* lässt sich dem Key Server beispielsweise ein neuer PGP-Schlüssel hinzufügen, mit *get* kann man gespeicherte Schlüssel abfragen [Gar95]. Die hohen Antwort-Zeiten haben dazu geführt, dass diese Schnittstelle mittlerweile kaum noch benutzt wird.

2.4.3 HTTP und FTP

Für die Abfrage von OpenPGP-Schlüsseln wird häufig das Hypertext Transfer Protocol (HTTP) benutzt. Frühere Key Server-Implementationen speicherten die Schlüssel im gleichen Format wie das Programm PGP für Endbenutzer, nämlich als Public Keyring (siehe Abschnitt 2.2.4). Sie boten nur über HTTP, File Transfer Protocol (FTP) und Email eine Schnittstelle zur Abfrage der Schlüssel. Das Format eines Public Keyring ist jedoch nicht für eine so große Anzahl von Schlüsseln ausgelegt und brachte in der Verwendung als Key Server Performance-Probleme. Die meisten Key Server basieren daher auf dem von Marc Horowitz im Rahmen seiner Abschlussarbeit entwickelten Key Server [Hor95], welches die Performance-Probleme mit einem effizienteren Speicherformat beseitigt.

Auch X.509-Zertifikate lassen sich über HTTP abfragen, die IETF hat 1999 den Standard RFC 2585 [HH99] herausgegeben, der zusätzlich noch die Abfrage über FTP regelt. Einige Nachteile dieses immer noch gültigen Standards wurden 2006 mit RFC 4387 [Gut06] ausgebessert. Abfragen über HTTP werden trotzdem bisher weit seltener genutzt als LDAP-Abfragen. Ähnlich wie beim OpenPGP Key Server werden im neueren Standard RFC 4387 aus der angefragten Uniform Resource Identifier (URI) die Suchparameter für den Schlüssel ermittelt.

2.4.4 Domain Name System (DNS)

Die IETF hat 1999 einen Standard zur Speicherung von Zertifikaten direkt im DNS vorgeschlagen [EG99]. Dabei gibt es einen zusätzlichen Eintragstyp „CERT“, bei dessen Abfrage direkt ein Schlüssel zurückgegeben werden kann. Neben anderen Formaten werden dabei sowohl X.509 als auch PGP unterstützt. Die Parameter zum Finden des entsprechenden Zertifikats sind auf die Abfrage eines DNS-Namens beschränkt, so dass der Standard nur eine gewisse Auswahl an Suchparametern bietet. Für ein Zertifikat, das sich auf eine Email-Adresse bezieht, muss man das Zertifikat ohne zusätzliche Vorkehrungen daher im DNS-Server des Email-Anbieters speichern, womit die Verwendbarkeit vom konkreten Email-Anbieter abhängt. Ein weiterer Grund für die bisher nicht sehr große Verbreitung des Systems könnte sein, dass mit Nutzung des Standards der DNS-Server deutlich stärker belastet wird als ohne. Während bei normalen DNS-Abfragen nur wenige Daten übertragen werden, muss mit dem im Standard vorgeschlagenen System ein ganzes Zertifikat von zum Teil mehreren Kilobyte Größe übertragen werden.

2.4.5 Proprietäre Lösungen

Es gibt auch proprietäre Software zur Abfrage von Schlüsseln. So könnte ein Programm je nach Umgebungsbedingungen direkt eine Datenbank abfragen oder auf eine lokale Zertifikatsdatenbank im Dateisystem zurückgreifen. Zum Beispiel kann das Programm PGP Desktop™ Schlüssel von einem so genannten Web Service des PGP Universal Server™ über die HTTP-Variante mit Verschlüsselung HTTP/Transport Layer Security (TLS) [Res00] abfragen.

Mit Ausnahme des Dateisystems sind mir für diese Verfahren keine Normen bekannt, dementsprechend funktionieren diese Abfragen nur bei einer sehr kleinen Auswahl von

Programmen. Trotzdem kann es Sinn machen, so eine Schnittstelle zu verwenden, beispielsweise als Backend für eine genormte Abfrage.

3 Vorhandene Systeme

Da die mathematischen Modelle zur Public-Key-Kryptographie schon seit 1976 existieren und Verschlüsselungsalgorithmen auf Rechnersystemen seit Jahren effizient durchführbar sind, gibt es eine Reihe von Systemen, die Kryptographie für Anwender nutzbar machen wollen. In diesem Kapitel werden solche Systeme vorgestellt, wobei sich die Beschreibungen auf den Bereich der Emailsicherheit beschränken. Da Angestellte großer Unternehmen häufig schon Teil einer PKI sind, wie sie in Kapitel 2 beschrieben wird, haben alle Beschreibungen außerdem den Hintergrund der Kryptographie für Privatanwender und Mitarbeiter kleiner Unternehmen. Diese können auf keine zentral administrierte PKI und Verschlüsselungssoftware zurückgreifen, stattdessen müssen sie auf die in diesem Kapitel dargestellten Alternativen zurückgreifen.

Die Benutzbarkeitsstudie „Why Johnny can't encrypt“ [WT99] nannte 1999 die zu komplizierte Handhabung der Kryptographie-Software als Ursache ihrer schlechten Verbreitung unter Privatanwendern. Mittlerweile hat sich das Problem verschoben, wie im Folgenden gezeigt wird.

Die dargestellten Verfahren lassen sich meist beliebig miteinander kombinieren. Das bedeutet, dass der Versender einer gesicherten Nachricht ein anderes Verfahren verwenden kann als der Empfänger.

3.1 Clientbasierte Zusatzprogramme zur Verschlüsselung

Auf dem Markt existieren verschiedene Programme, die auf einem Rechner installiert werden können, um kryptographische Funktionen nutzen zu können. Zum Beispiel bietet die PGP Corporation mit PGP Desktop eine Software für diverse Anwendungsbereiche (etwa Datei-, Festplatten- und Email-Verschlüsselung). Schon die nur auf Email-Verschlüsselung beschränkte Version kostet pro Jahr 59 €, was einen Privatanwender von der Nutzung abhalten kann. [PGP]

Das kostenlose GPG [GPG] bietet einen ähnlichen Funktionsumfang wie die kommerziellen Produkte. Da es sich lediglich um eine Kommandozeilenanwendung handelt, empfiehlt sich die Benutzung eines Graphical User Interface (GUI). Das ebenfalls kostenlose Mozilla Thunderbird-Add-In Enigmail bietet ein solches GUI und benutzt im Hintergrund GPG. Der Installationsaufwand ist insgesamt etwas höher als bei den kommerziellen Produkten.

Da eine zusätzliche Anwendung auf dem System installiert werden muss, sind Zusatzprogramme bei einigen unternehmensfremden Kommunikationspartnern nicht verwendbar. Viele dieser Kommunikationspartner wollen sich keine zusätzlichen Programme installieren oder können das sogar gar nicht, zum Beispiel wenn die Rechner zentral administriert werden, aber nur wenige Benutzer ihre Emails verschlüsseln wollen.

Da Privatanwender normalerweise keine vertrauenswürdigen X.509-Zertifikate haben (siehe Abschnitt 3.4), werden sie mit dem OpenPGP-Standard arbeiten. Dabei treten allerdings die in der eingangs erwähnten Benutzbarkeitsstudie „Why Johnny can't encrypt“ [WT99] erwähnten Benutzbarkeitsprobleme auf, die durch die Schlüsselverwaltung aufgeworfen werden. Eigene Schlüssel müssen generiert und gesichert werden, fremde Schlüssel beschafft, geprüft und signiert werden. Potentielle Benutzer können durch das erforderliche Fachwissen und den Zeitaufwand abgeschreckt werden.

3.2 Server-basierte Verschlüsselungssysteme

Statt die Emails erst am Rechner des Empfängers oder Senders zu ver- oder entschlüsseln, kann die Verschlüsselung auch an anderer Stelle auf dem Übertragungsweg durchgeführt werden. Dadurch hat man keine Ende-zu-Ende-Verschlüsselung, die Nachricht wird also nicht auf dem gesamten Weg vom Absender zum Empfänger verschlüsselt übertragen. Da die Nachricht eventuell auf den unverschlüsselten Strecken abgefangen werden kann, erkauft man sich die Vorteile in der Anwendbarkeit durch eine niedrigere Sicherheit.

Diese Art der Kryptographie nennt man Gateway-Verschlüsselung. Hierbei werden die Emails durch ein Verschlüsselungsgateway geleitet, das nach außen aussieht wie ein Mailserver. Dieses Gateway ver- und entschlüsselt die Emails, die vom Client selbst nicht ver- oder entschlüsselt werden können. Das bedeutet auch, dass die geheimen Schlüssel der Email-Empfänger auf dem Verschlüsselungsgateway gespeichert werden müssen. Das Gateway muss daher entsprechend gut vor Angreifern geschützt werden. Zudem ist eine Signatur im Sinne einer rechtsverbindlichen Unterschrift bei einem automatisch operierenden Gateway nicht möglich.

Wenn beispielsweise der Empfänger der Email-Nachricht ein Kryptographie-Programm installiert hat, wie es in Abschnitt 3.1 beschrieben ist, würde der Absender eine unverschlüsselte Email an das Gateway schicken, dieses würde die Email mit dem öffentlichen Schlüssel des Empfängers verschlüsseln und an einen herkömmlichen Mailserver weitergeben. Die verschlüsselte Email erreicht den Empfänger, dieser entschlüsselt den Inhalt mit seinem privaten Schlüssel (siehe Abbildung 3.1). Im umgekehrten Fall wird die Email verschlüsselt versendet und vom Verschlüsselungsgateway mit Hilfe des privaten Schlüssels des Empfängers entschlüsselt und danach unverschlüsselt an den eigentlichen Empfänger weitergegeben. Falls zwei Verschlüsselungsgateways existieren, wird die Email nur zwischen den beiden verschlüsselt übertragen, sonst unverschlüsselt.

Schon der Aufbau macht deutlich, dass Verschlüsselungsgateways praktisch nur von Organisationen eingesetzt werden können, immerhin ist ein zusätzlicher Server erforderlich. Innerhalb dieser Organisation fließen dann alle Emails über das Verschlüsselungsgateway. Der Nachteil, dass der Übertragungsweg zwischen Endnutzer und Gateway unverschlüsselt ist, wird dadurch etwas relativiert, dass dieser Übertragungsweg nur innerhalb des Unternehmens liegt. Privatnutzer können ein Verschlüsselungsgateway jedoch kaum sinnvoll einsetzen.

Etwas anders sieht es bei unternehmensfremden Kommunikationspartnern aus. Diese können eine kryptographische Client-Software direkt auf ihrem Rechner installieren.

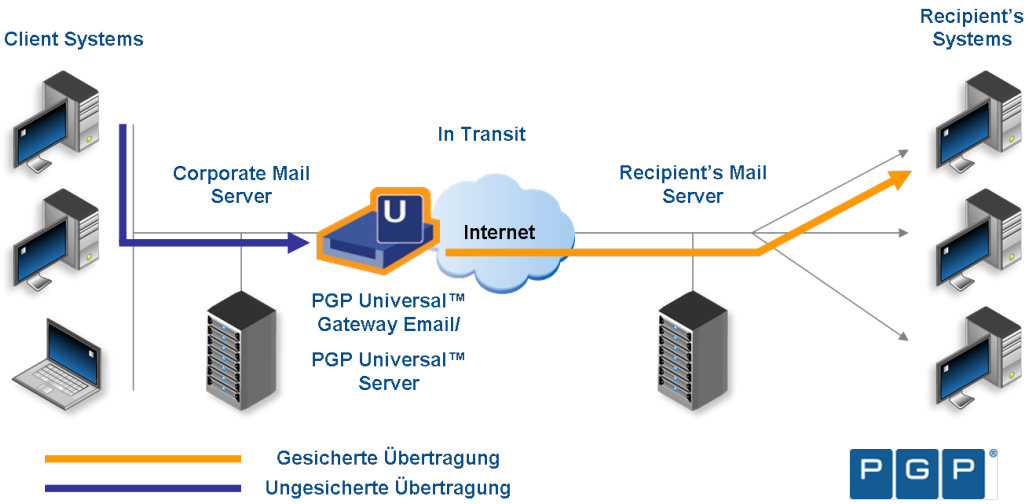

Abbildung 3.1: Übertragung einer Email über einen PGP Universal Server™ (hier in der Funktion eines Verschlüsselungsgateways) zu einem Client-Rechner mit kryptographischer Software. Quelle und Copyright: PGP Corporation

Je nach verwendetem Verschlüsselungsgateway kann diese Software auch vom Verschlüsselungsgateway den Kommunikationspartnern zur Verfügung gestellt werden. Alternativ bietet zum Beispiel der PGP Universal Server™ den so genannten Web Messenger an. Bei entsprechender Konfiguration des Universal Server erhalten die Kommunikationspartner statt einer Email mit dem eigentlichen Inhalt nur den Link zu einer Webseite, die auf dem PGP Universal Server™ liegt. Der Empfänger der Email kann deren Inhalt auf der verlinkten Webseite lesen, der Webseiteninhalt wird über eine verschlüsselte HTTP/TLS-Verbindung übertragen. Dies ist in Abbildung 3.2 dargestellt. Leider verliert der Empfänger hierbei die Möglichkeit, seine Emails lokal zu verwalten. Auch eine Verwaltung nur auf dem Verschlüsselungsgateway ist nicht möglich, da auf jedem Verschlüsselungsgateway nur die Emails gespeichert werden, die von oder zu einem Mitglied der Organisation des Verschlüsselungsgateways gesendet werden. Andere Absender haben entweder ihren eigenen Verschlüsselungsgateway oder die Mails werden im lokalen Postfach des Empfängers gespeichert. Außerdem werden in manchen Unternehmen HTTP/TLS-Verbindungen von der Firewall nicht zugelassen.

Ein großer Vorteil der Server-basierten Verschlüsselung ist ihre Transparenz. Absender und Empfänger haben den Eindruck, sie würden eine unverschlüsselte Email versenden und empfangen, ohne mit den möglichen Problemen konfrontiert zu werden.

3.3 Verschlüsselung im Webbrowser

Bei den meisten Email-Anbietern wie GMail[1], Yahoo[2], GMX[3] oder Hotmail[4] kann man die Emails seines Postfachs über die Webseite des Anbieters lesen (ein so genanntes

[1] `http://www.gmail.com`
[2] `http://www.yahoo.de`
[3] `http://www.gmx.de`
[4] `http://www.hotmail.com`

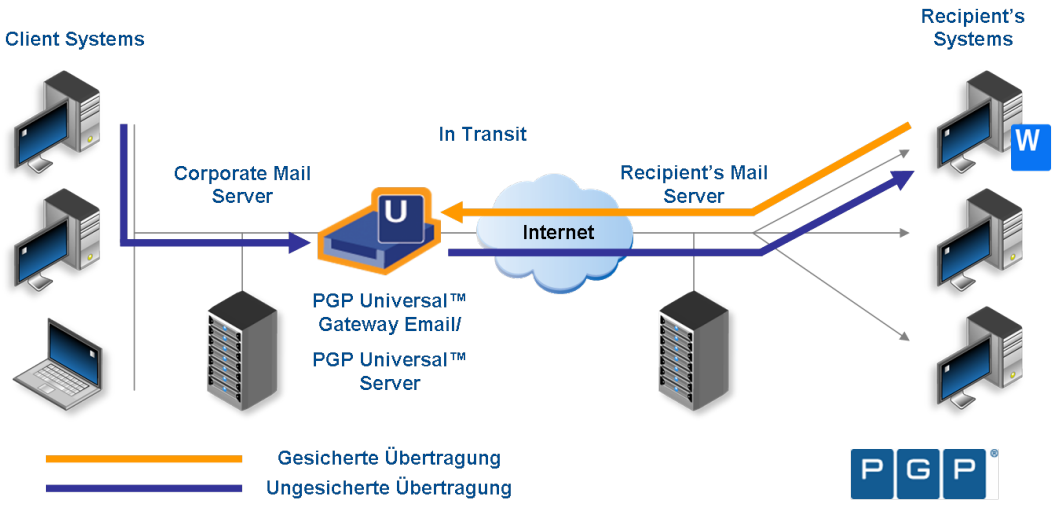

Abbildung 3.2: Übertragung einer Email über einen PGP Universal Server™ zu einem Client-Rechner ohne kryptographischer Software. Die Email wird per HTTP/TLS übertragen und in einem Webbrowser gelesen. Quelle und Copyright: PGP Corporation

Webmail-System). Mit den Systemen Hushmail, freenigma und FireGPG kann ein Benutzer auch in einem Webmail-System Emails ver- und entschlüsseln.

Während hushmail ein eigener Email-Anbieter mit integrierter Kryptographie ist, unterstützen freenigma und FireGPG die vorhandenen Email-Anbieter. Ein schon in der Idee verankerter Nachteil der letzten beiden Systeme ist die mangelnde Unterstützung von S/MIME, da die verschlüsselten Daten bei S/MIME binär übertragen werden und per Webmail gar nicht angezeigt werden. Im Gegensatz dazu werden OpenPGP-verschlüsselte Nachrichten bei der Übertragung per Email Base64-kodiert und können dargestellt werden. Dadurch hat ein Browser-Plugin Zugriff auf die verschlüsselten Daten und kann diese entschlüsselt darstellen.

3.3.1 freenigma

Die deutsche freenigma GmbH bietet ein kostenloses System zur Email-Verschlüsselung mit dem Namen freenigma an, das es erlaubt, in einem Browser verschlüsselte Emails zu lesen. Um das System nutzen zu können, muss man sich ein Plugin für den Browser Mozilla Firefox installieren. Dieses Plugin erkennt, wenn man die Webseite eines unterstützten Email-Anbieters im Browser öffnet. Das Plugin führt eine symmetrische Ver- oder Entschlüsselung auf die Nachricht aus. Der symmetrische Schlüssel wird zu einem freenigma-Server geschickt und dort asymmetrisch ver- oder entschlüsselt (abhängig davon, ob man Absender oder Empfänger ist). Da das asymmetrische Schlüsselpaar auf dem freenigma-Server gespeichert ist, kann man seine Emails an jedem Rechner lesen, der das entsprechende Firefox-Plugin installiert hat. Man muss also nicht seinen privaten Schlüssel mit sich führen, um verschlüsselte Emails zu lesen. [fre]

Zusätzlich zu dem eingangs erwähnten Problem der mangelnden S/MIME-Unterstützung, gibt es in freenigmas konkreter Implementierung noch weitere Nachteile. Zum

Beispiel kann nur an andere freenigma-Nutzer verschlüsselt werden und Firefox ist der einzige unterstützte Browser.

Da alle privaten Schlüssel von freenigma gespeichert werden, muss man dem Anbieter vertrauen. Der private Schlüssel ist nur durch ein Passwort geschützt, so dass ein Angreifer keinen physikalischen Zugang zur oder zumindest die Kontrolle über die Maschine des Benutzers erlangen muss, um den privaten Schlüssel zu erbeuten. Er muss lediglich das Passwort in Erfahrung bringen. Für den Datenschutz ist außerdem relevant, dass freenigma die Empfänger jeder verschlüsselten Email erfährt und sogar, wann man als Empfänger welche eigene verschlüsselte Email liest, da bei jedem Verschlüsseln die Email-Adressen der Empfänger an freenigma gesendet werden und bei jedem Entschlüsseln der symmetrische Schlüssel an freenigma übertragen wird.

3.3.2 FireGPG

Das ebenfalls kostenlose und sogar quelloffene FireGPG ist ein Firefox-Plugin, das mit PGP verschlüsselte Emails im Web-Mail-System von Gmail ver- und entschlüsseln kann. Das Plugin greift auf ein auf dem lokalen Rechner installiertes GPG [GPG] zurück. GPG übernimmt dabei alle kryptographischen Operationen. [CC]

Leider unterstützt FireGPG als einzigen Email-Anbieter Gmail, was die Zielgruppe stark einschränkt. Da es sich um ein recht junges Projekt handelt, die erste Version 0.1 ist im März 2007 erschienen, kann man jedoch hoffen, dass bald noch mehr Systeme unterstützt werden.

FireGPG ermöglicht die verschlüsselte Kommunikation mit allen Personen, die PGP benutzen. Es ist also nicht wie freenigma auf Benutzer des eigenen Systems beschränkt und erleichtert so den Ein- und Umstieg.

3.3.3 Hushmail

Der Email-Anbieter Hushmail hat Kryptographie nach dem OpenPGP-Standard bereits in das System integriert. Sowohl der private als auch der öffentliche Schlüssel werden auf dem Hushmail-Server gespeichert, der private Schlüssel ist aber symmetrisch mit einem Passwort verschlüsselt. Um eine Mail zu entschlüsseln, gibt der Benutzer sein Passwort in ein Browserfenster ein. Eine Java-Anwendung lädt den privaten Schlüssel von einem der Hushmail-Server, entschlüsselt den privaten Schlüssel mit dem eingegebenen Passwort und mit dem privaten Schlüssel schließlich die Email. [Husb]

Ein Vorteil des Systems ist die einfache Bedienung als Web-Anwendung. Da Hushmail auch über eine Internet Message Access Protocol (IMAP)-Schnittstelle verfügt, kann man wahlweise auch ein lokales Email-Programm verwenden und dort seine Emails verwalten. Obwohl eine Lösung wie die von Hushmail technisch auch S/MIME unterstützen könnte, kann Hushmails Implementierung nur mit OpenPGP umgehen.

Leider muss man bei Hushmail ein Email-Konto anlegen, um in den Genuss der kryptographischen Möglichkeiten des Systems zu kommen. Man kann also keine eventuell vorhandenen Email-Adressen bei anderen Email-Anbietern kryptographisch sichern. Die Sicherheit des Verfahrens beruht außerdem auf der Stärke des Passworts. Längere und damit hinreichend sichere Passwörter sind unbequemer zu handhaben als kürzere, aber leicht durch einen Brute-Force-Angriff zu ermittelnde Passwörter. Andere

Verschlüsselungssysteme sind bei hinreichend großer Schlüssellänge gegen Brute-Force-Angriffe immun, da der Angreifer keinen Zugriff auf den privaten Schlüssel des Angegriffenen hat. [Husa]

3.4 Kryptographische Funktionen in Email-Programmen

Die 2005 erschienene Studie „Johnny 2“ [GM05] stellt fest, dass die gängigen Email-Programme S/MIME auf einfache Weise unterstützen. Hier hat man jedoch das Problem, sich ein Zertifikat zu beschaffen. Auch Gutmann nennt die Beschaffung eines Zertifikats als größte Hürde für private Nutzer [Gut03].

Es gibt Anbieter von Zertifikaten zur privaten Email-Kryptographie (in der Tabelle 3.1 werden acht Anbieter gezeigt). Die Prozedur zur Beschaffung eines Zertifikats ist aufwändig, unter anderem weil zahlreiche für Email-Kryptographie unnötige Informationen an den Aussteller weitergegeben werden müssen [GM05]. Selbst technisch erfahrene Nutzer brauchen zwischen 30 Minuten und 4 Stunden, um ein Zertifikat zu erhalten [Gut03]. Für die Erstellung eines Zertifikats, das nur eine Email-Adresse verifiziert, sind Informationen wie der Name des Antragstellers jedoch gar nicht nötig. Auch im Hinblick auf Datenschutz ist die Erhebung dieser Informationen bedenklich.

Nachdem ein Nutzer sein Zertifikat erhalten hat, muss er sein Email-Programm noch so einstellen, dass es künftig kryptographisch abgesichert kommuniziert. Des Weiteren muss er eine Möglichkeit finden, die Zertifikate anderer Nutzer zu erhalten.

Zertifizierungsstelle	Mindestpreis	Windows	NSS	Webseite
CACert.org	0 €	Nein	Nein	www.CACert.org
Comodo	0 €	Ja	Ja	www.comodo.net
Digital Signature Trust	24,00 $	Ja	Ja	www.trustdst.com
GlobalSign	65,00 €	Ja	Ja	www.globalsign.net
GeoTrust	19,95 $	Ja	Ja	www.geotrust.com
IKS CA	7,50 €	Nein	Nein	www.iks-jena.de
Thawte	0 €	Ja	Ja	www.thawte.com
VeriSign	19,95 $	Ja	Ja	www.verisign.com

Tabelle 3.1: Beispielhafte Auswahl einiger Anbieter privater Email-Zertifikate und ob den CA-Zertifikaten der Anbieter in der Voreinstellung in Outlook/Outlook Express (Windows) und Thunderbird (NSS) vertraut wird. Datengrundlage sind die genannten Webseiten der Anbieter am 2007-12-02

3.5 Key Continuity Management

Auf Gutmanns Vorschlag hin [Gut04] wurde im CoPilot-Projekt [GM05] auf eine zentrale Zertifizierungsstelle verzichtet. Stattdessen wird eine so genannte opportunistische Verschlüsselung mittels Key Continuity Management (KCM) verwendet. Bei KCM wird bei der ersten Kommunikation mit einer fremden Email-Adresse jeder verfügbare Schlüssel automatisch angenommen. Dieser Schlüssel wird gespeichert und bei jeder

weiteren Kommunikation verwendet. Wenn sich der Schlüssel des Gegenübers ändert, kommt es zu einer Warnung, da eventuell ein Angreifer seinen eigenen Schlüssel als den des Gegenübers ausgibt.

Diese Methode kommt fast völlig ohne Benutzereingriffe aus, hat aber auch Nachteile. Es muss zusätzliche Software auf dem Zielsystem installiert werden und das System ist per Design für einige Man-In-The-Middle-Angriffe verwundbar, etwa wenn schon der erste Schlüsselaustausch abgefangen wurde.

Teil II

Entwicklung der Server-Komponenten

4 Entwicklung der Server-Komponenten

In diesem und dem nächsten Teil des Dokuments wird die Entwicklung der im Rahmen dieser Arbeit entstandenen PKI beschrieben. Dieser Teil handelt von den Server-seitigen Komponenten, der nächste Teil betrachtet das System aus der Perspektive der Clients.

Dieses Kapitel beschreibt überblicksartig den technischen Aufbau des Systems. Auf Grundlage dieser Beschreibung wird in den darauf folgenden Kapiteln näher auf die einzelnen Server-seitigen Komponenten eingegangen.

4.1 Motivation

Im Kapitel 3 wurden verschiedene Systeme vorgestellt, mit denen man Emails kryptographisch absichern kann. Die Systeme unterstützen unterschiedlich gut auch technisch unbedarfte Anwender im privaten Bereich, bisher hat sich allerdings kein durchschlagender Erfolg in dieser Zielgruppe abgezeichnet.

Die CoPilot-Autoren Garfinkel und Miller adressierten mit ihrem Projekt explizit die genannte Anwendergruppe, nannten als Alternative zu ihrem Programm aber die Idee, den Prozess der Zertifikatsbeantragung zu vereinfachen [GM05]. An diese Idee knüpft diese Arbeit an. Ein privater Nutzer soll mit dem für ihn geringstmöglichen Aufwand in eine PKI zur kryptographisch sicheren Email-Kommunikation eingebunden werden.

4.2 Struktur des Systems

Damit ein Email-Benutzer Kryptographie nutzen kann, müssen verschiedene Voraussetzungen erfüllt sein, wie sie in dem Kapitel 2 beschrieben sind. Das im Rahmen dieser Arbeit eingesetzte System ist in Abbildung 4.1 skizziert und wird im Folgenden beschrieben.

In einer funktionierenden PKI muss jeder Benutzer ein Schlüsselpaar bestehend aus öffentlichem und privatem Schlüssel besitzen. Der öffentliche Schüssel muss zusätzlich von einer vertrauenswürdigen CA signiert sein, es muss also ein Zertifikat für dieses Schlüsselpaar existieren.

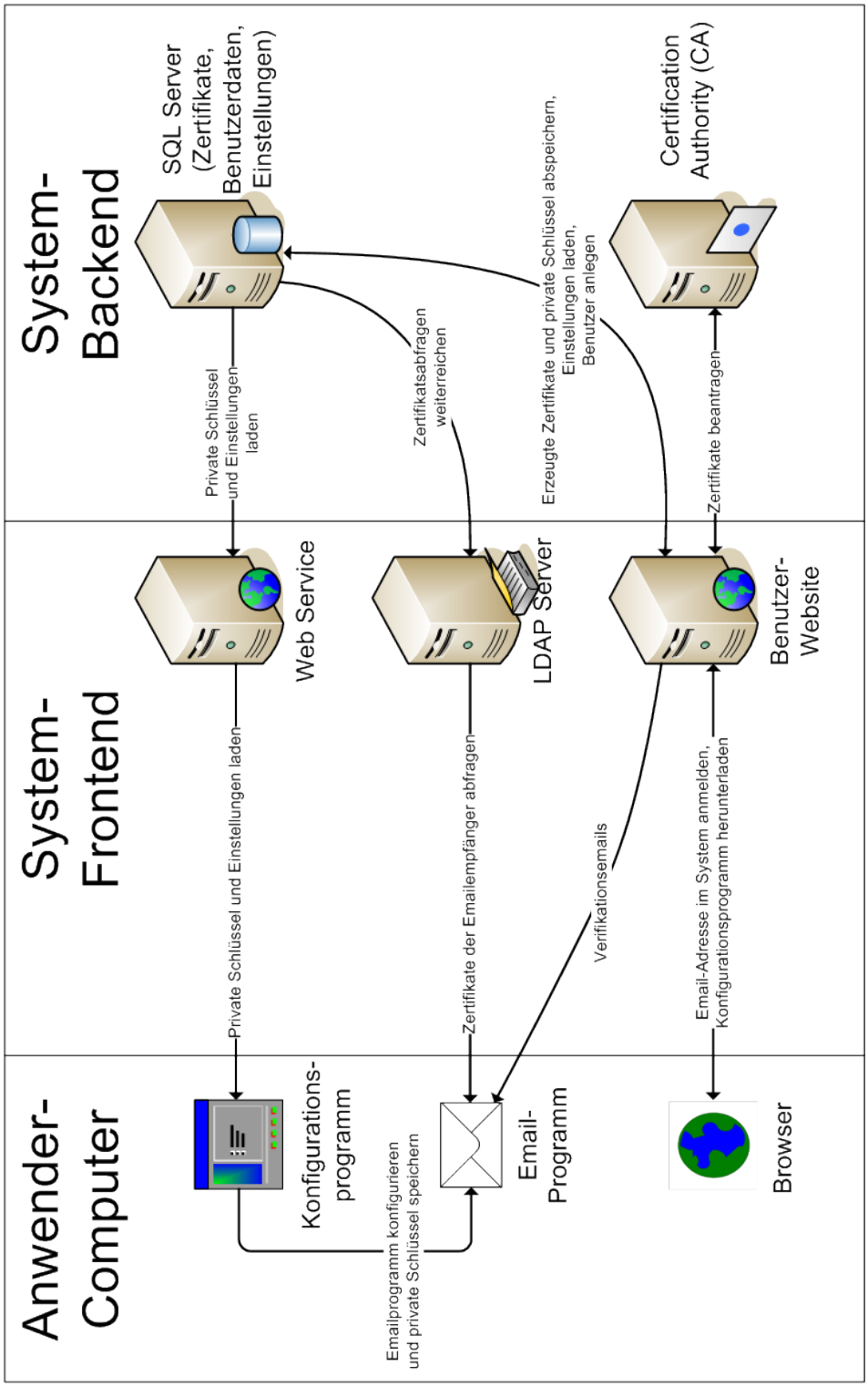

Abbildung 4.1: Darstellung der aus Benutzerperspektive beteiligten Komponenten der PKI. Die Pfeile geben die logische Datenflussrichtung der Kommunikation an und machen keine Aussage darüber, von welcher Komponente die Kommunikation ausgeht

Dieses Zertifikat muss allen Personen, die gesichert mit dem Benutzer kommunizieren wollen, zur Verfügung gestellt werden. Dies geschieht über ein Verzeichnis in Form eines LDAP-Servers. Dies ist auch die einzige Möglichkeit einer zentralen Speicherung der Zertifikate, die von Microsoft Outlook, Microsoft Outlook Express und Mozilla Thunderbird ohne Drittprogramme unterstützt wird.

Die über LDAP zur Verfügung gestellten Zertifikate müssen in einer Datenbank gespeichert werden. In einer Administrationsoberfläche sollen Berechtigte die Möglichkeit haben, sich einen Überblick über ausgestellte Zertifikate zu verschaffen und bei eventuellen Problemen eingreifen können.

Die normalen Benutzer des Systems sollen auf einer Webseite in die Benutzung des Systems eingeführt werden. Hier sollen sie die Möglichkeit haben, mit geringem Aufwand selbst am System teilzuhaben. Nach einer Registrierung sollten die Prozesse für den Benutzer also möglichst transparent ablaufen.

Damit das Email-Programm des Benutzers Emails entschlüsseln und signieren kann, muss es Zugriff auf das Schlüsselpaar des Benutzers haben. Die verbreiteten Email-Programme Outlook, Outlook Express und Thunderbird speichern ihr Schlüsselmaterial an unterschiedlichen Stellen in so genannten Stores: Outlook und Outlook Express nutzen den in Microsoft Windows eingebauten Store, Thunderbird greift auf das Schlüsselmaterial in der proprietären Datenbank der Network Security Services (NSS) zu[1]. Es muss also dafür Sorge getragen werden, dass sich Schlüsselmaterial in dem Store befindet, der für das verwendete Email-Programm relevant ist. Die drei genannten Email-Programme unterstützen Zertifikate nach dem Standard X.509 und Email-Verschlüsselung nach S/MIME, aber nicht OpenPGP. Die PKI ist daher auf eine PKIX-Umgebung, wie sie in Abschnitt 2.1 beschrieben wird, ausgerichtet.

Das Email-Programm des Benutzers muss außerdem so eingestellt werden, dass es versendete Emails automatisch an die richtige Person verschlüsselt und signiert. Dazu muss es das Zertifikat des Empfängers vom LDAP-Server herunterladen und den eigenen privaten Schlüssel aus seinem Store zur Signatur verwenden. Die passenden Einstellungen müssen im Email-Programm hinterlegt werden.

Die Konfiguration des Email-Programms und die Speicherung des Schlüsselmaterials am korrekten Ort übernimmt ein Konfigurationsprogramm, welches der Benutzer ausführt. Das Programm wird von der Webseite heruntergeladen und führt den Benutzer mit Hilfe eines Assistenten durch den Konfigurationsprozess. Die Entwicklung und Funktionsweise dieses Programms wird in Teil III detailliert beschrieben.

Die Zertifikate müssen auf einem zentralen Server erzeugt werden. Die Erzeugung der zugrunde liegenden Schlüssel kann auch auf dem Server oder auf den Rechnern der einzelnen Benutzer geschehen. In jedem Fall muss eine Kommunikation zwischen der CA-Komponente (da sie die Zertifikate erzeugt) und dem Rechner des Benutzers (da er den privaten Schlüssel speichert) stattfinden: Entweder muss die Server-Komponente sowohl privaten Schlüssel als auch Zertifikat erzeugen und dann den privaten Schlüssel an den Rechner des Benutzers übermitteln oder der private Schlüssel wird auf dem Rechner des Benutzers erzeugt. Dann muss aber der öffentliche Schlüssel an den Server übertragen werden, damit dieser ein Zertifikat erzeugt. Diese Kommunikation findet über einen Web Service auf dem Server über das Internet statt. Angesprochen wird der Web Service von dem auf dem Client ausgeführten Konfigurationsprogramm.

[1] Diese Schlüsseldatenbanken werden in den Abschnitten 8.3.4 und 8.5.4 eingehender beschrieben.

5 Certification Authority

Definitionsgemäß stellt die CA Zertifikate für die Benutzer aus. In diesem Kapitel wird allerdings zusätzlich noch die Generierung des privaten Schlüssels betrachtet, weil diese beiden Teile in der Systemstruktur eng zusammenhängen.

Um flexibel im Aufbau des Systems zu bleiben, wurden die einzelnen Komponenten modular entwickelt. Dadurch lässt sich der Teil des Programms zur Schlüsselgenerierung und derjenige zur Zertifizierung durch Neuprogrammierung eines Moduls austauschen. Dazu wurden Interfaces spezifiziert, die die Funktionen einer Schlüsselgenerierungskomponente und einer Zertifizierungskomponente festlegen. Konkrete Komponenten wurden durch Klassen realisiert, welche die Interfaces implementieren. Die entsprechenden Klassen und Komponenten wurden in einer Programmbibliothek zusammengefasst und können von externen Programmen wie dem in Kapitel 6 beschriebenen Web-Server verwendet werden.

Die Schlüsselgenerierungskomponente muss ein Schlüsselpaar bestehend aus öffentlichem und geheimem Schlüssel erzeugen. Den öffentlichen Schlüssel speichert sie mit einer Signatur des geheimen Schlüssels im PKCS #10-Format, so wie in Abschnitt 2.1.3 beschrieben. Diese Daten im PKCS #10-Format werden an die Zertifizierungskomponente weitergegeben, die mit den Daten ein Zertifikat erstellt. Dieses Zertifikat wird zusammen mit dem geheimen Schlüssel in einem Datenpaket des PKCS #12-Formats abgespeichert und kann dann von der externen Anwendung verwendet werden. In der vorliegenden Arbeit sind die im Kapitel 6 beschriebenen Web-Anwendungen diese externen Anwendungen.

5.1 Schlüsselgenerierung

Eine Schlüsselgenerierungskomponente kann auf dem Serversystem betrieben werden, die privaten Schlüssel müssen dann auf den Zielrechner übertragen werden, oder die Schlüsselgenerierungskomponente läuft direkt auf dem Zielrechner.

Vorteile einer Erzeugung und Speicherung der privaten Schlüssel auf dem Server sind die zentrale Verwaltung der Schlüsselgenerierungskomponente und das Backup der privaten Schlüssel. Falls ein Benutzer seinen privaten Schlüssel verliert, kann er das Backup herunterladen, womit verschlüsselte Daten nicht verloren sind. Der Nachteil ist die niedrigere Sicherheit, denn bei einem Angriff auf den Server könnten sämtliche privaten Schlüssel ermittelt werden. Außerdem haben Administratoren mit Datenbankzugriff Zugriff auf die privaten Schlüssel. Die Nachteile werden dadurch etwas relativiert, dass bei einem Angriff auf den Server die CA unter der Kontrolle des Angreifers steht und somit ohnehin neue Zertifikate für private Schlüssel im Besitz des Angreifers erzeugt werden könnten. Da die Vorteile eines Backups überwiegen, werden die privaten Schlüssel in der Datenbank gespeichert. Unter dieser Voraussetzung hat die Generierung der Schlüssel

auf dem Server keinen Nachteil mehr, so dass die Schlüsselgenerierungskomponente zu den Serverkomponenten zählt.

Die Schlüsselgenerierungskomponente muss aber nicht nur in der Lage sein, ein RSA-Schlüsselpaar zu generieren, sie muss daraus auch einen Zertifikatsantrag im PKCS #10-Format erzeugen und zusammen mit dem von der CA erzeugten Zertifikat ein Datenpaket im PKCS #12-Format mit privatem Schlüssel bilden.

Damit scheiden alle Systeme aus, die nur PGP verarbeiten können, zum Beispiel die stabilen Versionen von GPG [GPG]. Das .NET Framework enthält zwar Funktionen zur Erzeugung eines RSA-Schlüssels, nicht aber zur Speicherung im PKCS #10-Format und kommt somit ebenfalls nicht in Betracht [Mic05]. Die näher in Betracht kommenden Schlüsselgenerierungskomponenten werden im Folgen beschrieben.

5.1.1 OpenSSL

Im Rahmen des OpenSSL-Projekts wurde eine Bibliothek entwickelt, mit der man Secure Sockets Layer (SSL) und TLS nutzen kann. Mit diesem Hintergrund bietet OpenSSL allerdings noch weitere Funktionen, die nur mittelbar mit SSL und TLS in Verbindung stehen. Dazu gehören auch die Erzeugung von RSA-Schlüsselpaaren und der Umgang mit verschiedenen PKCS-Formaten. [Opea]

Zu OpenSSL gehören eine Kommandozeilenanwendung und zwei Bibliotheken. Durch direktes Laden der Bibliotheken lässt sich OpenSSL daher in eine Schlüsselgenerierungskomponente einbinden. Andererseits könnte man auch den Umweg über die Kommandozeilenanwendung gehen, was vielleicht eine einfachere Schnittstelle bedeuten würde.

OpenSSL ist quelloffen, allerdings sind sowohl die Kommandozeilenanwendung als auch die Bibliotheken nur unvollständig dokumentiert. Da OpenSSL immer noch weiter entwickelt wird, hätte die Verwendung dieser Komponente den positiven Effekt, dass eine Anpassung an neue Standards und das Beheben von Sicherheitslücken in der Komponente mit keinem bis geringem Aufwand möglich wäre. Man müsste in diesem Fall nur auf die neueste OpenSSL-Version aktualisieren.

5.1.2 Network Security Services

Die NSS [Moz07b] sind eine von Netscape für kryptographische Anwendungen entwickelte Bibliothek. Da diese Bibliothek auch im Konfigurationsprogramm zur Ansteuerung der Thunderbird-Schlüsseldatenbank verwendet wird, findet sich eine genauere Beschreibung der NSS in Abschnitt 8.5.5.

NSS ist wie OpenSSL quelloffen, aber unvollständig dokumentiert. Auch der relevante Funktionsumfang ist vergleichbar und wie OpenSSL kann NSS als Dynamic Link Library (DLL) geladen werden. NSS wird immer noch weiterentwickelt und an Neuerungen in den Standards angepasst.

5.1.3 Kommerzielle Systeme

Kommerzielle Systeme kosten im Allgemeinen mehr und sind an das Lizenzmodell des Entwicklers gebunden. Wenn sie nicht quelloffen sind, kann man im Falle mangelnder Dokumentation nicht auf den Quelltext zurückgreifen, um ein Problem zu lösen. Dafür erhält man normalerweise Beratung durch den Entwickler.

5.1.4 Auswahl eines Schlüsselgenerierungssystems

Da die Vorteile einer freien Lösung die einer kommerziellen überwiegen, musste für die Referenzrealisierung eine Entscheidung zwischen OpenSSL und NSS getroffen werden. Die beiden Systeme erscheinen für den Anwendungsfall gleichwertig. Da NSS auch für eine Client-Komponente im Konfigurationsprogramm verwendet wird, fiel die Entscheidung zu Gunsten von NSS aus. Das Gesamtprogramm ist damit von weniger externen Komponenten abhängig und der Quelltext verständlicher.

5.2 Zertifizierung

Die Zertifizierungskomponente soll mit Hilfe eines Zertifikatsantrags im PKCS #10-Format ein Zertifikat ausstellen können. Dazu muss sie den geheimen Schlüssel zu einem vertrauenswürdigen, öffentlichen Schlüssel besitzen, also einem vertrauenswürdigen CA-Zertifikat nach X.509-Standard.

Sie sollte außerdem eine Liste der ausgestellten und widerrufenen Zertifikate führen können, um CRLs erstellen und OCSP-Anfragen beantworten zu können, wie in Abschnitt 2.1.2 beschrieben. Die Liste der ausgestellten Zertifikate ist notwendig, damit keine Zertifikatsseriennummern doppelt vergeben werden.

Es gibt verschiedene Grundsysteme, aus denen man eine Zertifizierungskomponente entwickeln könnte. Die in der engeren Auswahl existierenden Komponenten werden nun vorgestellt.

5.2.1 Eigenentwickelte Signatur und Zertifikatsverwaltung

Als nahe liegende Möglichkeit könnte man OpenSSL, NSS oder ein ähnliches System dazu benutzen, die auszustellenden Zertifikate zu erzeugen und zu signieren. Die Zertifikatslisten müssten dann in einer Datenbank gespeichert und eine Verwaltung dieser Listen eigens programmiert werden. Falls auch OCSP gewünscht ist, müsste man die entsprechende Abfrageschnittstelle ebenfalls neu entwickeln.

Vorteil dieser Methode ist, dass man die Zertifikate beliebig detailliert ausgestalten und den Signaturablauf genau gemäß den Anforderungen gestalten kann. Einschränkungen bezüglich des Inhalts der Zertifikate sind nur durch die verwendete kryptographische Bibliothek gegeben. Beispielsweise NSS bietet aber sogar Methoden zur Steuerung auf sehr niedrigen Kodierungsebenen, wodurch faktisch keine Grenzen in Bezug auf mögliche Attributwerte und ähnliches gegeben sind. Da NSS und OpenSSL kostenlos eingesetzt werden dürfen und auch sonst keine kommerzielle Lösung bezahlt

werden muss, sind für diese Methode keine Kosten außer dem Arbeitsaufwand fällig.

Dieser Arbeitsaufwand ist dafür allerdings als recht hoch anzusetzen, da die einzelnen Teilkomponenten wie CA, CRL-Erzeugung und OCSP-Schnittstelle entwickelt werden müssen.

5.2.2 Externe Anbieter

Wenn die Zertifikatsgenerierung an einen externen Anbieter wie VeriSign delegiert werden würde, könnte man auf eine von vielen Programmen als vertrauenswürdig angesehene CA zurückgreifen. Die CA-Zertifikate einiger externer Anbieter werden sowohl in Outlook Express und Outlook, als auch in Thunderbird schon bei der Installation gespeichert. Damit wird auch allen Zertifikaten, die von solch einer Stelle signiert wurden, sofort vertraut.

Leider sind damit die Probleme nicht vermieden, die schon in Abschnitt 3.4 angesprochen wurden. Außerdem wird bei den genannten Anbietern keine automatisierte Zertifikatsbeantragung angeboten. Die Möglichkeit eines externen Anbieters als CA scheidet damit aus.

5.2.3 Microsoft Certificate Services

Bei den Microsoft Server Betriebssystemen werden als eine zusätzlich installierbare Komponente die Microsoft Certificate Services mitgeliefert. Diese fungieren als CA, die über ein dokumentiertes Application Programming Interface (API) angesprochen werden kann [Mic05]. In einer eigenen Datenbank werden die ausgestellten und widerrufenen Zertifikate verwaltet. Die CRLs können sowohl automatisch als auch manuell erstellt werden. Einsicht in die genannten Listen und ähnliche Verwaltungsaufgaben können über eine mitgelieferte grafische Oberfläche durchgeführt werden. Die Microsoft Certificate Services ab dem Microsoft Server-Betriebssystem mit dem Codenamen „Longhorn“ unterstützen das OCSP-Protokoll und bieten eine entsprechende Abfrageschnittstelle, davor ist ein Zertifikatswiderruf nur über CRLs möglich. Der private Schlüssel des CA-Zertifikats befindet sich in der Windows-Schlüsseldatenbank (siehe auch Abschnitt 8.3.4).

Um die Microsoft Certificate Services laufbereit zu machen, muss nur wenig Zeit aufgewendet werden. Die Programmierung ist ebenfalls mit geringen zeitlichen Ressourcen durchzuführen, da das System über ein gut dokumentiertes API angesprochen werden kann. Außerdem werden nach bestimmten Richtlinien Attribute in den ausgestellten Zertifikaten gesetzt, so dass die Möglichkeit eines schlecht aufgebauten Zertifikats verringert wird. Nachteilig ist daran, dass die Zertifikate bei der Ausstellung schwer oder sogar gar nicht verändert und den Wünschen des Administrators angepasst werden können. Die Microsoft Certificate Services sind zwar im Preis eines Windows Servers inbegriffen, dieser kostet ohne Vergünstigungen jedoch mindestens $999. Die kostengünstigere Web Edition kann für den geplanten Einsatzzweck nicht genutzt werden [Micb].

5.2.4 Auswahl einer Zertifizierungskomponente

Da eine Zertifizierungskomponente auf Basis der Microsoft Certificate Services deutlich schneller programmiert werden kann als eine Eigenentwicklung, wurde auf die höhere Anpassbarkeit der Eigenentwicklung verzichtet und die Microsoft Certificate Services als Kern der Referenz-Zertifizierungskomponente gewählt. Mittels Adapter-Pattern [GHJV95] kann das API an das definierte Interface einer Zertifizierungskomponente angepasst und so in das Zielsystem eingebaut werden. Die CRL wird über die Web- und Verzeichniskomponenten publiziert, so dass sie über LDAP und HTTP abrufbar ist.

6 Web-Server

In verschiedenen Prozessschritten kommunizieren die Server-Komponenten des Systems mit außerhalb liegenden Komponenten. Die Emailbenutzer müssen dem System über eine Weboberfläche ihre Email-Adresse bekannt machen, damit ein Zertifikat erstellt werden kann. Die Administratoren des Systems brauchen eine Oberfläche, um die Benutzer und ihre Zertifikate verwalten zu können. Das Konfigurationsprogramm auf den Rechnern der Benutzer muss mit dem Serversystem kommunizieren.

Die Kommunikation zwischen Emailbenutzern und dem Serversystem erfolgt über dynamische Webseiten und auch die Administrationsoberfläche wird als dynamische Webseiten realisiert. Im Falle der Emailbenutzer wurde das Kommunikationsmedium der Webseite gewählt, da fast jeder Emailbenutzer auch einen Browser besitzt und somit die Zielgruppe angesprochen werden kann.

Die Administration erfolgt ebenfalls über Webseiten, weil dadurch nur ein Programm entwickelt werden muss - nämlich der Code hinter der Webseite - und kein Client- und Serversystem. Außerdem müssen die Administratoren keine zusätzliche Software installieren und die Kommunikation über HTTP und seiner verschlüsselten Variante HTTP/TLS ist oft auch durch Firewalls, zum Beispiel aus Firmennetzwerken, möglich.

Das Konfigurationsprogramm braucht eine für die Maschine-Maschine-Kommunikation besser geeignete Schnittstelle als eine herkömmliche Webseite. Hierfür bietet sich ein Web Service an. Die Emailbenutzer müssen nur über HTTP nach außen kommunizieren, so dass eine etwaige Firewall mit einer geringeren Wahrscheinlichkeit die Kommunikation unterbindet. Web Services benutzen das auf HTTP aufsetzende Protokoll Simple Object Access Protocol (SOAP) zur Kommunikation. Da SOAP vom W3C standardisiert wurde, können zukünftig auch leicht weitere Konfigurationsprogramme entworfen werden, die mit dem Web Service kommunizieren, zum Beispiel um weitere Plattformen der Benutzer zu unterstützen [CCMW01, GHM+07].

Programmiersprache

Da alle drei Programme, Benutzeroberfläche, Administrationsoberfläche und Web Service, eine ähnliche Plattform haben - sie müssen auf einem Web-Server laufen - und auf die gleiche Datenbank zugreifen, liegt es nahe, für alle drei Programme die gleiche Programmiersprache oder zumindest das gleiche Framework zu verwenden.

Für Webseiten und Web Services bietet das .NET Framework mit dem Visual Studio 2005 zumindest in den Sprachen C# und Visual Basic .NET mehr Unterstützung als eine vergleichbare Java-Anwendung. Da ich andere Lösungen wie die Scriptsprache PHP nicht gut genug kenne, um sie in Erwägung zu ziehen, entschied ich mich dazu, C# als Programmiersprache für die drei Anwendungen zu benutzen. Die in Abschnitt 5.1 beschriebenen Klassen zur Schlüsselgenerierung wurden dagegen in

einer Bibliothek der Sprache Managed C++ zusammengefasst. In Managed C++ kann man den Quelltext aus C/C++ und .NET-Konstrukten zusammensetzen. Mit den C-Passagen kann man das NSS-API leichter benutzen, da dieses ebenfalls in C geschrieben wurde. Die Schnittstelle zu anderen .NET-Sprachen ist mit den .NET-Konstrukten leichter umzusetzen. Die Webseiten ebenfalls in Managed C++ zu schreiben, ist dagegen nicht möglich, denn diese Kombination wird vom Visual Studio 2005 nicht unterstützt.

Als Web-Server für die fertigen Webseiten dienen die Microsoft Internet Information Services (IIS). Sie können bei Windows Server 2003 und Windows „Longhorn“ schon bei der Installation des Betriebssystems mitinstalliert werden. Auf diesen beiden Server-Betriebssystemen enthalten die IIS auch gleich eine .NET-Schnittstelle, um .NET-Code dynamischer Seiten auszuführen. Mit den Active Server Pages .NET (ASP .NET) kann in Dateien mit der Endung *.aspx* der Aufbau der Seite in einer XML-Sprache vorgegeben werden, der zugehörige Quelltext wird in einer getrennten Datei gespeichert. Wenn man den Quelltext nicht auf dem Web-Server speichern möchte, kann man den Quelltext auch zu einer DLL übersetzen und nur noch die *.aspx*-Dateien und die DLL im Verzeichnis des Web-Servers ablegen.

6.1 Internet-Auftritt

Die Emailbenutzer können einen Internet-Auftritt nutzen, um ihre Email-Adressen einzutragen und dadurch den Konfigurationsprozess in Gang zu setzen. Der komplette Vorgang ist in Abbildung 6.1 dargestellt. Der Benutzer erhält eine Verifikations-Email, nachdem er seine Email-Adresse eingetragen hat. Klickt der Benutzer auf den in der Verifikations-Email enthaltenen Verifikations-Link, ist damit gezeigt, dass der Besitzer der Email-Adresse die Email empfangen hat. Schließlich kann jeder eine Email-Adresse eintragen, den Link kann nur der tatsächliche Besitzer betätigen. Dieser ist damit auch berechtigt, ein Zertifikat zu erhalten, das den Besitz der Email-Adresse zertifiziert. Wenn nur eine Email-Adresse bestätigt werden sollte, kann sich der Nutzer auf der verlinkten Webseite ein „markiertes“ Konfigurationsprogramm herunterladen und ausführen. Dem Binärcode des Konfigurationsprogramms wird beim Download eine Markierung hinzugefügt, die eindeutig kennzeichnet, von welchem Benutzer es heruntergeladen wurde. Bei Ausführung liest das Konfigurationsprogramm dann seine eigene Markierung aus und lädt den privaten Schlüssel und die durchzuführenden Einstellungen am Email-Programm des Benutzers von dem in Abschnitt 6.3 beschriebenen Web Service nach. Die genaue Funktionsweise des Konfigurationsprogramms ist im Teil III dieser Arbeit beschrieben. Da jeder mit Kenntnis der Markierung den privaten Schlüssel abfragen kann, muss die Markierung so beschaffen sein, dass sie nicht erraten werden kann. Die Sicherheit der Markierung wird in Abschnitt 9.3 bewiesen.

6.1.1 Widerruf und Erneuerung der Zertifikate

Wenn der private Schlüssel eines Emailbenutzers kompromittiert wurde, muss das System das zugehörige Zertifikat als widerrufen kennzeichnen, damit niemand mehr Emails mit dem Schlüssel verschlüsselt und mit diesem Schlüssel signierte Emails nicht mehr als authentisch eingestuft werden. Damit ein Benutzer einen Widerruf initiieren kann, muss er aber zuerst beweisen, dass er auch tatsächlich der Inhaber der zertifizierten

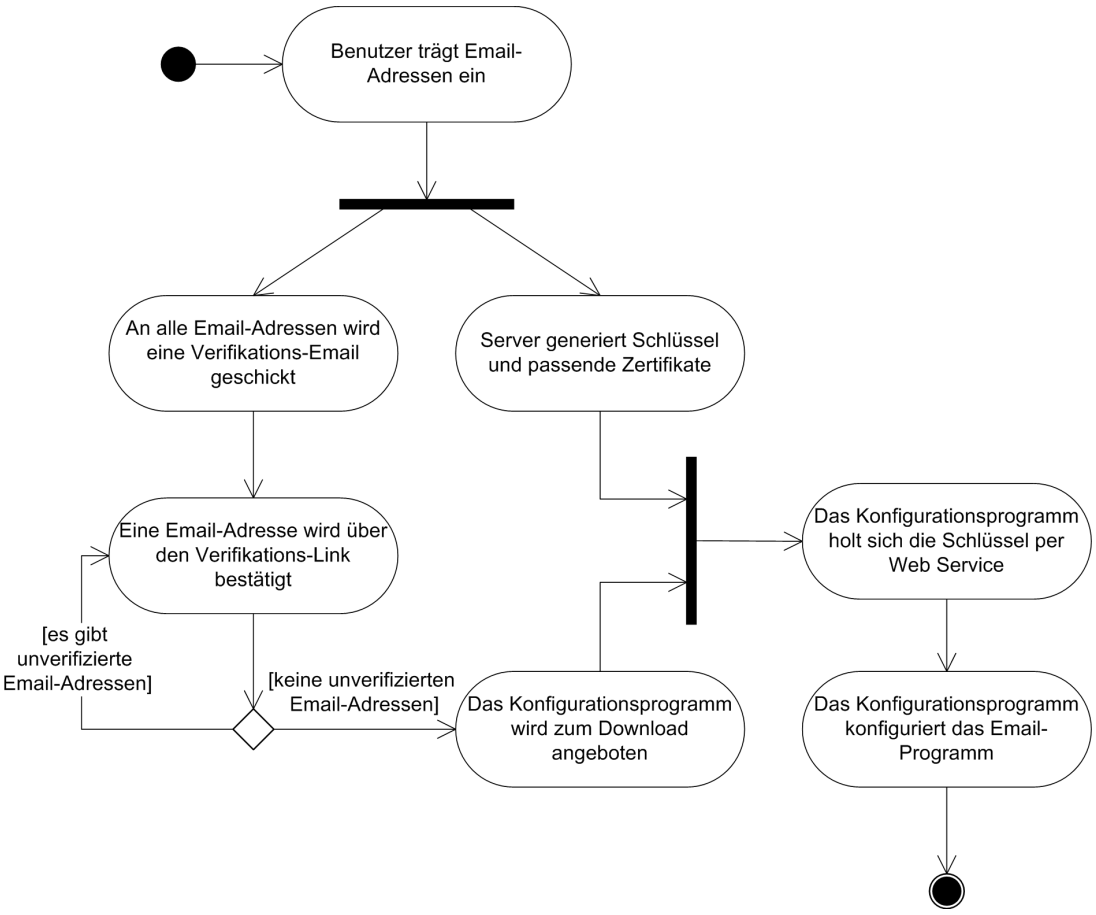

Abbildung 6.1: UML-Aktivitätsdiagramm zum Beantragungsprozess eines Benutzers

Email-Adresse ist. Dies geschieht wie bei der Beantragung über eine Verifikations-Email. Wird der Verifikations-Link in der Email angeklickt, gilt dies als Bestätigung des Widerrufs und das Zertifikat wird zurückgezogen, das heißt in die CRL aufgenommen.

Ein ähnliches Problem ergibt sich, wenn ein Zertifikat sein zum Ausstellungszeitpunkt festgelegtes Ablaufdatum erreicht. Wenn der Benutzer keine Maßnahmen ergreift, besitzt er kein gültiges Zertifikat mehr und erhält weder verschlüsselte Emails noch kann er seine versendeten Emails signieren. Eine Erneuerung des Zertifikats kann manuell über die Webseite erfolgen, der Benutzer sollte aber auch kurz vor Ablauf des Zertifikats vom System automatisch per Email gewarnt werden. In diesem Fall befindet sich der Verifikations-Link schon in der Warn-Email.

6.2 Administration

Damit die Administratoren nicht direkt auf die Datenbank zugreifen müssen, wenn sie Systeminformationen benötigen oder bestimmte Benutzerdaten ändern möchten, ist eine Administrationsoberfläche als Web-Applikation realisiert. Diese erleichtert die Bedienung, da die Daten automatisch interpretiert und entsprechend formatiert werden.

Außerdem werden Abhängigkeiten zwischen verschiedenen Tabellen schon bei der Darstellung ausgewertet und präsentiert.

Eine Seite der Administrationsoberfläche enthält eine Übersicht der dem System bekannten Email-Benutzer. Zu jeder Email-Adresse ist zum Beispiel aufgelistet, wann der Benutzer sich zuerst angemeldet und wie viele Zertifikate er heruntergeladen hat.

Ein weiterer wichtiger Teil der Administrationsoberfläche ist die Konfigurationsseite. Hier können globale Einstellungen eingesehen, verändert und in der Datenbank gespeichert werden. Die Einstellungen beeinflussen dann auch das Verhalten anderer Komponenten des Systems, wie des Konfigurationsprogramms und des Web Services. Solche Werte direkt im Quelltext festzulegen, senkt die Flexibilität des Systems, da bei einer Änderung der Randbedingungen alle betroffenen Programme neu kompiliert werden müssen. Konfigurierbare Einstellungen sind beispielsweise Texte und Namen, die nur die Darstellung beeinflussen, sowie die URI oder Internet Protocol (IP)-Adressen der am System beteiligten Server. Durch die letztgenannten Einstellungen bestimmt sich etwa der Mailserver, über den Emails des Systems zu ihrer Zieladresse geleitet werden.

6.2.1 Zugriffsberechtigung

Damit nur Administratoren Zugriff auf diese sensiblen Informationen haben, muss ein Authentifizierungsmechanismus den Zugriff auf die Administrationsoberfläche regeln.

Der IIS-Web-Server bietet schon ohne Anpassung einen Authentifizierungsmechanismus an. Man kann für jede Seite oder ganze Web-Applikationen einstellen, welche Benutzer darauf zugreifen dürfen, indem man die Berechtigungen auf Dateisystemebene verändert. Bei den Browsern Microsoft Internet Explorer und Mozilla Firefox geht beim Zugriff auf eine solcherart geschützte Webseite ein Dialog auf, in dem man Benutzernamen und Passwort eintragen kann. Passen die Daten zusammen und ist der Benutzer zum Zugriff auf die Seite berechtigt, dann wird die Seite angezeigt. Ansonsten wird ein Fehler gemeldet.

Eine andere Möglichkeit ist eine formularbasierte Authentifizierung. Hierbei erhält der zugreifende Benutzer auf Dateisystemebene, im Gegensatz zur Authentifizierung durch die IIS, immer die Rechte, um die angeforderte Seite zu erreichen. Der Code der Web-Applikation überprüft hierbei selbst, ob der Benutzer angemeldet ist, und wenn ja, ob er die Berechtigung hat, die entsprechende Seite zu sehen. Eine Seite zur Authentifizierung muss die Web-Applikation in diesem Fall selbst zur Verfügung stellen.

Bei der zweiten Möglichkeit ist eine feiner granulierte Rechtevergabe möglich, da beispielsweise auch innerhalb einer einzelnen Seite bestimmte Elemente für unberechtigte Benutzer ausgeblendet werden können. Außerdem kann die Authentifizierungsseite an das Aussehen der übrigen Seiten angepasst werden, um ein einheitlicheres Design zu erhalten. Berechtigungen und Benutzer können von der Web-Applikation selbst auch zur Laufzeit geändert werden, wenn man diese in einer Datenbank hinterlegt. Die erste Möglichkeit ist allerdings deutlich einfacher zu realisieren, der Quelltext einer so geschützten Web-Applikation muss nicht einmal verändert werden. Unter der Annahme, dass die

Zahl der Administratoren gering bleibt und keine fein granulierte Rechtevergabe nötig ist, wurde die Rechtevergabe direkt über die IIS gewählt. Wenn sich herausstellen sollte, dass ein anspruchsvolleres Rechtesystem nötig ist, kann das System erweitert werden.

6.3 Web Service

Das Konfigurationsprogramm auf den Client-Rechnern erhält seine veränderbaren Informationen über einen Web Service. Die Kommunikation zwischen Client-Rechner und Web Service ist mit dem SSL-Protokoll gesichert. Da die übertragenen Informationen durch SSL verschlüsselt werden, erfährt kein Dritter die übertragene Email-Adresse. Außerdem wird durch SSL sichergestellt, dass sich kein Dritter als Web Service ausgeben und dem Benutzer zum Beispiel einen korrumpierten privaten Schlüssel übertragen kann.

Allgemeine Einstellungen wie das einzutragende LDAP-Verzeichnis können über die Funktion *getLDAPConfig* abgerufen werden. Zertifikate können allgemein über *getPublicKey* abgerufen werden, hierbei dient die Email-Adresse des abzurufenden Zertifikats als Parameter. Die speziellen Parameter „currentRoot“ und „currentSub“ identifizieren die momentan gültigen CAs erster und zweiter Ebene, also eine Root CA und eine ihr untergeordnete CA, wie in Abschnitt 2.1.1 beschrieben. Tatsächlich ruft das Konfigurationsprogramm auch nur diese beiden CA-Zertifikate ab, denn die Zertifikate der anderen Benutzer werden später automatisch vom Email-Programm über LDAP abgefragt. Um den privaten Schlüssel über *getPrivateKey* abzurufen, muss die in Abschnitt 9.3 beschriebene Markierung als Authentifizierung übergeben werden.

Der Web Service überprüft mittels der im Abschnitt 7.2 beschriebenen Datenbank, ob eine angefragte Markierung tatsächlich auch in einem Konfigurationsprogramm herausgegeben wurde. Über die Markierung ist der Web Service in der Lage, den privaten Schlüssel des Anfragenden in der Datenbank zu finden und herauszugeben (siehe auch Abschnitt 7.2.1). Dabei wird in der Datenbank gespeichert, dass dieser Schlüssel nun heruntergeladen wurde und so verhindert, dass der Schlüssel ein zweites Mal heruntergeladen werden kann.

So kann unterbunden werden, dass ein Dritter den privaten Schlüssel herunterladen kann, wenn er die unverschlüsselte Verifikations-Email mitgelesen hat. Wenn er versucht, den privaten Schlüssel nach dem Benutzer herunterzuladen, erhält er eine Fehlermeldung. Wenn er den privaten Schlüssel vor dem Benutzer herunterlädt, wird der Benutzer darauf hingewiesen, dass sein privater Schlüssel schon einmal heruntergeladen wurde. Der Benutzer wird darauf aufmerksam gemacht, dass jemand die Email mitgelesen haben könnte. Im Anschluss kann er den in falsche Hände geratenen Schlüssel widerrufen lassen.

7 LDAP-Verzeichnisdienst und Datenbank

Die Serversysteme müssen verschiedene Arten von Daten speichern und Teile von ihnen der Außenwelt zur Verfügung stellen. Die beiden wichtigsten Datenbanken sind ein Structured Query Language (SQL)-Server und ein Verzeichnisdienst, wobei der Verzeichnisdienst auf dem SQL-Server beruht, was in diesem Kapitel näher erläutert wird. Während der SQL-Server Daten nur für die einzelnen Server-Komponenten zur Verfügung stellt, aber nicht direkt von außen erreichbar ist, dient der Verzeichnisdienst als Schnittstelle zwischen Daten und der Außenwelt. Die andere - deutlich indirektere - Schnittstelle der Außenwelt zu den Daten des Systems sind die in Kapitel 6 beschriebenen Web-Komponenten.

7.1 Verzeichnisdienst

Die LDAP-Komponente hat, wie in Abschnitt 2.4.1 beschrieben, die Aufgabe, den Benutzern der PKI die Zertifikate der anderen Benutzer zur Verfügung zu stellen. Die Email-Programme suchen beim Absenden einer Email nach Zertifikaten für die Email-Adressen, die im Empfänger-Feld stehen. Dazu macht das Programm eine Anfrage bei den in seinem Adressbuch stehenden LDAP-Servern. Der aufzusetzende LDAP-Server muss also fähig sein, ein Zertifikat zurückzuliefern, wenn man nach einer Email-Adresse sucht.

Der Server sollte aber auch nur dann einen Eintrag zurückliefern, wenn der Anfragende die genaue Email-Adresse kennt: In den Zertifikaten stehen die Email-Adressen ihrer Besitzer und so besteht die Gefahr, dass ein Dritter versuchen könnte, in Besitz der Email-Adressen der Benutzer des Systems zu gelangen, indem er viele Zertifikate sammelt. Hierdurch wird einerseits die Privatsphäre der Benutzer gefährdet, andererseits wird Spam zu den veröffentlichten Email-Adressen provoziert. Auf normalen LDAP-Servern kann man ungenaue Suchen durchführen, zum Beispiel alle Einträge mit Email-Adressen aus einer bestimmten Domäne. Solche ungenauen Suchen werden verhindert, ungenaue Abfragen dürfen also keine Zertifikate und damit Email-Adressen zurückliefern.

Das Server-System muss für den Betrieb der CA und der Web-Anwendungen verschiedene Daten speichern, darunter auch die privaten Schlüssel der Anwender. Die zu speichernden Daten sind also zum Teil sehr sensible Informationen, teilweise passen sie auch von ihrer Struktur her schlecht in einen Verzeichnisdienst. Zur Speicherung dieser Daten eignet sich eine getrennte Datenbank entsprechend besser. Um die Zertifikate nicht an mehreren Orten speichern und verwalten zu müssen, verwenden die Web-Anwendungen und der LDAP-Server die gleiche Datenbank. Eine Abfrage des LDAP-Servers sollte daher umgesetzt werden in eine Datenbankabfrage und ein etwaiges gefundenes Zertifikat wird wieder über LDAP zurückgeliefert.

7.1.1 Auswahl eines Programms

Um die LDAP-Komponente erweitern und den unterschiedlichen Anforderungen genügen zu können, muss der Quelltext für Anpassungen zur Verfügung stehen. Deswegen gibt es die Möglichkeit, die LDAP-Komponente selbst zu schreiben oder auf einen vorhandenen, freien LDAP-Server zurückzugreifen, bei dem der Quelltext ebenfalls erhältlich ist.

Die Wahl fiel schließlich auf stand-alone LDAP daemon (SLAPD), den LDAP-Server des OpenLDAP-Projektes [Opeb]. Dieser Server ist mit Quelltext erhältlich und unterstützt durch sein modulares Konzept ohne Änderung verschiedene Speichermöglichkeiten für die dem Verzeichnis zugrunde liegenden Daten. Hierzu werden verschiedene Backends unterstützt. Backends sind Module, die jeweils den Zugriff auf eine bestimmte Art von Datenbank unterstützen. Eines dieser Backends ist das SQL-Backend, mit dem der LDAP-Server auf eine SQL-Datenbank zugreifen kann. Im vorliegenden Fall sollte das möglichst die SQL-Datenbank sein, die auch von den Web-Anwendungen genutzt wird. SLAPD mit SQL-Backend ermöglicht also eine hohe Anpassbarkeit durch den Quelltext, während trotzdem ein geringerer Aufwand auf die Einrichtung der LDAP-Komponente entfällt, als dies bei einer Neuprogrammierung notwendig wäre. Zum Zeitpunkt der Auswahl der LDAP-Komponente beschränkte sich der geschätzte Aufwand im günstigsten Fall auf reine Konfiguration, wenn nämlich SLAPD mit SQL-Backend schon ohne Änderungen des Quelltextes den gegebenen Anforderungen genügt. Im ungünstigsten Fall müssen nur die Komponenten im Quelltext angepasst werden, die nicht mit den Anforderungen vereinbar sind.

7.1.2 Der OpenLDAP-Server

Die Web-Anwendungen sollten mit Hilfe der Microsoft ASP .NET 2.0-Technologie entwickelt werden. Für diese bot sich eine Windows-Plattform an, etwa das Server-Betriebssystem Microsoft Windows 2003 Server. Um nicht von unterschiedlichen Betriebssystemen abhängig zu sein, sollte der SLAPD auch auf einem Rechner mit dem Betriebssystem Windows 2003 laufen. Für diese Plattform ist OpenLDAP nur als Quelltext erhältlich; da es für Änderungen ohnehin erforderlich ist, den Server aus dem Quelltext neu zu bauen (dieser Vorgang wird aus dem Englischen Build genannt), ist das ein akzeptabler Nachteil. Die Installationsanleitung der OpenLDAP Foundation [Opeb] zum Build des SLAPD bezieht sich nur auf Unix-Plattformen, das heißt, man muss den SLAPD auf einer Unix-Plattform bauen. Die Zielplattform kann aber eine andere sein, das heißt, dass das erstellte Programm zum Beispiel unter Windows läuft. Wenn sich die Zielplattform von der Plattform des Builds unterscheidet, nennt man dies Cross Compile. Die OpenLDAP Foundation empfiehlt in ihren Frequently Asked Questions (FAQ) zur Erstellung des SLAPD für die Zielplattform Windows die Cygnus' Cygwin Umgebung für den Build zu verwenden, aber mittels eines Cross Compiles Minimalist GNU for Windows (MinGW) als Zielplattform zu verwenden. Alternativ wird vorgeschlagen, aber nicht empfohlen, das Minimal System (MSYS) als Umgebung des Builds zu benutzen, und mit MinGW die Binärdateien zu erzeugen. Diese Vorgehensweise wird von Lucas Bergman im mittlerweile eingestellten Projekt *OpenLDAP for Win32* angewendet [BM]. Mit allen Varianten gab es Probleme zu bewältigen, daher wurden verschiedene Möglichkeiten probiert, um einen leicht zu realisierenden Weg zu finden, der trotzdem den Ansprüchen genügt. Die letzte Möglichkeit, also MinGW zu benutzen,

habe ich zurückgestellt, nachdem Probleme den erwarteten Zeitaufwand hoch erscheinen ließen. Später konnte der LDAP-Server mit Ziel- und Quellplattform Cygwin (also ohne Cross Compile) erfolgreich gebaut werden, so dass den weiteren Möglichkeiten nicht weiter nachgegangen wurde.

Wie in der OpenLDAP-FAQ und im OpenLDAP Admin Guide beschrieben, muss man zuerst Cygwin [Cyg] mit allen benötigten Paketen auf einem Windows-System installieren. In dieser Umgebung baut und installiert man GNU Regex [GNU]. Schließlich kann man den Quelltext von OpenLDAP bei [Opeb] herunterladen und das Programm bauen - allerdings werden so noch nicht alle Backends und insbesondere nicht das SQL-Backend unterstützt.

Tatsächlich wird das SQL-Backend bei Windows-Plattformen offiziell gar nicht unterstützt [Opeb] und auch Bergman beschrieb die Nutzung dieses Backends nicht [BM]. Erwartungsgemäß hat man es hier also mit mehr und schwierigeren Problemen zu tun als bei den restlichen Schritten des Installationsprozesses. Eine Beschreibung der Probleme mit OpenLDAP und ihrer Lösung wird in Anhang A dargestellt.

7.1.3 Datenschutz

Wie zu Anfang des Kapitels erwähnt, sollte auf ein Zertifikat nur zugegriffen werden können, wenn man schon dessen exakte Email-Adresse kennt. Damit soll verhindert werden, dass Dritte an eine Liste der Email-Adressen aller Nutzer des Systems gelangen können.

Das Protokoll LDAP sieht eine Beschränkung der maximal zurückgegebenen Einträge vor, die man in der OpenLDAP-Implementation auch auf eins festlegen kann [WHK97]. Somit könnte man aber immer noch das erste Zertifikat eines Suchergebnisses herunterladen und mit geeigneter Wahl von Suchparametern trotzdem an alle Zertifikate gelangen, indem man beispielsweise die Suche auf bestimmte Anfangsbuchstaben in den Email-Adressen einschränkt.

Vielversprechender ist hier schon die Möglichkeit, eine „administrative Schranke“ zu setzen. In OpenLDAP ist dies so implementiert, dass gar kein Ergebnis zurückgeliefert wird, wenn die Anzahl der Suchergebnisse eine festgelegte Schranke überschreitet [Ope07]. Trotzdem kann man aus dem Resultat der Suche erkennen, ob die Suche keinen, einen oder mehrere Treffer ergab. Ein Angreifer könnte wie in einem Suchbaum Suchen mit immer restriktiveren Filtern durchführen, um schließlich zu einem gültigen Zertifikat zu gelangen. Ein möglicher Algorithmus wird als Programmablaufplan in Abbildung 7.1 dargestellt. Wenn man die dargestellte Funktion *durchsucheLDAP* mit einem leeren Text als Parameter aufruft, gibt sie alle Email-Adressen im Verzeichnis aus. In dem Suchbaum stellen die Knoten Textfilter dar, denen Email-Adressen entsprechen, die Blätter sind Textfilter, denen genau eine Email-Adresse entspricht. Eine Suche mit dem Filter eines Blatts gibt also ein Zertifikat zurück. Die Tiefe t des Suchbaums ist maximal so groß wie die Anzahl der Zeichen der längsten Email-Adresse, womit das Finden eines Zertifikats in Zeit $O(\log t)$ gelingt. Folglich können auf diese Weise sogar alle Email-Adressen effizient gefunden werden, nämlich in Zeit $O(\log t \cdot n)$, wenn im Verzeichnis n Zertifikate gespeichert sind.

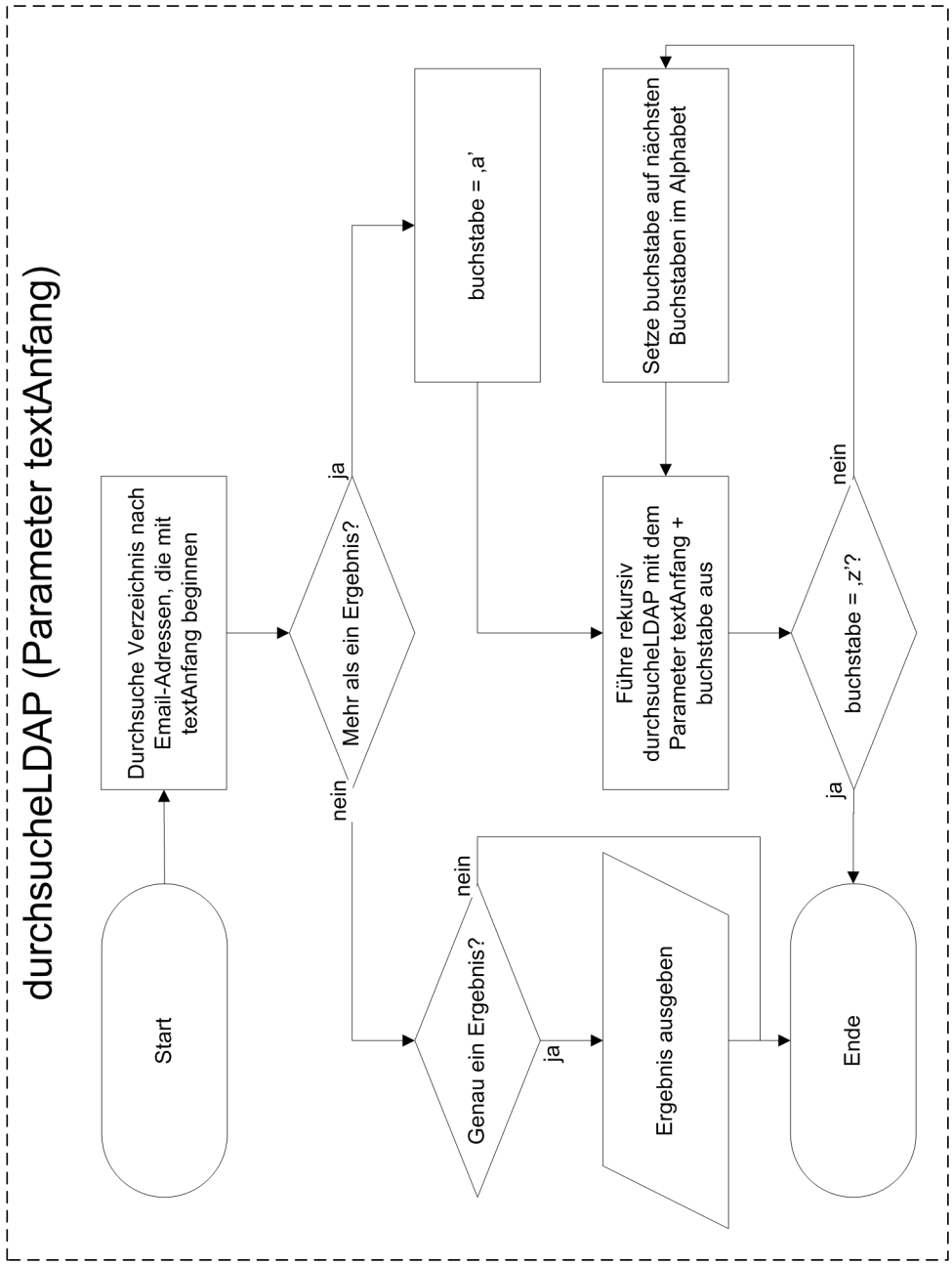

Abbildung 7.1: Algorithmus, um in einem OpenLDAP-Verzeichnis mit administrativer Schranke 1 alle Email-Adressen zu finden

Da ich keine weitere Einstellung gefunden habe, mit der das Ermitteln aller Email-Adressen in OpenLDAP verhindert werden könnte, muss hierfür der Quelltext angepasst werden. Hierzu werden alle Filter auf Teilstrings nicht wie herkömmlich so interpretiert, dass ein Attributwert den Filtertext enthalten muss, sondern der Attributwert muss sogar mit dem Substring identisch sein. Damit sind einerseits noch ungefilterte Suchen möglich, diese geben wegen der administrativen Schranke eine Fehlermeldung zurück, es ist aber kein Rückschluss auf die Verzeichnisstruktur möglich. Andererseits sind noch Suchen auf exakte Attributwerte möglich.

Kompatibilität mit Suchfiltern

OpenLDAP ist so implementiert, dass zuerst das Backend bei einer Suche eine Liste von „Kandidaten“ zurückliefert und die Kernmodule von OpenLDAP dann diese Kandidaten nochmals auf den Filter der aktuellen Suche hin überprüfen. Dabei kann es also sein, dass die Suche des Backends mehr Kandidaten zurückliefert als tatsächlich zurückgegeben werden sollen. Das SQL-Backend nimmt dies für verschiedene Arten von Suchen in Kauf, vermutlich um die Programmierung des SQL-Backends zu vereinfachen. Das ist normalerweise kein Problem, die falschen Treffer werden von den OpenLDAP-Kernmodulen wieder herausgefiltert. Eine etwaige administrative Schranke wird aber nicht auf die tatsächliche Anzahl der zurückgegebenen Werte hin geprüft, sondern auf die Anzahl der vom Backend zurückgelieferten Kandidaten. Bei einer Suchanfrage mit exakter Email-Adresse könnte es deswegen vorkommen, dass mehr als ein Kandidat vom SQL-Backend zurückgeliefert wird und die OpenLDAP-Kernmodule dann nicht das einzige Suchergebnis von den falschen Treffern filtern, sondern wegen der administrativen Schranke einfach eine Fehlermeldung zurückgeben. Die Suchfunktionen des SQL-Backends mussten deswegen so optimiert werden, dass sie strenger filtern, so dass bei einer Suche nach einer exakten Email-Adresse nur genau ein Kandidat vom SQL-Backend zurückgeliefert wird. Das SQL-Backend wurde daher so angepasst, dass zumindest bei den Suchanfragen der in dieser Arbeit behandelten Email-Programme Outlook, Outlook Express und Mozilla Thunderbird keine Fehlermeldung zurückgegeben wird, wenn der Suchfilter exakt auf die zu findende Email-Adresse schließen lässt.

7.2 Datenbank

Das Serversystem speichert fast alle veränderbaren Daten in einer zentralen SQL-Datenbank. Nur solche Einstellungen und Benutzerdaten, deren Speicherung sich nicht auf einfache Weise in einen SQL-Server übertragen lassen, verwenden andere Formate. Dies betrifft beispielsweise die interne Datenbank einer vorgefertigten CA (siehe Kapitel 5) oder einige Grundeinstellungen des LDAP-Servers (Abschnitt 7.1).

7.2.1 Benutzerdaten

Jedem Benutzer des Systems wird ein eindeutiger Global Unique Identifier (GUID) zugewiesen und der GUID wird in der Tabelle *cert_ users* gespeichert (siehe Abbildung 7.2). Auf diese Weise wird ein Benutzer in der Datenbank eindeutig identifiziert. Ein Benutzer kann mehrere Email-Adressen haben, die in der Tabelle *cert_ emails* verwaltet

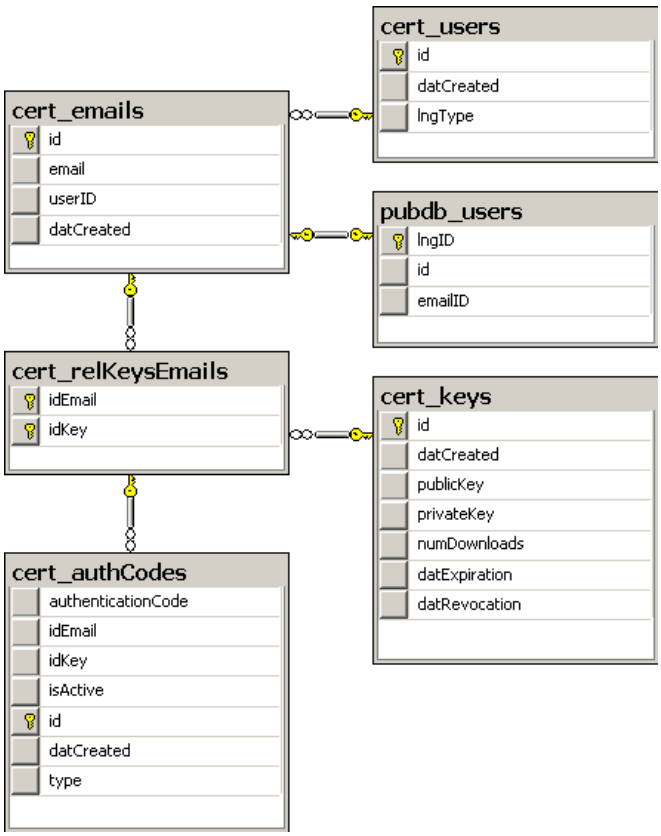

Abbildung 7.2: Datentabellen, die die Benutzerinformationen im SQL-Server speichern

werden. Zwischen Email-Adressen und ausgestellten Zertifikaten besteht eine n-zu-m-Relation. Das heißt, dass ein Zertifikat auch für mehrere Email-Adressen ausgestellt sein kann und es umgekehrt für eine Email-Adresse mehrere Zertifikate geben kann. Letzteres ist der Fall, wenn ein altes Zertifikat widerrufen wurde oder abgelaufen ist und ein neues Zertifikat für die Email-Adresse erstellt werden musste. Diese Relation zwischen den Tabellen *cert_ emails* und *cert_ keys* wird in der Tabelle *cert_ relKeysEmails* verwaltet.

Die im Kapitel 6 beschriebene Weboberfläche lässt zwar noch nicht zu, dass ein Zertifikat für mehrere Email-Adressen ausgestellt wird, da jedoch die Oberfläche leichter zu verändern ist als die Datenbankstruktur, wurde die Datenbank schon für eine solche zukünftige Veränderung ausgelegt.

Die Tabelle *cert_ authCodes* enthält die Authentifizierungscodes, mit denen das Konfigurationsprogramm wie in Abschnitt 9.3 beschrieben markiert wird. Durch die Relation zur Tabelle *cert_ relKeysEmails* wird festgelegt, an welche Email-Adresse der Authentifizierungscode verschickt und für welches Zertifikat er erzeugt wurde. Wenn das Konfigurationsprogramm dem Web Service den Authentifizierungscode schickt, kann der Web Service das Zertifikat und den privaten Schlüssel ermitteln und an das Konfigurationsprogramm weitergeben. Dieser Vorgang wird auch im Abschnitt 6.3 beschrieben.

7.2.2 OpenLDAP-Metadaten

Das SQL-Backend von OpenLDAP benutzt sechs Tabellen mit fest definierten Spalten, um zu ermitteln, wo die tatsächlichen Daten in der Datenbank gefunden werden können [Yod04]. Diese Tabellen können, wie in Anhang A.2.1 beschrieben, mit SQL-Skripten angelegt werden, die dem SQL-Backend von OpenLDAP schon beiliegen.

Relationstabellen zwischen OpenLDAP und Benutzerdaten

Die Tabelle *ldap_ entries* wurde hierbei durch eine so genannte View ersetzt, um den Besonderheiten des Systems gerecht zu werden. Vom OpenLDAP-SQL-Backend ist vorgesehen, dass *ldap_ entries* die *distinguished names* der LDAP-Objekte enthält und so den Typ der LDAP-Objekte und ihre Position in der LDAP-Hierarchie bestimmt. Durch die Verwendung einer View statt einer festen Tabelle können die Tabelleninhalte dynamisch aus anderen Tabellen zusammengestellt werden. Die View kann man so aufrufen wie eine Tabelle, im SQL-Server ist sie allerdings nicht durch feste Datensätze, sondern durch eine SQL-Abfrage festgelegt. Bei jedem Aufruf der View wird das Ergebnis der SQL-Abfrage zurückgeliefert und die zurückgegebenen Daten werden so behandelt, als wären sie in einer festen Tabelle mit dem Namen der View enthalten. Auf diese Weise müssen die *distinguished names* und die Anzahl der Benutzer nicht festgelegt werden, sondern bestimmen sich aus den in den anderen Tabellen enthaltenen Benutzerdaten.

Die erwähnte View *ldap_ entries* greift auf die Tabelle *pubdb_ ou* zu, die Verweise auf die zu publizierenden Zertifikate in den Tabellen mit Benutzerdaten enthält. Damit muss nur ein Verweis auf ein Zertifikat in die Tabelle *pubdb_ ou* eingetragen werden, damit das Zertifikat im LDAP-Verzeichnis publiziert wird.

7.2.3 Einstellungen

Die im Kapitel 6 beschriebenen Web-Applikationen werten die in der Datenbank hinterlegten Einstellungen aus. Hier stehen beispielsweise die IP-Adressen oder URIs der beteiligten Server. Wenn ein Link in einer der Web-Applikationen erzeugt werden soll, etwa um ihn in einer Email zu verschicken, werden diese Rechnernamen ausgewertet, um das Ziel des Links zu bilden. Die Email wird dann an den konfigurierten Simple Mail Transfer Protocol (SMTP)-Server verschickt.

Teil III

Entwicklung der Client-Komponenten

8 Problemanalyse

In diesem Teil der Arbeit wird die Entwicklung der Client-Komponenten des Systems beschrieben. Thema der nächsten beiden Kapitel ist also das Konfigurationsprogramm, das auf den Rechnern der Benutzer des Systems ausgeführt wird, um die kryptographischen Funktionen vorhandener Email-Programme einzurichten.

8.1 Motivation

Damit ein Email-Nutzer an einer PKI teilnehmen und kryptographisch gesicherte Emails austauschen kann, müssen verschiedene Bedingungen erfüllt sein. Nun soll untersucht werden, wie diese Bedingungen in einer für den Benutzer möglichst einfachen Weise verwirklicht werden können.

Als Ziel soll dem Benutzer ein System zur Verfügung stehen, das verschlüsselte und signierte Emails versenden und empfangen kann. Ein Ansatz wäre es, Kryptographie-Plugins für die verschiedenen Email-Programme an die Benutzer weiterzugeben. Die Email-Programme Microsoft Outlook, Microsoft Outlook Express und Mozilla Thunderbird bieten diese Funktionen allerdings auch selbst schon an, wenn sie entsprechend konfiguriert sind. Da der Benutzer einerseits mit den Schnittstellen seines gewohnten Email-Programms besser zurecht kommt als mit einem völlig neuen Plugin, und um andererseits den Programmier- und Wartungsaufwand zu senken, werden die vorhandenen Funktionen der Email-Programme ausgenutzt. Das Konfigurationsprogramm muss also nur das Email-Programm des Benutzers korrekt einrichten und wird dann nicht mehr ausgeführt, insbesondere für die spätere Ver- und Entschlüsselung sowie die Signaturen ist es nicht mehr nötig.

8.2 Anforderungen an ein Email-Programm mit Kryptographie

Um die beschriebenen Funktionen des Email-Programms nutzen zu können, sind eine Reihe von Operationen nötig, etwa die privaten und öffentlichen Schlüssel zu importieren und passende Ablaufparameter des Email-Programms zu setzen. Damit der Benutzer diese Schritte nicht von Hand durchführen muss, nimmt ihm ein im Rahmen dieser Arbeit entwickeltes Konfigurationsprogramm alle Vorgänge ab, die ohne sein Eingreifen auskommen. Danach soll das System auch ohne das Konfigurationsprogramm lauffähig sein, das heißt, dass alle Operationen dann direkt vom Email-Programm durchgeführt werden.

Um die Akzeptanz der kryptographischen Funktionen zu erhöhen, sollte der Benutzer in seiner täglichen Arbeit so wenig wie möglich beeinträchtigt werden. Das heißt, dass er wie gewohnt Emails schreiben können sollte, diese dann aber ohne weiteren Eingriff

verschlüsselt und signiert werden sollen. Dazu müssen in den Email-Programmen bestimmte Einstellungen gesetzt werden, die Verschlüsselung und Signaturen „anschalten“. Andererseits müssen auch die Voraussetzungen für Verschlüsselung und Signaturen gegeben sein, das heißt die öffentlichen Schlüssel der Nachrichtenempfänger müssen verfügbar und vertrauenswürdig sein und ein eigener geheimer Schlüssel muss benutzt werden können.

Die Veränderungen am Email-Programm bestehen daher im Wesentlichen aus vier Schritten,

- der in Abschnitt 7.1 beschriebene LDAP-Server muss als „Adressbuch“ eingerichtet werden, damit das Email-Programm dort nach öffentlichen Schlüsseln für die Empfänger seiner Emails suchen kann.
- Der Vertrauensanker (die Root CA) des Systems muss in die Schlüsseldatenbank des Email-Programms eingespielt werden, damit den öffentlichen Schlüsseln der anderen Benutzer vertraut wird.
- Ein geheimer Schlüssel muss dem Programm zur Verfügung gestellt werden. Mit diesem können ausgehende Emails signiert und eingehende Emails entschlüsselt werden.
- Verschlüsselung und Signaturen müssen aktiviert und gegebenenfalls programmspezifische Einstellungen vorgenommen werden.

Je nach Email-Programm müssen verschiedene Besonderheiten beachtet werden. Auf den Aufbau der drei betrachteten Email-Programme Outlook, Outlook Express und Thunderbird wird deshalb im Folgenden eingegangen.

8.3 Microsoft Outlook

Microsoft Office Outlook ist eine von der Microsoft Corporation entwickelte Groupware, die vor allem auch als Email-Programm verwendet werden kann [Mic07]. Es wird seit Microsoft Office 97 mit den Office-Paketen ausgeliefert [HY97]. Das Programm zielt auf den professionellen Einsatz in Firmen ab. Für das Betriebssystem Mac OS wird Outlook unter dem Namen Entourage angeboten. Die aktuelle Version ist Outlook 2007 beziehungsweise Outlook 12.0.

8.3.1 Aufbau

Die Windows-Versionen von Outlook speichern die meisten ihrer Einstellungen in der Registry, einer von allen Anwendungen erreichbaren Datenbank mit Baumstruktur. Einzelne Einträge in der Registry heißen Schlüssel. Mit dem Programm RegMon der Windows SysInternals [RC07] sind bei der Verwendung von Outlook Zugriffe auf den Registry-Schlüssel

HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles

zu sehen.

8.3.2 Profile

Man kann in Outlook unterschiedliche Profile anlegen, zum Beispiel für mehrere Benutzer. Jedem dieser Profile entspricht ein Profilschlüssel unterhalb des oben genannten Registry-Schlüssels. Welches Profil vorausgewählt ist, wird über den Wert *DefaultProfile* bestimmt (siehe Abbildung 8.1). Der Profilschlüssel eines neuen Profils enthält zehn Schlüssel, die die Konfiguration von Outlook bestimmen. Konfigurierte Profile können weniger oder mehr Schlüssel enthalten. Jeder dieser Schlüssel ist mit einer 32-stelligen Sedezimalzahl benannt; manche Zahlen sind bei jeder Installation gleich, andere variieren. Eine Dokumentation der Bedeutung einiger Schlüssel hat James McWhinney in Form eines Visual Basic-Scripts im Code Project veröffentlicht [McW06]. Zu anderen Dokumentationen schreibt er im Abschnitt „Points of Interest“

> As far as I can tell, this section of the Outlook registry was undocumented until now.

Da ich ebenfalls keine weiteren Dokumentationen, speziell auch zu den Sicherheitseinstellungen, gefunden habe, musste ich mit RegMon überwachen, in welche Registry-Einträge geschrieben wird, wenn man in Outlook eine Einstellung ändert. Die Ergebnisse sind im Anhang B zu finden.

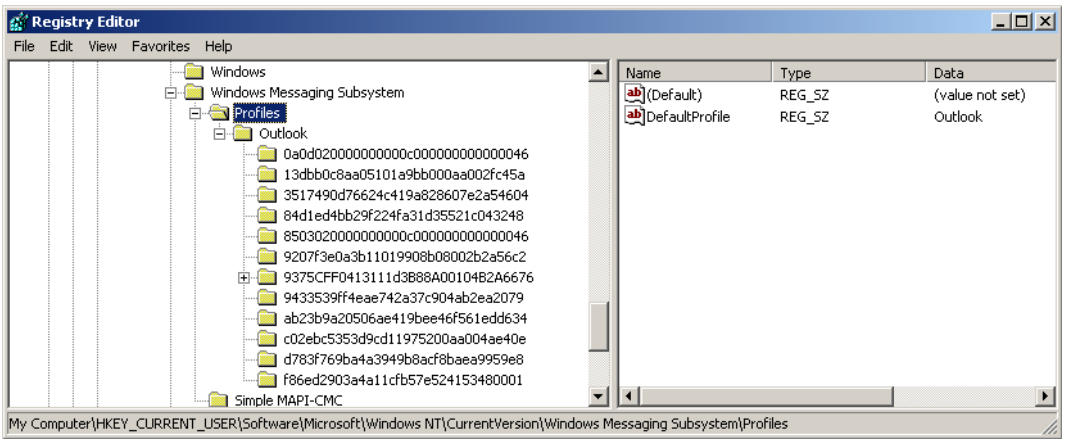

Abbildung 8.1: Screenshot auf die Outlook-Konfiguration in der Registry. Hier ist „Outlook“ das einzige Profil und damit auch das DefaultProfile

8.3.3 Einstellungen und Adressbücher

Die kryptographisch relevanten Einstellungen von Outlook werden im Konfigurationsschlüssel *C02EBC5353D9CD11975200AA004AE40E* unterhalb des Profilschlüssels gespeichert (eine genauere Beschreibung der möglichen Einträge findet sich in der Tabelle B.1). Dieser Schlüssel existiert allerdings nur, wenn die Standardeinstellungen verändert wurden.

Jedes LDAP-Adressbuch hat zwei Schlüssel unterhalb des Profilschlüssels, welche die Einstellungen des Adressbuches bestimmen. Die Sedezimalnummern dieser Schlüssel können beim Anlegen beliebig gewählt werden (siehe Tabellen B.4 und B.2). Die beiden Schlüssel haben unter anderem auch Einträge über die Nummer des anderen Schlüssels, so dass es reicht, einen der beiden Schlüssel zu kennen, um an die Informationen

über das Adressbuch zu gelangen (siehe Abbildungen 8.2 und 8.3). Die Liste aller Konten (das sind alle Adressbücher und Email-Postfächer in Outlook) besteht aus Unterschlüsseln des Konfigurationsschlüssels *9375CFF0413111d3B88A00104B2A6676* (Beschreibung dieses Konfigurationsschlüssels in Tabelle B.5), danach muss der Eintrag wie weiter unten beschrieben als aktives Adressbuch gekennzeichnet werden. Die Unterschlüssel von *9375CFF0413111d3B88A00104B2A6676* sind mit achtstelligen Sedezimalzahlen benannt, die bei *00000001* beginnend fortlaufend nummeriert werden und die Konten eindeutig kennzeichnen (die Unterschlüssel für Adressbücher sind in B.6 beschrieben). Der Schlüssel für ein Adressbuch enthält einen Anzeigenamen und einen Verweis auf einen der beiden Konfigurationsschlüssel des besagten Adressbuchs (siehe Abbildung 8.4). Die beiden Konfigurationsschlüssel (in den Abbildungen 8.2 und 8.3 zu sehen) enthalten in ihren Einträgen Einstellungen, wie den Servernamen und Protokollparameter. Da einige Registry-Einträge der Outlook-Konfiguration den Typ *REG_BINARY* haben, werden Textinformationen allerdings im Registrierungseditor nicht richtig angezeigt, sondern nur deren Byte Codes.

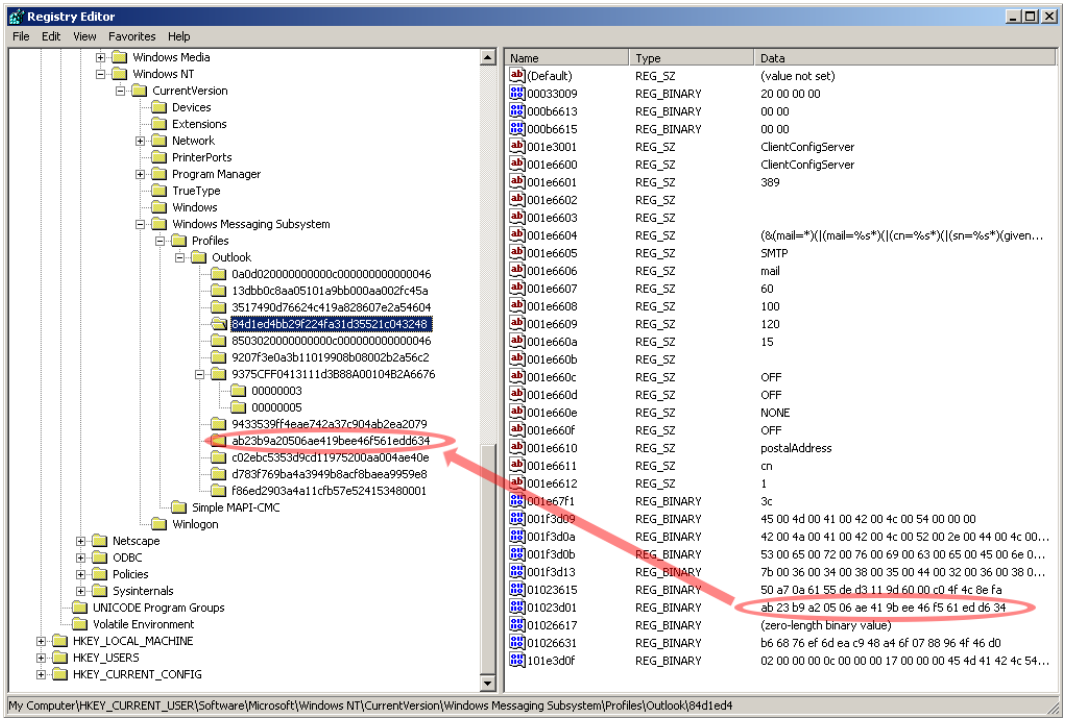

Abbildung 8.2: Der erste der beiden Adressbuch-Konfigurationsschlüssel. Er enthält einen Verweis zum zweiten Konfigurationsschlüssel (rot markiert)

Im Schlüssel *9375CFF0413111d3B88A00104B2A6676* muss der Eintrag *NextAccountID* größer als die laufende Nummer des Adressbucheintrags sein, damit ein Adressbuch als aktiv gilt. Außerdem muss die Nummer des Adressbuchs im Eintrag *{ED475419-B0D6-11D2-8C3B-00104B2A6676}* aufgelistet sein (Siehe Abbildung 8.5). Hier werden für jedes aktive Adressbuch die vier Byte zur Darstellung der achtstelligen, sedezimalen laufenden Nummern des Adressbuchs hintereinander gespeichert. Da die Speicherung in der Little Endian-Reihenfolge erfolgt, erscheint in der Darstellung der niederwertigste Teil der Nummer auf der linken Seite. Für das Adressbuch mit der Nummer 5 lautet der Eintrag daher nicht `00 00 00 05`, sondern `05 00 00 00`, wie in Abbildung 8.5 zu sehen ist.

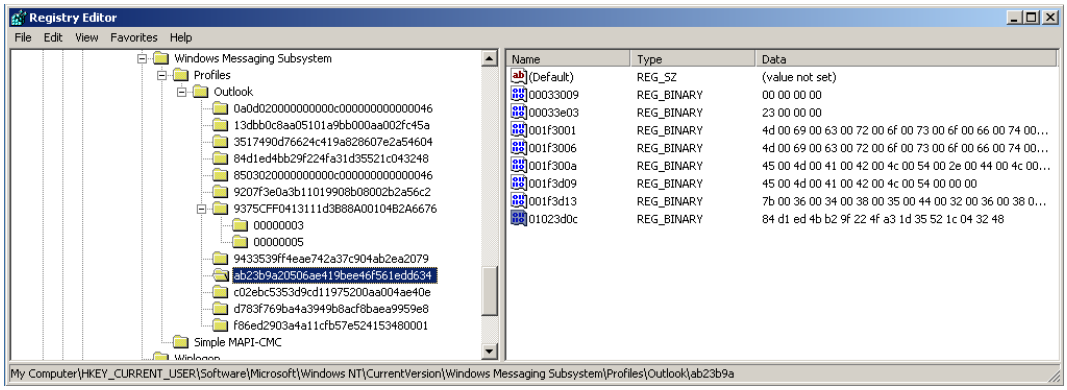

Abbildung 8.3: Der zweite Konfigurationsschlüssel. Er enthält einen Verweis auf den ersten Konfigurationsschlüssel, dieser ist im Bild aber nicht markiert

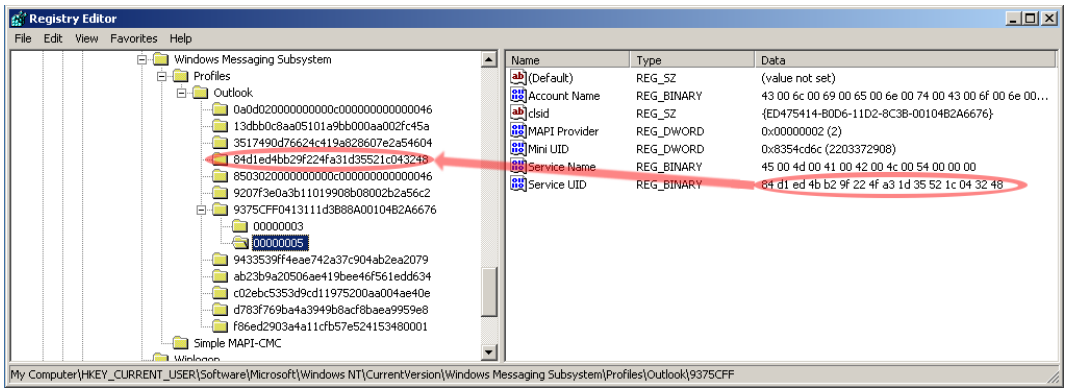

Abbildung 8.4: Der Eintrag eines Adressbuchs ist ausgewählt. Der Verweis auf den Konfigurationsschlüssel ist rot markiert.

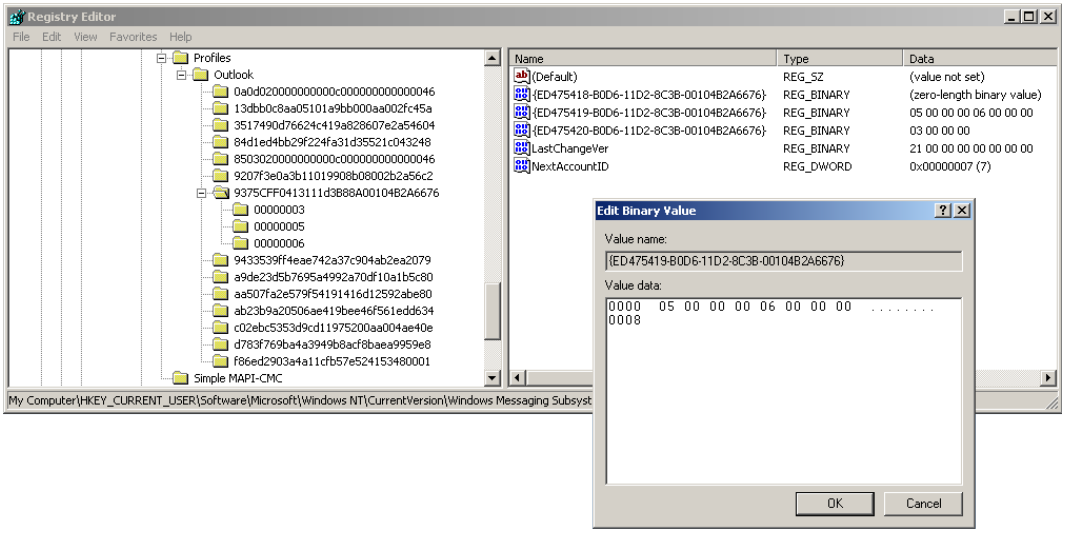

Abbildung 8.5: Liste der aktiven Adressbücher, Adressbuch 5 und 6 sind ausgewählt. In dieser Abbildung wurde ein zusätzliches Adressbuch gegenüber den vorigen Abbildungen hinzugefügt

Schließlich sollte das Adressbuch noch in Outlooks Backup-Schlüssel (Tabelle B.3) gespeichert werden, sonst wird das Adressbuch beim nächsten Outlook-Start womöglich wieder gelöscht.

8.3.4 Schlüsseldatenbank

Jede Kopie des Microsoft Windows Betriebssystems führt eine zentrale Datenbank mit X.509-Zertifikaten und dazu passenden geheimen Schlüsseln. Programme wie Outlook, Outlook Express und der Internet Explorer benutzen diese gemeinsame Datenbank, anstatt eine eigene zu pflegen. Die Datenbank kann ihre Zertifikate an verschiedenen Orten speichern. Die Standardeinstellung ist die Registry, aber falls eine Smartcard vorhanden ist, ist diese auch ein gültiger Speicherungsort.

Obwohl man auch direkt auf die Registry zugreifen kann, um Schlüssel hinzuzufügen, zu löschen oder zu verändern, bietet Microsoft diese Funktion auch in einer Bibliothek namens CryptoAPI an. Auf dem System, auf dem das CryptoAPI benutzt werden soll, muss die DLL *Crypt32.dll* vorhanden sein. Diese DLL gehört zu den mit Windows installierten Dateien, so dass sie nur in Ausnahmefällen nicht vorhanden ist. Die CryptoAPI bietet Funktionen zum Zugriff auf Microsofts Schlüsseldatenbank, es lassen sich Informationen aus X.509 Zertifikaten auslesen und kryptographische Funktionen ausführen (Verschlüsselung, Entschlüsselung, Signaturen). Die CryptoAPI ist in der Microsoft Developer Network (MSDN) Library dokumentiert [Mic05].

8.4 Microsoft Outlook Express

Outlook Express ist ein Email-Programm und Newsgroup-Reader von Microsoft. Es unterscheidet sich technisch stark von Outlook, was sich schon in der Zielgruppe der Privatanwender ausdrückt. Outlook Express wird mit dem Internet Explorer 4.x und seinen Nachfolgern ausgeliefert, durch die Integration des Internet Explorer in Windows ist es daher ab Windows 98 bei allen Desktop-Windows-Versionen vorinstalliert [Mic07]. In Windows Vista wurde Outlook Express durch Windows Mail abgelöst [Mica].

8.4.1 Aufbau

Outlook Express hält seine Betriebsparameter an verschiedenen Stellen der Windows-Registry. Während die Adressbücher global im Schlüssel

HKEY_CURRENT_USER\Software\Microsoft\Internet Account Manager\Accounts

gespeichert werden, legt Outlook Express die sicherheitsrelevanten Einstellungen pro Profil (die in Outlook Express Identitäten heißen) ab [Ins].

Outlook Express benutzt die gleiche Windows-Schlüsseldatenbank, die auch vom Internet Explorer und von Outlook benutzt werden. Dies ist in Abschnitt 8.3.4 näher beschrieben.

8.4.2 Profile und Einstellungen

Für jedes Profil gibt es einen mit einem GUID benannten Schlüssel unterhalb von *HKEY_CURRENT_USER\Identities*. Dort ist auch der GUID des Standard-Profils gespeichert. Im Unterschlüssel

Software\Microsoft\Outlook Express\5.0\Mail

befinden sich unter anderem die sicherheitsrelevanten Einstellungen. Der Teil „5.0“ des Schlüssels ist übrigens auch für Outlook Express 5.5 und 6.0 richtig.

Im Gegensatz zu den Outlook-Einstellungen sind bei Outlook Express die Namen der Einträge Hinweise auf ihre Bedeutung. Die Einträge *Digitally Sign Messages* und *Encrypt Messages* bestimmen daher erwartungsgemäß, ob ausgehende Nachrichten signiert oder verschlüsselt werden sollen.

8.4.3 Adressbücher

Für jedes Adressbuch gibt es einen Unterschlüssel des Schlüssels

HKEY_CURRENT_USER\Software\Microsoft\Internet Account Manager\Accounts

Die Unterschlüssel können beliebige Namen tragen, die drei voreingestellten Adressbücher Bigfoot, VeriSign und WhoWhere verwenden eine Kurzform des Anzeigenamens. Jeder Unterschlüssel enthält Einträge mit sprechenden Namen zur Konfiguration des Adressbuchs (siehe Abbildung 8.6).

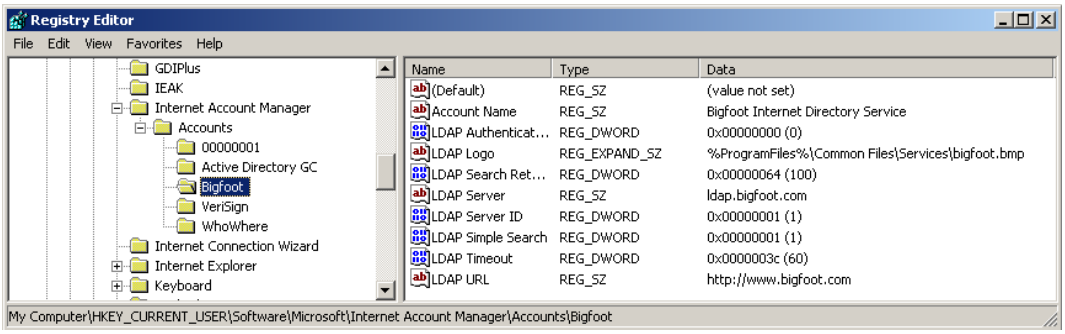

Abbildung 8.6: Registry-Einstellungen zum Adressbuch Bigfoot

8.5 Thunderbird

Mozilla Thunderbird ist ein kostenloses und quelloffenes Email-Programm, das von der Mozilla Foundation [Mozb] aus dem Quelltext von Netscape entwickelt wurde [Mozc]. Thunderbird ist auf verschiedenen Plattformen wie Windows und Linux erhältlich. Es unterstützt in der Standardinstallation Email-Verschlüsselung mit S/MIME. Durch ein Pluginsystem kann Thunderbird auf weitere Verschlüsselungsstandards erweitert werden. Bis zum 2007-11-30 wurde es knapp 58 Millionen Mal heruntergeladen [Moza]. Die Beschreibung in diesem Abschnitt bezieht sich auf die Versionen 1.5 und 2.0.

8.5.1 Aufbau

Auf Grund seiner Plattformunabhängigkeit kann Thunderbird seine lokalen Daten wie Einstellungen, Emails und X.509-Zertifikate nicht in der nur in Windows vorhandenen Registry speichern, sondern benutzt lokale Dateien. Dazu wird bei der Installation im Verzeichnis *%appdata%* ein Unterverzeichnis *Thunderbird* angelegt, in dem sich die weiteren Daten befinden.

8.5.2 Profile

Thunderbird unterstützt mehrere Benutzer, jedem wird ein eigenes so genanntes Profil zugeordnet. Nach der Installation bemerkt man davon allerdings nichts, in der Voreinstellung wird ohne Nachfrage ein Standardprofil geladen. Erst wenn man Thunderbird mit dem Kommandozeilenparameter *-profilemanager* aufruft, startet sich ein Profilmanager, mit dem man die verschiedenen Profile der Installation verwalten und starten kann [moz07a]. Im Startmenü gibt es einen Eintrag, mit dem man den Profilmanager direkt (das heißt ohne Kommandozeile) starten kann. Die Profileinstellungen werden vom Konfigurationsprogramm in die Datei *profiles.ini* im *Thunderbird*-Konfigurationsverzeichnis geschrieben und dort beim Starten von Thunderbird ausgelesen. In dieser *.ini*-Datei ist auch festgehalten, wo die Daten der einzelnen Thunderbird-Profile lokalisiert sind. Jedes Profil kann in einem beliebigen Verzeichnis auf der Festplatte gespeichert werden, in der Voreinstellung wird jedoch ein Unterverzeichnis des Ordners *Profiles* im Konfigurationsverzeichnis *Thunderbird* verwendet.

8.5.3 Einstellungen und Adressbücher

Im Profil-Konfigurationsverzeichnis befindet sich eine JavaScript-Datei *prefs.js* und eventuell eine Datei *user.js*. Diese beiden Dateien werden bei jedem Start von Thunderbird ausgeführt. Dabei wird die Funktion *user_pref* mehrmals aufgerufen, wobei jedes Mal eine Einstellung verändert wird. Zu den Einstellungen gehört die Wahl des für Signaturen benutzten Schlüssels, aber auch die Details der verwendeten Adressbücher. Da die Datei *user.js* zuletzt aufgerufen wird, werden bei eventuellen Konflikten die Einstellungen der *user.js* verwendet. Beim Beenden von Thunderbird werden die aktuellen (und eventuell im Programm veränderten) Einstellungen in die Datei *prefs.js* zurückgeschrieben. Konfigurationsänderungen in der Datei *prefs.js* sind zur Laufzeit damit nicht möglich, denn in der Datei veränderte Werte werden von Thunderbird beim Beenden wieder mit den alten Werten überschrieben. Änderungen in der Datei *user.js* sind dagegen zur Laufzeit möglich, andererseits lassen sich Einstellungen in *user.js* über das GUI überhaupt nicht mehr ändern, denn es wird immer nur die *prefs.js* neu geschrieben, Einstellungen in der *user.js* haben aber höhere Priorität.

Fast alle Konfigurationen, die man über das GUI vornehmen kann, haben ihre Entsprechung in *prefs.js*. Auch die Einstellungen zu den Adressbüchern werden in dieser Datei festgelegt. Kryptographische Daten wie private und öffentliche Schlüssel werden jedoch nicht per JavaScript festgelegt und können damit nicht in der *prefs.js* verändert werden.

8.5.4 Schlüsseldatenbank

Thunderbird verwendet für seine kryptographischen Operationen die NSS, eine von der Mozilla Foundation zu Sicherheitszwecken entwickelte Bibliothek [Moz07b]. In Thunderbirds Schlüsselverwaltung spiegelt sich die NSS-Datenbankstruktur wider: In der obersten Abstraktionsebene werden verschiedene PKCS #11 Module (siehe Abschnitt 2.1.3) angesprochen, die jeweils mehrere so genannte Slots enthalten können. Die Slots enthalten Schlüsselobjekte. Ein Modul ist in diesem Kontext das Modell für einen bestimmten Typ von Zertifikatsspeicher, zum Beispiel Smartcards eines bestimmten Herstellers. Ein Slot entspricht dann einer konkreten Instanz dieses Zertifikatsspeichers, also zum Beispiel einer konkreten SmartCard. Ein Schlüsselobjekt besteht meist aus einem X.509-Zertifikat (wird in Abschnitt 2.1 beschrieben) und gegebenenfalls einem dazugehörigen privaten Schlüssel.

Thunderbird legt bei der Installation zwei Module an, das nicht beschreibbare „Builtin Roots Module“ mit den von Mozilla als vertrauenswürdig eingeschätzten Root CA Zertifikaten und das lese- und schreibfähige „NSS Internal PKCS #11 Module“ mit den eigenen privaten Schlüsseln und dazugehörigen Zertifikaten. Die Art der vorhandenen Module und die Inhalte des „NSS Internal PKCS #11 Module“ werden in einem proprietären Format in den Dateien *secmod.db*, *cert8.db* und *key3.db* gespeichert.

8.5.5 Architektur

Die Mozilla Foundation strukturiert ihre Entwicklung in funktionale Module (das hat nichts mit den oben beschriebenen Modulen der Schlüsseldatenbank zu tun). Zum Zeitpunkt der Erstellung dieses Dokuments gab es 79 verschiedene Module [Mozb]. NSS gehört zusammen mit den Java Security Services zum security-Modul, Thunderbird ist ein eigenes Modul. Zwischen den einzelnen Modulen gibt es zahlreiche Abhängigkeiten. Es soll hier nur der programmtechnische Aufbau des security-Moduls erläutert werden und hiervon nur die NSS.

Da die Mozilla-Projekte den Anspruch haben, plattformübergreifend zu funktionieren, gibt es ein Modul Netscape Portable Runtime (NSPR), das alle Zugriffe auf Betriebssystemressourcen (wie Dateien, aber auch schon Arbeitsspeicherallokation) kapselt. Dementsprechend greift auch NSS bei seinen betriebssystemabhängigen Operationen auf das NSPR zurück. NSS arbeitet auf einer niedrigen Abstraktionsebene, weswegen es noch Komponenten auf höherer Ebene gibt, die wiederum auf NSS-Funktionen zugreifen. Diese Hierarchie ist in Abbildung 8.7 dargestellt, die Komponenten SSL und S/MIME sind Beispiele für von NSS abhängige Komponenten [Moz07b].

Unter Windows gibt es für jede Komponente eine DLL, die vom Thunderbird-Hauptprogramm geladen wird. Die Namen der DLLs setzen sich zusammen aus einem Kürzel für die Komponente und der Hauptversionsnummer. Dementsprechend sind die Funktionen der S/MIME-Komponente in der DLL *smime3.dll*, die SSL-Funktionen in der DLL *ssl3.dll*, die NSS-Kernfunktionen in *nss3.dll* und die NSPR-Funktionen in *nspr4.dll*. Diese DLLs befinden sich im Thunderbird-Installationsverzeichnis.

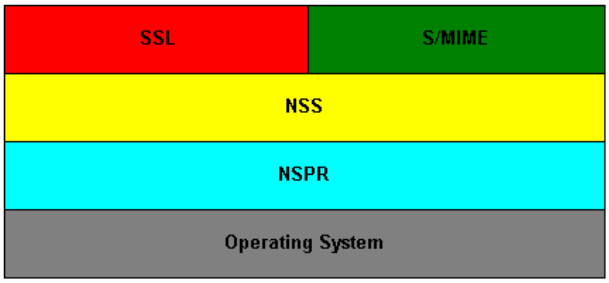

Abbildung 8.7: Schnittstellen zwischen NSS und NSPR (Quelle: Mozilla Foundation, http://www.mozilla.org/projects/security/pki/nss/intro.html)

9 Programmentwicklung

In diesem Kapitel wird der Aufbau und die Entwicklung der Client-Komponenten, also des Konfigurationsprogramms, beschrieben. Hierbei werden einige aufgetretene Probleme hervorgehoben.

9.1 Programmiersprache

Um einen hohen Anteil der Anwender zu erreichen, soll das Konfigurationsprogramm möglichst auf beliebig konfigurierten Windows-Betriebssystemen funktionieren. Da das Microsoft .NET Framework nur dann verfügbar ist, wenn es nachträglich installiert wurde, sollte der Client auch ohne dieses auskommen. Somit scheiden alle reinen .NET-Programmiersprachen wie C#, Visual Basic .NET und J# aus. Die Sprache C++ kommt nur in Frage, wenn der Quelltext keine von .NET abhängigen Teile enthält. Es darf also kein Managed C++ verwendet werden. Die Sprache Java hat den Nachteil, dass eine Java Virtual Machine auf der Zielmaschine installiert sein muss. Unter den Programmiersprachen, die von vorinstallierten Frameworks unabhängigen Code erzeugen, beispielsweise Delphi, habe ich mit C++ die meiste Erfahrung. Das Windows-API in der MSDN Library und das Mozilla-API auf der Mozilla-Webseite sind in C++-Syntax beschrieben. Diese Vorteile führten zu der Entscheidung, die Client-Komponente in C++ zu schreiben.

9.2 Grafische Oberfläche

Man kann das Programm zur Ausführung in verschiedene Umgebungen einbetten. Da der Benutzer das Programm ohnehin von einer Webseite bezieht, sollte man in Betracht ziehen, den Code im Browser des Benutzers laufen zu lassen, zum Beispiel mit Hilfe eines ActiveX Controls. Alternativ kann man eine Windowsanwendung als Oberfläche wählen. Wegen der höheren Benutzerfreundlichkeit bietet sich hier ein „Wizard“ oder Deutsch „Assistent“ an. Vorteil des ActiveX Controls ist, dass der Benutzer nichts bewusst herunterzuladen braucht. Von Nachteil sind die Berechtigungsprobleme: Mit den Voreinstellungen lässt sich das ActiveX Control nicht ausführen. Außerdem hängt die Ausführbarkeit vom verwendeten Browser ab. Aus diesen Gründen wurde das Programm als Windowsanwendung entwickelt, die in Form eines Wizards abläuft.

9.2.1 Bibliothek für die grafische Oberfläche

Für C++ gibt es die Windows Template Library (WTL) zur Entwicklung von Windowsanwendungen und speziell ihrer grafischen Oberfläche. Diese Bibliothek wurde von Microsoft entwickelt und der Quelltext schließlich freigegeben. WTL ist eine Erweiterung

der Active Template Library (ATL), die ebenfalls von Microsoft entwickelt wurde. Ein anderes, weit verbreitetes Framework zur Entwicklung von Benutzeroberflächen unter Windows sind die Microsoft Foundation Classes (MFC). Als weitere Alternative kann man Benutzeroberflächen auch direkt und ohne Zwischenschicht mit dem Win32-API entwickeln. Die ersten beiden Möglichkeiten - WTL und MFC - sind komfortabler, weil die Funktionen auf einer höheren Ebene ansetzen. Die MFC machen die ausführbaren Dateien aber deutlich größer, da der MFC-Binärcode in jede Datei mit hinein kopiert wird. WTL basiert auf Templates und Makros und erzeugt so Binärcode, der direkt das Win32-API anspricht. WTL und das Win32-API erzeugen demnach sehr kleine ausführbare Dateien. Leider hat die WTL keine offizielle Dokumentation und der Programmierer ist in manchen Situationen auf sich allein gestellt. Nach Abwägung der Argumente habe ich mich bei diesem Projekt für die WTL entschieden.

Die WTL bietet schon ohne zusätzliche Module Unterstützung für Wizards, Michael Dunn gibt im achten Teil seiner Artikelreihe „WTL for MFC Programmers“ sogar Beispiele mit Quelltext [Dun06]. Die WTL sieht vor, dass für den Wizard eine Klasse existiert, die von *CPropertySheet* erbt. In diesem Projekt heißt diese Klasse *CMainDlg*. Jeder Schritt im Wizard wird durch ein Objekt vom Typ *CPropertyPage* repräsentiert. Außer bei sehr einfachen Schritten ist es hierbei sinnvoll, eine eigene von *CPropertyPage* erbende Klasse zu schreiben und zur Laufzeit je ein Objekt jeder Klasse zu erzeugen. In den einzelnen Schritten des Wizards

1. stellt die Klasse *CIntroDlg* eine Seite mit einleitendem Text dar,
2. kann der Benutzer durch *CInstTypeDlg* zwischen einer automatischen Installation und einer Installation mit vielen Konfigurationsmöglichkeiten wählen,
3. können die zu konfigurierenden Email-Anwendungen mit der Klasse *CTargetChooseDlg* ausgewählt werden. Bei einer automatischen Installation wird dieser Schritt übersprungen und es werden automatisch alle auf dem System installierten Anwendungen ausgewählt.
4. In *CExecDlg* wird die eigentliche Installation durchgeführt und der Installationsverlauf in Form eines Fortschrittbalkens dargestellt und
5. abschließend wird durch die Klasse *CFinalDlg* eine Seite mit Erfolgs- oder Fehlermeldung ausgegeben.

9.2.2 Objektmodell

Aus der Perspektive der grafischen Oberfläche wird das Zusammenspiel der Komponenten durch das Design Pattern Model/View/Controller (MVC) bestimmt [FF04]. Die View setzt sich aus den verschiedenen mit WTL interagierenden Klassen zusammen, das Model besteht aus den von *IAppConfigurator* erbenden Klassen und der Controller ist durch die Klasse *CEmailClientManager* repräsentiert. Die beiden letztgenannten Klassen werden nun beschrieben.

Das Model *IAppConfigurator*

IAppConfigurator ist ein Interface, eine abstrakte Klasse, die generisch ein konfigurierbares Email-Programm darstellt. Sie enthält nur die Prototypen von Methoden zur

Konfiguration eines Email-Programms, aber keinen konkreten Code, um die Konfigurationen durchzuführen.

Für jedes konkrete Email-Programm hat *IAppConfigurator* eine Tochterklasse, die den Code enthält, um dieses bestimmte Email-Programm zu konfigurieren. Die Klasse *IAppConfigurator* und ihre Tochterklassen sind in Abschnitt 9.4 im Detail beschrieben.

Der Controller *CEmailClientManager*

CEmailClientManager ist ein Singleton [GHJV95], das eine Liste von *IAppConfigurator*-Objekten führt. Die Objekte werden im Constructor von *CEmailClientManager* erzeugt, so dass der Constructor als einzige Methode abhängig ist von den konkreten *IAppConfigurator*-Realisierungen.

CEmailClientManager merkt sich, welche *IAppConfigurator* ausgewählt wurden, also welche Email-Programme konfiguriert werden sollen. Die Auswahl kann über die Methoden *selectClient*, *removeClient* und *isClientSelected* verwaltet werden.

Bei Aufruf der Methode *processClients* werden die ausgewählten *IAppConfigurator* angewiesen, ihre Email-Programme zu konfigurieren. Dazu werden jeweils ihre Konfigurationsmethoden (*setLDAPServer*, *enableCryptoSettings* und so weiter) aufgerufen.

Gemäß MVC ist *CEmailClientManager* aber auch das Subject in einem Observer Pattern. Die Observer können auf diese Weise den Konfigurationsprozess überwachen. In der View ist die Klasse *ExecDlg* der einzige Observer, sie zeigt in einem Fortschrittsbalken den Verlauf des Installationsprozesses an.

9.3 Eindeutige Markierung der Konfigurationsprogramme

Bis jetzt wurde nur allgemein von einer Markierung des Konfigurationsprogramms geschrieben, nun soll detaillierter dargestellt werden, wie diese Markierung funktioniert.

Das Konfigurationsprogramm liegt in binärer Form vor und soll beim Herunterladen so verändert werden können, dass es die genannte Markierung an den in Abschnitt 6.3 beschriebenen Web Service überträgt. Nur dann kann der Web Service den richtigen privaten Schlüssel zurück übertragen. Außerdem sollte die Markierung immer eindeutig sein, das bedeutet, dass zwei heruntergeladene Konfigurationsprogramme immer unterschiedliche Markierungen besitzen müssen. Schließlich ist noch wichtig, dass niemand die Markierung eines anderen erraten kann und so zu dem privaten Schlüssel dieser Person kommt.

Um zukünftige Änderungen zu erleichtern, wurde die Art der Markierung im Konfigurationsprogramm selbst offen gelassen. Einige Start- und einige Ende-Zeichen markieren Anfang und Ende eines bestimmten Teilbereichs des Binärcodes des Konfigurationsprogramms. Dieser Teilbereich zwischen Anfang- und Ende-Zeichen wird vom Konfigurationsprogramm selbst ausgelesen und als Authentifizierungscode an den Web

Service übertragen. Beim Download wird der vorher willkürliche Text des Teilbereichs durch den tatsächlichen Authentifizierungscode ersetzt. Server-seitig wird der Authentifizierungscode aus einem GUID gebildet.

9.3.1 Sicherheit bei Verwendung von GUIDs

Eine Voraussetzung für die Sicherheit der GUIDs ist ihre Zufälligkeit oder Pseudo-Zufälligkeit. Pseudo-Zufallszahlen sind nicht völlig zufällig, sondern werden errechnet. Ihre Berechnung ist aber von einem Außenstehenden nicht nachzuvollziehen, so dass sie für diesen zufällig *wirken*. Im Folgenden werden Pseudo-Zufallszahlen und Zufallszahlen gleich behandelt, da die Sicherheit von Pseudo-Zufallszahlen nicht Kern dieser Arbeit ist. GUIDs sind zwar nicht völlig zufällig oder pseudo-zufällig, die in aktuellen Windows-Versionen und insbesondere in Windows 2003 und Windows Longhorn verwendeten GUIDs in Version 4 enthalten aber immerhin 122 pseudo-zufällige Bits [LMS05].

Durch die hohe Zahl möglicher GUIDs, nämlich $2^{122} \approx 5,31 \cdot 10^{36}$, ist ein Erraten des Authentifizierungscodes praktisch unmöglich: Die Wahrscheinlichkeit, dass ein zufällig herausgesuchter GUID ein ausgegebener GUID ist, entspricht dem Verhältnis von ausgegebenen GUIDs zur Gesamtzahl der GUIDs. Bei einer Billion ausgegebenen GUIDs wäre sowohl das Verhältnis der ausgegebenen GUIDs zur Gesamtzahl aller GUIDs als auch die Wahrscheinlichkeit p, dass ein zufällig herausgesuchter GUID ein ausgegebener GUID ist, damit

$$p = (\frac{5,31 \cdot 10^{36}}{10^{12}})^{-1} = (5,31 \cdot 10^{24})^{-1} \approx 1,88 \cdot 10^{-25}$$

Selbst wenn man von 10^{30} GUIDs schon wüsste, dass sie nicht ausgegeben wurden, ändert sich diese Wahrscheinlichkeit nur vernachlässigbar in der Größenordnung $10^{-6} \cdot p$, da

$$\frac{5,31 \cdot 10^{36} - 10^{30}}{10^{12}} \approx 5,31 \cdot 10^{24}$$

Bei einer Abfrage von einer Million GUIDs pro Sekunde hätte man nach 10 Billionen Jahren $10^6 \cdot 60 \cdot 60 \cdot 24 \cdot 365 \cdot 10^{13} = 3,1536 \cdot 10^{26}$ GUIDs abgefragt. Selbst wenn sich alle diese GUIDs unterscheiden würden, wäre also bei jedem Test von maximal $3,1536 \cdot 10^{26}$ GUIDs bekannt, dass sie nicht herausgegeben sind. Die Wahrscheinlichkeit, dass eine bestimmte dieser getesteten GUIDs ein ausgegebener GUID ist, ist demnach immer noch ungefähr p. Im Folgenden wird der eben beschriebene Fehler in Kauf genommen und davon ausgegangen, dass jeder Test die gleiche Wahrscheinlichkeit p hat, einen herausgegebenen GUID zu finden. Dies ist möglich, da dieser Fehler die Wahrscheinlichkeit p viel weniger stark verändert als die Anzahl der ausgegebenen GUIDs, die, wie eingangs gesagt, proportional zu p ist.

Wenn man bei der Suche nach einem ausgegebenen GUID also nur die ersten 10 Billionen Jahre berücksichtigt, handelt es sich wahrscheinlichkeitstheoretisch beim Ausgang des Tests eines GUID um eine Bernoulli-Verteilung, bei der gesamten Suche nach GUIDs um einen Bernoulli-Prozess, da die Wahrscheinlichkeit, einen herausgegebenen GUID zu finden, für jeden getesteten GUID konstant p ist [Dra67]. Somit

gibt die geometrische Verteilung die Anzahl der erforderlichen Versuche an, um einen ausgegebenen GUID zu finden. Da es sich um die geometrische Verteilung handelt, ist der Erwartungswert der zu testenden GUIDs vor dem ersten Treffer ungefähr $p^{-1} = 5,31 \cdot 10^{24}$. Die erwartete Zeit, bis zu der die Suche den ersten tatsächlich herausgegebenen GUID gefunden hat, ist damit etwa 168,5 Milliarden Jahre. Dies ist die Zeit, bis man bei Überprüfung von einer Million GUIDs pro Sekunde $5,31 \cdot 10^{24}$ GUIDs überprüft hat.

Selbst wenn sich in Zukunft noch Umstände ergeben, die eine schnellere Überprüfung der GUIDs zulassen, ist damit ein gewisser Puffer vorhanden, der die Sicherheit des Systems gewährleistet.

9.4 Konfiguration der Email-Programme

In diesem Abschnitt werden die Klassen beschrieben, die die unmittelbaren Veränderungen an den Email-Programmen herbeiführen, das sind die abstrakte Klasse *IAppConfigurator* und ihre Tochterklassen.

9.4.1 Objektmodell

Ein Zeichen guter Programmierung ist das Abtrennen sich verändernder Programmteile, damit eine Änderung funktionierenden Code nicht beeinträchtigt [FF04]. Da anzunehmen ist, dass in Zukunft auch andere Email-Programme oder andere Versionen der vorhandenen Email-Programme unterstützt werden müssen, ändert sich der Code, der die einzelnen Email-Programme konkret bedient, sehr häufig. Um diesen Code abzukapseln, wird eine Variation des Design Patterns „Template Method“ [GHJV95] benutzt. Es gibt daher ein Interface *IAppConfigurator*, das die einzelnen Schritte der Konfiguration eines Email-Programms als Funktionsköpfe enthält (siehe Abbildung 9.1). Im Gegensatz zum klassischen Template Method Pattern gibt es allerdings gar keine TemplateMethod im Interface. Diese Methode wird stattdessen in einer anderen externen Klasse implementiert, da der Algorithmus insgesamt von vielen Parametern abhängt und die Übersichtlichkeit der Klasse gelitten hätte, wenn eine TemplateMethod mit so vielen Parametern deklariert worden wäre. Außerdem soll der Fortschritt des Prozesses überwacht werden, was im Falle einer TemplateMethod innerhalb von *IAppConfigurator* umständlicher gewesen wäre.

Die Tochterklassen der Basisklasse *IAppConfigurator* repräsentieren jeweils einen Typ von Email-Programm und enthalten den Code zur Konfiguration dieses Email-Programms.

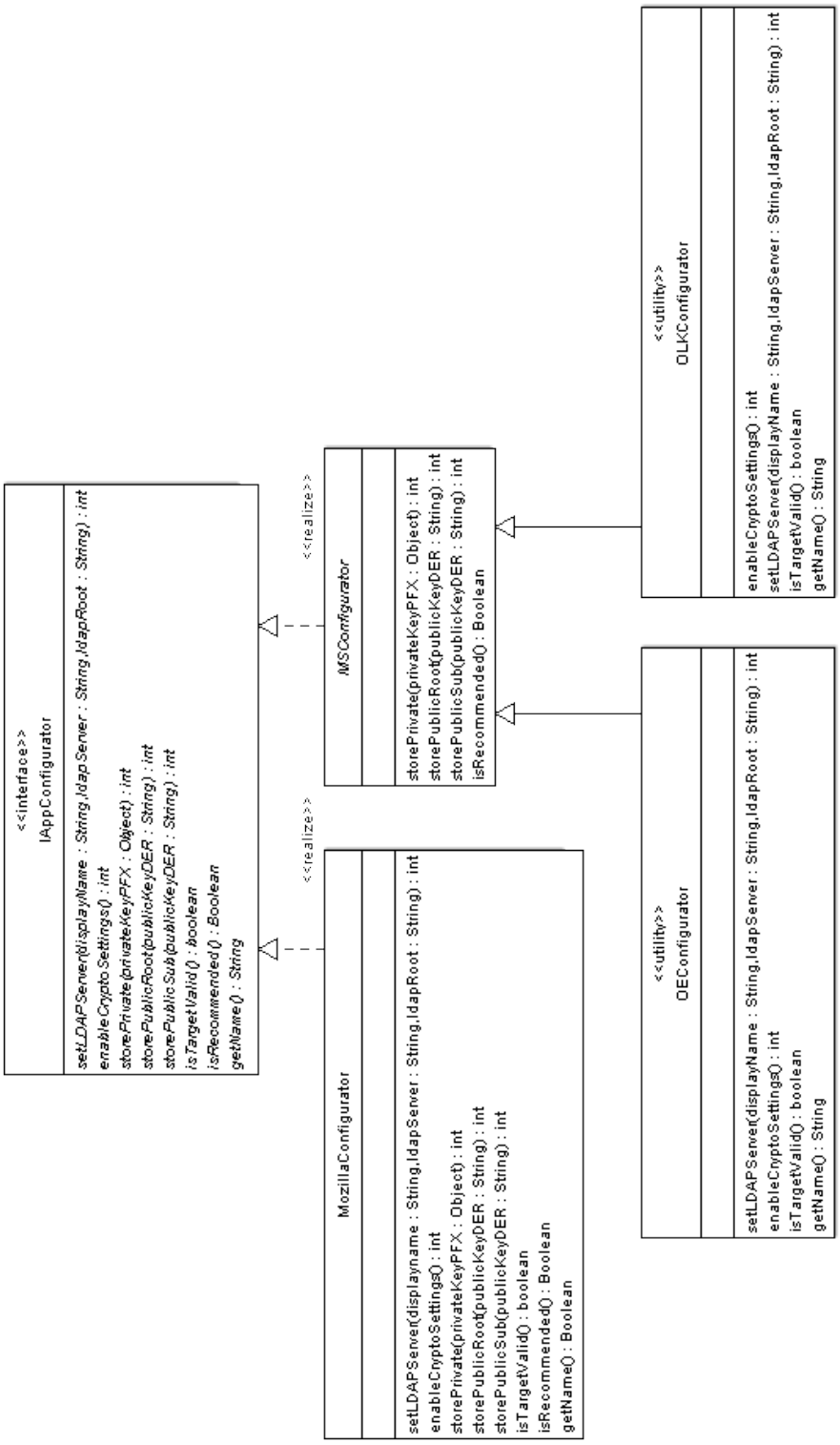

Abbildung 9.1: Vereinfachtes UML-Klassendiagramm der Konfigurationsklassen für die Email-Programme

Aufbau des Interface *IAppConfigurator*

Auf einem System können auch mehrere Email-Programme installiert sein und der Benutzer möchte eventuell in allen Email-Programmen verschlüsselt kommunizieren. Für jedes Email-Programm gibt es daher eine von *IAppConfigurator* erbende Klasse, so dass die TemplateMethod für jede dieser Klassen einmal ausgeführt werden muss. Um die Objekte der verschiedenen Tochterklassen zu verwalten, gibt es die Methoden *isTargetValid*, *isRecommended* und *getName*. Die Methode *isTargetValid* gibt dabei an, ob das passende Email-Programm auf diesem Rechner installiert ist und konfiguriert werden kann, *isRecommended* gibt in diesem Fall an, ob es vorausgewählt sein soll, dieses Email-Programm zu konfigurieren. Die Methode *getName* zeigt einen lesbaren Namen des Email-Programms an, damit dem Benutzer die verfügbaren Programme zur Auswahl gestellt werden können.

Die Methode *setLDAPServer* trägt das übergebene Adressbuch im Email-Programm ein, *enableCryptoSettings* aktiviert Verschlüsselung und Signaturen in den Einstellungen des Email-Programms. Die Funktionen *storePrivate*, *storePublicRoot* und *storePublicSub* speichern einen Schlüssel in der Schlüsseldatenbank des Email-Programms. Dabei legt *storePrivate* einen geheimen Schlüssel fest, *storePublicRoot* und *storePublicSub* speichern öffentliche CA-Zertifikate, nämlich die Root CA (Vertrauensanker) beziehungsweise eine untergeordnete CA. Geheime Schlüssel werden als binäre Parameter im PKCS #12-Format (siehe Abschnitt 2.1.3 dieser Arbeit) übergeben, die öffentlichen Schlüssel als DER-kodierte X.509-Zertifikate (siehe Abschnitt 2.1).

Die Klassen *MozillaConfigurator*, *OEConfigurator* und *OLKConfigurator* zur Konfiguration der Email-Programme

Die Klassen *MozillaConfigurator*, *OEConfigurator* und *OLKConfigurator* repräsentieren im Objektmodell die Email-Programme Mozilla Thunderbird, Microsoft Outlook Express und Microsoft Outlook. Da Outlook Express und Outlook eine gemeinsame Schlüsseldatenbank besitzen, erben sie von einer zusätzlichen abstrakten Klasse *MSConfigurator*, die die Methoden zur Speicherung der Schlüssel implementiert. *MozillaConfigurator* kann dagegen direkt von *IAppConfigurator* erben, weil sie mit den übrigen Klassen nichts gemeinsam hat.

Man kann Outlook und Outlook Express nur einmal im System installieren, weswegen das Singleton Pattern [GHJV95] für ihre Erzeugung verwendet wurde. Da man Thunderbird theoretisch mehrfach installieren kann, erzeugt man die Klasse über einen Constructor. So kann man ein Objekt für jedes installierte Thunderbird erzeugen und alle installierten Thunderbird-Programme in einem Durchgang konfigurieren. Im GUI wurde diese Möglichkeit allerdings noch nicht umgesetzt, hier ist nur die Konfiguration eines installierten Thunderbird-Programms möglich.

9.4.2 Outlook

Microsoft Outlook wird über die Klasse *OLKConfigurator* konfiguriert, indem die in Abschnitt 8.3 angedeuteten und in Anhang B aufgeschlüsselten Änderungen in der

Registry durchgeführt werden. Dazu wird die Klasse *RegistryKey* verwendet, die eine objektorientierte Kapselung der Windows-API-Funktionen zum Registry-Zugriff ist. Outlook legt bei der Installation in der Registry den Schlüssel mit einigen Einstellungen an. Um herauszufinden, ob Outlook installiert ist, wird daher mit Hilfe der Klasse *RegistryKey* die Existenz des Konfigurationsschlüssels abgefragt.

Die Microsoft Schlüsseldatenbank

Wie zuvor erwähnt benutzen Outlook und Outlook Express eine gemeinsame Schlüsseldatenbank. Die Konfigurationsklassen *OLKConfigurator* und *OEConfigurator* erben daher von der gemeinsamen Basisklasse *MSConfigurator*, die für Zugriffe auf die Windows Schlüsseldatenbank zuständig ist. Die drei Methoden *storePrivate*, *storePublicRoot* und *storePublicSub* operieren auf kryptographischen Schlüsseln und sind direkt mittels Zugriffen auf das CryptoAPI (siehe Abschnitt 8.3.4) implementiert. Von den PKCS #12-Daten für die Methode *storePrivate* wird erwartet, dass sie entweder nicht verschlüsselt sind, verschlüsselt mit einem leeren String oder verschlüsselt mit dem Passwort „password“. Eine stärkere Verschlüsselung wird nicht gewählt, da die Daten nie über einen unsicheren Kanal übertragen werden und das Passwort über den gleichen Kanal wie die Daten übertragen werden muss. Insofern hat ein Angreifer ohnehin auch das Passwort, wenn er Zugriff auf die Daten erhält. Ursprünglich war es nicht geplant, auch „password“ zu unterstützen, wegen Problemen Thunderbirds mit passwortlosen PKCS #12-Daten wurde dieses Passwort jedoch auch noch aufgenommen (siehe Abschnitt 9.4.4).

Wenn man einen privaten Schlüssel in der Microsoft Schlüsseldatenbank speichert, hat man die Möglichkeit, eine in der Oberfläche so genannte „strong private key protection“ einzuschalten. Ist diese eingeschaltet, öffnet sich bei jedem Zugriff auf den privaten Schlüssel ein Fenster, in dem der Benutzer gefragt wird, ob der private Schlüssel tatsächlich verwendet werden darf. Bei der Aktivierung der „strong private key protection“ kann der Benutzer zusätzlich ein Passwort festlegen, das in diesem Fall immer eingegeben werden muss, wenn der private Schlüssel benutzt wird. Das in dieser Arbeit beschriebene Konfigurationsprogramm nutzt keine der beiden Möglichkeiten, den privaten Schlüssel zu schützen, damit der Benutzer nicht bei jedem Verfassen einer Email eine zusätzliche Meldung erhält. Auf diese Weise kann er wie gewohnt Emails schreiben, ohne sich auf zusätzliche Veränderungen einstellen zu müssen.

9.4.3 Outlook Express

Die Klasse *OEConfigurator* verändert die Registry-Einträge an den in 8.4 beschriebenen Stellen, um zum Beispiel ein Adressbuch hinzuzufügen oder ausgehende Emails zu verschlüsseln und zu signieren. Im Lauf der Untersuchungen stellte sich heraus, dass Outlook Express leider nur dann nach dem Zertifikat des Empfängers einer Email im Adressbuch sucht und die Email verschlüsselt, wenn der Empfänger explizit aus dem Adressbuch ausgewählt wurde. Wenn einfach nur die Email-Adresse des Empfängers in das Empfängerfeld einer Email eingegeben wurde, wird nicht überprüft, ob der Empfänger nicht vielleicht trotzdem ein Zertifikat im Adressbuch gespeichert hat.

Auch *OEConfigurator* bedient sich der Klasse *RegistryKey*. So kann auch überprüft werden, ob Outlook Express überhaupt schon einmal benutzt wurde, denn bei der ersten Benutzung wird der Registry-Schlüssel

HKEY_CURRENT_USER\Software\Microsoft\Internet Account Manager

angelegt. Die Outlook Express-Konfiguration war deutlich einfacher umzusetzen als in Outlook, insbesondere ein Adressbuch hinzuzufügen. Der Rumpf der Funktion zum Hinzufügen eines Adressbuchs umfasst in *OLKConfigurator* 293 Zeilen, in *OEConfigurator* nur 32 Zeilen.

9.4.4 Thunderbird

In Abschnitt 8.5 wird beschrieben, dass die Einstellungen in der Datei *prefs.js* verändert und die verschiedenen Schlüssel in der Thunderbird-Schlüsseldatenbank, bestehend aus den Dateien *semod.db*, *cert8.db* und *key3.db*, gespeichert werden müssen, um Thunderbird zu konfigurieren. Dazu muss das Profilverzeichnis gefunden werden, in dem die zu verändernden Dateien liegen. Um Kompatibilitätsprobleme zu vermeiden, sollten nicht zwei verschiedene Versionen von NSS auf dieselbe Schlüsseldatenbank zugreifen. Daher sollte die vom Konfigurationsprogramm eingesetzte Version von NSS der entsprechen, die auch das produktive Thunderbird einsetzt. Daher und wegen des geringeren Speicherplatzbedarfs wird der NSS-Code nicht statisch in die ausgelieferte Datei des Konfigurationsprogramms eingefügt, sondern es wird auf die durch Thunderbird mitinstallierten NSS-DLLs zurückgegriffen. Diese liegen im Installationsverzeichnis von Thunderbird, so dass auch dieses ermittelt werden muss, bevor mit der eigentlichen Konfiguration begonnen werden kann.

Auf einem Rechner können mehrere Email-Programme installiert sein, die dem Mozilla Thunderbird sehr ähnlich sind und mit der Klasse *MozillaConfigurator* konfiguriert werden könnten. Daher lassen sich die Pfade des Mozilla-Installationsverzeichnisses und des Profilverzeichnisses mit einem Aufruf der Funktion *setPaths* setzen (siehe auch Abbildung 9.2). Da auf vielen Zielsystemen nur ein solches Email-Programm installiert ist, ermittelt die Methode *autoDetectPaths* des *MozillaConfigurator* selbst geschickte Werte für die beiden Pfade. Das Mozilla-Installationsverzeichnis wird dabei über die Registry ermittelt, da Windows dort die auf dem System installierten Programme speichert, wenn sie in der Windows Systemsteuerung unter „Software“ (beziehungsweise in der englischen Version „Add or Remove Programs“) aufgelistet werden.

In Mozilla Thunderbird 1.5 und 2.0 kann man einstellen, dass Emails in der Voreinstellung nie verschlüsselt werden sollen oder dass Verschlüsselung zwingend notwendig ist. Es gibt jedoch keine Option wie bei Outlook und Outlook Express, mit der Emails verschlüsselt werden, wenn das Zertifikat des Empfängers gefunden werden kann, und sonst die Email unverschlüsselt verschickt wird. Das Konfigurationsprogramm stellt deswegen ein, dass Emails voreingestellt nie verschlüsselt werden. Der Benutzer muss vor dem Versenden jeder Email auswählen, dass die Email verschlüsselt werden soll. Dies läuft zwar dem Ziel entgegen, dass Kryptographie möglichst einfach und ohne Benutzereingriff erfolgen soll, die zwingende Verschlüsselung ist aber dennoch die schlechtere Wahl: Mit zwingender Verschlüsselung können Emails an Empfänger ohne Zertifikate gar nicht mehr abgeschickt werden. Signieren ausgehender Emails und Entschlüsseln eingehender Emails sind von diesem Problem aber unberührt und funktionieren ohne Benutzereingriff.

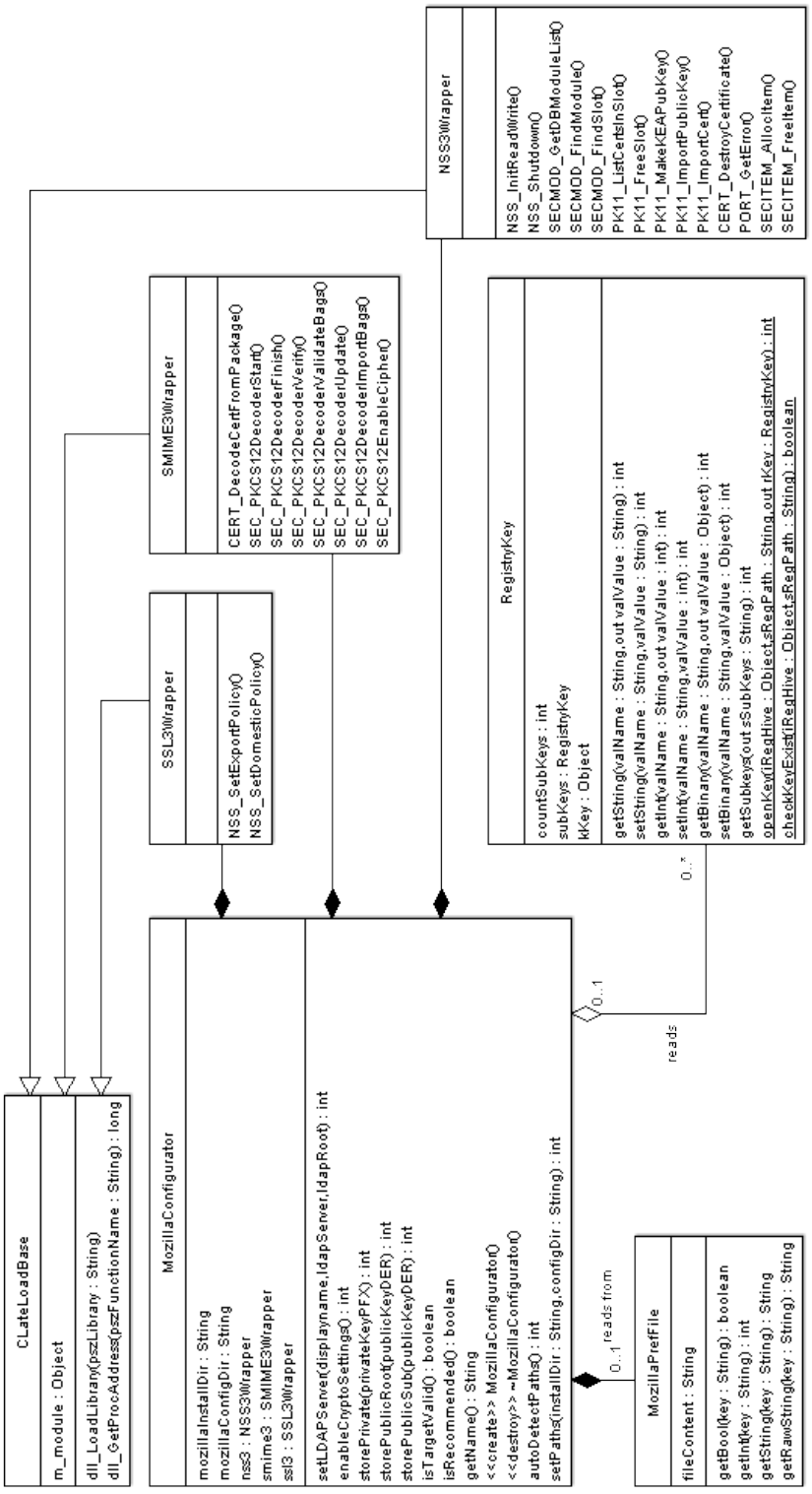

Abbildung 9.2: UML-Klassendiagramm der Klassen rund um *MozillaConfigurator*

Zugriff auf die Thunderbird DLLs

Um auf die Schlüsseldatenbank zuzugreifen, werden die in Thunderbird mitinstallierten DLLs *nss3.dll*, *ssl3.dll* und *smime3.dll* verwendet. Der Zugriff auf die einzelnen DLLs erfolgt jeweils über eine eigene Klasse. Diese drei kapselnden Klassen *NSS3Wrapper*, *SSL3Wrapper* und *SMIME3Wrapper* erben von der Klasse *CLateLoadBase*. Die drei genannten Klassen wurden nicht von Grund auf selbst geschrieben, sondern werden mit den LateLoad-Makros von Jason de Arte [DA04] beim Übersetzen des Quelltextes erzeugt. Die Datei, in der die Makros definiert werden, beinhaltet auch die Definition der Klasse *CLateLoadBase*. Mit Hilfe dieser Klassen können die DLLs zur Laufzeit geladen werden, das heißt, dass die DLLs erst in den Speicher kopiert werden, wenn sie auch tatsächlich gebraucht werden. Der andere Weg, das Laden einer DLL zur Startzeit eines Programms, hat den Nachteil, dass das Programm nicht startet, wenn die DLL nicht gefunden wird [Gal99]. Da nicht davon ausgegangen werden kann, dass auf jedem Zielsystem Thunderbird installiert ist, war das Laden zur Startzeit keine Alternative. Das Laden zur Laufzeit ist jedoch deutlich umständlicher, da für den Aufruf einer DLL-Funktion zuerst ein Funktionszeiger ermittelt werden muss, was einen zusätzlichen Funktionsaufruf kostet. Außerdem können bestimmte in die Entwicklungsumgebung Microsoft Visual Studio integrierte Hilfsfunktionen beim Laden zur Laufzeit nicht eingesetzt werden. Ein Beispiel für eine solche Hilfsfunktion ist eine Kurzbeschreibung der Parameter, die erscheint, während der Programmierer einen Funktionsaufruf schreibt. Die LateLoad-Makros adressieren diese Probleme, indem mit Makros jeweils eine Klasse zusammengesetzt wird, die für jede Funktion in der DLL, die man benutzen möchte, eine Methode definiert. Nun muss man nur noch die Methode der Klasse aufrufen und das Makro delegiert den Aufruf an die DLL. Um die speziellen Anforderungen der Thunderbird-DLLs zu erfüllen (zum Beispiel liegen die DLLs nicht am gleichen Ort wie die ausführbare Datei), mussten die Makros geringfügig angepasst werden.

Adressbücher und Konfigurationen anpassen

Wie in Abschnitt 8.5 erwähnt, werden Adressbücher und Einstellungen in der Konfigurationsdatei *prefs.js* gespeichert. Es reicht allerdings nicht aus, nur in diese Datei zu schreiben, sondern man muss auch aus ihr lesen, da bestimmte Einstellungen von den vorhandenen abhängig sind. Man muss beispielsweise für jedes Email-Konto getrennt die kryptographischen Einstellungen vornehmen und dazu den Namen des Kontos kennen. Diese Namen müssen vorher aus der Konfigurationsdatei ausgelesen werden. Die schreibenden Zugriffe sind recht einfach zu handhaben; man muss lediglich an das Dateiende neue Befehle einfügen, bei der nächsten Ausführung Thunderbirds werden die Befehle automatisch an eine sinnvolle Stelle sortiert. Die lesenden Zugriffe erfordern jedoch ein etwas aufwändigeres Parsen, so dass diese Zugriffe in eine eigene Klasse *MozillaPrefFile* ausgelagert wurden.

10 Fazit

Als Ziel dieser Arbeit sollte untersucht werden, wie Privatpersonen ihre Emails kryptographisch sichern können. Auf den Ergebnissen aufbauend sollte ermittelt werden, ob vorhandene S/MIME-fähige Email-Programme zur Aufbau einer PKI genutzt werden können und diese Möglichkeit sollte mit den vorhandenen Systemen verglichen werden.

10.1 Zusammenfassung

Es wurden vorhandene Systeme vorgestellt, mit denen Emails kryptographisch gesichert werden können. Besonderes Augenmerk wurde entsprechend der Zielsetzung auf die Gruppe der Email-Benutzer außerhalb großer Organisationen gelegt. Für große Organisationen lohnt sich die Einführung einer zentralen PKI, Mitarbeiter kleiner Organisationen und Privatanwender können normalerweise auf keine PKI zurückgreifen. Die vorhandenen Lösungen für die Zielgruppe der Email-Benutzer ohne PKI haben unterschiedliche Nachteile und haben sich nicht in großem Umfang durchgesetzt. Es hat sich gezeigt, dass eine brauchbare Lösung sehr einfach zu bedienen und kostenlos sein muss, außerdem muss sie schon mit geringem Aufwand einzurichten sein.

Um diese Probleme zu beheben, wurde eine frei benutzbare PKI eingeführt, die auch dem System Unbekannte benutzen können. Dazu wird nicht aufwändig die gesamte Identität des Benutzers überprüft, stattdessen wird nur die Zusammengehörigkeit von öffentlichem Schlüssel und Email-Adresse sichergestellt. Diese Zusammengehörigkeit lässt sich leicht überprüfen, indem eine Email an die angegebene Adresse verschickt wird und der Empfänger den Erhalt der Email über eine im Textkörper enthaltene eindeutige Zeichenfolge bestätigen muss. Diese Zeichenfolge ist in einem Link eingebettet, so dass der Benutzer seine Identitätsprüfung mit einem Mausklick abschließen kann.

Zusammen mit dem Zertifikat und dem Schlüsselpaar erhält der Benutzer ein Konfigurationsprogramm, das das Email-Programm des Benutzers so vorbereitet, dass es zukünftig versendete Emails verschlüsseln und signieren kann. Eingehende, verschlüsselte Emails können entschlüsselt werden. Da kein zusätzliches Programm für die kryptographischen Operationen auf dem Rechner läuft, können die Funktionalität und Stabilität anderer Programme auf dem Rechner auch nicht durch ein Kryptographie-Programm beeinträchtigt werden. Außerdem muss sich der Benutzer nicht an die Oberfläche eines neuen Programms gewöhnen, sondern kann alle kryptographischen Funktionen über sein bekanntes Email-Programm steuern.

Dass die Entwicklung eines solchen Konfigurationsprogrammes möglich ist, wurde mit der Entwicklung eines Prototypen mit den entsprechenden Funktionen gezeigt. Dabei stellte sich heraus, dass Microsoft Outlook 2003 schwierig automatisiert zu konfigurieren ist, da die Konfiguration auf einer Reihe von Schlüsseln in der Registry beruht, die nicht dokumentiert sind und keine sinntragenden Namen haben. Die Konfiguration von

Microsoft Outlook Express 6 ist deutlich einfacher, da weniger Schlüssel in der Registry verändert werden müssen. Diese Schlüssel haben sinnvolle Namen und sind zumindest inoffiziell dokumentiert. Die Konfiguration von Mozilla Thunderbird funktioniert anders als bei den beiden Microsoft-Programmen und ist daher kaum mit diesen vergleichbar. Schwierigkeiten bereitet hier ebenfalls eine mangelnde Dokumentation, die jedoch teilweise durch den frei verfügbaren Quelltext ausgeglichen wird.

Als fertig konfigurierte Email-Programme gibt es immer noch Unterschiede zwischen den drei betrachteten Programmen. Alle Programme haben mit Entschlüsselungen und dem Signieren von Emails kein Problem, die Benutzbarkeit unterscheidet sich aber bei der Verschlüsselung ausgehender Emails. Während Outlook die gewünschten Operationen automatisch durchführt, muss bei Outlook Express der Empfänger explizit aus dem Adressbuch ausgewählt werden, damit an ihn verschlüsselt wird. In Thunderbird muss der Benutzer entweder für jede Email einzeln festlegen, dass sie verschlüsselt werden soll, oder die Verschlüsselung erzwingen. Bei erzwungener Verschlüsselung können jedoch keine Emails mehr an Empfänger ohne oder mit nicht aufzufindenden Zertifikaten verschickt werden.

10.2 Ausblick

Obwohl das in dieser Arbeit entwickelte System funktioniert, dient es vor allem dem Ziel, die grundsätzliche Möglichkeit der Nutzung S/MIME-fähiger Email-Programme in einer PKI zu beweisen. Entsprechend sind noch einige Verbesserungen möglich.

Vor einem Einsatz auf einer hohen Bandbreite von Zielsystemen sollte die Zahl der unterstützten Email-Programme erhöht werden. Das Konfigurationsprogramm müsste auch ältere Versionen von Microsoft Outlook und das neue Microsoft Outlook 2007 unterstützen. Mit wachsender Verbreitung des Betriebssystems Microsoft Windows Vista sollte der in Vista integrierte Outlook Express-Nachfolger Windows Mail ebenfalls unterstützt werden.

Als Referenz wurde das Konfigurationsprogramm und die Webseite nur in Englisch implementiert. Im Hinblick auf eine leichtere Benutzbarkeit sollten auch andere verbreitete Sprachen unterstützt werden.

Neben der Verwendung für Privatpersonen gibt es auch weitere, in dieser Arbeit nicht betrachtete Anwendungsbereiche des Konfigurationsprogramms. Man könnte das Programm auch gezielt in Unternehmen einsetzen, die die Verwendung zusätzlicher Software zur Absicherung ihrer Emails ablehnen, zum Beispiel um Inkompatibilitäten mit vorhandener Software zu vermeiden. Hierzu sollte der Benutzer seine Einstellungen aber nicht durch einen Assistenten festlegen müssen, sondern das Konfigurationsprogramm sollte unsichtbar und im Hintergrund alle Einstellungen durchführen können. Dazu müsste das Konfigurationsprogramm so erweitert werden, dass es auf Kommandozeilenebene alle relevanten Optionen annehmen kann. Die Administratoren des Unternehmens würden dann vor der Installation an zentraler Stelle die gewünschten Parameter vorgeben.

Verschlüsselung von Emails würde eine deutlich höhere Verbreitung finden, wenn eine zukünftige Version eines verbreiteten Email-Programms schon so vorkonfiguriert wäre, dass Emails verschlüsselt werden. Dies könnte erreicht werden, wenn die Schritte, die

das in dieser Arbeit entwickelte Konfigurationsprogramm durchführt, in die Installation integriert wären.

Anhang

A Anpassung und Einrichtung des OpenLDAP-Servers

Dieser Anhang beschreibt, welche Anpassungen und besonderen Konfigurationen am OpenLDAP-Server SLAPD nötig sind, um den in Abschnitt 7.1 beschriebenen Anforderungen zu genügen.

A.1 Maschinencode erzeugen

OpenLDAP besteht aus verschiedenen Komponenten, darunter verschiedene Programme zur Nutzung von Client-Funktionen (ldapsearch, ldapmodify) und Serverprogrammen (SLAPD, Stand-Alone LDAP Update Replication Daemon (SLURPD)). SLAPD öffnet einen Port, normalerweise den Standard-LDAP-Port 389, und beantwortet LDAP-Anfragen, die den Rechner erreichen. Verschiedene so genannte Backends dienen als Datenquelle für die Antworten. Welches Backend benutzt wird, entscheidet die Konfigurationsdatei *slapd.conf*. SLAPD ist in der Programmiersprache C geschrieben, die im Programm verfügbaren Backends müssen schon zum Zeitpunkt des Linkens bestimmt werden. Um ein Backend benutzen zu können, muss der Quelltext übersetzt und die erzeugte Bibliothek mit dem Rest des Programms verlinkt werden. Die Details dieser Arbeit werden dem Benutzer aber vom OpenLDAP Buildsystem abgenommen: Es gibt ein Bourne Again Shell (BASH)-Script *configure*, welches aus den mitgelieferten, auf jedes System passenden *make.in*-Dateien speziell auf das verwendete System (hier Cygwin) angepasste *make*-Dateien erzeugt. Diese *make*-Dateien werden vom Programm make (für OpenLDAP braucht man GNU make) ausgewertet und die einzelnen Schritte (Übersetzen, Linken, Kopiervorgänge, ...) durchgeführt. Während diese Schritte obligatorisch sind, ist es normalerweise nicht nötig, aber möglich, die Datei *configure* mit dem Programm autoconf aus einer Datei *configure.in* zu erzeugen. Dem BASH-Script *configure* kann man zahlreiche Parameter übergeben, um die Vorgehensweise beim Build, also dem Übersetzen und Linken des Programms, genauer zu bestimmen. So lässt sich auch festlegen, welche Backends tatsächlich verwendet werden sollen. *configure* überprüft in einem gewissen Rahmen, ob das System die Voraussetzungen der gewählten Backends erfüllt.

A.1.1 Probleme mit dem SQL-Backend

Wenn man das SQL-Backend verwenden möchte, muss ein Open Database Connectivity (ODBC)-Treiber auf dem System installiert sein. Windows stellt zwar selbst einen solchen ODBC-Treiber zur Verfügung, aber dieser ist in der für den Build verwendeten Cygwin-Umgebung nicht ohne Weiteres zu erreichen. Windows stellt eine dynamische Bibliothek mit Funktionen für ODBC-Zugriffe in Form einer Win32 DLL, *odbc32.dll*, zur Verfügung. Die Funktionen werden in den Header-Dateien *sql.h* und

sqltypes.h des Windows-API beschrieben. Zur DLL passende Header-Dateien finden sich beispielsweise im Platform SDK der entsprechenden Windows-Version (erhältlich auf `www.microsoft.com`). Um unter Cygwin auf die DLL zugreifen zu können, muss es eine statische Bibliothek geben, die den Zugriff auf die DLL kapselt. Diese statische Bibliothek muss nach außen Funktionen zur Verfügung stellen, die den gleichen Funktionsnamen haben wie die Funktionen, die die DLL zur Verfügung stellt. Ein Funktionsaufruf wird dann nur weiter delegiert an die DLL, die Funktion der statischen Bibliothek ruft also die Funktion der DLL mit dem gleichen Namen auf. Cygwin enthält schon statische Bibliotheken für viele in Windows vorhandene DLLs. So findet sich in der Cygwin-Umgebung im Verzeichnis */lib/w32api* die Datei *libodbc32.a*. Die Dateien der statischen Bibliotheken, die die Zugriffe auf die DLLs kapseln, folgen alle dem gleichen Benennungsschema: Dem Namen der DLL wird ein lib vorangestellt und die Endung wird von .dll auf .a geändert. Da die DLL in Windows *odbc32.dll* heißt, ist es zwar konsequent, die statische Bibliothek auch *libodbc32.a* zu nennen, aber OpenLDAP erwartet, dass die Bibliothek mit dem ODBC-Treiber den Namen *libodbc.a* hat (vermutlich in Unix-Umgebungen üblich). Es ist daher nötig, eine Kopie der Bibliothek mit Namen *libodbc.a* zu erstellen oder das *configure*-Script anzupassen. Ohne weitere Eingriffe lässt sich der OpenLDAP Server mit SQL-Backend aber noch nicht bauen. Zum Verständnis der weiter unten beschriebenen Probleme ist der nachfolgende Einschub nötig, der die Methode beschreibt, mit der Cygwin-Entwickler die statischen Bibliotheken erzeugen und mit der man auch selbst solche statischen Bibliotheken erzeugen kann.

Eine statische Bibliothek für eine vorhandene DLL lässt sich mit dem Programm dlltool beziehungsweise dllwrap erzeugen. Dieser Vorgang wird von Colin Peters [Pet01] und Mark Schoenberg [Sch06] beschrieben. Zur Erstellung der Bibliothek braucht man die zu kapselnde DLL und eine Liste der in der DLL vorhandenen Funktionen, die .def-Datei. In der .def-Datei stehen Zeile für Zeile die Funktionsnamen der von der DLL zur Verfügung gestellten Funktionen. Hierbei ist zu beachten, dass sich die Funktionsnamen, die der Linker erwartet, von den Funktionsnamen im Quelltext etwas unterscheiden [Wik07]. In der .def-Datei müssen die Funktionsnamen so aufgeführt sein, dass der Linker sie erkennt, allerdings ohne den führenden Unterstrich. Man kann eine Liste der Funktionsnamen auf verschiedene Weise erhalten, beispielsweise kann man die Namen mit dem im Microsoft Visual Studio enthaltenen Programm Dependency Walker heraus kopieren (man muss dann nur noch Kopf und Fuß der .def-Datei von Hand hinzufügen).

Das Problem der Aufrufkonvention

Eine Funktion in C kann auf verschiedene Arten aufgerufen werden [Wik07]. Welche Art des Aufrufs verwendet wird, wird durch eine so genannte Aufrufkonvention bestimmt. Der Compiler erzeugt für jede Quelltext-Datei eine Objekt-Datei, in der neben dem übersetzten Maschinencode die Namen der Funktionen stehen, deren Aufrufe noch eingefügt werden müssen. Dabei werden aber nicht die Funktionsnamen wie im Quelltext verwendet, sondern es wird die Aufrufkonvention in den Namen hinein codiert. Im Quelltext muss man stattdessen ein Schlüsselwort angeben, um die Art des Aufrufs festzulegen. Bei *cdecl* muss im Code nach dem Aufruf der Stack von den übergebenen Parametern bereinigt werden, bei *stdcall* reinigt die aufgerufene Funktion selbst den Stack. Das bedeutet auch, dass der Stack korrumpiert wird, wenn eine Funktion falsch

aufgerufen wird. Ohne Schlüsselwort wird die Aufrufvariante *cdecl* benutzt, in Windows' ODBC-Treiber werden die Funktionen als *stdcall* aufgerufen. Der Compiler schreibt in die Objektdatei einen Unterstrich vor den Funktionsnamen, wenn die Funktion nach *cdecl* aufgerufen werden soll. Wenn die Funktion nach *stdcall* aufgerufen werden soll, schreibt der Compiler zusätzlich ein @ und dezimal die Gesamtgröße aller Parameter in Byte hinter den Funktionsnamen. Eine Funktion foo mit einem 4 Byte großen Parameter (etwa einem int) würde demnach bei Aufrufkonvention *stdcall* kodiert werden zu _foo@4.

Da die ODBC-DLL Aufrufe nach *stdcall* erwartet, haben auch in *sql.h* und *sqltypes.h* des Windows-API alle Funktionsaufrufe das Schlüsselwort *stdcall*. Das OpenLDAP-SQL-Backend erwartet daher, dass die statische Bibliothek die Funktionen als *stdcall* anbietet oder anders gesagt, die Funktionsnamen müssen @n als Suffix haben, wobei n die Zahl der Byte aller Parameter ist. Dies ist bei der Datei *libodbc32.a*, die mit Cygwin mitinstalliert wird, auch der Fall. Das Script *configure* testet das Vorhandensein einer funktionierenden ODBC-Bibliothek aber damit, dass ein C-Programm compiliert und gelinkt wird, welches einen Funktionsaufruf auf die Bibliothek macht. In diesem Programm wird aber nicht die Header-Datei *sql.h* oder *sqltypes.h* eingebunden, sondern die Funktion wird neu deklariert. Hierbei wird allerdings nicht das Schlüsselwort *stdcall* verwendet, der Funktionsaufruf wird also zum *cdecl*. Eine andere ODBC-Bibliothek für Unix benutzt tatsächlich *cdecl*-Aufrufe, vermutlich ist diese Aufrufkonvention in Linux üblich. Als Ergebnis scheitert das Linken des Testprogramms im *configure*-Script und der Buildvorgang wird abgebrochen. Man kann mit dlltool eine statische Bibliothek als Kapselung der *odbc32.dll* erzeugen, die zwar die Funktionen in der DLL mit *stdcall* aufruft, ihre eigenen Funktionen werden aber als *cdecl* aufgerufen (dies erreicht man mit dem Parameter -k). dlltool sollte mit dem Parameter -A sogar statische Bibliotheken erzeugen können, die sowohl als *stdcall*, als auch als *cdecl* aufgerufen werden können, dies ließ sich aber auf dem hier benutzten System nicht nachvollziehen. Mit einer statischen Bibliothek, die nur *cdecl*-Aufrufe bietet, lässt sich das SQL-Backend aber nicht erzeugen, da das SQL-Backend auf der Header-Datei *sql.h* beruht. Die *sql.h* des Windows API erwartet aber *stdcall*-Funktionen. Das Problem kann man auf drei verschiedene Arten lösen, nämlich

1. durch Änderung des *configure*-Scripts, so dass es in seinem Testprogramm einen *stdcall*-Aufruf vornimmt. Dabei wird die in Cygwin enthaltene statische Bibliothek benutzt.
2. Alternativ werden die Dateien *sql.h* und *sqltypes.h* angepasst, so dass sie *cdecl*-Aufrufe erwarten, hierbei wird eine selbst erstellte statische Bibliothek verwendet oder
3. das *configure*-Script nutzt eine selbst erstellte statische Bibliothek, die *cdecl*-Aufrufe erwartet. Danach wird die statische Bibliothek gegen die in Cygwin enthaltene ausgetauscht und das eigentlich Programm samt Backend gebaut.

Die erste Methode hat den Nachteil, dass man von einem veränderten *configure*-Script abhängig ist. Wenn man eine neuere Version von OpenLDAP benutzen möchte, muss man die Änderungen einzeln übertragen. Bei der zweiten Methode muss man eine funktionierende statische Bibliothek erstellen, während die in Cygwin enthaltene Bibliothek schon getestet ist. Dazu ist es unter anderem notwendig, die Gesamtgröße der Parameter in Byte jeder einzelnen Funktion zu ermitteln, um in der .def-Datei den korrekten Funktionsnamen mit @ und der Größe schreiben zu können. Die dritte Methode

ermöglicht, dass die selbst erstellte statische Bibliothek nur eine Fassade mit korrekten Funktionsnamen in *cdecl*-Aufrufkonvention sein kann. Sie muss nicht funktionieren - das *configure*-Script kompiliert das Testprogramm nur und führt es nicht aus. Solch eine statische Bibliothek ist einfach zu erzeugen und muss nicht getestet werden. Nachteil der dritten Methode ist es, dass man einen zusätzlichen Schritt im Buildvorgang hat. In diesem Projekt wurde auf die dritte Möglichkeit zurückgegriffen, ausschlaggebend war die Einfachheit der Lösung.

Das Problem der unvollständigen ODBC-Header-Datei

Nach Beseitigung der oben genannten Probleme würde der Buildvorgang beim Aufruf von make mit einem Fehler stehenbleiben. Das SQL-Backend kann noch nicht gebaut werden, da der Compiler bestimmte Symbole in den Dateien *sql.h.* und *sqltypes.h* nicht erkennt. In Programmen, die das Windows-API benutzen, wird meistens die Header-Datei *windows.h* eingebunden. In dieser wird über Umwege die Header-Datei *wtypes.h* eingebunden, in der die unbekannten Symbole definiert werden. Obwohl die Header-Datei *wtypes.h* eine Direktive enthält, die sie vor mehrfachem Laden schützt, wird sie nicht noch einmal in *sqltypes.h* eingebunden. Ein Programm, welches nur auf die Funktionen in *sql.h* und *sqltypes.h* zurückgreifen möchte, muss also trotzdem *wtypes.h* einbinden. Das OpenLDAP SQL-Backend tut dies nicht, da es plattformunabhängig ist. Nachdem man in *sqltypes.h* die Datei *wtypes.h* eingebunden hat, lässt sich der OpenLDAP Server mit SQL-Backend bauen.

A.2 Konfiguration des SLAPD

In der Datei *slapd.conf* werden globale Einstellungen des LDAP-Servers SLAPD vorgenommen. Hier wird auch festgelegt, auf welche Backends der Server bei einer Anfrage zurückgreift. Es wurde festgelegt, dass für einen bestimmten Zweig des LDAP-Verzeichnisbaumes auf einen SQL-Server zugegriffen werden soll. Auch die Zugangsdaten zum SQL-Server, wie der Name des Benutzers, aber auch die verwendete Datenbank, werden durch die Datei *slapd.conf* festgelegt. Eine Beschreibung der möglichen Einstellungen findet man im OpenLDAP Administrator's Guide auf der OpenLDAP-Webseite [Ope07].

A.2.1 Konfiguration des SQL-Backend

Der Aufbau der Datenbank für den Verzeichnisdienst ist vom SQL-Backend nicht statisch festgelegt. Vielmehr kann eingestellt werden, welche Information in welcher Tabelle steht. Die Konfiguration erfolgt über bestimmte Hilfstabellen, die in der gleichen Datenbank existieren müssen, in der auch die Daten für das Backend gespeichert werden und die in der Datei *slapd.conf* festgelegt wurde (in [Yod04] und Abschnitt 7.2.2 finden sich genauere Beschreibungen der einzelnen Tabellen). Mit dem SQL-Backend des SLAPD werden SQL-Skripte mitgeliefert, die die erwähnten Hilfstabellen anlegen und mit Beispieldaten füllen.

Das Problem der binären Attributdaten

Pierangelo Masarati, einer der Entwickler des OpenLDAP SQL-Backends, räumt ein, dass das SQL-Backend nicht für die Speicherung und Übertragung binärer Daten ausgelegt ist [Mas06].

Bei SQL-Abfragen weist das SQL-Backend den SQL-Server an, die zurückgelieferten Daten nach ihrem Inhalt zu sortieren. Wenn binäre Attributdaten abgefragt werden, führt dies zumindest beim Microsoft SQL-Server zu einem Fehler, denn nach Spalten mit binärem Inhalt kann nicht sortiert werden. Da für den konkreten Anwendungsfall (Abfrage von Zertifikaten für Email-Verschlüsselung) die Reihenfolge der zurückgelieferten Daten keine Rolle spielt, wurde der Quelltext des SQL-Backends dahingehend angepasst, dass gar keine Sortierung durchgeführt wird.

Trotzdem funktioniert die Abfrage binärer Daten auch nach diesen Anpassungen nicht in allen Fällen korrekt. Das SQL-Backend behandelt alle Daten wie Strings. Insbesondere werden die Daten nur bis zur ersten auftretenden binären Null zurückgeliefert, der so genannten terminierenden Null. Da DER-kodierte Zertifikate durchaus auch binäre Nullen enthalten können, werden diese Zertifikate nur teilweise zurückgeliefert, was dazu führt, dass sie nicht mehr als gültig erkannt werden. Auch für dieses Problem war eine Änderung des Quelltextes des SQL-Backends notwendig. Der Quelltext musste so angepasst werden, dass die Daten in ihrer gesamten Länge und nicht nur bis zur ersten Null übertragen werden.

Die Behandlung der Daten als Strings bewirkt außerdem, dass Binärdaten des SQL-Servers erst in einen String umgewandelt werden müssen. Hierbei werden die Binärdaten in ihre sedezimale Repräsentation überführt, was die Daten für die meisten Drittprogramme unverwertbar macht. Obwohl ein DER-kodiertes Zertifikat binär vom SQL-Server zu OpenLDAP übertragen wird, müssen die Daten daher als String gekennzeichnet werden, um eine weitere Umwandlung durch OpenLDAP zu verhindern.

Das Problem der LDAP-Option *binary*

Die LDAP-Spezifikation [WHK97] sieht vor, dass Attribute mit binären Daten bei ihrem Abruf mit einer zusätzlichen Option *binary* versehen werden. Anstatt direkt das Attribut über seinen Attributnamen *userCertificate* abzufragen, der für X.509-Zertifikate vorgesehen ist, soll bei binären Daten die Option *binary* angehängt werden. Optionen werden vom Attributnamen mit einem Semikolon getrennt. Bei der Abfrage wird also eine Anfrage auf das Attribut mit der so genannten AttributeDescription *userCertificate;binary* gestellt. Im LDAP-Protokoll wird diese AttributeDescription übertragen wie ein Attributname, das Vorhandensein von Optionen ist somit nur am Vorhandensein eines Semikolons zu erkennen. Die LDAP-Komponente von SLAPD gibt diese AttributeDescription unverarbeitet weiter an das SQL-Backend. Dieses erkennt nicht, dass die abgefragte AttributeDescription kein Attributname, sondern ein Attributname mit Optionen ist.

Da die drei Email-Programme Microsoft Outlook, Microsoft Outlook Express und Mozilla Thunderbird bei der Suche eines Zertifikates die Option *binary* verwenden, muss diese Option aber unterstützt werden. Die LDAP-Spezifikation schreibt vor, dass die Option *binary* nur die Übertragungsweise der Daten bestimmt, nicht aber, wie die Daten gespeichert werden. Die sauberste Lösung wäre es gewesen, das SQL-Backend so

zu erweitern, dass Optionen als solche erkannt und getrennt behandelt werden. Da diese Lösung einen erheblichen Programmieraufwand bedeutet hätte, wurde die einfachere, dafür weniger flexible Lösung gewählt, *userCertificate;binary* als zusätzliches Attribut einzutragen. Dieses Attribut verweist auf die selbe Datentabelle wie *userCertificate*, die Zertifikate müssen also nicht doppelt gespeichert werden. Sowohl bei einer Abfrage nach *userCertificate* als auch bei einer Abfrage nach *userCertificate;binary* erhält man also ein Zertifikat. Falls ein LDAP-Client alle Attribute eines Eintrags abfragt, wird der Client allerdings sowohl *userCertificate* als auch *userCertificate;binary* zurückgeliefert bekommen. Das ist hinnehmbar, da zumindest die drei Email-Programme Outlook, Outlook Express und Thunderbird nur explizit *userCertificate;binary* abfragen. Eine Syntax-Überprüfung innerhalb des OpenLDAP-SQL-Backends muss hierfür allerdings mit einer Quelltextänderung ausgehebelt werden. Das Überspringen dieser Überprüfung ist allerdings vorgesehen, so dass hierfür nur ein so genanntes define für den Precompiler gesetzt werden muss.

Leider ist das Semikolon in SQL ein reserviertes Zeichen, es grenzt zwei aufeinanderfolgende Befehle voneinander ab. Im SQL-Backend werden bei den Abfragen des SQL-Servers temporäre Tabellen angelegt. Dabei wird auch eine Tabelle verwendet, deren Spaltennamen den Attributnamen entsprechen. Hierbei wurde nicht berücksichtigt, dass ein Attributname reservierte Zeichen enthalten könnte, weswegen die Abfrage fehlschlägt. Dies lässt sich mit Quoting vermeiden, man muss den Spaltennamen mit eckigen Klammern umklammern. Diese Umklammerung erfordert eine Änderung des Quelltextes des SQL-Backends.

LDAP Controls

Der Standard zu LDAP sieht vor, dass bei LDAP-Abfragen so genannte Controls spezifiziert werden können, die die Art der Abfrage beeinflussen [WHK97]. Jedes Control wird durch eine eindeutige Nummer, ein so genannter Object Identifier (OID), festgelegt und bei der Abfrage mit übertragen. Zusätzlich kann ein Schalter für das Critical Flag gesetzt werden. Dieses Critical Flag beeinflusst das Verhalten des Servers, falls er das Control nicht kennt oder nicht benutzen kann. Ist das Critical Flag nicht gesetzt, ignoriert der Server das unbekannte Control einfach, ist es dagegen gesetzt, bricht der Server die Abfrage mit einer Fehlermeldung ab. Um schon vorher herauszufinden, ob ein bestimmtes Control benutzt werden kann, lädt der Client ein vordefiniertes Objekt des Servers, die *root DSE*, in dem alle unterstützten Controls aufgelistet werden.

Wenn Outlook ein Zertifikat auf einem LDAP-Server sucht, prüft es zunächst, welche Controls dieser Server unterstützt. Unterstützt er das so genannte Paging-Control mit dem OID *1.2.840.113556.1.4.319*, dann benutzt Outlook dieses Paging-Control bei jeder zukünftigen Abfrage und markiert das Control zusätzlich mit dem Critical Flag. Das Control bewirkt, dass ab einer bestimmten Menge an Suchergebnissen diese nur portionsweise zurückgegeben werden. Die Netzwerklast kann so vermindert werden, denn der Client muss aktiv die nächste Portion anfordern, beziehungsweise kann auch darauf verzichten, falls die zurückgegebenen Ergebnisse schon reichen.

SLAPD unterstützt eine Reihe von Controls, unter anderem auch das Paging-Control, jedoch werden nicht alle diese Controls auch von allen Backends unterstützt. Speziell

das SQL-Backend unterstützt das Paging-Control nicht. Beim Zugriff auf die *root DSA-specific Entry (DSE)*[1] gibt der OpenLDAP-Server zurück, dass das Paging-Control unterstützt wird. Wenn über LDAP das Verzeichnis durchsucht wird und auch das SQL-Backend Suchergebnisse zurückliefern soll, allerdings das Paging-Control gesetzt und mit dem Critical Flag markiert ist, wird die Suche mit einer Fehlermeldung abgebrochen.

Dieser Fehler tritt also insbesondere auch dann auf, wenn Outlook ein Zertifikat in SLAPD sucht. Das Problem lässt sich vermeiden, wenn die Liste mit unterstützten Controls auf dem SLAPD gelöscht wird. Dies lässt sich mit einem Eintrag in der *slapd.conf* erreichen, der Zugriffe auf die Liste in der *root DSE* verbietet.

[1] DSA bezeichnet hier den LDAP-Server [WHK97]

B Konfigurationseinträge für Microsoft Outlook im Detail

Tabelle B.1: Kryptographische Einstellungen von Outlook in der Registry (c02ebc5353d9cd11975200aa004ae40e)

Schlüssel	Wert	Zweck
00030354	Vier Byte[2]	Niederwertigstes Bit gibt an, ob Emails verschlüsselt werden sollen, Zweitniederwertigstes Bit gibt an, ob Emails signiert werden sollen
11020355	Binärdaten	Genauer Aufbau unbekannt, enthält aber zumindest die Einstellungen der Outlook „Security Settings“

Tabelle B.2: Zweiter Konfigurationsschlüssel eines Outlook Adressbuchs

Schlüssel	Wert	Zweck
00033009	20 00 00 00[2]	Unbekannt, aber bei allen beobachteten Adressbüchern identisch
000b6613	00 00 oder 01 00[2]	Bei 01 00 wird SSL verwendet, bei 00 00 nicht
000b6615	00 00 oder 01 00[2]	Bei 01 00 wird „Secure Password Authentication“ verwendet, bei 00 00 nicht
001e3001	String	Anzeigename des Adressbuchs in Outlook
001e6600	String	DNS-Name des LDAP-Servers
001e6601	String	Port des LDAP-Servers als String
001e6602	String	Benutzername, mit dem gegen den LDAP-Server authentifiziert wird
001e6603	String	Der LDAP-Pfad, in dem sich die Kontakte befinden
001e6604	String	LDAP-Filter, der die berücksichtigten Kontakte bestimmt
001e6605	„SMTP“	-[1]
001e6606	„mail“	-[1]
001e6607	String	Timeout für die LDAP-Suche in Sekunden als String
001e6608	String	Anzahl der zurückgegebenen Werte pro Suche als String

Fortsetzung auf der nächsten Seite

[1] Auswirkung des Werts unbekannt, aber bei allen beobachteten Adressbüchern identisch
[2] Wert ist in der Registry als REG_BINARY typisiert, auch wenn der Inhalt eigentlich nicht diesen Typ hat
[3] Wert ist auch in der Registry als REG_SZ (Null-terminierter String) typisiert
[4] Wert ist in der Registry als REG_DWORD typisiert

Tabelle B.2 – Fortsetzung der vorigen Seite

Schlüssel	Wert	Zweck
001e6609	„120“	-[1]
001e660a	„15“	-[1]
001e660b	„“	-[1]
001e660c	„OFF“	-[1]
001e660d	„OFF“	-[1]
001e660e	„NONE“	-[1]
001e660f	„OFF“	-[1]
001e6610	„postalAddress“	-[1]
001e6611	„cn“	-[1]
001e6612	„1“	-[1]
001e67f1	Ein Byte[2]	Wenn es diesen Eintrag nicht gibt, öffnet sich beim nächsten Start von Outlook ein Konfigurationsdialog für das Adressbuch. Der eigentliche Wert des Eintrags scheint allerdings keinen Einfluss auf das Verhalten des Adressbuchs zu haben
001f3d09	„EMABLT“[2]	-[1]
001f3d0a	„BJABLR.DLL“[2]	-[1]
001f3d0b	„ServiceEntry“[2]	-[1]
001f3d13	„{6485D268-C2AC-11D1-AD3E-10A0C911C9C0}“[2]	-[1]
01023615	50 a7 0a 61 55 de d3 11 9d 60 00 c0 4f 4c 8e fa[2]	-[1]
01023d01	GUID (binär)[2]	Ein Verweis auf den ersten Konfigurationsschlüssel (Tabelle B.4) des Adressbuchs
01026617	Binärdaten	Das verschlüsselte Passwort, mit dem sich Outlook gegen den LDAP-Server authentifiziert
101e3d0f	02 00 00 00 0c 00 00 00 17 00 00 00 EMABLT.DLL 00 OBJABLR.DLL 00[2]	-[1]

Der Eintrag 11023d05 des Schlüssels 9207f3e0a3b11019908b08002b2a56c2 bestimmt, ob die Empfänger-Email-Adressen abgesendeter Nachrichten noch einmal mit den Einträgen in Adressbüchern gegengeprüft werden sollen. Der Eintrag gibt daher eine Liste von solchen zu durchsuchenden Adressbüchern an; die Reihenfolge der Adressbücher in der Liste ist auch die Reihenfolge, in der Outlook die Adressbücher beim Absender einer Email überprüft.

Der Eintrag 11023d05 ist vom Typ REG_BINARY und besteht in der folgenden Reihenfolge aus

- einer vier Byte großen Litte-Endian-Zahl, die die Anzahl der durchsuchten Adressbücher angibt,
- für jedes Adressbuch zwei vier Byte große Little-Endian-Zahlen, wobei die erste angibt, wie viele Byte das spätere Binary Large Object (BLOB) mit den Adressbuchinformationen umfasst, die zweite gibt den Offset innerhalb des Registry-Schlüssels an, an dem der BLOB zu dem Adressbuch gespeichert ist und
- für jedes Adressbuch ein BLOB mit weitergehenden Informationen.

Die Struktur des BLOB für ein Adressbuch richtet sich nach der Art des Adressbuchs. Für jedes LDAP-Verzeichnis sind die ersten 29 Byte konstant, danach kommt der DNS-Name des Servers (so wie in Eintrag 001e6600 des zweiten Adressbuch-Konfigurationsschlüssels festgelegt, siehe dazu Tabelle B.2). Da die BLOBs immer an einem durch vier teilbaren Offset beginnen, kann es einen nicht benutzten Bereich bis zum Beginn des nächsten BLOBs geben. Dieser Bereich ist mit binären Nullen aufgefüllt.

Tabelle B.3: Schlüssel mit Backup-Daten für Outlook (9207f3e0a3b11019908b08002b2a56c2)

Schlüssel	Wert	Zweck
000b000d	01 00[2]	unbekannt
01023d00	GUIDs (binär)[2]	unbekannt
01023d01	GUIDs (binär)[2]	Die konkatenierten ersten Konfigurationsschlüssel aller Adressbücher
01023d02	GUIDs (binär)[2]	unbekannt
01023d08	GUIDs (binär)[2]	unbekannt
01023d11	GUIDs (binär)[2]	unbekannt
01023d0e	GUIDs (binär)[2]	Die konkatenierten zweiten Konfigurationsschlüssel aller Adressbücher
11023d05	unbekannt[2]	unbekannt

Tabelle B.4: Erster Registry-Konfigurationsschlüssel eines Outlook Adressbuchs

Schlüssel	Wert	Zweck
00033009	00 00 00 00[2]	-[1]
00033e03	23 00 00 00[2]	-[1]
001f3001	„Microsoft LDAP Directory“[2]	-[1]
001f3006	„Microsoft LDAP Directory“[2]	-[1]
001f300a	„EMABLT.DLL“[2]	-[1]
001f3d09	„EMABLT“[2]	-[1]
001f3d13	„{6485D268-C2AC-11D1-AD3E-10A0C911C9C0}“[2]	-[1]
01023d0c	GUID (binär)[2]	Ein Verweis auf den zweiten Konfigurationsschlüssel (Tabelle B.2) des Adressbuchs

Tabelle B.5: Schlüssel mit Outlook-Konten (9375CFF0413111d3B88A00104B2A6676)

Schlüssel	**Wert**	**Zweck**
ED475418-B0D6-11D2-8C3B-00104B2A6676	Konkatenierte 4-Byte-Zahlen im Little-Endian-Format[2]	unbekannt
ED475419-B0D6-11D2-8C3B-00104B2A6676	Konkatenierte 4-Byte-Zahlen im Little-Endian-Format[2]	Die Zahlen geben die Nummern der aktiven (in irgendeiner Weise benutzten) Adressbücher an
ED475420-B0D6-11D2-8C3B-00104B2A6676	Konkatenierte 4-Byte-Zahlen im Little-Endian-Format[2]	Die Zahlen geben die Nummern der aktiven (in irgendeiner Weise benutzten) Email-Postfächer an
LastChangeVer	unbekannt[2]	unbekannt
NextAccountID	Vier-Byte-Zahl (DWORD)	Gibt die Nummer an, die das nächste erzeugte Konto (Email-Postfach oder Adressbuch) haben wird
00000001 bis ffffffff	Unterschlüssel	Jeder Unterschlüssel entspricht einem Konto und ist mit der Nummer des Kontos benannt. Der Aufbau eines Unterschlüssels für ein Adressbuch ist in Tabelle B.6 beschrieben

Tabelle B.6: Dritter Konfigurationsschlüssel eines Outlook Adressbuchs (Unterschlüssel des Schlüssels 9375CFF0413111d3B88A00104B2A6676, siehe Tabelle B.5)

Schlüssel	**Wert**	**Zweck**
Account Name	String[2]	Angezeigter Name des Adressbuchs als Unicode-String
clsid	„{ED475414-B0D6-11D2-8C3B-00104B2A6676}“[3]	-[1]
MAPI Provider	2[4]	-[1]
Mini UID	Vier-Byte-Zahl (DWORD)	unbekannt, Abwesenheit zeigt aber keine Auswirkungen
Service Name	„EMABLT“[2]	-[1]
Service UID	GUID (binär)[2]	Ein Verweis auf den zweiten Konfigurationsschlüssel (Tabelle B.2) des Adressbuchs
XP Capabilities	1[4]	-[1]

C Beschreibung des Quelltextes

Dieser Anhang ist ein Führer durch die Quelltexte der im Rahmen dieser Arbeit entwickelten Programme. Die einzelnen Dateien der Teilprojekte werden tabellarisch aufgelistet und ihr Zweck kurz umrissen.

C.1 DALCryptoLayer

Tabelle C.1 zeigt die Dateien der Bibliothek *DALCryptoLayer*. Die in C++ geschriebene Bibliothek dient als Schnittstelle zwischen den NSS und den .NET-Komponenten auf dem Server. Sie enthält daher .NET-Klassen mit Unmanaged C++-Quelltext. Sie gehört zu der in Kapitel 5 beschriebenen Schlüsselgenerierungskomponente.

Tabelle C.1: Die Bibliothek DALCryptoLayer

Datei	**Codezeilen**	**Inhalt**
Assemblyinfo.cpp	40	Versionsinformationen
ICertificationAuthority.h	20	Interface zur Erzeugung beantragter Zertifikate
MemoryOutput.h und MemoryOutput.cpp	14 + 33	Hilfsklasse, um Strings zusammenzusetzen
NSSKeyGenerator.h und NSSKeyGenerator.cpp	16 + 242	Erzeugt mit den NSS Schlüsselpaare
Stdafx.h und Stdafx.cpp	5 + 5	Erforderlich für bestimmte Compileroptimierungen
Gesamt	375	

C.2 BIKeyDistributor

Die C#-Bibliothek *BIKeyDistributor* erfüllt verschiedene Funktionen. Sie dient als Schnittstelle zur in Abschnitt 7.2 beschriebenen Datenbank, enthält Klassen für die in Abschnitt 5 eingeführten kryptographischen Operationen und verschiedene kleinere Backend-Dienste für die Web-Applikationen, die in Abschnitt 6 dargestellt werden. Die einzelnen Dateien werden in Tabelle C.2 beschrieben.

C.3 certadm

Die C#-Web-Applikation *certadm* realisiert die in Abschnitt 6.2 beschriebene Administrationsoberfläche und besteht aus den in Tabelle C.3 beschriebenen Dateien.

Tabelle C.2: Die Bibliothek BIKeyDistributor

Datei	**Codezeilen**	**Inhalt**
Assemblyinfo.cs	35	Versionsinformationen
CertConfig.cs	148	Zugriff auf globale Konstanten und Einstellungen
CertFuncs.cs	234	Statische Methoden auf einer hohen Abstraktionsebene
DbAuthCodes.cs	280	Repräsentiert eine Verifikations-Email
DbEmail.cs	140	Repräsentiert eine Email-Adresse
DbKey.cs	269	Repräsentiert ein Schlüsselpaar
DbObject.cs	108	Repräsentiert ein Datenbankobjekt, Basisklasse der übrigen Db...-Klassen
DbUser.cs	107	Repräsentiert einen registrierten Benutzer
IKeyGenerator.cs	20	Interface zur Erzeugung von Schlüsselpaaren
MailManager.cs	59	Versenden der Verifikations-Emails
MSCA.cs	93	Realisiert ICertificationAuthority aus *DALCryptoLayer*, steuert die Microsoft CA an
NSSKeyGenerator.cs	29	Adapter zwischen IKeyGenerator und NSSKeyGenerator aus *DALCryptoLayer*
SqlWrapper.cs	79	Schnittstelle zur Datenbank
Gesamt	1601	

Tabelle C.3: Die Web-Applikation certadm

Datei	**Codezeilen**	**Inhalt**
admin.master.cs und admin.master	21 (+ 25)	Grundstruktur aller Seiten der Web-Applikation
cakeys.aspx.cs und cakeys.aspx	98 (+ 16)	Übersicht der CA-Zertifikate
certadmin.css	(82)	Cascading Style Sheet (CSS), definiert Aussehen der Web-Applikation
configuration.aspx.cs und configuration.aspx	18 (+ 18)	Globale Einstellungen festlegen
downloadSysObj.aspx.cs und downloadSysObj.aspx	36 (+ 3)	Herunterladen der CA-Zertifikate
users.aspx.cs und users.aspx	18 (+ 19)	Übersicht der ausgegebenen Zertifikate
web.config	(49)	Einstellungen des Webservers
Gesamt	191 (+ 212)	

Tabelle C.4: Die Web-Applikation certreq

Datei	**Codezeilen**	**Inhalt**
certreq.css	(156)	Definiert Aussehen der Web-Applikation
changeCertificate.aspx.cs und changeCertificate.aspx	89 (+ 7)	Dialog bei wiederholter Beantragung
download.aspx.cs und download.aspx	41 (+ 3)	In Verifikations-Email verlinkt, Herunterladen des Konfigurations-programms
downloadCertificate.aspx.cs und downloadCertificate.aspx	34 (+ 3)	Herunterladen eines öffentlichen Zertifikats
findCertificate.aspx.cs und findCertificate.aspx	55 (+ 8)	Erste Dialogseite beim Suchen eines Zertifikats
findCertificate2.aspx.cs und findCertificate2.aspx	30 (+ 5)	Zweite Dialogseite beim Suchen eines Zertifikats
GenericReqPage.cs	39	Basisklasse für alle Seiten der Web-Applikation
info.aspx.cs und info.aspx	18 (+ 20)	Informationen zu Kryptographie
recoverFinish.aspx.cs und recoverFinish.aspx	18 (+ 5)	Bestätigungsdialog für mehrmaliges Versenden der Verifkations-Emails
renewFinish.aspx.cs und renewFinish.aspx	18 (+ 5)	Bestätigungsdialog für Verlängerung der Zertifikatsgültigkeit
request.master.cs und request.master	25 (+ 37)	Grundstruktur aller Seiten der Web-Applikation
revoke.aspx.cs und revoke.aspx	36 (+ 10)	Eingabedialog für Widerruf eines Zertifikats
revokeFinish.aspx.cs und revokeFinish.aspx	18 (+ 5)	Bestätigungsdialog zum Widerruf eines Zertifikats
subscribe.aspx.cs und subscribe.aspx	61 (+ 9)	Beantragung eines Zertifikats
subscribeFinish.aspx.cs und subscribeFinish.aspx	18 (+ 5)	Bestätigungsdialog zur Beantragung eines Zertifikats
twiceDownload.aspx.cs und twiceDownload.aspx	18 (+ 5)	Warnmeldung bei mehrmaligem Herunterladen eines privaten Schlüssels
web.config	(49)	Einstellungen des Webservers
Gesamt	518 (+ 332)	

C.4 certreq

Der in Abschnitt 6.1 beschriebene Internet-Auftritt wird durch die in Tabelle C.4 aufgelistete C#-Web-Applikation *certreq* realisiert.

C.5 keyloader

Die Funktion des in C# geschriebenen Web Services *keyloader* wird in Abschnitt 6.3 beschrieben. Er setzt sich aus den in Tabelle C.5 gezeigten Dateien zusammen.

Tabelle C.5: Der Web Service keyloader

Datei	**Codezeilen**	**Inhalt**
KeyLoader.asmx.cs und KeyLoader.asmx	76 (+ 1)	Definition der Web Service-Funktionen
KeyLoader.wsdl	-	Automatisch erzeugt, Web Service Description Language (WSDL)-Beschreibung des Web Services
web.config	(48)	Einstellungen des Webservers
Gesamt	76 (+ 49)	

C.6 WTLClientConfig

Das in Teil III dieser Arbeit beschriebene Konfigurationsprogramm wird durch das C++-Programm *WTLClientConfig* realisiert. Die Dateien dieses Programms werden in Tabelle C.6 aufgelistet.

Tabelle C.6: Das Konfigurationsprogramm WTLClientConfig

Datei	**Codezeilen**	**Inhalt**
import/*.h	-	Aus NSS kopierte Header-Dateien
EmailClientManager.h und EmailClientManager.cpp	65 + 298	Singleton Pattern, Controller im MVC
ErrorHandling.h	6	Makros zur einfachen Fehlerbehandlung
ExecDlg.h und ExecDlg.cpp	41 + 64	Dialog, der während der Konfiguration angezeigt wird
FinalDlg.h und FinalDlg.cpp	33 + 49	Abschließender Dialog
IAppConfigurator.h	34	Interface zur Konfiguration eines Email-Programms, Model im MVC

Fortsetzung auf der nächsten Seite

Tabelle C.6 – Fortsetzung der vorigen Seite

Datei	Codezeilen	Inhalt
IInstallObserver.h	8	Interface des Observer Pattern, der Observer
InstTypeDlg.h und InstTypeDlg.cpp	35 + 44	Dialog zur Auswahl automatischer oder manueller Installation
IntroDlg.h und IntroDlg.cpp	34 + 17	Einleitender Dialog
LateLoad.h	956	Fremderstelltes Makro zum Laden der NSS-DLLs
MainDlg.h und MainDlg.cpp	59 + 40	Dieses Fenster beinhaltet alle Dialoge, View im MVC
MozillaConfigurator.h und MozillaConfigurator.cpp	57 + 732	Realisiert IAppConfigurator, konfiguriert Mozilla Thunderbird
MozillaPrefFile.h und MozillaPrefFile.cpp	21 + 52	Liest und interpretiert die *prefs.js* zur Konfiguration Thunderbirds
MSConfigurator.h und MSConfigurator.cpp	21 + 107	Realisiert IAppConfigurator, ist aber abstrakt. Importiert Schlüssel und Zertifikate in Microsofts Schlüsseldatenbank
netscapeTypes.h	10	Makros zum Import der NSS-Header-Dateien
nspr4.h	13	Erzeugt mit den LateLoad-Makros eine Klasse zum Laden der *nspr4.dll*
nss3.h	40	Erzeugt mit den LateLoad-Makros eine Klasse zum Laden der *nss3.dll*
OEConfigurator.h und OEConfigurator.cpp	34 + 193	Realisiert IAppConfigurator, erbt von MSConfigurator, konfiguriert Outlook Express
OLKConfigurator.h und OLKConfigurator.cpp	50 + 668	Realisiert IAppConfigurator, erbt von MSConfigurator, konfiguriert Outlook 2003
RegistryKey.h und RegistryKey.cpp	37 + 178	Zugriff auf Einträge der Windows-Registry
resource.h	34	Automatisch erzeugt, Konstanten zum Zugriff auf die Dialogfenster
ResourceMgr.h und ResourceMgr.cpp	18 + 67	Singleton, verwaltet die Ressourcen in *WTLClientConfig.rc*
smime3.h	21	Erzeugt mit den LateLoad-Makros eine Klasse zum Laden der *smime3.dll*
ssl3.h	14	Erzeugt mit den LateLoad-Makros eine Klasse zum Laden der *ssl3.dll*
stdafx.h und stdafx.cpp	62 + 9	Erforderlich für bestimmte Compiler-optimierungen
TargetChooseDlg.h und TargetChooseDlg.cpp	39 + 139	Dialog zur Auswahl der konfigurierten Email-Programme
TestData.h	323	Testdaten, die bei der Fehlersuche geladen werden können

Fortsetzung auf der nächsten Seite

Tabelle C.6 – Fortsetzung der vorigen Seite

Datei	**Codezeilen**	**Inhalt**
WizardManager.h und WizardManager.cpp	14 + 23	Singleton zur Verwaltung des Zustands des Konfigurationsprozesses
WSKeyLoaderClient.h und WSKeyLoaderClient.cpp	18 + 218	Teilweise aus älteren Projekten übernommen, regelt den Zugriff auf den Web Service
WTLClientConfig.cpp	53	Hauptprogramm
WTLClientConfig.rc	-	Ressourcen des Programms (Texte, Dialoge, ...)
Gesamt	5048	

D Benutzeranleitung

Ohne zusätzliche Einstellungen verschlüsseln und signieren gängige Email-Programme versendete Emails nicht. Damit haben zumindest die verwendeten Email-Anbieter und Internet-Provider Zugriff auf den Inhalt der Emails. Außerdem kann jeder in fremdem Namen Emails versenden, ohne dass eine Überprüfung möglich ist, ob die Email tatsächlich vom angegebenen Absender stammt. Mit dem System TrustedRoot.org können bei Verwendung der Email-Programme Microsoft Outlook 2003, Microsoft Outlook Express 6 oder Mozilla Thunderbird 1.5/2.0 Emails verschlüsselt und signiert werden. Eine signierte Mail kann nicht in fremdem Namen verschickt werden. Die Verschlüsselung sorgt dafür, dass Emails von niemand anderem als dem richtigen Empfänger gelesen werden können.

D.1 Registrierung

Im ersten Schritt muss die zu verwendende Email-Adresse im System registriert werden. Auf der Internetseite `www.TrustedRoot.org` ist für die Email-Adresse ein Feld vorgesehen, wie in Abbildung D.1 zu sehen ist. Die Eingabe wird mit einem Klick auf „Verify email address“ bestätigt.

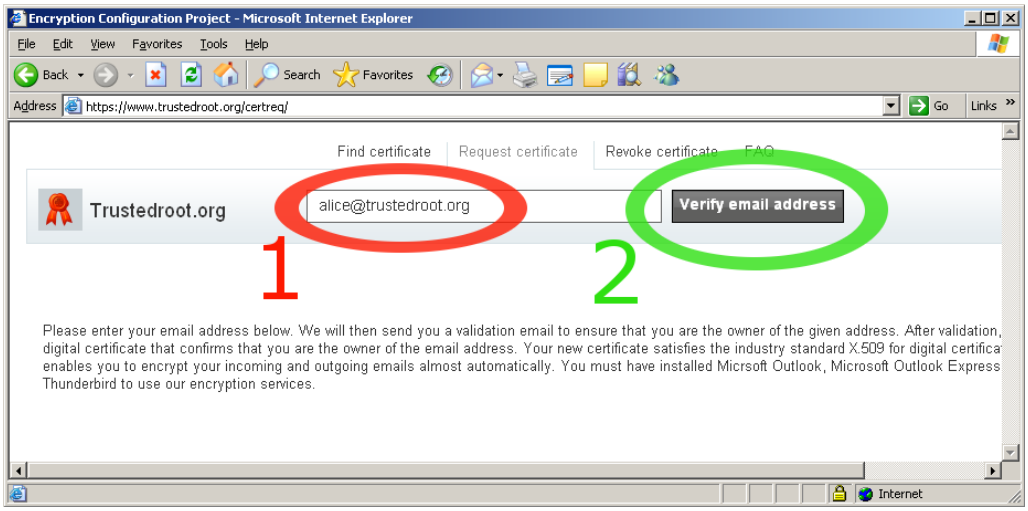

Abbildung D.1: Registrierung auf `www.TrustedRoot.org`

Im Postfach der eingegebenen Email-Adresse befindet sich kurz darauf die in Abbildung D.2 zu sehende Email. Ein Klick auf den enthaltenen Link lädt ein Konfigurationsprogramm herunter, das alle Einstellungen am verwendeten Email-Programm vornimmt. Jeder Browser zeigt eine andere Meldung an, mit der man das Programm starten kann. Im Microsoft Internet Explorer wird die in Abbildung D.3 gezeigte Warnung gemeldet.

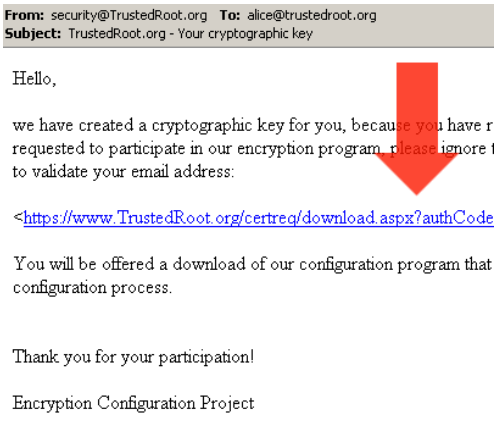

Abbildung D.2: Ausschnitt der Registrierungsemail

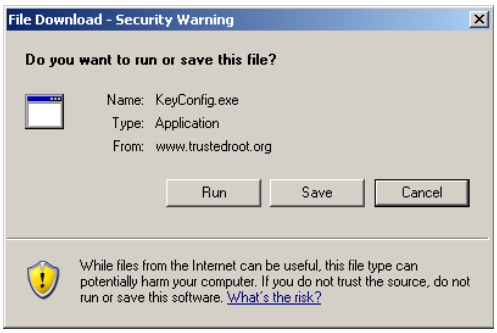

Abbildung D.3: Warnmeldung des Microsoft Internet Explorer

D.2 Konfigurationsprogramm

Das Konfigurationsprogramm ermittelt automatisch alle erforderlichen Einstellungen, die zum Verschlüsseln und Signieren von Emails notwendig sind. Die in der Abbildung D.4 zu sehenden Meldungen können daher mit „Next“ und „Finish“ bestätigt werden. Insbesondere Mozilla Thunderbird sollte aber vor der Installation unbedingt beendet werden.

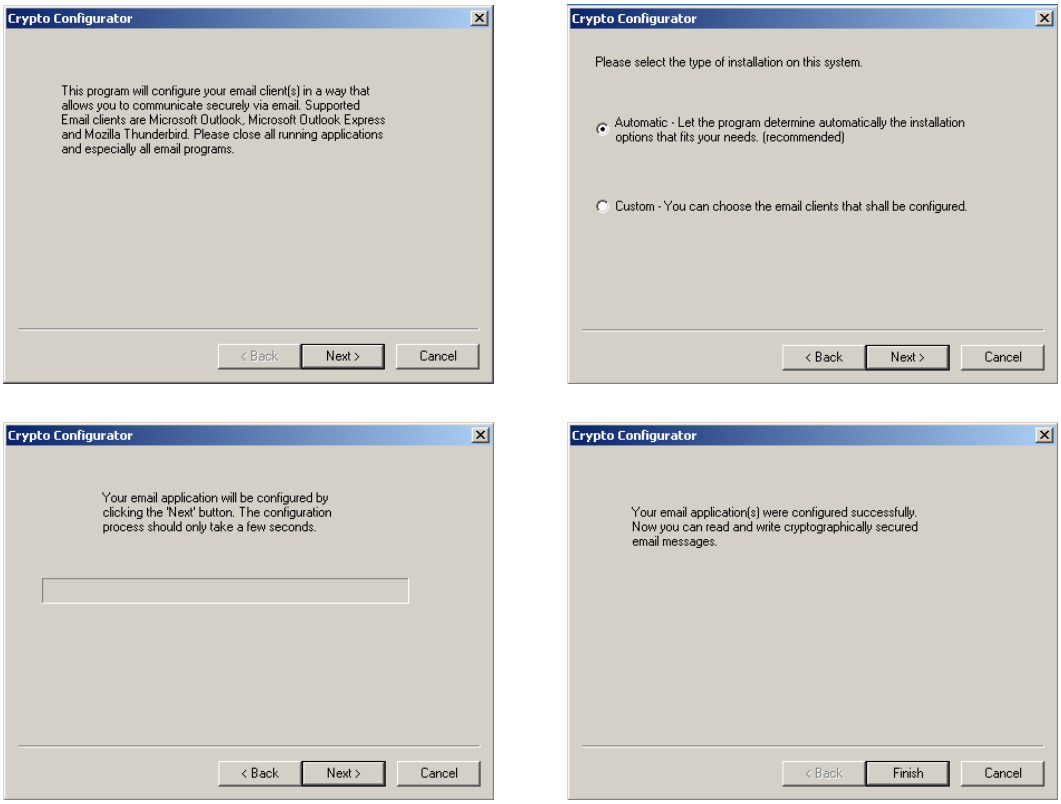

Abbildung D.4: Die Schritte des Konfigurationsprogramms

D.3 Versenden und Empfangen verschlüsselter Emails

Die Email-Programme Microsoft Outlook, Microsoft Outlook Express und Mozilla Thunderbird unterscheiden sich beim Versenden verschlüsselter Emails erheblich, das Empfangen und Entschlüsseln eingehender Emails ist dagegen bei allen Programmen recht einfach. Im Folgenden wird beschrieben, wie sich die drei Email-Programme verhalten, wenn sie durch TrustedRoot.org konfiguriert wurden.

D.3.1 Microsoft Outlook

Bei Microsoft Outlook ändert sich die Benutzbarkeit durch die Verschlüsselungsfunktionen kaum. Wenn der Empfänger einer Email diese entschlüsseln kann, dann verschlüsselt Outlook automatisch und im Hintergrund. Nur wenn eine Email nicht verschlüsselt werden konnte, zum Beispiel weil der Empfänger nicht an einem Kryptographiesystem teilnimmt, erscheint die in Abbildung D.5 dargestellte Meldung. Nach Bestätigung wird die Email in diesem Fall unverschlüsselt, aber signiert gesendet.

Abbildung D.5: Outlook warnt, wenn an den Empfänger nicht verschlüsselt werden kann

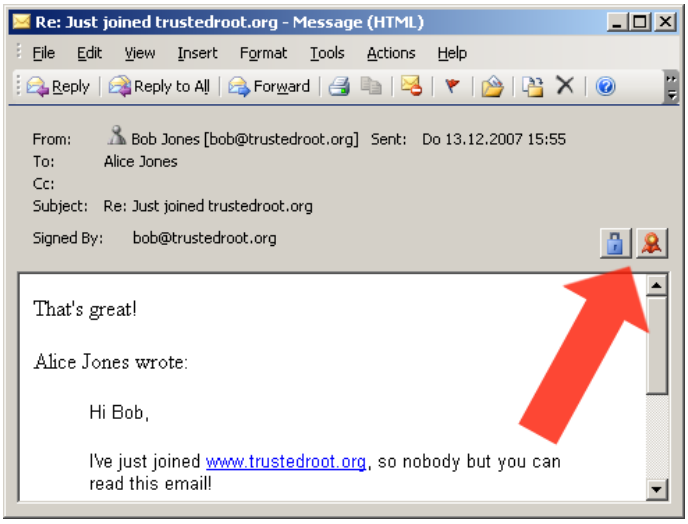

Abbildung D.6: Die zwei markierten Symbole einer eingehenden Email zeigen an, dass die Email verschlüsselt und signiert ist

Bei eingehenden Emails zeigt Outlook mit jeweils einem Symbol an, ob die Email verschlüsselt oder signiert ist. Eine verschlüsselte und signierte Email ist in Abbildung D.6 zu sehen.

D.3.2 Microsoft Outlook Express

In Microsoft Outlook Express werden ausgehende Emails nach der Konfiguration immer signiert. Zur Verschlüsselung von Emails wird aber das Zertifikat des Empfängers benötigt. Dieses ist nach früherer verschlüsselter Kommunikation mit dem Empfänger möglicherweise schon gespeichert, ansonsten muss das Zertifikat erst aus dem Internet heruntergeladen werden. Dazu darf die Email-Adresse nicht wie üblich sofort in das Empfängerfeld eingetippt werden, sondern sie muss über einen Klick auf den Schalter „To:“ neben dem Empfängerfeld und dann auf den Schalter „Find...“ gesucht werden, wie in Abbildung D.7 zu sehen ist. Es öffnet sich ein Dialog, in dem eine Suche im „TrustedRoot.org LDAP Server“ gestartet werden kann. Wenn die Email-Adresse des Empfängers in das Feld „E-mail“ eingetragen wurde, kann die Suche mit „Find Now“ durchgeführt werden. Das Fenster erweitert sich um einen unteren Teil, in dem die gefundene Email-Adresse aufgelistet ist. Mit einem Klick auf „To: ->“ wird die Adresse in das Empfängerfeld der Email kopiert, woraufhin die Email verschlüsselt versendet werden kann.

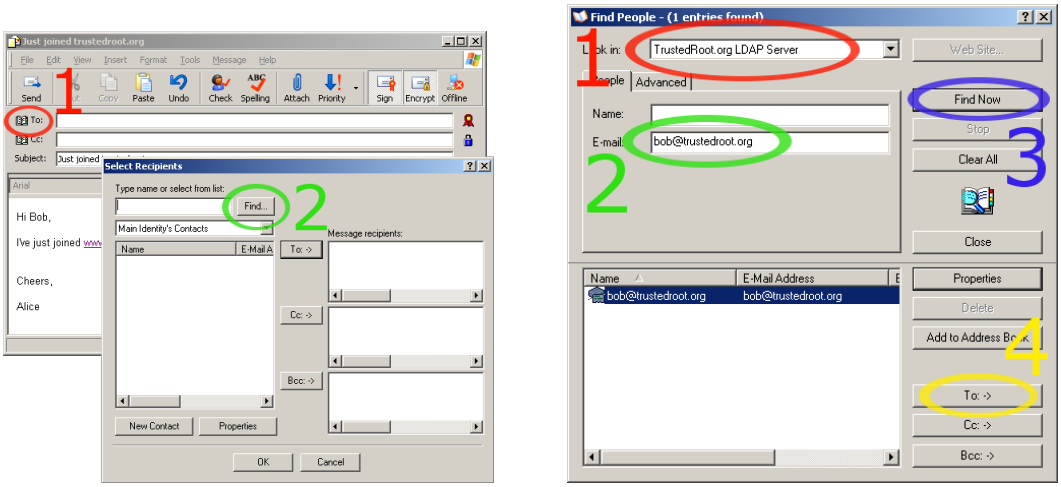

Abbildung D.7: Die notwendigen Schritte, um mit Microsoft Outlook Express eine verschlüsselte Email zu versenden.

Die erste verschlüsselte Nachricht an einen Empfänger wird in Outlook Express nur schwach verschlüsselt, Outlook Express gibt hierzu jedoch auch einen entsprechenden Hinweis aus.

Beim Empfang einer signierten oder verschlüsselten Nachricht erklärt Outlook Express dem Benutzer die Bedeutung von Signaturen und Verschlüsselung, wie in Abbildung D.8 zu sehen ist. Die beiden markierten Symbole weisen darauf hin, dass die eingehende Nachricht signiert und verschlüsselt ist.

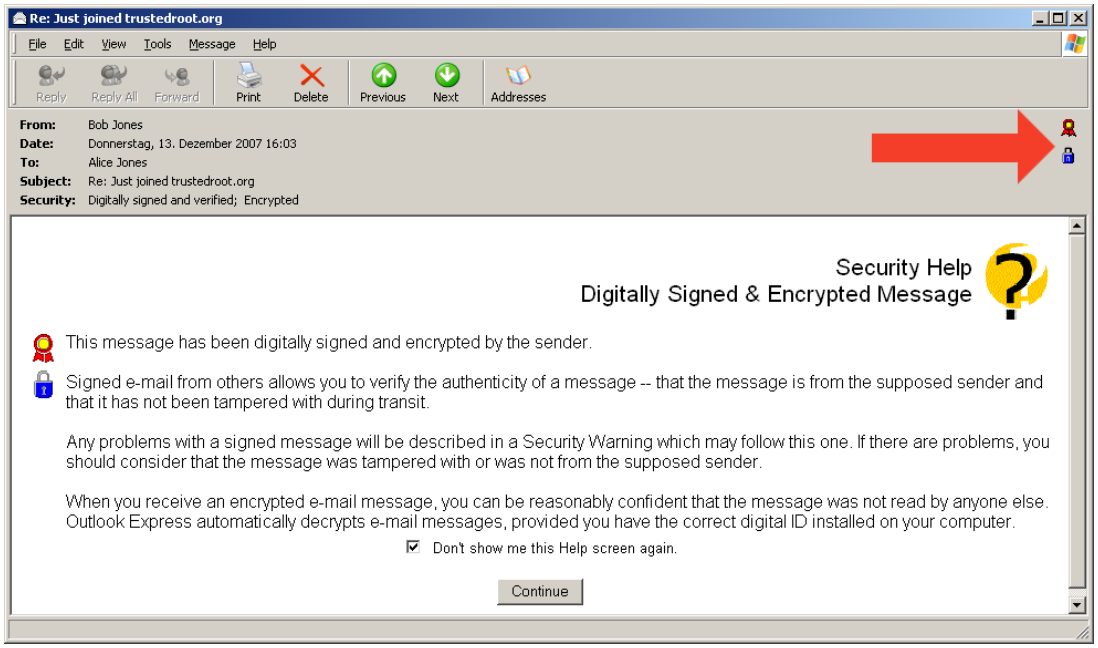

Abbildung D.8: Microsoft Outlook Express informiert über Signaturen und Verschlüsselung

D.3.3 Mozilla Thunderbird

Auch Mozilla Thunderbird signiert automatisch und ohne Zutun des Nutzers alle ausgehenden Emails. Soll eine Email zusätzlich noch verschlüsselt werden, öffnet ein Klick auf den Pfeil rechts neben dem Symbol „Security“ das in Abbildung D.9 gezeigte Kontextmenü. In diesem Menü muss „Encrypt This Message“ ausgewählt werden. Die Email wird dann beim Versenden automatisch verschlüsselt.

Auch in Thunderbird weisen zwei Symbole in eingehenden Emails auf Signatur und Verschlüsselung hin, wie in Abbildung D.10 zu sehen ist.

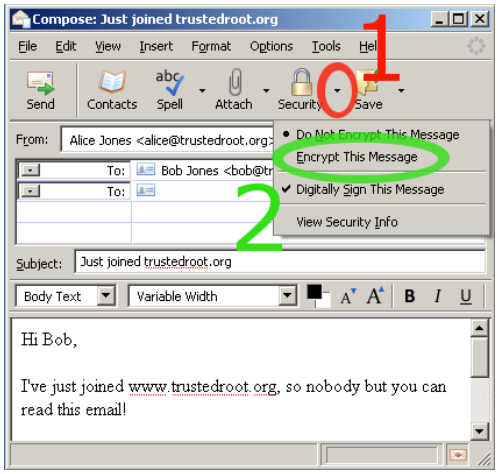

Abbildung D.9: Versenden verschlüsselter Emails in Mozilla Thunderbird

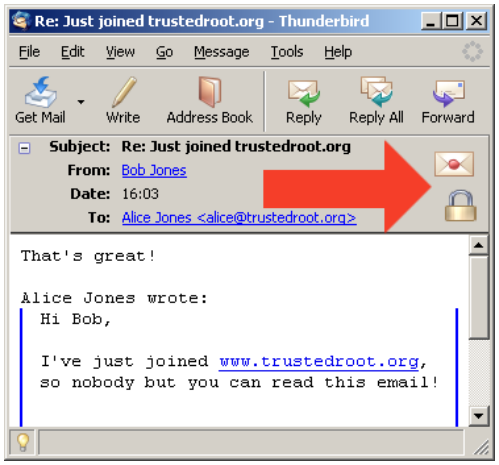

Abbildung D.10: Der Pfeil zeigt auf die Symbole für Signatur und Verschlüsselung

Glossar

Backend

Server-Komponente, die keine direkte Verbindung zu den Benutzern hat (etwa über HTTP, LDAP, oder ähnliches), sondern nur mit anderen Server-Komponenten kommuniziert.

Base64

Ein ursprünglich namenloses Verfahren, mit dem Binärdaten kodiert werden. Die kodierten Daten enthalten nur 64 verschiedene Zeichen, die so gewählt sind, dass sie beispielsweise in Emails oder auf Webseiten dargestellt werden können. Durch die Base64-Kodierung vergrößert sich die Datenmenge um ein Drittel.

Build

Alle Operationen, die durchgeführt werden müssen, um aus dem Quelltext eines Programms das benutzbare Programm in Maschinensprache zu erzeugen. Dazu gehört insbesondere das Übersetzen und Linken, aber auch das Ausführen etwaiger Skripte.

Certification Authority

Certification Authority oder kurz CA ist der Name einer Instanz, die digitale Schlüssel zusammen mit Informationen über seinen Besitzer kryptographisch signiert und damit zertifiziert, dass Schlüssel und Besitzer zusammengehören. Im X.509-Kontext ist jedes Zertifikat von genau einer CA ausgestellt, bei OpenPGP kann es auch mehrere CAs für einen Schlüssel geben. Eine CA, die ihr eigenes Zertifikat ausgestellt hat, nennt man Root CA.

CryptoAPI

Eine Bibliothek mit kryptographischen Funktionen wird in verschiedenen Betriebssystemen zur Verfügung gestellt. In dieser Arbeit bezieht sich der Begriff CryptoAPI auf die in Microsoft Windows vorhandene kryptographische Bibliothek und die durch sie erreichbare Schlüsseldatenbank.

Design Pattern

In der Softwareentwicklung nennt man formalisierte, wiederverwendbare Lösungswege für Probleme Design Patterns. Es gibt Kataloge, die Design Patterns für verschiedene Problembereiche enthalten, es gibt aber auch Design Patterns für allgemeine Probleme.

DLL

Eine Dynamic Link Library, kurz DLL, ist eine in Windows benutzbare Programmbibliothek, die Klassen und Funktionen enthalten kann. Sie heißt dynamisch, weil sie nicht fester Bestandteil des Programmcodes ist, sondern erst beim Starten des Programms oder zur Laufzeit in den Speicher geladen wird. Je nach

Programm und Bibliothek wird die Bibliothek auch nur einmal in den Speicher geladen, obwohl mehrere Programme auf sie zugreifen.

GUI

Akronym für Graphical User Interface, beschreibt eine mit der Maus zu bedienende Oberfläche eines Programms.

GUID

Global Unique Identifier oder kurz GUIDs sind 128-Bit-Zahlen, die nach ihrer Erzeugung weltweit eindeutig sind. Diese Eindeutigkeit wird durch die enorme Größe des Zahlenraums sichergestellt.

Interface

In der Programmierung ist ein Interface die Menge der Funktionsdeklarationen einer Klasse. In verschiedenen Programmiersprachen ist ein Interface genauer eine abstrakte Klasse ohne konkrete Implementierung in Quelltext, es werden also nur die Funktionsköpfe angegeben.

JavaScript

Scriptsprache, die in ihrer Syntax der Programmiersprache Java ähnlich ist. JavaScript wird von den meisten Web-Browsern unterstützt und kann in einer Webseite definiert und ausgeführt werden. Ein JavaScript ist aber zum Beispiel auch direkt unter Windows ausführbar.

Managed Code

Wenn eine Programmierumgebung eine Garbage Collection bietet, der Programmierer sich also nicht um die Speicherfreigabe kümmern muss, so nennt man diesen Bereich des Programms Managed Code. Umgekehrt heißt ein Programmbereich ohne Garbage Collection Unmanaged Code. Meist bestimmt sich schon durch die verwendete Programmiersprache, ob Managed Code (C#, Visual Basic 6.0, Visual Basic .NET, Java) oder Unmanaged Code (PASCAL, C) entsteht. In C++ gibt es eigentlich keine Garbage Collection, in der Microsoft-Spracherweiterung Managed C++, auch bezeichnet als C++ mit .NET Extensions, wird der Sprache neben den übrigen Features des .NET Framework auch eine Garbage Collection hinzugefügt.

Parsen

Das Analysieren eines Textes auf bestimmte Merkmale hin wird als Parsen bezeichnet.

Quoting

Als Quoting wird eine Umklammerung (eventuell mit Anführungsstrichen) eines Ausdrucks in einer Sprache bezeichnet. Diese Umklammerung kennzeichnet, dass dieser Ausdruck nicht weiter interpretiert werden soll. Hierdurch ist es möglich, auch Ausdrücke zu verwenden, die reservierte Zeichen oder reservierte Wörter enthalten.

Reservierter Ausdruck

Ein reservierter Ausdruck ist ein Wort oder Zeichen, das in einer Sprache als Steuersymbol einer Kontrollstruktur verwendet wird. Man kann diese Ausdrücke in der Sprache nicht (ohne Weiteres) für andere Zwecke als ihre Steuerfunktion benutzen. In manchen Sprachen lässt sich das Symbol mit einem Mechanismus wie Quoting doch noch verwenden.

Smartcard

Ursprünglich bezeichnete Smartcard eine Chipkarte, deren Chip eigenständige Berechnungen durchführen kann, also nicht nur ein reiner Datenträger ist. Dies erlaubt nicht nur die Speicherung privater Schlüssel auf der Smartcard, sondern auch die Ausführung kryptographischer Operationen wie Entschlüsselungen und Signaturen. So kann man einen privaten Schlüssel auf einer Smartcard generieren und verwenden, ohne dass dieser jemals die Smartcard verlässt. In jüngerer Zeit werden Universal Serial Bus (USB)-Token mit den gleichen Funktionen eingesetzt, gegenüber Smartcards hat man den Vorteil, dass man keinen Kartenleser braucht. Diese USB-Tokens werden manchmal ebenfalls als Smartcard bezeichnet.

Store/Schlüsseldatenbank

Store oder auch Key Store bezeichnet eine üblicherweise lokale Datenbank zur Speicherung von kryptographischem Schlüsselmaterial und digitalen Zertifikaten. Viele kryptographische Programme benutzen proprietäre Formate für ihre Stores, einige Formate wie PGP Keyrings sind auch standardisiert.

Web-Applikation

Eine Web-Applikation ist eine Anwendung, die auf einem Web-Server läuft, und über eine HTTP-Schnittstelle gesteuert wird. Sie unterscheidet sich von einer normalen Anwendung also durch die Art ihrer Steuerung. Von einer Web-Seite unterscheidet sie sich durch ihre höhere Komplexität und die dynamischen Inhalte. Eine Web-Applikation stellt nicht nur Informationen dar, sondern interagiert mit dem Benutzer.

X.509-Zertifikat

Ein Zertifikat ist im Kontext der Kryptographie ein Datenpaket mit digitaler Signatur. Ein X.509-Zertifikat wurde nach dem Standard X.509 erzeugt; das Datenpaket besteht aus einem öffentlichen Schlüssel und Informationen über den Besitzer, etwa seine Email-Adresse. Eine CA erzeugt die Signatur und bezeugt damit, dass der öffentliche Schlüssel und die Besitzerinformationen auch tatsächlich zusammengehören.

Abkürzungsverzeichnis

API
: Application Programming Interface

ASN.1
: Abstract Syntax Notation One

ASP .NET
: Active Server Pages .NET

ATL
: Active Template Library

BASH
: Bourne Again Shell

BLOB
: Binary Large Object

CA
: Certification Authority

CRL
: Certificate Revocation List

CSS
: Cascading Style Sheet

DAP
: X.500 Directory Access Protocol

DER
: Distinguished Encoding Rules

DLL
: Dynamic Link Library

DN
: Distinguished Name

DNS
: Domain Name System

DSA
: Digital Signature Algorithm

DSE

DSA-specific Entry

ECDSA

Elliptic Curve Digital Signature Algorithm

FAQ

Frequently Asked Questions

FTP

File Transfer Protocol

GNU

GNU is not Unix

GPG

GNU Privacy Guard

GUI

Graphical User Interface

GUID

Global Unique Identifier

HTTP

Hypertext Transfer Protocol

IETF

Internet Engineering Taskforce

IIS

Microsoft Internet Information Services

IMAP

Internet Message Access Protocol

IP

Internet Protocol

ITU

International Telecommunications Union

ITU-T

ITU Telecommunication Standardization Sector

KCM

Key Continuity Management

LDAP

Lightweight Directory Access Protocol

MFC
: Microsoft Foundation Classes

MIME
: Multipurpose Internet Mail Extensions

MinGW
: Minimalist GNU for Windows

MSDN
: Microsoft Developer Network

MSYS
: Minimal System

MVC
: Model/View/Controller

NIST
: National Institute of Standards and Technology

NSS
: Network Security Services

OCSP
: Online Certificate Status Protocol

ODBC
: Open Database Connectivity

OID
: Object Identifier

PEM
: Privacy Enhanced Mail

PGP
: Pretty Good Privacy

PKCS
: Public Key Cryptography Standards

PKI
: Public Key-Infrastruktur oder Public Key Infrastructure

PKIX
: PKI (X.509)

RDN
: Relative Distinguished Name

RFC
: Request for Comments

SLAPD
: stand-alone LDAP daemon

SLURPD
: Stand-Alone LDAP Update Replication Daemon

S/MIME
: Secure Multipurpose Internet Mail Extensions

SMTP
: Simple Mail Transfer Protocol

SOAP
: Simple Object Access Protocol

SPKI
: Simple PKI

SQL
: Structured Query Language

SSL
: Secure Sockets Layer

TCP/IP
: Transmission Control Protocol/Internet Protocol

TLS
: Transport Layer Security

URI
: Uniform Resource Identifier

WSDL
: Web Service Description Language

WTL
: Windows Template Library

W3C
: World Wide Web Consortium

XKMS
: XML Key Management Specification

XML
: Extended Markup Language

Literaturverzeichnis

[AL99] Carlisle Adams und Steve Lloyd, *Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations*, Technology Series, Macmillan Technical Publishing, 1999.

[ASZ96] D. Atkins, W. Stallings und P. Zimmermann, *PGP Message Exchange Formats*, RFC 1991, Internet Engineering Taskforce, August 1996.

[BBB+07] Elaine Barker, William Barker, William Burr, William Polk und Miles Smid, *Recommendation for Key Management - Part 1: General (Revised)*, NIST Special Publication 800-57, National Institute of Standards and Technology, März 2007.

[BGB06] *Bürgerliches Gesetzbuch*, Dezember 2006.

[BM] Lucas Bergman und Matthias Mohr, *OpenLDAP for Win32*, `http://lucas.bergmans.us/hacks/openldap/`, Zugriff am 2007-01-04.

[Buc99] Johannes Buchmann, *Einführung in die Kryptographie*, Springer-Verlag, 1999.

[CC] Achraf Cherti und Maximilien Cuony, *Webseite des FireGPG-Plugins*, `http://firegpg.tuxfamily.org/`, Zugriff am 2007-11-09.

[CCMW01] Erik Christensen, Francisco Curbera, Greg Meredith und Sanjiva Weerawarana, *Web Services Description Language (WSDL)*, W3C Recommendation, World Wide Web Consortium, März 2001.

[CDF+07] Jon Callas, Lutz Donnerhacke, Hal Finney, D. Shaw und Rodney Thayer, *OpenPGP Message Format*, RFC 2440, Internet Engineering Taskforce, November 2007.

[CDFT98] Jon Callas, Lutz Donnerhacke, Hal Finney und Rodney Thayer, *OpenPGP Message Format*, RFC 2440, Internet Engineering Taskforce, November 1998.

[Cyg] *Webseite der Cygwin-Plattform*, `http://www.cygwin.com/`, Zugriff am 2007-12-04.

[DA04] Jason De Arte, *LateLoad DLL Wrapper*, `http://www.codeproject.com/KB/DLL/LateLoad.aspx`, März 2004, Zugriff am 2007-12-06.

[DH76] W. Diffie und M. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), no. 6, 644–654.

[Dra67] Alvin W. Drake, *Fundamentals of applied probability theory*, McGraw-Hill Series in Probability and Statistics, McGraw-Hill Book Company, 1967.

[Dun06] Michael Dunn, *WTL for MFC Programmers*, The Code Project (2006), `http://www.codeproject.com/KB/wtl/wtl4mfc1.aspx`, Zugriff am 2007-12-06.

[EG99] D. Eastlake und O. Gudmundsson, *Storing Certificates in the Domain Name System (DNS)*, RFC 2538, Internet Engineering Taskforce, März 1999.

[Elg85] Taher Elgamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, Proceedings of CRYPTO 84 on Advances in cryptology (New York, NY, USA), Springer-Verlag New York, Inc., 1985, pp. 10–18.

[FF04] Eric Freeman und Elisabeth Freeman, *Design Patterns*, Head First, O'Reilly, Oktober 2004.

[FHBF+01] Warwick Ford, Phillip Hallam-Baker, Barbara Fox, Blair Dillaway, Brian LaMacchia, Jeremy Epstein und Joe Lapp, *XML Key Management Specification (XKMS)*, W3C Note, World Wide Web Consortium, März 2001.

[fre] *Webseite des Plugins freenigma*, `http://www.freenigma.com/`, Zugriff am 2007-12-04.

[Gal99] Michael Galkovsky, *DLLs the Dynamic Way*, MSDN Library, Microsoft Corporation, November 1999.

[Gar95] Simson Garfinkel, *PGP*, O'Reilly, 1995.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides, *Design Patterns*, Professional Computing Series, Addison-Wesley, 1995.

[GHM+07] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar und Yves Lafon, *SOAP Version 1.2*, W3C Recommendation, World Wide Web consortium, April 2007.

[GM05] Simson Garfinkel und Robert Miller, *Johnny 2: a user test of key continuity management with S/MIME and Outlook Express*, SOUPS '05: Proceedings of the 2005 Symposium on Usable Privacy and Security (New York, NY, USA), ACM Press, 2005, pp. 13–24.

[GNU] *GNU File Server*, `http://ftp.gnu.org/old-gnu/`, Zugriff am 2007-12-04.

[GPG] *Webseite des Gnu Privacy Guard Projekts*, `http://www.gnupg.org/`, Zugriff am 2007-12-04.

[Gut03] Peter Gutmann, *Plug-and-Play PKI: A PKI Your Mother Can Use*, 12th USENIX Security Symposium, 2003, `http://www.cs.auckland.ac.nz/~pgut001/pubs/usenix03.pdf`.

[Gut04] ———, *Why isn't the internet secure yet, dammit*, Computer Security: Are we there yet?, AusCERT Asia Pacific Information Technology Security Conference, 2004, `http://www.cs.auckland.ac.nz/~pgut001/pubs/dammit.pdf`.

[Gut06] ———, *Internet X.509 Public Key Infrastructure Operational Protocols: Certificate Store Access via HTTP*, RFC 4387, Internet Engineering Taskforce, Februar 2006.

[HH99] R. Housley und P. Hoffman, *Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP*, RFC 2585, Internet Engineering Taskforce, May 1999.

[Hor95] Marc Horowitz, *A PGP Public Key Server*, Undergraduate thesis, Massachusetts Institute of Technology, 1995, `http://www.mit.edu/people/marc/pks/pks.html`.

[Hou04] R. Housley, *Cryptographic Message Syntax (CMS)*, RFC 3852, Internet Engineering Taskforce, Juli 2004.

[Hou07] ———, *Cryptographic Message Syntax (CMS) Multiple Signer Clarification*, RFC 4853, Internet Engineering Taskforce, April 2007.

[HPFS02] R. Housley, W. Polk, W. Ford und D. Solo, *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, RFC 3280, Internet Engineering Taskforce, April 2002.

[Husa] *Webseite des Hushmail Webangebots*, `http://www.hushmail.com/`, Zugriff am 2007-11-21.

[Husb] Hush Communications, *Webmail using the hush encryption engine*, `https://www.hushmail.com/public_documents/Webmail%20Using%20the%20Hush%20Encryption%20Engine.pdf`, Zugriff am 2007-11-21.

[HY97] Michael Halvorson und Michael Young, *Microsoft Office 97 Professionell nutzen*, Microsoft Press, 1997.

[Ins] *Inside Outlook Express - OE Registry Keys*, `http://www.insideoe.com/files/regkeys.htm`, Zugriff am 2007-12-04.

[Jos06] S. Josefsson, *The Base16, Base32, and Base64 Data Encodings*, RFC 4648, Internet Engineering Taskforce, Oktober 2006.

[kes06] *Lagebericht zur Informationssicherheit*, <kes> **6** (2006), no. 2, 48–54.

[Lin93] J. Linn, *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*, RFC 1421, Internet Engineering Taskforce, Februar 1993.

[LMS05] P. Leach, M. Mealling und R. Salz, *A Universally Unique IDentifier (UUID) URN Namespace*, RFC 4122, Internet Engineering Taskforce, July 2005.

[MAM+99] M. Myers, R. Ankney, A. Malpani, S. Galperin und C. Adams, *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*, RFC 2560, Internet Engineering Taskforce, Juni 1999.

[Mas06] Pierangelo Masarati, *Eintrag im OpenLDAP-Forum: slapd-sql with unixodbc - userCertificate retrieval problem*, `http://www.openldap.org/lists/openldap-software/200606/msg00058.html`, Juni 2006, Zugriff am 2007-12-04.

[McW06] James McWhinney, *Adding an LDAP address book to MS Outlook*, The Code Project (2006), `http://www.codeproject.com/KB/vbscript/Add_ldap_addrbook.aspx`, Zugriff am 2007-12-06.

[Mica] *Where is Outlook Express?*, Windows help and how-to, Microsoft Corporation, `http://windowshelp.microsoft.com/Windows/en-US/Help/49b9f100-92fd-418c-aa04-9c0d575a19a31033.mspx`, Zugriff am 2007-12-04.

[Micb] *Windows Server 2003 R2 Pricing*, `http://www.microsoft.com/windowsserver2003/howtobuy/licensing/pricing.mspx`, Zugriff am 2007-12-04.

[Mic05] Microsoft Corporation, *MSDN Library*, October 2005.

[Mic07] *Differences between Outlook and Outlook Express*, Knowledge Base Article 257824, Microsoft Corporation, `http://support.microsoft.com/kb/257824`, Mai 2007, Zugriff am 2007-12-04.

[Moza] *RSS Feed mit den gezählten Thunderbird-Downloads*, `http://feeds.spreadfirefox.com/downloads/thunderbird.xml`, Zugriff am 2007-11-30.

[Mozb] *Webseite des Mozilla Project*, `http://www.mozilla.org`, Zugriff am 2007-11-27.

[Mozc] Mozilla Foundation, *Thunderbird FAQ*, `http://www.mozilla.org/support/thunderbird/faq`, Zugriff am 2007-12-04.

[moz07a] *Profile Manager*, mozillaZine, Oktober 2007, `http://kb.mozillazine.org/Profile_Manager`, Zugriff am 2007-12-04.

[Moz07b] Mozilla Foundation, *Network Security Services (NSS)*, Juli 2007, `http://www.mozilla.org/projects/security/pki/nss/`, Zugriff am 2007-12-04.

[NIS94] *Digital Signature Standard (DSS)*, FIPS PUB 186, National Institute of Standards and Technology, Mai 1994.

[Opea] *Webseite des OpenSSL-Projekts*, `http://www.openssl.org/`, Zugriff am 2007-12-04.

[Opeb] *Webseite des OpenLDAP-Projekts*, `http://www.OpenLDAP.org/`, Zugriff am 2007-12-04.

[Ope07] The OpenLDAP Project, *OpenLDAP 2.3 Admin Guide*, Januar 2007, `http://www.openldap.org/doc/admin23/`, Zugriff am 2007-09-28.

[Pet01] Colin Peters, *Programming Win32 with GNU C and C++*, `http://www.emmestech.com/software/cygwin/pexports-0.43/download_tutorial.html`, May 2001, Zugriff am 2007-12-04.

[PGP] *Webseite der PGP Corporation*, `http://www.pgp.com/`, 2007-12-04.

[Ram04] B. Ramsdell, *Secure/multipurpose Internet mail extensions (S/MIME) version 3.1 message specification*, RFC 3851, Internet Engineering Taskforce, Juli 2004.

[RC07] Mark Russinovich und Bryce Cogswell, *Windows Sysinternals*, `http://www.microsoft.com/technet/sysinternals`, Mai 2007, Zugriff am 2007-12-04.

[Res00] E. Rescorla, *HTTP Over TLS*, RFC 2818, Internet Engineering Taskforce, Mai 2000.

[Res01] P. Resnick, *Internet Message Format*, RFC 2822, Internet Engineering Taskforce, April 2001.

[RSA] *The RSA Factoring Challenge*, `http://www.rsa.com/rsalabs/node.asp?id=2092`, Zugriff am 2007-12-02.

[RSA78] R. L. Rivest, A. Shamir und L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems*, Communications of the ACM **21** (1978), no. 2, 120–126.

[RSA93] *PKCS 7: Cryptographic Message Syntax Standard*, Tech. Report Version 1.5, RSA Laboratories, November 1993.

[RSA99] *PKCS 12 v1.0: Personal Information Exchange Syntax*, Tech. report, RSA Laboratories, Juni 1999.

[RSA00] *PKCS 10 v1.7: Certification Request Syntax Standard*, Tech. report, RSA Laboratories, Mai 2000.

[Sch96] Bruce Schneier, *Angewandte Kryptographie*, vol. 1, Addison-Wesley, 1996.

[Sch06] Mark Schoenberg, *A Moron's Guide to Using Microsoft DLL's when Compiling Cygwin or Mingw Programs*, `http://www.emmestech.com/software/cygwin/pexports-0.43/moron1.html`, May 2006, Zugriff am 2007-12-04.

[Sig07] *Gesetz über Rahmenbedingungen für elektronische Signaturen*, Februar 2007.

[WHK97] M. Wahl, T. Howes und S. Kille, *Lightweight Directory Access Protocol (v3)*, RFC 2251, Internet Engineering Taskforce, Dezember 1997.

[Wik07] *Name mangling*, Wikipedia, Januar 2007, `http://en.wikipedia.org/wiki/Name_mangling`, Zugriff am 2007-12-04.

[WT99] Alma Whitten und J. D. Tygar, *Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0*, Proceedings of the 8th USENIX Security Symposium, USENIX Association, 1999, pp. 169–184.

[X.505a] *Information technology - Open Systems Interconnection - The Directory: Overview of concepts, models and services ITU-T Recommendation X.500*, ITU-T Recommendation X.500, International Telecommunication Union, August 2005.

[X.505b] *Information technology - Open Systems Interconnection - The Directory: Public-Key and attribute certificate frameworks*, ITU-T Recommendation X.509, International Telecommunication Union, August 2005.

[X.602] *Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*, ITU-T Recommendation X.690, International Telecommunication Union, Juli 2002.

[Yod04] S. Yoder, *Setting up LDAP with back-sql*, `http://www.flatmtn.com/computer/Linux-LDAP.html`, August 2004, Zugriff am 2007-12-04.

[Zei06] K. Zeilenga, *Lightweight Directory Access Protocol (LDAP) Schema Definitions for X.509 Certificates*, RFC 4523, Internet Engineering Taskforce, Juni 2006.

Eidesstattliche Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Zitate und dem Sinne nach anderen Quellen entnommene Textpassagen sind entsprechend gekennzeichnet. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

_____, den _____

_____

Christoph Hannebauer