Fail-Safe-Konzept für Public-Key-Infrastrukturen

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades
Doctor rerum naturalium (Dr. rer. nat.)

von
Dipl.-Math. Jan Sönke Maseberg
aus Bremen

Referenten:	Prof. Dr. J. Buchmann
	Prof. Dr. C. Eckert
Tag der Einreichung:	29. Mai 2002
Tag der mündlichen Prüfung:	30. Juli 2002

Darmstadt 2002
D17

An dieser Stelle möchte ich Herrn Professor Johannes Buchmann herzlich dafür danken, dass ich an seinem Lehrstuhl diese Arbeit erstellen konnte. Er nahm sich stets die Zeit, meine Arbeit durch Ratschläge, Diskussionen und Verbesserungsvorschläge zu unterstützen.

Ferner möchte ich Frau Professor Claudia Eckert für die Übernahme des Koreferats und die anregenden Diskussionen danken.

Herr Bruno Struif hat mich auf die Themenstellung aufmerksam gemacht und zahlreiche Anregungen gegeben. Ich danke ihm herzlich für seine Hilfe.

Mein besonderer Dank gebührt dem Fraunhofer-Institut für Sichere Telekooperation (SIT), das mir diese Arbeit ermöglichte. Ferner möchte ich den Lehrstuhl von Professor Buchmann an der Technischen Universität Darmstadt und das „FairPay“-Projekt erwähnen, in dessen Rahmen wesentliche Teile der Arbeit entstanden.

Wenke bin ich dankbar für ihre moralische und fachliche Unterstützung zu dieser Arbeit.

Nicht zuletzt für ihr sorgfältiges Korrekturlesen möchte ich an dieser Stelle Brigitte und Bernhard Wuwer danken.

Schließlich danke ich besonders herzlich meinen lieben Freundinnen und Freunden, Kolleginnen und Kollegen von SIT und der TU, die mir durch die stete Bereitschaft zu Diskussionen und durch viele konstruktive Vorschläge sehr geholfen haben; namentlich erwähnen möchte ich Michael Hartmann, Igor Kalenderian, Markus Tak, Markus Ruppert, Bernd Baumgarten, Carsten Kunz, Joachim Nadler, Matthias Bormann, Jürgen Repp und Carsten Rudolph.

Zusammenfassung

Public-Key-Infrastrukturen sind von zentraler Bedeutung für eine sichere elektronische Kommunikation in offenen Netzen. Unternehmen, Behörden und Privatleute nutzen Anwendungen mit dieser Technik in zunehmendem Maße und vertrauen auf ihre Zuverlässigkeit. Public-Key-Infrastrukturen bergen aber auch Risiken, weil die Verfahren der zugrunde liegenden Public-Key-Kryptographie nicht beweisbar sicher sind und weil Implementierungsfehler nie ausgeschlossen werden können. Ein bestehender Mechanismus, um im Schadensfall die Sicherheit einer Infrastruktur wiederherzustellen, ist, die vom Schaden betroffenen Bereiche zu sperren, was je nach Ausmaß des Schadens den Ausfall der Funktionsfähigkeit einer – auch wichtigen – Anwendung verursachen und damit beträchtliche wirtschaftliche und gesellschaftliche Folgen nach sich ziehen kann. Heutige Public-Key-Infrastrukturen arbeiten meist mit fest installierten kryptographischen Verfahren, so dass es zur Wiederherstellung ihrer vollen Funktionsfähigkeit erforderlich sein kann, Komponenten neu zu entwickeln, zu produzieren und zu verteilen. Daneben könnte ein Schaden die Beweiskraft digitaler Signaturen verletzen und den Verlust der Vertraulichkeit verschlüsselter Daten implizieren. Zur Lösung der geschilderten Probleme wird in dieser Dissertation ein Fail-Safe-Konzept für Public-Key-Infrastrukturen vorgestellt.

Abstract

Public key infrastructures (PKIs) play an important role in secure electronic communication via the internet. More and more governments, companies and customers use this technology and trust in the provided security, but PKIs have risks. On the one hand the mathematical problems on the hardness of which public key cryptography depends are not provably hard, on the other hand failures in implementations can happen. A mechanism to restore the security of an infrastructure is to revoke damaged areas. Depending on the extent a breakdown would restrict the functionality of the PKI which would imply costly consequences for business and society. To maintain the PKI, new components would be developed, produced, distributed, and installed which would cost time and money. Furthermore a failure would imply the loss of the provability of digital signatures and of the confidence in sensitive data. In this thesis the described problems will be solved by a fail safe concept for public key infrastructures.

Inhaltsverzeichnis

Einleitung

> „Geheimer Kreditkartenschlüssel im Internet veröffentlicht – Der Traum von der ‘Yescard‘
> Der Traum vom unbeschwerten Leben hat für Tüftler und Computerfreaks einen Namen, er heißt ‘Yescard‘. Das ist die Kreditkarte, die immer ‚Ja‘ sagt. Ganz gleich, ob Geld auf dem Konto ist, oder ob überhaupt ein Konto existiert. Der Automat sagt ganz einfach und immer ‚Ja‘, er zahlt die Rechnung im Restaurant oder an der Tankstelle.“ [RZ]

Was war passiert? Im März 2000 wurde im Internet ein Code veröffentlicht, mit dem sich französische Kreditkarten nachmachen lassen. Die “Cartes Bancaires“ sind – anders als die Magnetstreifenkarten in Deutschland – Chipkarten, auf denen geheime Schlüssel abgespeichert sind und kryptographische Verfahren ablaufen. Zum Fälschen einer “Yescard“ werden ein Kartenleser, zwei Cartes Bancaires und ein Chipkarten-Rohling benötigt. Die so erzeugte Karte verweist auf ein nicht existierendes Konto und funktioniert deshalb nur bei Offline-Terminals. Geschädigt wird die Bank. Stehen aber zusätzliche reale Kreditkartendaten zur Verfügung, so wie sie auf weggeworfenen Belegen zu finden sind, dann kann eine existierende Karte dupliziert werden, so dass auch bei Online-Terminals – sofern die Karte nicht gesperrt ist – Zahlungen erfolgen können. Geschädigt wird in diesem Fall direkt die Person, welcher das Konto gehört. Es wird geschätzt, dass auf das Kreditkartenkonsortium “Groupement d’Intérêt Économique – Cartes Bancaires (GIE CB)“ Kosten in Höhe von mehr als 1,5 Mrd. Euro für den Austausch der Karten und die Umstellung der Offline- auf Online-Terminals zukommen [Irel]. Desweiteren braucht diese Umstellung vor allem Zeit, in der etwaige Schäden weiter beglichen werden müssen. Übrigens verhandelte GIE CB im März 2000 auch über eine Verlängerung der Versicherung gegen Betrugsfälle...

Dieses Beispiel zeigt zweierlei: Erstens kann das plötzliche Auftreten eines Schadens, der das System kompromittiert, nicht ausgeschlossen werden, und zweitens stehen keine Mechanismen zur Verfügung, um das System im laufenden Betrieb auf Veränderungen an den Kryptoverfahren anzupassen.

Das Beispiel der “Yescard“ muss kein Einzelfall bleiben, denn die Symptome „plötzliches Auftreten eines Schadens“ und „inflexible Integration von Algorithmen und Schlüsseln“ ist durchaus typisch für moderne Informationssysteme.

In immer mehr Bereichen finden elektronische Kommunikationsformen ihre Verwendung. Zum Beispiel beim „E-Commerce“ zwischen Unternehmen oder zwischen Unternehmen und Privatleuten, beim „E-Government“ zwischen Behörden und Bürgern, bei der firmen- oder behördeninternen Kommunikation, oder um berechtigten Personen oder Rechnern einen realen resp. virtuellen Zutritt zu gestatten. Sicherheitsrelevante Anwendungen verlangen dabei nach Sicherheit, insbesondere wenn die Kommunikation über offene Netze verläuft – wie etwa das Internet. D. h. je nach Anwendung sollen übertragene Daten tatsächlich vom behaupteten Absender stammen und unverändert und vertraulich übermittelt werden.

Die Public-Key-Kryptographie bietet hierzu einen hervorragenden Lösungsansatz durch digitale Signaturen und Verschlüsselungen, weil das Schlüsselmanagement durch Public-Key-Methoden elegant organisiert werden kann. Die für den praktischen Gebrauch notwendige Infrastruktur wird in einer so genannten Public-Key-Infrastruktur (PKI) realisiert. Public-Key-Infrastrukturen bergen aber auch Risiken, weil die Verfahren nicht beweisbar sicher sind. Der Grund ist, dass nicht bekannt ist, ob die zugrunde liegenden mathematischen Probleme, z. B. das Faktorisierungsproblem oder das Diskrete-Logarithmus-Problem, die die Basis für Kryptoalgorithmen RSA resp. DSA darstellen, wirklich schwierig sind. Es ist nur bekannt, dass bei geeigneter Schlüssel- und Systemparameterwahl unter Berücksichtigung der heute verfügbaren Rechenleistung keine effizienten Algorithmen zum Faktorisieren großer Zahlen oder Berechnen diskreter Logarithmen bekannt sind. Diese Empfehlungen über kryptographisch geeignete Algorithmen, Schlüssel und Systemparameter werden u. a. vom Bundesamt für Sicherheit in der Informationstechnik (BSI) zusammengestellt [BSI].

Das Problem ist, dass sich diese Empfehlungen und Voraussagen nur auf das Wissen der Vergangenheit und Gegenwart beziehen, dass aber Fortschritte bei der Entwicklung effizienter Algorithmen oder Implementierungsfehler in der Zukunft nicht auszuschließen sind. Das BSI schreibt: „Solche Prognosen [...] können sich natürlich jederzeit aufgrund unvorhersehbarer dramatischer Entwicklungen als falsch erweisen“ [BSI01]. Zum Beispiel verbesserte 1988 John Pollards Zahlkörpersieb die Faktorisierungsalgorithmen deutlich [LeLe93], Hans Dobbertin zeigte 1996, dass die Hashfunktion MD4 nicht kollisionsresistent ist [Dobb96] und Daniel Bleichenbacher präsentierte 1998 einen Angriff auf RSA mit PKCS#1-Padding [Blei98]. Eine andere Klasse von Angriffen zielt auf das Ausspähen geheimer Schlüssel, z. B. sind Chipkarten einer Reihe von logischen Angriffen ausgeliefert, die Reaktionen auf gezielte Fehler, den Stromverbrauch oder Berechnungszeiten analysieren [Koch96] [JaJK99].

Das Auftreten eines Fehlers kann zur Kompromittierung digitaler Signaturen führen. Das bedeutet, dass digitale Signaturen ihre Authentizität und Integrität und damit ihre Beweiskraft verlieren könnten, wenn durch einen plötzlichen Schaden keine Möglichkeit mehr bestehen würde, Daten anderweitig zu sichern oder kryptographische Verfahren und Parameter anzupassen. Betroffen wären nicht nur rechtsverbindliche digitale Signaturen, deren Zeitstempel keine gesicherte Aussage über den Erstellungszeitpunkt mehr geben, wenn auch sie kompromittiert wurden, sondern auch die für eine PKI notwendigen Zertifikate und Revokationsmechanismen. Zertifikate stellen in einer PKI die Verbindung von öffentlichem Schlüssel und Teilnehmer her und durch Revokationsmechanismen – wie etwa eine Sperrliste – können Benutzer vor dem Akzeptieren kompromittierter Schlüssel gewarnt werden. Da Zertifikate und Revokationsinformationen digital signiert sind, können auch sie von einem Schaden betroffen sein: Es können neue Zertifikate mit fiktiven Seriennummern in Umlauf gebracht werden, die der Zertifizierungsinstanz nicht bekannt sind und somit auch nicht revoziert werden können, und Revokationsinformationen können ihre Aussagekraft verlieren. Daneben sind die existierenden Revokationsmechanismen nur für die Sperrung einzelner Zertifikate ausgelegt, was bei Sperrung sehr vieler Zertifikate den Umgang mit Sperrlisten unhandlich machen würde.

Darüber hinaus könnte ein Kompromittieren von Schlüsseln kurzfristige Verschlüsselungen enthüllen. Langfristige Verschlüsselungen werden hier nicht betrachtet, da ein Abhören und Mitschneiden von Daten über offene Netze und ein Entschlüsseln nach hinreichend langer Zeit nicht zu verhindern ist. Kurzfristige Verschlüsselungen sind Geheimnisse, die nach einiger Zeit ohnehin öffentlich bekannt oder uninteressant werden.

Das Hauptproblem ist aber, dass durch Revokation von Zertifikaten zwar die Sicherheit hergestellt wird, dies aber u. U. gleichzeitig den Verlust der Verfügbarkeit einer Anwendung bedeuten

kann. Denn das in einer PKI notwendige Vertrauen wird über den öffentlichen Schlüssel der Zertifizierungsinstanz, welcher der Zertifikatsinhaber vertraut, und durch Zertifikate und Zertifikatsketten gebildet. Werden Zertifikate in dieser Kette revoziert, kann u. U. eine digitale Signatur nicht mehr als gültig validiert werden, was die Funktionsfähigkeit der PKI einschränken kann. Da in Public-Key-Infrastrukturen heutzutage meist fest installierte kryptographische Verfahren arbeiten, ist es oft unmöglich, auf Veränderungen zu reagieren. Zur Wiederherstellung der Verfügbarkeit der gesamten PKI kann es dann notwendig sein, neue Komponenten zu entwickeln, zu produzieren, zu verteilen und zu installieren, was Zeit und Geld kosten würde, wie das Beispiel der "Yescard" auch belegt. Sind beispielsweise Chipkarten involviert, müssten diese erst bestellt und u. U. produziert werden, was gerade dann besonders lange dauern dürfte, wenn das Problem mehrere Anwendungen trifft und auf dem Markt eine entsprechende Nachfrage einsetzt. Da sich die heutige Wirtschaft und Gesellschaft auf funktionierende Informationssysteme verlässt, können solche Schäden beträchtliche wirtschaftliche und gesellschaftliche Folgen nach sich ziehen.

Es bleiben also die vier offenen Probleme:

1. Verlust der Verfügbarkeit von PKI-Anwendungen
2. Verlust der Beweisbarkeit digitaler Signaturen
3. Verlust von praktikablen Revokationsmechanismen
4. Verlust der Vertraulichkeit verschlüsselter Daten

Der in dieser Dissertation verfolgte Ansatz zur Lösung der Probleme besteht in einem Fail-Safe-Konzept für Public-Key-Infrastrukturen.

Fail-Safe bedeutet: Wenn ein sicherheitskritisches System ausfällt („fail"), dann bleibt es dennoch in einem sicheren Zustand („safe"). Im Falle von Public-Key-Infrastrukturen bedeutet dies konkret, dass die Verfügbarkeit einer PKI auch im Schadensfall gewährleistet bleibt und die Eigenschaften von digitalen Signaturen, Revokationsmechanismen und Verschlüsselungen erhalten bleiben. Eine wesentliche Voraussetzung für das Konzept ist die flexible Integration der Kryptoverfahren in die Public-Key-Infrastruktur, welche es ermöglicht, Verfahren im laufenden Betrieb hinzuzufügen und zu löschen. An der Technischen Universität Darmstadt entsteht im Projekt „FlexiPKI" eine flexible Public-Key-Infrastruktur, die den Austausch kryptographischer Basistechnologien und damit ein Aktualisieren der in der PKI verwendeten Komponenten gestattet [FlexiPKI].

Das Fail-Safe-Konzept sieht vor, mehrere voneinander unabhängige Kryptoverfahren mit Schlüsseln und Zertifikaten schon vor einem Schaden in die Infrastruktur zu integrieren, so dass im Schadensfall, wenn sozusagen ein „Teil" der PKI ausfällt, die zusätzlichen, vom Schaden nicht betroffenen Komponenten den Betrieb aufrecht erhalten und damit die Verfügbarkeit sichern. Die PKI besteht somit aus mindestens zwei parallelen „Teil"-PKIs. Durch Einsatz multipler Kryptographie kann auch dem Verlust von Verbindlichkeit und Vertraulichkeit entgegen gewirkt werden, indem digitale Signaturen mehrfach parallel und Verschlüsselungen mehrfach iterativ angewendet werden. Anwendungen von multiplen digitalen Signaturen sind insbesondere Zeitstempel und Revokationsinformationen. Letztere werden darüber hinaus so erweitert, dass mit einem einzigen Sperreintrag mehrere Zertifikate gesperrt werden können. Ein wesentlicher Bestandteil des Fail-Safe-Konzepts ist die Möglichkeit, Komponenten der PKI im laufenden Betrieb und auf sichere Weise hinzufügen und löschen zu können, um auf Veränderungen reagieren zu

können. Diese Veränderungen sind insbesondere nach Auftreten eines Schadens und eventuellem Ausfall eines Teils einer PKI nötig, um in der PKI wieder multiple Kryptographie installieren zu können.

Zusammengefasst besteht das Fail-Safe-Konzept für Public-Key-Infrastrukturen aus den vier Bausteinen

- Integration multipler Kryptographie in eine PKI mit dynamischer Aktualisierbarkeit,
- Einsatz multipler digitaler Signaturen,
- Verwendung erweiterter Revokationsmechanismen und
- Nutzung iterativer Verschlüsselungen,

die je eines der vier offenen Probleme beantworten, so dass eine konkrete Umsetzung des Konzepts anwendungsabhängig realisiert werden kann.

Das Fail-Safe-Konzept wird an einer besonderen PKI – der „Fail-Safe-PKI“ – konzeptionell und exemplarisch in einer Implementierung demonstriert. Dadurch wird der Proof-of-Concept erbracht, dass eine Public-Key-Infrastruktur möglich ist, in welcher die genannten Probleme nicht auftreten.

In der Literatur existieren bereits Ideen zur multiplen Kryptographie mit multiplen Signaturen und iterativen Verschlüsselungen [Wohl00] [Pfit01] [FJPP95], oder wie mit gefälschten digitalen Signaturen umgegangen werden könnte [PePf97]. Die Motivation für diese Überlegungen beschränkt sich allerdings auf das Kompromittieren einzelner Schlüssel, nicht eines kryptographischen Verfahrens. Bislang waren die hier thematisierten Probleme – und insbesondere das Hauptproblem der nachhaltigen Verfügbarkeit – nicht gelöst worden.

Das Dokument gliedert sich wie folgt: Im ersten Kapitel werden Informationen zur Kryptographie, zu Public-Key-Infrastrukturen und zu derzeitigen Anwendungen vermittelt. Die Definitionen und Beispiele zu den kryptographischen Verfahren sind für die Darstellung der Problemexposition im zweiten Kapitel wichtig. Dort werden die Gründe für die Probleme dargelegt und die Folgen diskutiert. Das dritte Kapitel widmet sich der Darstellung des Fail-Safe-Konzepts und wie es die zuvor geschilderten Probleme lösen kann. Kapitel 4 stellt die Realisierung des Konzepts anhand der Fail-Safe-PKI im Detail vor, indem die PKI und ihre Funktionsweise auf mathematisch motivierte Weise beschrieben wird. Die exakten Datenstrukturen sind in die Appendizes ausgelagert. Das fünfte Kapitel diskutiert zwei konkrete Beispiele und das sechste Kapitel stellt die geleisteten Implementierungen vor. Eine Zusammenfassung mit Ausblick rundet die Dissertation zum Fail-Safe-Konzept für Public-Key-Infrastrukturen ab.

Kapitel 1

Grundlagen zu Public-Key-Infrastrukturen

Elektronische Kommunikation in offenen Netzen kann Sicherheit erfordern, zum Beispiel die unverfälschte, authentische oder vertrauliche Übertragung elektronischer Daten. Die Public-Key-Kryptographie bietet hierzu einen hervorragenden Lösungsansatz, der zum praktischen Einsatz eine entsprechende Infrastruktur erfordert: die Public-Key-Infrastruktur (PKI). Public-Key-Infrastrukturen sind das Forschungsobjekt dieser Dissertation. Sie werden heutzutage in diversen Umgebungen benutzt, die sich in Signatur- und Authentisierungs-Anwendungen klassifizieren lassen.

In diesem Kapitel wird das Forschungsobjekt erklärt. Beginnend mit den Anforderungen an Sicherheit in verteilten Systemen werden die Grundlagen der Public-Key-Kryptographie erläutert sowie Aufbau und Funktionsweise einer Public-Key-Infrastruktur geschildert. Beispiele heutiger Anwendungen runden dieses Kapitel ab.

1.1 Sicherheitsanforderungen

Kommunikation in verteilten Systemen – wie etwa dem Internet – kann Sicherheit erfordern. Dieser Abschnitt diskutiert mögliche Sicherheitsziele.

Die heutige Informationsgesellschaft nutzt elektronische Kommunikation in vielfältiger Weise. Einige Schlagworte: E-Mail, Datenverschlüsselung, Homebanking, digitale Wahlen, E-Commerce oder E-Government. Einige dieser Anwendungen erfordern die folgende Sicherheit [ISO89]:

- Authentizität übertragener Daten (Herkunft)
- Integrität übertragener Daten (Unversehrtheit)
- Zurechenbarkeit zu einem Benutzer (Verbindlichkeit)
- Verschlüsselung sensibler Daten (Vertraulichkeit)
- Verfügbarkeit von Daten oder Dienstleistungen
- Authentisierung von Benutzern oder Rechnern (Zugangsberechtigung)

Ein Beispiel verdeutlich diese Anforderungen.

Alice und Bob
Alice und Bob sind zwei sicherheitsbewusste Internet-Nutzer. Sie wollen auf sichere Weise miteinander kommunizieren:

- Bob will sicher sein, dass eine via E-Mail kommunizierte Nachricht, die den Anschein hat, von Alice zu sein, auch tatsächlich von Alice stammt und dass der Inhalt der Nachricht nicht verändert wurde. Diese Beweise sollen auch einen Richter überzeugen, so dass Alice ihre Aussagen nachträglich nicht abstreiten kann.
- Alice und Bob legen großen Wert auf eine vertrauliche Kommunikation und verschlüsseln deshalb ihre E-Mails.
- Neben der Kommunikation via E-Mail kommunizieren die Rechner der beiden miteinander in einer Client-Server-Architektur. Bob stellt auf seinem Rechner – als Server – eine Dienstleistung zur Verfügung. Er möchte sichergehen, dass nur Alices Rechner – als Client – auf diesen Dienst zugreifen kann.
- Alice und Bob erwarten eine Verfügbarkeit der sicheren Kommunikation und Dienstleistung.

Kryptographie ist eine Basis zur Realisierung dieser Sicherheitsanforderungen.

1.2 Public-Key-Kryptographie

Da die Übertragungswege über offene Netze – wie etwa das Internet – unsicher sind, werden Sicherheitsmechanismen benötigt. Methoden zur Verschlüsselung und zur Gewährleistung von Daten-Authentizität und -Integrität lassen sich mit Hilfe der Kryptographie erschaffen. Dieser Abschnitt widmet sich den Grundlagen der Public-Key-Kryptographie.

Sensible Daten können verschlüsselt übertragen werden, so dass ein Unbefugter den Inhalt einer Nachricht zwar lesen, aber nicht verstehen kann. Es gibt symmetrische Verschlüsselungsverfahren, die zum Ver- und Entschlüsseln denselben geheimen Schlüssel verwenden, den nur Sender und Empfänger kennen. Symmetrische Verschlüsselungsverfahren sind effizient, haben aber den Nachteil des Schlüsselmanagements, denn Sender und Empfänger müssen den geheimen Schlüssel zuvor ausgetauscht haben. Dieses Manko eliminieren asymmetrische Verschlüselungsverfahren. Es gibt zwei Schlüssel: einen privaten Schlüssel (den “Private Key“), den der Benutzer geheim hält, und einen dazu passenden öffentlichen Schlüssel (den “Public Key“). Dabei soll es praktisch nicht möglich sein, aus dem öffentlichen Schlüssel den privaten zu berechnen. Eine vertrauliche Nachricht kann mit dem öffentlichen Schlüssel des Empfängers verschlüsselt werden, so dass nur dieser aus dem erhaltenen Kryptogramm den Klartext extrahieren kann. Die asymmetrischen Verfahren heißen auch Public-Key-Verfahren. Da sie deutlich langsamer als symmetrische Verfahren sind, werden in der Praxis meist hybride Verfahren angewendet: Zunächst wird der Klartext mit einem symmetrischen Verfahren und einem geheimen Schlüssel, der extra für diese Verschlüsselung zufällig erzeugt wurde, effizient verschlüsselt. Anschließend wird dieser geheime Schlüssel unter Nutzung eines asymmetrischen Verfahrens und dem öffentlichen Schlüssel des Empfängers in ein Kryptogramm verwandelt. Das Public-Key-Verfahren dient damit dem Schlüsselaustausch.

Um Daten-Authentizität und -Integrität zu garantieren, kann eine Art von elektronischer Unterschrift genutzt werden – die digitale Signatur. Durch eine digitale Signatur zu einer Nachricht wird ihre Authentizität und Integrität gewährleistet. Digitale Signaturen basieren auf Public-Key-Kryptographie. Ein Sender hat wieder einen öffentlichen und einen privaten Schlüssel, wobei sich der private aus dem öffentlichen Schlüssel effizient nicht berechnen lässt. Er signiert eine Nachricht, indem aus der Nachricht und seinem privaten Schlüssel eine digitale Signatur generiert wird. Ein Empfänger kann sich anhand des öffentlichen Schlüssels davon überzeugen, dass die Signatur korrekt ist, d. h. dass die übermittelte Nachricht nicht verändert wurde und mit dem dazu passenden privaten Schlüssel erzeugt wurde.

Für Details zur Kryptographie sei auf die Standardwerke von Bruce Schneier [Schn96], Alfred Menezes, Paul van Oorschot und Scott Vanstone [MeOV97] oder Johannes Buchmann [Buch99] verwiesen. In diesem Abschnitt werden nur die für die Arbeit relevanten Fakten zusammengefasst.

Definition 1
Ein **Signaturalgorithmus** besteht aus

- einem **Algorithmus zur Signaturerzeugung** σ und
- einem **Algorithmus zur Signaturverifikation** τ.

Dazu gehören

- ein **öffentlicher Schlüssel** – der **Public Key** – puK und
- ein **privater Schlüssel** – der **Private Key** – prK.

Aus einem Bitstring b, dessen Länge von σ abhängt, und dem privaten Schlüssel prK erzeugt σ eine **digitale Signatur**

$$s = \sigma(b, \mathrm{prK}).$$

τ berechnet aus einem Bitstring s', dem öffentlichen Schlüssel puK und einem Bitstring b'

$$\tau(b', s', \mathrm{puK}) = \begin{cases} \mathrm{TRUE} \\ \mathrm{FALSE} \end{cases}$$

Es gilt stets: $\tau(b, s, \mathrm{puK}) = \mathrm{TRUE} \iff s = \sigma(b, \mathrm{prK})$. Ein Signaturalgorithmus wird auch mit σ bezeichnet und damit dem Umstand Rechnung getragen, dass τ und σ stets zusammengehören.

An einen Signaturalgorithmus σ und Schlüsselpaare $(\mathrm{prK}, \mathrm{puK})$ werden die Anforderungen gestellt, dass es praktisch nicht möglich ist, den privaten Schlüssel zu einem öffentlichen Schlüssel zu berechnen oder ohne Kenntnis des privaten Schlüssels prK eine digitale Signatur $s = \sigma(b, \mathrm{prK})$ mit $\tau(b, s, \mathrm{puK}) = \mathrm{TRUE}$ zu erzeugen.

Beispiel (RSA-Signaturalgorithmus)
Der derzeit am häufigsten benutzte Signaturalgorithmus ist RSA, der nach seinen Erfindern Ron Rivest, Adi Shamir und Len Adleman benannt ist [Buch99]:

Seien p, q prim. Sei $n = pq$. Sei die natürliche Zahl e so gewählt, dass gilt: $1 < e < \varphi(n)$ mit $\varphi(n) = (p-1)(q-1)$ und $\gcd(e, \varphi(n)) = 1$. Sei die natürliche Zahl d so gewählt, dass gilt: $1 < d < \varphi(n)$ und $de \equiv 1 \bmod (p-1)(q-1)$.

Dann funktioniert RSA wie folgt:

Private Key	d
Public Key	(n, e)
Zu signieren	$m \in \{1, ..., n-1\}$
Signatur	s mit $s = \sigma(m, d) \equiv m^d \bmod n$
Verifikation	$\tau(m, s, (n, e)) =$ TRUE, wenn $m \equiv s^e \bmod n$ gilt, sonst FALSE

Bei RSA ist es möglich, aus der Signatur s das signierte m zurückzugewinnen: $m \equiv s^e \bmod n$. Darüber hinaus ist es aufgrund der multiplikativen Eigenschaft bei RSA möglich, eine digitale Signatur ohne Kenntnis des zugehörigen privaten Schlüssels zu erzeugen; siehe dazu Seite 5.

Die Schwierigkeit des RSA-Problems, d zu berechnen oder s zu fälschen, hängt – vermutlich, da noch nicht mathematisch bewiesen – mit dem Faktorisierungsproblem zusammen, welches das Problem beschreibt, aus $n = pq$ die Faktoren p und q zu ermitteln.

Beispiel (Digital Signature Algorithm (DSA))
Ein weiterer beliebter Signaturalgorithmus ist der Digital Signature Algorithm (DSA) [Buch99]:

Sei p eine Primzahl mit $2^{1023+64t} < p < 2^{1024+64t}$, $t \in \{0, 1, 2, ..., 8\}$. Betrachte $(\mathbb{Z}/p\mathbb{Z})^*$. Seien $q \in \mathbb{N}$, $g \in \mathbb{N}$ so gewählt, dass gilt: $2^{159} < q < 2^{160}$, $q \mid p-1$ und $g \equiv x^{(p-1)/q} \bmod p$. Sei $a \in \{1, 2, ..., q-1\}$ zufällig gewählt. Setze $A \equiv g^a \bmod p$.

DSA funktioniert nun folgendermaßen:

Private Key	a
Public Key	(p, q, g, A)
Zu signieren	$m \in \{1, ..., n-1\}$
Signatur	$\sigma(m, a) = (r, s)$ mit: Wähle $k \in \{1, 2, ..., q-1\}$ zufällig. Berechne $r \equiv (g^k \bmod p) \bmod q$. Berechne $s \equiv k^{-1}(m + ar) \bmod q$.
Verifikation	$\tau(m, (r, s), (p, q, g, A)) =$ TRUE, wenn gilt: 1. $1 \le r \le q-1$, $1 \le s \le q-1$ 2. $r \equiv ((g^{(s^{-1}m) \bmod q} A^{((rs^{-1}) \bmod q}) \bmod p) \bmod q$ sonst FALSE

DSA basiert auf der Schwierigkeit, diskrete Logarithmen in endlichen Körpern zu berechnen (DL-Problem in endlichen Körpern). DSA ist ein effizientes ElGamal-Signaturverfahren.

Beispiel (Elliptic Curve Digital Signature Algorithm (ECDSA))
ECDSA (Elliptic Curve DSA) ist eine Variante von DSA, die auf der Punktegruppe $E(\mathbb{K})$ einer elliptischen Kurve E über einem endlichen Körper $\mathbb{K}$ basiert. $\mathbb{K}$ kann ein endlicher Primkörper F_p, p prim, oder ein endlicher Körper F_{2^m} der Charakteristik 2 sein, $m \in \mathbb{N}$.

Signaturalgorithmen lassen als Input für die Signaturerzeugung und -Verifikation nur einen beschränkt langen Bitstring zu. Da zu signierende Dokumente länger als dieser Input sein können, gibt es Hashfunktionen:

Definition 2
Eine **Hashfunktion** ist eine Funktion κ mit folgenden Eigenschaften:

- *Compression* – κ bildet einen beliebig langen Bitstring b (Input) auf einen String fester Länge $\kappa(b)$ (Output) ab. $\kappa(b)$ heißt **Hashwert** von b.
- *Easy to compute* – Zu gegebenem κ und b ist $\kappa(b)$ leicht zu berechnen.
- *Preimage resistance* – Es gibt keinen effizienten Algorithmus, um zu einem Output y ein b' zu berechnen, so dass $y = \kappa(b')$ gilt.
- *2nd-preimage resistance* – Es gibt keinen effizienten Algorithmus, um zu einem Input b ein zweites b' zu finden, so dass $\kappa(b) = \kappa(b')$ gilt.
- *Collision resistance* – Es gibt keinen effizienten Algorithmus, um zwei Inputs b und b' zu finden, die auf denselben Output $\kappa(b) = \kappa(b')$ abgebildet werden [MeOV97].

Eine Hashfunktion gewährleistet die Integrität übertragener Daten: Wird der Hashwert der übertragenen Daten sicher übermittelt, kann die Veränderung der Daten zwar nicht verhindert, aber festgestellt werden. Ein Bitstring b mit Hashwert $\kappa(b)$ heißt **integer**, wenn die Hashfunktion die obigen Anforderungen erfüllt.

Beispiele (Hashfunktionen)
MD4 (Message Digest 4) ist eine 128-Bit-Hashfunktion, die beliebig lange Bitstrings auf 128 Bit lange Hashwerte abbildet. MD4 arbeitet in vier Schritten, *Definition of constants*, *Preprocessing*, *Processing* und *Completion*, in denen der Input durch Anwendung einiger Shift-Operationen und Funktionen mit gewissen Initial-Werten in drei Runden komprimiert wird. Aufgrund einer schwachen Kollisionsresistenz wurde eine Weiterentwicklung notwendig. MD5 unterscheidet sich von MD4 durch andere Shift-Operationen und Funktionen sowie durch Hinzufügen einer weiteren Runde. Auch die 160-Bit-Hashfunktionen SHA-1 (Secure Hash Algorithm 1) und RIPEMD-160 (RACE Integrity Primitives Evaluation 160) basieren auf der Basisroutine von MD4, allerdings verfeinert durch andere Initial-Werte, vier Runden und anderen Funktionen. SHA-1 und RIPEMD-160 gelten als stärker als MD5. Die heute üblichen Hashfunktionen sind standardisiert und über einen Object Identifier (OID) weltweit eindeutig gekennzeichnet: OID = {1 3 14 3 2 26} kennzeichnet SHA-1, OID = {1 2 840 113549 2 5} gehört zu MD5 und OID = {1 3 36 3 2 1} führt zu RIPEMD-160. Für weitere Informationen sei auf [MeOV97] oder [Schn96] verwiesen.

Signaturalgorithmen sollen die Authentizität von Daten gewährleisten, so dass nachweisbar derjenige eine digitale Signatur erzeugt hat, der den privaten Schlüssel besitzt. Signaturalgorithmen sind Angriffen ausgesetzt. Zwei Angriffe auf die Authentizität von Signaturen funktionieren wie folgt:

- Ausnutzen der multiplikativen Eigenschaft des Signaturalgorithmus
  Die multiplikative Eigenschaft bei einem Signaturalgorithmus bedeutet, dass für zwei Nachrichten m_1 und m_2 gilt: $\sigma(m_1, \mathrm{prK}) * \sigma(m_2, \mathrm{prK}) = \sigma(m_1 * m_2, \mathrm{prK})$. RSA besitzt diese Eigenschaft: $(m_1^d \bmod n) * (m_2^d \bmod n) \equiv (m_1 * m_2)^d \bmod n$. Bei diesem Angriff setzt der Angreifer zwei Signaturen zu einer neuen, nicht authentischen Signatur zusammen.
- Existenzielle Fälschungen
  Voraussetzung für eine existenzielle Fälschung ist, dass sich die Algorithmen von Signaturerzeugung und -verifikation vertauschen lassen, wie etwa bei RSA. Ein Angreifer wählt ein zufälliges s und berechnet mit dem öffentlichen Schlüssel (n, e) von – beispielsweise – Alice: $m \equiv s^e \bmod n$. Ist m ein sinnvoller Text, wäre $s \equiv m^d \bmod n$ eine glaubwürdige digitale Signatur von Alice zu m.

Um diese Angriffe zu verhindern, gibt es folgende Gegenmaßnahmen:

- Einsatz einer Hashfunktion κ
  Denn: κ besitzt selbst nicht die multiplikative Eigenschaft (Andernfalls würde für ein beliebiges m aus $\kappa(m * 1) = \kappa(m) * \kappa(1)$ folgen: $\kappa(1) = 1$, was den Eigenschaften einer Hashfunktion widerspricht), weshalb unter der Annahme, σ hätte die multiplikative Eigenschaft, $\sigma(\kappa(m_1), \mathrm{prK}) * \sigma(\kappa(m_2), \mathrm{prK}) = \sigma(\kappa(m_1) * \kappa(m_2), \mathrm{prK}) \neq \sigma(\kappa(m_1 * m_2), \mathrm{prK})$ gilt. κ verhindert auch die existenzielle Fälschung, denn das über den Verifikationsalgorithmus berechnete m wäre kein sinnvoller Text, sondern der Hashwert eines Textes, der effizient nicht zu berechnen ist.

- Einsatz einer **Formatierung** ϕ
  Eine Formatierung oder Formatierungsfunktion nutzt eine festgelegte Struktur aus. Beispiel: Ein zu signierender Text beginnt stets mit einem festgelegten Wert, beispielsweise $\phi(m) = 00 \,|\, m$ für einen Bitstring m. ϕ verhindert ein Ausnutzen der multiplikativen Eigenschaft, weil ϕ sie selbst nicht besitzt, $\phi(m_1) * \phi(m_2) \neq \phi(m_1 * m_2)$ für $m_1, m_2 \neq 00$, und sich diese Eigenschaft auf σ überträgt. ϕ verhindert existenzielle Fälschungen, weil für ein m gelten müsste: es gibt ein m' mit $\phi(m') = m$.

- Einsatz einer besonderen Formatierung – **Redundanzfunktion** ρ
  Beispiel für eine Redundanzfunktion: Die Konkatenation von Bitstrings: $\rho(m) = m \,|\, m$ für einen Bitstring m. ρ verhindert die beiden Angriffe: ρ besitzt selbst nicht die multiplikative Eigenschaft, $\rho(m_1) * \rho(m_2) \neq \rho(m_1 * m_2)$ für zwei verschiedene m_1 und m_2, was sich – wie oben – auf σ überträgt. ρ verhindert ebenfalls existenzielle Fälschungen, weil für das m gelten müsste: es gibt ein m' mit $\rho(m') = m$.

- Wenn der Hashwert deutlich kürzer als der Input des Signaturalgoritmus ist, kann es sinnvoll sein, sowohl Hashfunktion als auch Formatierung zu nutzen.

In der Praxis gebräuchliche Formatierungen sind PKCS#1, ISO 9796-1 oder ISO 9796-2, die z. B. in [PKCS1b] und [MeOV97] detailliert beschrieben sind. Formatierungen werden auch **Padding** genannt.

Aus den drei Komponenten Signaturalgorithmus, Hashfunktion und Formatierung setzt sich dann ein Signaturverfahren zusammen, so wie es in heutigen Anwendungen zu finden ist.

Definition 3
Ein **Signaturverfahren**[1] besteht aus den beiden Teilen „Erzeugung einer digitalen Signatur“ und „Verifikation einer digitalen Signatur“.

Zur **Erzeugung einer digitalen Signatur** wird der Algorithmus sign genutzt, der sich aus den kryptographischen Primitiven Algorithmus zur Signaturerzeugung σ, Hashfunktion κ und Formatierung ϕ sowie einer Zufallszahl k zusammensetzen kann. sign erzeugt aus einer beliebig langen Nachricht m und privatem Schlüssel prK die **digitale Signatur** fester Länge

$$\mathrm{sign}(m, \mathrm{prK}).$$

[1] Die Begriffe „Signaturverfahren“ und „Signaturalgorithmus“ sind ähnlich und werden im allgemeinen Sprachgebrauch oft synonym benutzt, sind aber nicht immer dasselbe: Ein Signaturverfahren kann ausschließlich aus einem Signaturalgorithmus bestehen oder zusätzlich über eine Hashfunktion und/oder Formatierung verfügen.

Zum **Verifizieren einer digitalen Signatur** wird der Algorithmus verify genutzt, der sich aus Algorithmus zur Signaturverifikation τ, Hashfunktion κ und Formatierung ϕ zusammensetzen kann. Aus Nachricht m', Signatur s' und öffentlichem Schlüssel puK liefert verify das Verifikationsergebnis TRUE oder FALSE:

$$\text{verify}(m', s', \text{puK}) = \begin{cases} \text{TRUE} \\ \text{FALSE} \end{cases}$$

Die Bedingungen an die Verifikation hängen vom spezifischen Verfahren ab.

Eine digitale Signatur s zum Dokument m heißt **mathematisch korrekt** oder **gültig**, falls gilt:

$$\text{verify}(m, s, \text{puK}) = \text{TRUE}$$

Ein Signaturverfahren wird auch mit sign bezeichnet und damit dem Umstand Rechnung getragen, dass verify und sign stets zusammengehören.

Definition 3 entspricht einer "signature suite" aus [EU A9C01], die definiert ist als: "combination of a signature algorithm with its parameters, a key generation algorithm, a padding method, and a cryptographic hash function".

Das folgende an Victor Shoup [Shou98] angelehnte **Signatur/Verifikations-Modell** fasst die wichtigsten Punkte zusammen:

Private Key	prK
Public Key	puK
Dokument	D
Digitale Signatur	s mit $s = \text{sign}(D, \text{prK})$
Verifikation	$\text{verify}(D, s, \text{puK}) = \text{TRUE}$ oder FALSE

Beispiel (Signatur/Verifikations-Modell für RSA, MD5 und PKCS#1)
RSA ist derzeit mit 1024 Bit langen Schlüsseln als kryptographisch geeignet anzusehen. Das Signatur/Verifikations-Modell lässt sich für RSA folgendermaßen spezialisieren:

Private Key	d
Public Key	(n, e)
Dokument	D
Signatur	s mit $s = \text{sign}(D, (d, n)) = \sigma(\phi(\kappa(D)), (d, n)) \equiv \phi(\kappa(D))^d \bmod n$ mit Hashfunktion $\kappa \mathrel{\widehat{=}}$ MD5 und Formatierung $\phi \mathrel{\widehat{=}}$ PKCS#1
Verifikation	$\text{verify}(D, s, (n, e)) = \text{TRUE}$, falls $\phi(\kappa(D)) \equiv s^e \bmod n$, andernfalls ist $\text{verify}(D, s, (n, e)) = \text{FALSE}$

Beispiel (Signatur/Verifikations-Modell für DSA und SHA-1)
Unter kryptographischen Sicherheitsaspekten muss p mindestens 1024 und q mindestens 160 Bit lang sein. Das Signatur/Verifikations-Modell für DSA:

Private Key	a
Public Key	(p, q, g, A)
Dokument	D
Signatur	(r, s) mit $(r, s) = \text{sign}(D, (p, q, g, a)) = \sigma(\kappa(D), (p, q, g, a))$ $\equiv ((g^k \bmod p) \bmod q, k^{-1}(\kappa(D) + ar) \bmod q)$ mit Hashfunktion $\kappa \mathrel{\widehat{=}}$ SHA-1 und Zufallszahl $k \in \{1, 2, ..., q-1\}$
Verifikation	$\text{verify}(D, s, (p, q, g, A)) = \text{TRUE}$, genau dann, wenn gelten: 1. $1 \leq r \leq q-1$ 2. $1 \leq s \leq q-1$ 3. $r \equiv ((g^{(s^{-1}\kappa(D)) \bmod q} A^{((rs^{-1}) \bmod q)}) \bmod p) \bmod q$

Zur Vollständigkeit sei erwähnt: Das in Definition 3 auf Seite 6 definierte Signaturverfahren ist ein Verfahren *ohne Message Recovery* oder *mit Appendix*, denn aus der Signatur lässt sich der ursprüngliche Text nicht wieder extrahieren. Demgegenüber gibt es Verfahren *mit Message Recovery*, die statt der Hashfunktion ausschließlich eine Redundanzfunktion nutzen und die eine Rückgewinnung der Nachricht aus der Signatur zulassen. Solche Signaturverfahren werden hier im Weiteren nicht betrachtet.

In Tabelle 1.1 sind einige Signaturverfahren aufgeführt. Zur eindeutigen Kennzeichnung werden diese mit weltweit eindeutigen Object Identifiern, so genannten OIDs, versehen. Jeder OID besteht aus einer Ziffernfolge und einem Distinguished Name. Der zugehörigen Beschreibung eines OID sind weitere Informationen zu entnehmen, wie z. B. Angaben zur Formatierung, die manchmal nicht aus dem Distinguished Name ersichtlich sind. Da mehrere Organisationen OIDs vergeben, kann es vorkommen, dass ein Verfahren unter verschiedenen Object Identifiern bekannt ist.

Für einen einheitlichen Aufbau des Signaturverfahrens aus kryptographischen Primitiven kann sign formal folgendermaßen gesehen werden: $\text{sign} = \sigma \circ \phi \circ \kappa$, wobei ϕ und κ in Abhängigkeit des Signaturverfahrens auch die identische Abbildung id sein können, wenn sie in sign nicht enthalten sind. Drei Beispiele: rsasignatureWithripemd160 = RSA ∘ PKCS#1 ∘ RIPEMD-160, RSA encryption = RSA ∘ PKCS#1 ∘ id oder ecdsa-with-SHA1 = ECDSA ∘ id ∘ SHA-1.

An ein Signaturverfahren sign und Schlüsselpaare (prK, puK) werden folgende Anforderungen gestellt, siehe auch [MeOV97], [Buch99] oder [Schn96]:

1. Es ist praktisch nicht möglich, den privaten Schlüssel zu einem öffentlichen Schlüssel zu berechnen, auch nicht, wenn zusätzliches Datenmaterial zur Verfügung steht:
   (a) Es gibt keine effiziente Funktion f, die aus öffentlichem Schlüssel den privaten Schlüssel berechnet: $f(\text{puK}) = \text{prK}$.
   (b) Es gibt keine effiziente Funktion f', die aus öffentlichem Schlüssel und einer Menge von Bitstring-Signatur-Paaren $\{(b, s)\}$ den privaten Schlüssel berechnet: $f'(\text{puK}, \{(b, s)\}) = \text{prK}$.
2. Es ist praktisch nicht möglich, eine korrekte Signatur ohne Kenntnis des privaten Schlüssels zu erzeugen, auch nicht, wenn zusätzliches Datenmaterial zur Verfügung steht:
   (a) Es gibt keine effiziente Funktion g, die aus öffentlichem Schlüssel ein Bitstring-Signatur-Paar (b', s') effizient berechnet, für welches gilt: $\text{verify}(b', s', \text{puK}) = \text{TRUE}$.

Tabelle 1.1: Object Identifier für Signaturverfahren (Auswahl)

OID-Ziffernfolge	OID-Distinguished Name	Beschreibung
{2 5 8 1 1}	id-ea-rsa	RSA Encryption
{1 2 840 113549 1 1 1}	RSA encryption	RSA mit PKCS#1
{1 2 840 113549 1 1 5}	sha1WithRSAENC	RSA mit SHA-1 und PKCS#1-Formatierung
{1 3 14 3 2 29 }	SHA1WITHRSA	RSA mit SHA-1
{1 3 36 3 3 1 2 }	rsasignatureWithripemd160	RSA mit RIPEMD-160 und PKCS#1-Formatierung
{1 3 36 3 4 2 2 1 }	sigS_ISO9796-2Withsha1	RSA mit SHA-1 und ISO 9796-2-Formatierung
{1 3 36 3 4 2 2 2 }	sigS_ISO9796-2Withripemd 160	RSA mit RIPEMD-160 und ISO 9796-2-Formatierung
{1 3 36 3 4 3 2 1 }	sigS_ISO9796-2rndWithsha1	RSA mit SHA-1 und ISO 9796-2-Formatierung mit Zufallszahl
{1 3 36 3 4 3 2 2 }	sigS_ISO9796-2rndWith ripemd160	RSA mit RIPEMD-160 und ISO 9796-2-Formatierung mit Zufallszahl
{1 2 840 113549 1 1 4}	MD5WITHRSA	RSA mit MD5 und PKCS#1-Formatierung
{1 2 840 10040 4 1 }	DSA	DSA
{1 2 840 10040 4 3 }	SHA1WITHDSA	DSA mit SHA-1
{1 3 14 3 2 27 }	DSAWithSHA1	DSA mit SHA-1
{1 2 840 10045 1 }	ecdsa-with-SHA1	ECDSA mit SHA-1
{1 3 36 3 3 2 1 }	ecSignWithsha1	ECDSA mit SHA-1
{1 3 36 3 3 2 2 }	ecSignWithripemd160	ECDSA mit RIPEMD-160
{1 3 6 1 4 1 8301 3 1 1 2}	SHA1/IQDSA Signature Algorithm	IQDSA mit SHA-1
{1 3 6 1 4 1 8301 3 1 1 3}	RipeMD160/IQDSA Signature Algorithm	IQDSA mit RIPEMD-160

(b) Es gibt keine effiziente Funktion g', die aus öffentlichem Schlüssel und einer Menge von Bitstring-Signatur-Paaren $\{(b, s)\}$ ein Bitstring-Signatur-Paar (b', s') effizient berechnet, für welches gilt: $\text{verify}(b', s', \text{puK}) = \text{TRUE}$.

Unter diesen Bedingungen heißt ein Bitstring b mit Signatur s **authentisch**. D. h. digitale Signaturen gewährleisten die Authentizität übertragener Daten.

Damit gewährleisten Signaturverfahren die Authentizität und Integrität übertragener Daten.

Nun zu den Verschlüsselungsalgorithmen und -verfahren.

Definition 4
Ein **symmetrischer Verschlüsselungsalgorithmus** η_sym ist ein Algorithmus, der einen beliebig langen Klartext (Plaintext) p mit einem geheimen Schlüssel (dem Secret Key) sK zum **Kryptogramm**

$$c = \eta\_\text{sym}(p, \text{sK})$$

verschlüsselt. Ein Kryptogramm wird auch Ciphertext, Chiffretext oder Verschlüsselung genannt. Zu η_sym existiert ein **Entschlüsselungsalgorithmus** δ_sym, der aus Kryptogramm c' und geheimem Schlüssel sK

$$p' = \delta\_\text{sym}(c', \text{sK})$$

berechnet. Es gilt stets $p = \delta\_\text{sym}(\eta\_\text{sym}(p, \text{sK}), \text{sK})$ für jeden Klartext p.

An einen symmetrischen Verschlüsselungsalgorithmus η_sym und geheime Schlüssel sK wird die Anforderung gestellt, dass es praktisch nicht möglich ist, Schlüssel oder Klartexte aus Kryptogrammen zu berechnen.

Beispiele (symmetrische Verschlüsselungsalgorithmen)
DES (Data Encryption Standard), Triple-DES, IDEA (International Data Encryption Algorithm), SAFER (Secure And Fast Encryption Routine), der als DES-Nachfolger gekürte AES (Advanced Encryption Standard) Rijndael und weitere AES-Kandidaten wie MARS, RC6, SAFER+, Serpent, Twofish oder Magenta sind Beispiele für symmetrische Verschlüsselungsalgorithmen. Details zur Funktionsweise sind beispielsweise bei [MeOV97] oder unter [NIST] nachzulesen.

Definition 5
Ein **asymmetrischer Verschlüsselungsalgorithmus** η_asym ist ein Algorithmus, der einen beliebig langen Klartext p mit einem öffentlichen Schlüssel puK (des Empfängers) zum Kryptogramm

$$c = \eta\_\text{asym}(p, \text{puK})$$

verschlüsselt. Zu η_asym existiert ein **Entschlüsselungsalgorithmus** δ_asym, der aus einem Kryptogramm c' und privatem Schlüssel prK

$$p' = \delta\_\text{asym}(c', \text{prK})$$

berechnet. Es gilt stets $p = \delta\_\text{asym}(\eta\_\text{asym}(p, \text{puK}), \text{prK})$ für jeden Klartext p.

An einen asymmetrischen Verschlüsselungsalgorithmus η_asym und Schlüsselpaare (prK, puK) wird die Anforderung gestellt, dass es praktisch nicht möglich ist, private Schlüssel zu öffentlichen Schlüsseln oder Klartexte zu Kryptogrammen zu berechnen.

Beispiel (RSA-Verschlüsselung)
Die RSA-Verschlüsselung ist ein asymmetrischer Verschlüsselungsalgorithmus, der mit den wie im RSA-Signaturverfahren (Seite 3) gewählten privaten und öffentlichen Schlüsseln wie folgt abläuft:

Private Key	d
Public Key	(n, e)
Klartext	p beliebiger Länge
Verschlüsselung	Aufteilen von p in k Blöcke. Jeder Block p_i hat Länge $0 \leq m < n$ Berechne pro Block: $c_i \equiv p_i^e \bmod n$ für alle $i = 1, ..., k$
Kryptogramm	$c_1, ..., c_k$
Entschlüsselung	Berechne pro Block: $p_i \equiv c_i^d \bmod n$ für alle $i = 1, ..., k$ und setze p wieder zusammen

Beispiel (ElGamal-Verschlüsselung)
Die ElGamal-Verschlüsselung ist ein asymmetrischer Verschlüsselungsalgorithmus in endlichen Körpern und funktioniert wie folgt: Sei p prim. Betrachte $(\mathbb{Z}/p\mathbb{Z})^*$. Sei g eine Primitivwurzel modulo p. Wähle zufällig und gleichverteilt ein $a \in \{1, ..., p-2\}$. Berechne $A \equiv g^a \bmod p$.

Private Key	a
Public Key	(p, g, A)
Klartext	P beliebiger Länge
Verschlüsselung	Aufteilen von P in k Blöcke: $p_i \in \{0, 1, 2, ..., p-1\}$, $i = 1, ..., k$ Wähle zufälliges $b \in \{1, 2, ..., p-2\}$ Berechne: $B \equiv g^b \bmod p$ Berechne pro Block: $c_i \equiv A^b p_i \bmod p$ für alle $i = 1, ..., k$
Kryptogramm	$(B, c_1, ..., c_k)$
Entschlüsselung	Berechne $x = p - 1 - a$ Berechne pro Block: $p_i \equiv B^x c_i \bmod p$ für alle $i = 1, ..., k$ und rekonstruiere P

Beispiel (ECIES-Verschlüsselung)
Die Verschlüsselung mit Hilfe einer elliptischen Kurve wird mit dem **Elliptic Curve Integrated Encryption Scheme (ECIES)** realisiert, was noch nicht endgültig standardisiert ist. ECIES ist ein Verschlüsselungsschema, das einen auf elliptischen Kurven basierenden Diffie-Hellman-Schlüsselaustausch (ECDH) mit einer symmetrischen Verschlüsselung kombiniert. Für Details sei auf [P1363a01] verwiesen.

Ziel von Verschlüsselungen ist die Geheimhaltung von übermittelten Daten, so dass nur diejenigen, die den geheimen bzw. privaten Schlüssel kennen, die Daten entschlüsseln können. Ein Angriff auf eine asymmetrische Verschlüsselung kann gelingen, falls zu wenige Möglichkeiten für den Input vorliegen. Ein Extremfall verdeutlicht dies: Ein Geheimnis p, das entweder 0 oder 1 ist, soll vertraulich verschickt werden. Der Absender verschlüsselt $p \in \{0, 1\}$ mit dem öffentlichen Schlüssel des Empfängers: $c = \eta\_\text{asym}(p, \text{puK})$. Angenommen, einem Angreifer sei bekannt, dass die zu verschlüsselnde Nachricht entweder 0 oder 1 ist, und weiter angenommen, er könne c mithören. Dann kann der Angreifer herausbekommen, ob 0 oder 1 verschlüsselt wurde, indem er $c_1 = \eta\_\text{asym}(0, \text{puK})$ und $c_2 = \eta\_\text{asym}(1, \text{puK})$ berechnet und c mit c_1 und c_2 vergleicht.

Eine Lösung ist die **Randomisierung**, so dass ein Klartext bei jeder Verschlüsselung auf einen anderen Chiffretext abgebildet wird. Eine mögliche Randomisierung ist das Konkatenieren einer Zufallszahl k an den zu verschlüsselnden Klartext p. Verschlüsselt wird dann $p \,|\, k$. Der Empfänger berechnet $p \,|\, k$ aus dem Kryptogramm und extrahiert den Klartext p heraus. Damit

der Empfänger weiß, wie er die Randomisierung vom Klartext trennen kann, muss ein standardisiertes Format benutzt werden, wie beispielsweise PKCS#1. Die Randomisierung ist eine spezielle Formatierung.

Auch bei symmetrischen Verschlüsselungsalgorithmen kann eine Randomisierung sinnvoll sein, so dass zwei identische Klartexte zu unterschiedlichen Chiffretexten verschlüsselt werden. Eine Möglichkeit ist, einen Initialisierungsvektor IV in die Berechnung einfließen zu lassen.

Ein Verschlüsselungsverfahren vereinigt Verschlüsselungsalgorithmus und Formatierung.

Definition 6
Ein **Verschlüsselungsverfahren**[2] besteht aus einem (symmetrischen oder asymmetrischen) Ver- und Entschlüsselungsalgorithmus und optional einer Formatierung.

Zur **Verschlüsselung** wird der Algorithmus enc_sym bzw. enc_asym genutzt, der sich aus den kryptographischen Primitiven Verschlüsselungsalgorithmus η_sym resp. η_asym und Formatierung ϕ zusammensetzen kann. enc_sym resp. enc_asym erzeugt aus einem beliebig langen Klartext p und geheimem Schlüssel sK resp. öffentlichem Schlüssel puK das Kryptogramm

$$c = \text{enc\_sym}(p, \text{sK}) \text{ resp. } c = \text{enc\_asym}(p, \text{puK}).$$

Zum **Entschlüsseln** wird der Algorithmus dec_sym bzw. dec_asym genutzt, der sich aus Verschlüsselungsalgorithmus δ_sym resp. δ_asym und Formatierung ϕ zusammensetzen kann. Aus einem Kryptogramm c' und geheimem Schlüssel sK resp. privatem Schlüssel prK wird berechnet:

$$p' = \text{dec\_sym}(c', \text{sK}) \text{ resp. } p' = \text{dec\_asym}(c', \text{puK}).$$

Es gilt stets $p = \text{dec\_sym}(\text{enc\_sym}(p, \text{sK}), \text{sK})$ bzw. $p = \text{dec\_asym}(\text{enc\_asym}(p, \text{puK}), \text{prK})$ für jeden Klartext p.

In Tabellen 1.2 und 1.3 sind einige asymmetrische und symmetrische Verschlüsselungsverfahren aufgeführt.

Tabelle 1.2: Object Identifier für asymmetrische Verschlüsselungsverfahren (Auswahl)

OID-Ziffernfolge	OID-Distinguished Name	Beschreibung
{2 5 8 1 1}	id-ea-rsa	RSA Encryption
{1 2 840 113549 1 1 1}	RSA encryption	RSA, PKCS#1-Randomisierung
{1 3 6 1 4 1 8301 3 1 1 11}	IQElGamal Encryption	IQElGamal-Algorithmus

Für einen einheitlichen Aufbau der Verschlüsselungsverfahren aus kryptographischen Primitiven kann enc_sym bzw. enc_asym formal als enc_sym = η_sym $\circ$ ϕ resp. enc_asym = η_asym $\circ$ ϕ betrachtet werden, wobei ϕ in Abhängigkeit der Verschlüsselungsverfahren auch die identische Abbildung id sein kann, wenn sie nicht enthalten ist; z. B. RSA encryption = RSA $\circ$ PKCS#1, DES = DES $\circ$ id oder DESECB_ISOpad = DESECB $\circ$ ISO-Padding.

In der Praxis kommt aus Effizienzgründen meist eine Mischung aus symmetrischem und asymmetrischem Verschlüsselungsverfahren zum Einsatz:

[2] Die Begriffe „Verschlüsselungsverfahren“ und „Verschlüsselungsalgorithmus“ sind ähnlich und werden im allgemeinen Sprachgebrauch oft synonym benutzt, sind aber nicht immer dasselbe.

Tabelle 1.3: Object Identifier für symmetrische Verschlüsselungsverfahren (Auswahl)

OID-Ziffernfolge	OID-Distinguished Name	Beschreibung
{1 3 36 3 1 1}	DES	DES-Algorithmus
{1 3 36 3 1 1 1}	DESECB	DES im electronic codebook (ECB)-Modus
{1 3 36 3 1 1 2}	DESCBC	DES im cipher block chaining (CBC)-Modus
{1 3 36 3 1 1 2 1}	DESCBC_pad	DES (CBC) mit Padding
{1 3 36 3 1 1 2 1 1}	DESCBC_ISOpad	DES (CBC) mit ISO-Padding
{1 3 36 3 1 3}	DES_3	Triple-DES-Algorithmus
{1 3 36 3 1 3 2 1}	DES_3CBC_pad	Triple-DES (CBC) mit Padding
{1 3 36 3 1 3 2 1 1}	DES_3CBC_ISOpad	Triple-DES (CBC), ISO-Padding
{2 16 840 1 101 3 4 1 1}	id-aes128-ECB	AES (128 Bit Key), ECB-Modus
{2 16 840 1 101 3 4 1 2}	id-aes128-CBC	AES (128 Bit Key), CBC-Modus
{2 16 840 1 101 3 4 1 22}	id-aes192-CBC	AES (192 Bit Key), CBC-Modus
{2 16 840 1 101 3 4 1 42}	id-aes256-CBC	AES (256 Bit Key), CBC-Modus
{1 3 36 3 1 2}	IDEA	IDEA-Algorithmus

Definition 7
Ein **(hybrides) Verschlüsselungsverfahren** encrypt setzt sich aus einem symmetrischen und einem asymmetrischen Verschlüsselungsverfahren zusammen.

Ein Klartext p wird mit dem symmetrischen Verfahren enc_sym und einem zufällig gewählten Schlüssel sK – dem geheimen Session Key – verschlüsselt: $c = \text{enc\_sym}(p, \text{sK})$. Zum Schlüsselaustausch von sK wird das asymmetrische Verfahren genutzt: sK wird mit enc_asym und dem öffentlichen Schlüssel des Empfängers in das Kryptogramm $\text{sK}' = \text{enc\_asym}(\text{sK}, \text{puK})$ verwandelt. Übertragen wird

$$(c, \text{sK}').$$

Der Empfänger entschlüsselt ein Kryptogramm c' mittels Entschlüsselungsverfahren decrypt, welches die Berechnung

$$p' = \text{dec\_sym}(c', \text{dec\_asym}(\text{sK}', \text{prK}))$$

durchführt. Es gilt $p = \text{dec\_sym}(\text{enc\_sym}(p, \text{sK}), \text{dec\_asym}(\text{enc\_asym}(\text{sK}, \text{puK}), \text{prK}))$ für jeden Klartext p.

Zusammenfassend: **Ver-/Entschlüsselungs-Modell bei einer hybriden Verschlüsselung**:

Private Key des Empfängers	prK
Public Key des Empfängers	puK
Zu verschlüsselnder Klartext	p

Session Key (zufällig)	sK
Verschlüsselung	Kryptogramm c mit $c = \eta\_\mathrm{sym}(p, \mathrm{sK})$ verschlüsselter Session Key $\mathrm{sK}' = \eta\_\mathrm{asym}(\mathrm{sK}, \mathrm{puK})$
Übertragen	(c, sK')
Entschlüsselung	$sK = \delta\_\mathrm{asym}(\mathrm{sK}', \mathrm{prK})$ $p = \delta\_\mathrm{sym}(c, sK)$

An ein asymmetrisches Verschlüsselungsverfahren enc_asym und ein zugehöriges Schlüsselpaar $(\mathrm{prK}, \mathrm{puK})$ werden folgende Anforderungen gestellt:

1. Es ist praktisch nicht möglich, den privaten Schlüssel zu einem öffentlichen Schlüssel zu berechnen, auch nicht, wenn zusätzliches Datenmaterial zur Verfügung steht:
   (a) Es gibt keine effiziente Funktion f, die aus öffentlichem Schlüssel den privaten Schlüssel berechnet: $f(\mathrm{puK}) = \mathrm{prK}$.
   (b) Es gibt keine effiziente Funktion f', die aus öffentlichem Schlüssel und einer Menge von Klartext-Chiffretext-Paaren $\{(p, c)\}$ den privaten Schlüssel berechnet: $f'(\mathrm{puK}, \{(p, c)\}) = \mathrm{prK}$.
2. Es ist praktisch nicht möglich, aus einem Kryptogramm den zugehörigen Klartext ohne Kenntnis des privaten Schlüssels zu gewinnen, auch nicht, wenn zusätzliches Datenmaterial zur Verfügung steht:
   (a) Es gibt keine effiziente Funktion g, die aus einem Kryptogramm c den Klartext p berechnet: $g(c) = p$.
   (b) Es gibt keine effiziente Funktion g', die aus einem Kryptogramm c und einer Menge von Klartext-Chiffretext-Paaren $\{(p', c')\}$ den Klartext p berechnet: $g(c, \{(p, c)\}) = p$.

An ein symmetrisches Verschlüsselungsverfahren enc_sym wird die Anforderungen gestellt, dass es praktisch nicht möglich ist, aus einem Kryptogramm den zugehörigen Klartext ohne Kenntnis des geheimen Schlüssels zu gewinnen, auch nicht, wenn zusätzliches Datenmaterial zur Verfügung steht:

1. Es gibt keine effiziente Funktion g, die aus einem Kryptogramm c den Klartext p berechnet: $g(c) = p$.
2. Es gibt keine effiziente Funktion g', die aus einem Kryptogramm c und einer Menge von Klartext-Chiffretext-Paaren $\{(p', c')\}$ den Klartext p berechnet: $g(c, \{(p, c)\}) = p$.

Unter diesen Bedingungen bleibt ein Kryptogramm c zum Klartext p **vertraulich**. D. h. Verschlüsselungen gewährleisten die Vertraulichkeit übertragener Daten.

Dieser Abschnitt zeigte, wie mittels Public-Key-Kryptographie durch digitale Signaturen und Verschlüsselungen die Sicherheitsziele Herkunft, Unversehrtheit und Vertraulichkeit erfüllt werden können. Für die praktische Nutzung ist die Zuordnung eines Public Keys zu einem Benutzer entscheidend. Mit der dazu notwendigen Infrastruktur beschäftigt sich der nächste Abschnitt.

1.3 Public-Key-Infrastruktur

Zur Nutzung digitaler Signaturen und verschlüsselter Nachrichten mit Public-Key-Kryptographie ist die Kenntnis der öffentlichen Schlüssel der Kommunikationspartner notwendig und das Vertrauen in diese Schlüssel. Die Infrastruktur, die dieses leistet, ist eine Public-Key-Infrastruktur (PKI).

In diesem Abschnitt werden die Grundzüge einer PKI erklärt. Für Details sei auf den de facto Standard [Road00] oder die Literatur, wie z. B. [Buch99], verwiesen.

1.3.1 Aufbau einer Public-Key-Infrastruktur

Dieser Abschnitt beschäftigt sich mit der Frage, aus welchen Komponenten eine Public-Key-Infrastruktur bestehen muss, um Public-Key-Kryptographie praktisch einsetzen zu können.

Zunächst wird die Fragestellung durch ein Beispiel geschildert.

Alice und Bob
Bob hat Alice eine digital signierte E-Mail geschickt. Alice möchte die digitale Signatur prüfen. Dazu benötigt Alice Bobs echten öffentlichen Schlüssel. Es reicht nicht, dass Bob einfach seinen öffentlichen Schlüssel an die E-Mail anhängt, denn das könnte auch jemand getan haben, der nicht Bob ist. Bob könnte Alice seinen öffentlichen Schlüssel auf andere Weise mitteilen, z. B. bei einem Treffen. Das ist aber unpraktisch, wenn Bob von Alice weit weg wohnt.

Das benötigte Schlüsselmanagement wird über Zertifikate organisiert, in denen einem Benutzer ein öffentlicher Schlüssel zugeordnet wird. Zertifikate sind digital signiert. Nun hat sich das Problem verschoben, denn um den authentischen öffentlichen Schlüssel zu einem Teilnehmer zu erhalten, muss jetzt die digitale Signatur des Zertifikats verifiziert werden. Der Vorteil ist aber, dass nur noch einem(!) Schlüssel – dem Schlüssel der „Zertifizierungsinstanz“ – Vertrauen entgegengebracht werden muss, um mit sämtlichen Teilnehmern, die auch solche Zertifikate haben, sicher kommunizieren zu können.

Definition 8
Ein **Public-Key-Zertifikat** oder kurz **Zertifikat (Certificate)** ist eine von einer vertrauenswürdigen Instanz bestätigte „Verbindung“ zwischen einem **Zertifikatsinhaber (Certificate Holder – CH)** und einem öffentlichen Schlüssel und ist digital signiert.

Ein de facto Standard für Zertifikate ist das X.509-Zertifikat, das am weitesten verbreitet ist. Ein X.509-Zertifikat beinhaltet im Wesentlichen folgende Angaben, wobei für Details auf Appendix A.1 auf Seite 187 verwiesen sei:

- *Version* – Versionsnummer dieses Zertifikatstyps
- *Serial Number* – Seriennummer des Zertifikats
- *Signature Algorithm* – Signaturverfahren, mit dem dieses Zertifikat signiert wurde
- *Issuer* – Name der zertifizierenden Instanz
- *Validity* – Gültigkeitszeitraum
- *Subject* – Identität des Zertifikatsinhabers, was bei natürlichen Personen Land, Name und E-Mail-Adresse und bei Chipkarten die Chipkarten-Seriennummer sein kann

- *Subject Public Key Info* – Informationen zum öffentlichen Schlüssel des Zertifikatsinhabers, die aus dem öffentlichen Schlüssel und dem dazu gehörenden Signatur- oder Verschlüsselungsalgorithmus bestehen. Zum Signieren bzw. Verschlüsseln können dann verschiedene Verfahren genutzt werden, die diesen Algorithmus enthalten
- *Extensions* – Erweiterungen, die beispielsweise den Verwendungszweck (Key Usage) des Schlüssels ("Non-Repudiation" für Nicht-Abstreitbarkeit, "Key-Cert-Sign" und "CRL-Sign" für spezielle Anwendungen von "Non-Repudiation", "Digital Signature" in Authentisierungsprotokollen, "Key-Encipherment" und "Key-Agreement" beim Schlüsseltransport und -austausch und "Data-Encipherment" für Datenentschlüsselung), monetäre Einschränkungen („Bestellungen nur bis 500 Euro"), alternative Namen oder Eigenschaften des Zertifikatsinhabers (Arzt oder Mitarbeiter von Firma xyz.com) angeben
- *Signature Algorithm* – Signaturverfahren, mit dem dieses Zertifikat von der zertifizierenden Instanz signiert wurde
- *Signature Value* – Digitale Signatur, die das Zertifikat absichert

Es gibt noch weitere Arten von Zertifikaten, z. B. Attributzertifikate, die – ähnlich wie die *Erweiterungen* – eine bestimmte Funktion oder Eigenschaft zu einem Public-Key-Zertifikat bestätigen. Somit lassen sich temporäre Eigenschaften einfacher handhaben.

Ein Zertifikat ist zur Realisierung des in Abschnitt 1.1 auf Seite 1 erwähnten Sicherheitsziels „Zurechenbarkeit zu einem Benutzer" wichtig: Eine digitale Signatur kann Verbindlichkeit und damit Nicht-Abstreitbarkeit herstellen, weil die Zurechenbarkeit von einem öffentlichen Schlüssel zu einem Benutzer über ein Zertifikat von einer vertrauenswürdigen Instanz vorgenommen wurde, weil der Sender seinen privaten Schlüssel unter seiner alleinigen Kontrolle hält und weil er die Signieraktion vorher mit Passwort, PIN oder einem biometrischen Verfahren autorisieren musste.

Beispiel Zertifikat
Ein X.509-Zertifikat hat folgende Form:

```
Certificate:
   Data:
      Version: 3 (0x2)
      Serial Number: 94:ee:00:00:00:02:56:c4:db:87:87:42:22:25
      Signature Algorithm: md5WithRSAEncryption
      Issuer: C=DE, ST=Hamburg, L=Hamburg, O=TC TrustCenter for Security in
         Data Networks GmbH, OU=TC TrustCenter Class 1
         CA/Email=certificate@trustcenter.de
      Validity
         Not Before: Jan 28 13:18:41 2002 GMT
         Not After : Jan 28 13:18:41 2003 GMT
      Subject: C=DE, CN=Soenke Maseberg/Email=maseberg@sit.fraunhofer.de
      Subject Public Key Info:
         Public Key Algorithm: rsaEncryption
         RSA Public Key: (512 bit)
            Modulus (512 bit):
               00:c7:e1:9f:91:15:d8:51:ce:cf:3e:39:6b:de:64:
               b0:5a:14:d3:64:dc:a0:d2:b8:b3:3a:7f:5b:a1:73:
               75:4d:5f:a5:ee:cf:7e:68:dc:d9:8f:c6:d5:b6:fb:
               81:b0:09:26:43:09:69:c2:52:86:b6:d9:35:9d:2e:
```

```
            5d:cc:83:4f:f7
          Exponent: 65537 (0x10001)
    X509v3 extensions:
       Netscape CA Policy Url: http://www.trustcenter.de/guidelines
       Netscape Cert Type: SSL Client, S/MIME
       Netscape Revocation Url: https://www.trustcenter.de
          /cgi-bin/check-rev.cgi/94EE0000000256C4DB8787422225?
Signature Algorithm: md5WithRSAEncryption
Signature Value:
09:cd:b0:f6:cb:62:f2:c1:eb:de:ea:df:00:d3:80:fc:5e:3a:
2a:b3:59:70:6f:4f:be:a2:31:01:8d:da:79:01:52:c6:c5:a7:
cf:4e:3f:bd:74:f9:67:00:7c:f0:d0:64:e4:9e:6e:a8:cb:19:
10:08:e4:c5:3b:ad:b8:0f:3a:c8:22:95:be:e1:b1:03:0c:5c:
c2:5d:09:f0:d6:31:f0:20:07:fe:bc:ca:19:30:f8:dd:c5:21:
56:d7:b5:b5:2a:8b:27:52:22:df:33:3d:38:4d:81:06:c3:3c:
4b:74:b0:db:5c:a0:07:1d:8a:24:96:a7:2e:dd:24:0f:ee:20:
d3:25
```

Definition 9
Eine **Zertifizierungsinstanz (Certification Authority – CA)** gibt Zertifikate heraus und revoziert sie, falls sich Änderungen in den Zertifikatsangaben ergeben oder die Eignung der kryptographischen Verfahren erlischt. Die CA kann Schlüsselpaare erzeugen. Die CA prüft Schlüssel auf ihre kryptographische Eignung und auf Dublettenfreiheit.

Definition 10
Eine **Registrierungsinstanz (Registration Authority – RA)** bürgt für die Verbindung zwischen öffentlichem Schlüssel und Identitäten und Attributen der Zertifikatsinhaber. In der Registrierung wendet sich eine Person an eine RA und beantragt ein Zertifikat.

Definition 11
Eine **Sperrliste (Certificate Revocation List – CRL)** beinhaltet Zertifikatsnummern, die vor Ende ihrer Gültigkeit zurückgezogen – revoziert – wurden [CCRL99].

Ein Standard einer CRL ist in Appendix A.3 auf Seite 192 beschrieben.

Definition 12
Ein **Verzeichnis (Directory – DIR)** speichert Zertifikate und Sperrlisten und stellt sie zur Verfügung.

Ein Standard zum Zugriff auf das DIR-Verzeichnis ist das **Lightweight Directory Access Protocol (LDAP)** [LDAP99].

Eine Alternative zu Sperrlisten sind Zertifikats-Status-Anfragen. Ein Standard ist das **Online Certificate Status Protocol (OCSP)** (siehe Appendix A.4 auf Seite 194 und [OCSP99]). Mit OCSP kann bei einem OCSP-Verzeichnis der momentane Status eines Zertifikats erfragt werden. Da ein OCSP-Server eine CRL-Sperrliste als Datenbasis zur Beantwortung nutzen kann, muss OCSP nicht zwingend aktueller als CRL sein.

Werden Zertifikate auf ihre Gültigkeit – z. B. via LDAP oder OCSP – überprüft, so wird von der **Validierung** einer digitalen Signatur oder des zugehörigen Zertifikats gesprochen.

Definition 13
In einem **Trust Center** sind die Funktionalitäten aus Certification Authority und Registration Authority zusammengefasst sowie die Dienstleistungen für Verzeichnis, Sperrlisten und Zertifikats-Status-Anfragen. Einzelne Aufgabenbereiche können auch ausgelagert werden.

Der Zertifikatsinhaber benötigt zum Umgang mit Zertifikaten einen **Client**, der anwendungsabhängig

- digitale Signaturen erzeugt,
- digitale Signaturen und Zertifikate ausgehend vom öffentlichen Schlüssel der vertrauenswürdigen CA, dem **Sicherheitsanker – Trust Anchor**, verifiziert,
- Zertifikate validiert,
- Kryptogramme entschlüsselt und
- Klartexte verschlüsselt.

Zertifikate werden von der CA mit dem privaten Schlüssel, der zum Sicherheitsanker gehört, digital signiert. Diese Sicherheitsanker können als Schlüssel oder Selbstzertifikat beim Client abgelegt sein. Ein **Selbstzertifikat** ist ein Zertifikat, dass mit dem darin zertifizierten öffentlichen Schlüssel zu verifizieren ist.

Sicherheitsanker, private Schlüssel des Zertifikatsinhabers und kryptographische Verfahren werden in einer „sicheren Umgebung“ – einer so genannten **Personal Security Environment (PSE)** – aufbewahrt. In einer PSE werden auch die kryptographischen Operationen ausgeführt. Eine PSE soll dem Benutzer die Authentizität des Sicherheitsankers sowie den alleinigen Zugriff auf seine privaten Schlüssel und die Unauslesbarkeit der privaten Schlüssel garantieren. Letzteres ist wichtig, weil andernfalls beweistechnische Probleme auftreten können, wenn ein CH seinen privaten Schlüssel selbst veröffentlicht und behauptet, eine digitale Signatur nicht selbst geleistet zu haben.

PSEs können in Software oder in Hardware realisiert sein, wobei letztere eine höhere Sicherheit versprechen können. Eine besonders geeignete Hardware-PSE ist die **Chipkarte**, auch **Smart Card** oder **Integrated Circuit Card (ICC)** genannt. In der sicheren Umgebung des Trust Centers wird sie mit dem Sicherheitsanker versehen und auf den CH personalisiert. In diesem Prozess wird die Chipkarte mit Schlüsselpaaren und Zertifikaten versehen.

Es gibt zwei Modelle zur Schlüsselerzeugung: Entweder wird das Schlüsselpaar auf der Karte erzeugt, so dass der private Schlüssel die Karte nie verlässt, oder das Schlüsselpaar wird außerhalb der Karte erzeugt und in die Karte eingebracht. Obwohl neben Chipkarten weitere Hardware-PSEs, wie z. B. USB-Token oder Trusted Pocket Signer (TruPoSign) [TPS] existieren oder in der Entwicklung sind, wird im Folgenden stets stellvertretend von der ICC als Standard-Hardware-PSE gesprochen.

Abstrakt gesehen, hält ein Client kryptographische Verfahren, Sicherheitsanker und Schlüssel in einer PSE sicher verwahrt. Die PSE kann als Software oder als Chipkarte realisiert sein. Abbildung 1.1 illustriert dieses Modell.

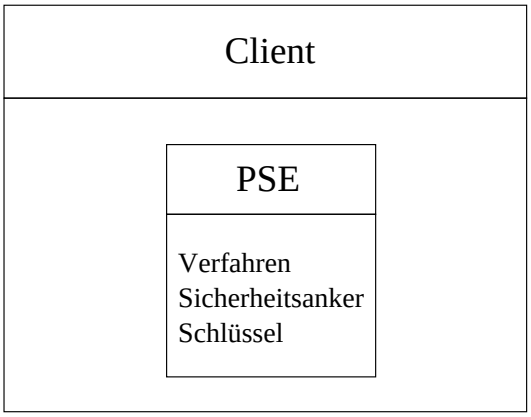

Abbildung 1.1: Modell eines Clients mit PSE

Mit dem Sicherheitsanker können Zertifikate verifiziert werden, so dass damit die authentischen öffentlichen Schlüssel anderer Teilnehmer vorliegen. Deshalb ist es theoretisch nicht nötig, fremde Zertifikate lokal abzuspeichern. Aus Effizienzgründen ist es aber üblich, Zertifikate lokal abzulegen.

Wieweit ein Benutzer einem Zertifikat vertrauen und damit die Zurechenbarkeit von Schlüssel zu Nutzer herstellen kann, hängt neben den kryptographischen Gesichtspunkten davon ab, wie die Zertifikatsinhaber im Trust Center authentisiert werden, nach welchen Prozeduren die CA Zertifikate erzeugt, wie die Sicherheit im Trust Center gewährleistet wird, wie die privaten Schlüssel verwahrt werden und welche rechtlichen Bedingungen gelten. All diese Aspekte werden in einer **Zertifikatspolicy (Certificate Policy)** oder kurz **Policy** festgehalten. Eine Policy wird auch ein **Certification Practice Statement (CPS)** genannt. Eine Policy ist eine Menge von Regeln, unter denen Zertifikate gewissen Sicherheitsforderungen genügen, so dass Vertrauen und Verbindlichkeit in ein Zertifikat entstehen kann. [CP99] gibt ein Muster vor, wie eine Policy aussehen kann:

- *Introduction* kennzeichnet Benutzer und Applikationen, für die diese Policy gedacht ist
- *General Provisions* legt rechtliche und finanzielle Bedingungen fest
- *Identification and Authentication* beschreibt den Ablauf, wie Zertifikatsinhaber im Trust Center authentisiert werden
- *Operational Requirements* definiert organisatorische Forderungen an Schlüssel- und Zertifikatsmanagement
- *Physical, Procedural, and Personnel Security Controls* beschäftigt sich mit der physischen Sicherheit des Trust Centers und Kriterien an das Personal und ihre Rollenaufteilung
- *Technical Security Controls* beschreibt Kriterien für die kryptographischen Komponenten
- *Certificate and CRL Profile* gibt Profile für Zertifikate und Sperrlisten vor
- *Specification Administration* behandelt das Management dieser Policies

Für Details sei auf Appendix A.9 auf Seite 215 verwiesen.

Die Gesamtheit dieser Komponenten wird in einer Public-Key-Infrastruktur zusammengefasst:

Definition 14
Eine **Public-Key-Infrastruktur** (PKI) ist eine Menge aus Hardware, Software, Beteiligten, Richtlinien (Policies) und Verfahren, um auf Public-Key-Kryptographie basierende Zertifikate zu erstellen, zu handhaben, zu verwahren, zu verteilen und zu revozieren [Road00].

Ein de facto Standard für Public-Key-Infrastrukturen ist "Internet X.509 Public Key Infrastructure (PKIX)", der von der "Internet Engineering Task Force (IETF) Working Group" erstellt wird [PKIX] [IETF]. Daneben gibt es noch weitere standardisierte Public-Key-Infrastrukturen, wie z. B. Pretty Good Privacy [PGP]. Die vorliegende Dissertation orientiert sich am PKIX-Standard.

Die "PKIX Roadmap" [Road00] gibt einen Überblick über die Aktivitäten von PKIX wieder. Abbildung 1.2 zeigt, wie sich PKIX eine PKI graphisch vorstellt: Es gibt eine Registration Authority (RA), eine Certification Authority (CA) und Certificate Holder (CH). RA, CA und ein CH kommunizieren untereinander über Managementtransaktionen. RA und CA versorgen ein Verzeichnis (DIR) mit Zertifikaten und Sperrlisten und ein CH kann sich Informationen dort abholen – z. B. via LDAP oder OCSP. Zwei Certificate Holder CH_1 und CH_2 nutzen eine Anwendung, über die sie elektronisch kommunizieren, und die PKI stellt die Sicherheitsfunktionen bereit.

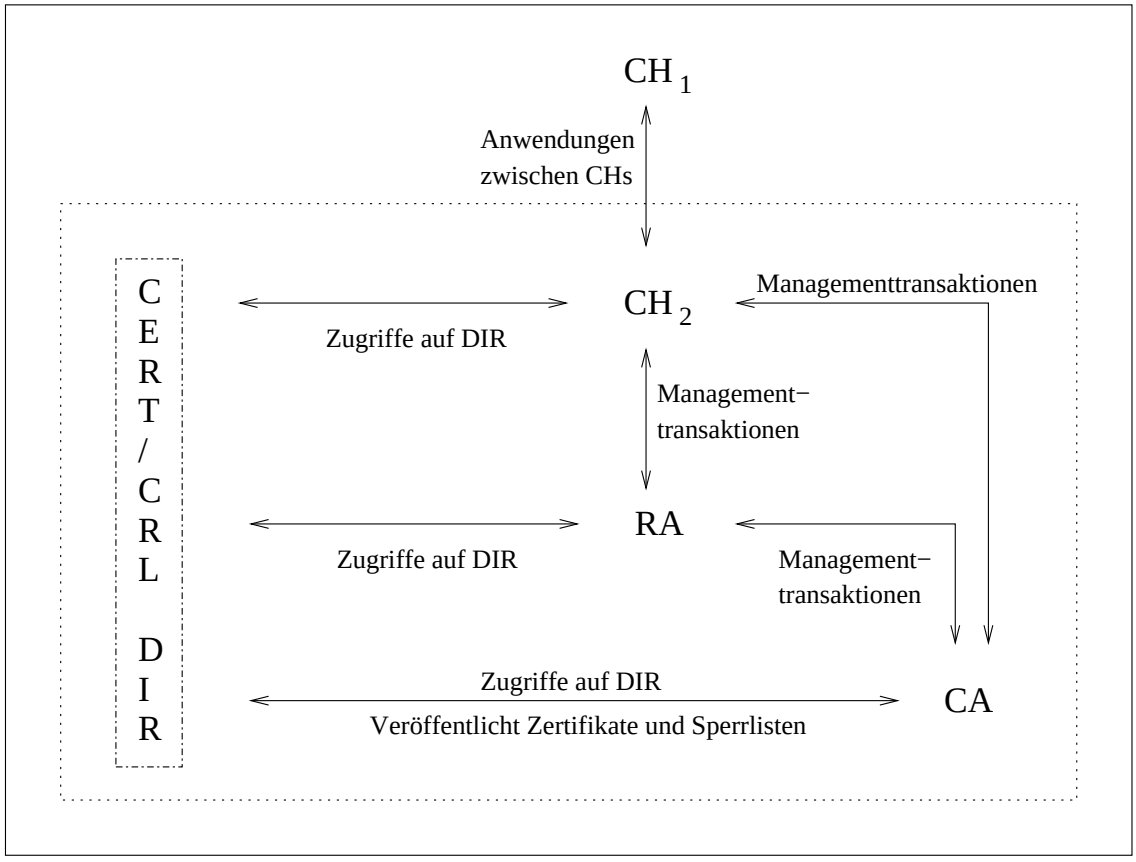

Abbildung 1.2: PKI-Struktur

Im einfachsten Fall zertifiziert die Zertifizierungsinstanz (CA) ausschließlich Zertifikatsinhaber (CH_1, CH_2, ...), so dass die PKI eine einstufige Hierachie ist (Abb. 1.3).

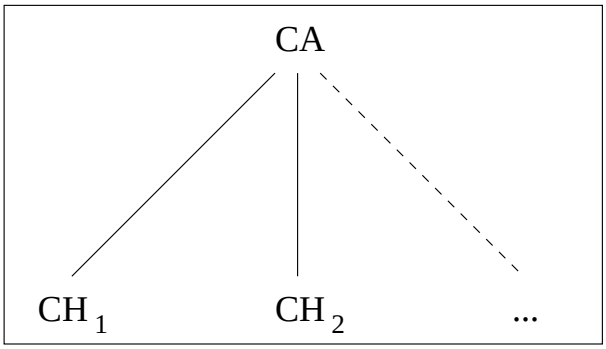

Abbildung 1.3: Eine einstufige Hierachie

Eine einstufige PKI lässt sich auf beliebig viele Stufen verallgemeinern, indem Zertifizierungsinstanzen wieder CAs zertifizieren. Abbildung 1.4 zeigt eine zweistufige PKI-Hierachie. Die – in diesem Bild – oberste CA heißt **Wurzelzertifzierungsinstanz (RootCA)**, die zwei CAs und einen CH zertifiziert. CA_1 zertifiziert ausschließlich Zertifikatsinhaber, während CA_2 zusätzlich eine weitere untergeordnete CA beglaubigt.

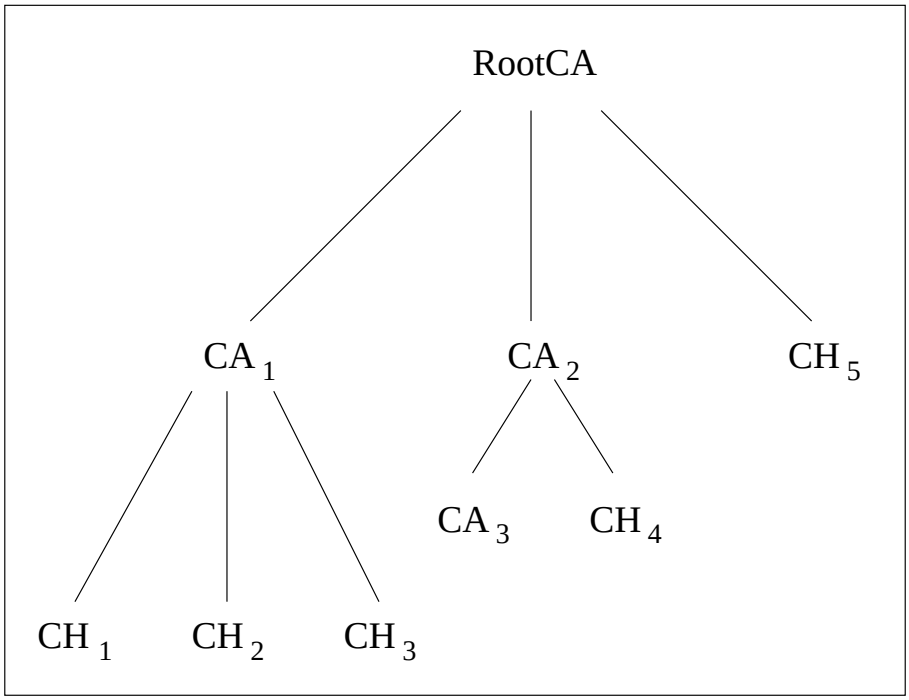

Abbildung 1.4: Eine zweistufige Hierachie

Vertrauen in einer PKI bildet sich vom Sicherheitsanker über Zertifikate und die daraus entstehende **Zertifizierungs-** bzw. **Zertifikatskette**.

Bei der Verifikation einer digitalen Signatur muss der gesamte **Verifikationspfad** entlang der

Zertifizierungskette verifiziert werden. Zur Verifikation einer digitalen Signatur gibt es zwei verschiedene **Gültigkeitsmodelle** [GiSc00]:

1. **Schalenmodell (Shell Model)**
   Beim Schalenmodell ist die digitale Signatur eines Dokuments D gültig, wenn die Signaturen aller Zertifikate im Verifikationspfad zum Signierzeitpunkt t_0 von D gültig waren. Dabei heißt ein Zertifikat dann gültig, wenn t_0 im Gültigkeitszeitraum des Zertifikats liegt und das Zertifikat zum Zeitpunkt t_0 nicht revoziert war. Das Schalenmodell wird auch als Hybrid- oder ISO-Modell bezeichnet.

2. **Kettenmodell (Chain Model)**
   Beim Kettenmodell ist die digitale Signatur eines Dokuments D' gültig, wenn alle im Verifikationspfad enthaltenen Zertifikate zum Zeitpunkt ihrer(!) Erstellung – also nicht zum Erstellungszeitpunkt von D' – gültig waren.

Grundsätzlich bleibt eine digitale Signatur auch über den Gültigkeitszeitraum des zugehörigen Zertifikats gültig, so lange die Signaturverfahren als kryptographisch geeignet gelten [GiSc00]. Ein Akzeptieren könnte jedoch beweistechnische Probleme nach sich ziehen, wenn sich nicht eindeutig feststellen lässt, dass der Signierzeitpunkt innerhalb des Gültigkeitszeitraumes des Zertifikats lag. Um diese Probleme zu vermeiden, können Signaturen „nachsigniert“ werden. Das Thema des Nachsignierens oder „Re-Signierens“ wird in Abschnitt 2.1.3 auf Seite 45 behandelt.

Die Sicherheit, das Vertrauen und die Verbindlichkeit in einer PKI beruhen auf drei Faktoren:

1. Kryptographisch geeignete Verfahren
2. Authentizität der Sicherheitsanker
3. Private Schlüssel, die unter der alleinigen Kontrolle des zugehörigen CHs liegen und die niemandem bekannt sind

1.3.2 Funktionsweise einer Public-Key-Infrastruktur

Auf Basis einer Public-Key-Infrastruktur lassen sich Authentizität und Integrität übertragener Dokumente durch Einsatz digitaler Signaturen, Vertraulichkeit durch Verschlüsselungen und die Prüfung von Identitäten durch Authentisierungsmechanismen realisieren.

In diesem Abschnitt werden die Techniken digitale Signatur, Verschlüsselung und Authentisierung erklärt sowie das in der Praxis dazu eingesetzte technische Equipment in verschiedenen Varianten diskutiert.

Technik der digitalen Signaturen

Digitale Signaturen gewährleisten Authentizität und Integrität eines Dokuments. Am Beispiel von Alice und Bob wird dargestellt, wie diese Technik funktioniert.

Alice und Bob
Alice will Bob von der Authentizität und Integrität eines Dokuments überzeugen. Angenommen, Alice und Bob verfügen über dasselbe Signaturverfahren, den Sicherheitsanker und jeweils über einen eigenen privaten Schlüssel und ein Zertifikat, das für die Nutzung als “Non-Repudiation“ vorgesehen ist. Non-Repudiation bedeutet Nicht-Abstreitbarkeit und wird benutzt, wenn digitale Signaturen im Sinne einer Unterschrift eingesetzt werden.

1. Alice erzeugt ein Dokument D.
2. Alice erzeugt mit einem Signaturverfahren eine digitale Signatur S zum Dokument D.
3. Alice verschickt (D, S) und optional zusätzlich ihr Zertifikat.
4. Bob besorgt sich das Zertifikat von Alice vom Verzeichnisdienst, falls er es nicht schon hat oder es mitgeliefert wurde.
5. Bob verifiziert mit seinem Sicherheitsanker das Zertifikat von Alice, wozu er bei einer mehrstufigen PKI-Hierachie die gesamte Zertifikatskette verifiziert.
6. Bob verifiziert mit dem ihm nun bekannten authentischen öffentlichen Schlüssel von Alice die Signatur S des Dokuments D.
7. Bob validiert die Zertifikate entsprechend Schalen- oder Kettenmodell, indem er Sperrlisten anfordert und prüft, ob auch keines der Zertifikate revoziert ist. Alternativ nutzt Bob die Zertifikats-Status-Anfrage.
8. Ist keines der geprüften Zertifikate gesperrt und sind die Signaturen korrekt, kann Bob davon ausgehen, dass niemand die Nachricht von Alice verfälscht hat und dass auch wirklich Alice diese Nachricht signiert hat.
9. Möchte Bob den Zeitpunkt eindeutig festhalten, um später nachweisen zu können, dass es zum jetzigen Zeitpunkt gültig signiert war, kann er das Dokument von einem Zeitstempeldienst mit einem Zeitstempel versehen lassen.

Eine **Mail** steht für jegliche Dokumente, die elektronisch verschickt werden – prominentes Beispiel ist die E-Mail. Aber auch Nachrichten in einer Bank-Anwendung sind eine Art von Mail. Eine Mail wird über den Mail Client des Absenders zu seinem Mail Server gesendet, welcher die Mail an den Mail Server des Empfängers weiterleitet, von wo der Empfänger die Mail mit seinem Mail Client abrufen kann. Ein Zertifikatsinhaber benötigt also – wie auf Seite 18 ausgeführt – zum Umgang mit digitalen Signaturen einen Client und eine PSE. Abbildung 1.5 illustriert das

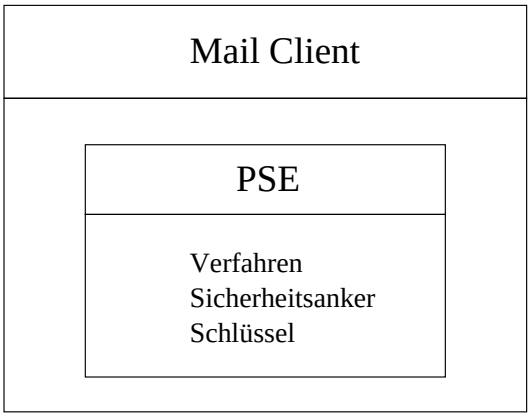

Abbildung 1.5: Modell eines Clients mit PSE

abstrakte Modell hierzu: Der Client hält die kryptographischen Verfahren, Sicherheitsanker und Schlüssel in einer PSE sicher verwahrt. Ein Anwendungsbeispiel dieses Modells ist der allein genutzte PC eines Benutzers mit einem E-Mail Client, wie etwa Netscape Messenger.

In der Praxis kommen darüber hinaus mehrere Varianten dieses Modells zur Anwendung:

Variante a)

Der Client wird von nicht nur einem, sondern von mehreren Benutzern genutzt – ein sogenannter Multi User Client. Deshalb sind in der (Software-)PSE im Client mehrere Profile der Benutzer angelegt. Das Profil eines jeden Zertifikatsinhabers stellt die Verbindung zu seinen Schlüsseln, Sicherheitsankern und Zertifikaten her. Ein Anwendungsbeispiel ist die Nutzung eines E-Mail Clients, wie z. B. Microsoft Outlook oder Netscape Messenger, von mehreren Benutzern. Abbildung 1.6 stellt diese Variante graphisch dar.

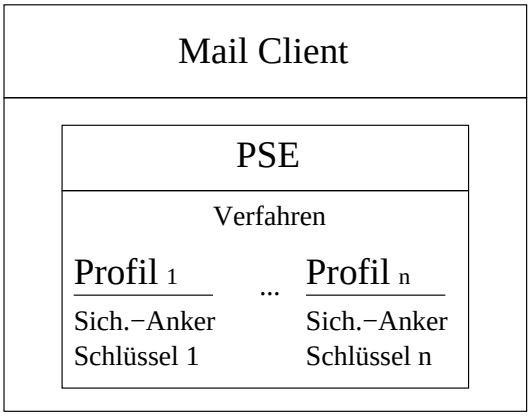

Abbildung 1.6: Variante a) des Modells: Client mit mehreren Profilen in einer (Software-)PSE

Hardware-PSEs können eine erhöhte Sicherheit gegenüber Software-PSEs bieten. Sie benötigen allerdings einen Client, mit dem sie angesprochen werden oder Kontakt zur „Außenwelt“ aufnehmen können. Vier Varianten des Modells mit Hardware-PSEs kommen in der Praxis zur Anwendung. Stellvertretend für Hardware-PSEs wird hier die Chipkarte (ICC) diskutiert.

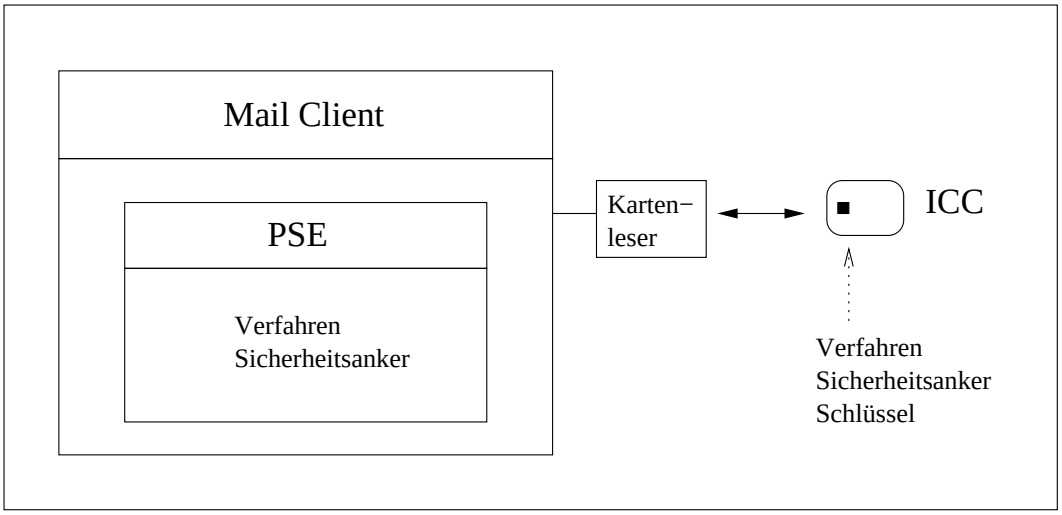

Abbildung 1.7: Variante b) des Modells: Client mit PSE und ICC

Variante b)

Variante b) unterscheidet sich vom Modell (Abb. 1.5) dadurch, dass der Benutzer eine ICC als

(Hardware-)PSE nutzt. Das Problem der Chipkarte sind ihre eingeschränkten Ressourcen, was Rechen- und Speicherkapazität angeht. Deshalb werden aus Effizienz- und Praktikabilitätsgründen gewisse Funktionalitäten der Chipkarte vollständig oder zum Teil nach außen auf den Client verlagert: Symmetrische Verfahren zum Ver- und Entschlüsseln, Hashfunktionen zum Hashen zu signierender oder verifizierender Daten in Arbeitsteilung (d. h. Client und ICC teilen sich das Hashen), asymmetrische Verfahren zum Verifizieren von digitalen Signaturen oder Entschlüsseln von Session Keys zusammen mit Sicherheitsankern, die in der (Software-)PSE im Client abgelegt sind. Ein Anwendungsbeispiel ist die Nutzung einer Signaturkarte, also einer von einem Trust Center personalisierten Chipkarte am eigenen PC. Abbildung 1.7 zeigt dieses Szenario.

Variante c)
Variante c) vereinigt die Varianten a) und b): Mehrere Benutzer teilen sich einen Client, d. h. es werden Profile der Benutzer mit einer (Software-)PSE angelegt, und jeder Benutzer hat eine ICC. Ein Anwendungsbeispiel ist die Nutzung von Signaturkarten an einem Client, der von mehreren Benutzern genutzt wird – etwa im Büro. Abbildung 1.8 illustriert diese Variante.

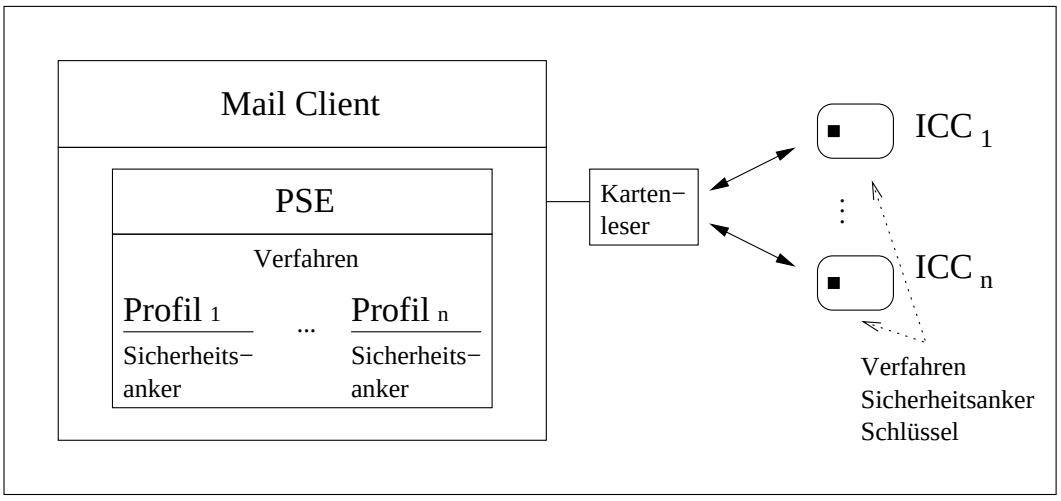

Abbildung 1.8: Variante c) des Modells: Client mit mehreren Profilen in einer (Software-)PSE und mehreren ICCs

Variante d)
Signaturkarten sollen auch an öffentlichen Terminals, etwa in Rathäusern oder Internet-Cafés genutzt werden können. In diesem Fall wird die ICC an einem für den Benutzer anonymen Client genutzt. Nach der DIN-Sig-Spezifikation [DINSig99] ist eine Authentisierung zwischen Client und Chipkarte nötig. Abbildung 1.9 zeigt den Aufbau dieser Variante.

Variante e)
Variante e) unterscheidet sich von Variante b) darin, dass im Client keine PSE mit Sicherheitsankern vorhanden ist. Optional sind auch keine Verfahren existent, so dass der Client ausschließlich für die Kommunikation der Chipkarte zur Außenwelt dient. Diese Variante kommt in besonders sicherheitskritischen Anwendungen zum Einsatz, wenn nur die Chipkarte kryptographische Operationen ausführen darf. Abbildung 1.10 zeigt den Aufbau dieser Variante.

Für Kommunikationen mit digitalen Signaturen gibt es zwei Standards: Secure/Multipurpose Internet Mail Extensions (S/MIME) und Public Key Cryptographic Standard #7 (PKCS#7) (siehe Appendizes A.7 und A.6 auf den Seiten 209 und 206).

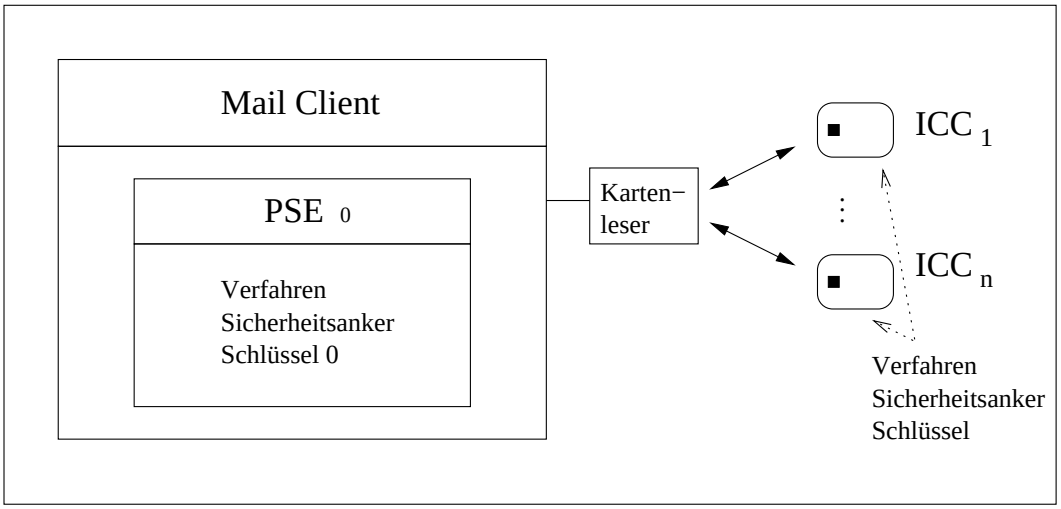

Abbildung 1.9: Variante d) des Modells: Client mit einer (Software-)PSE und mehreren ICCs

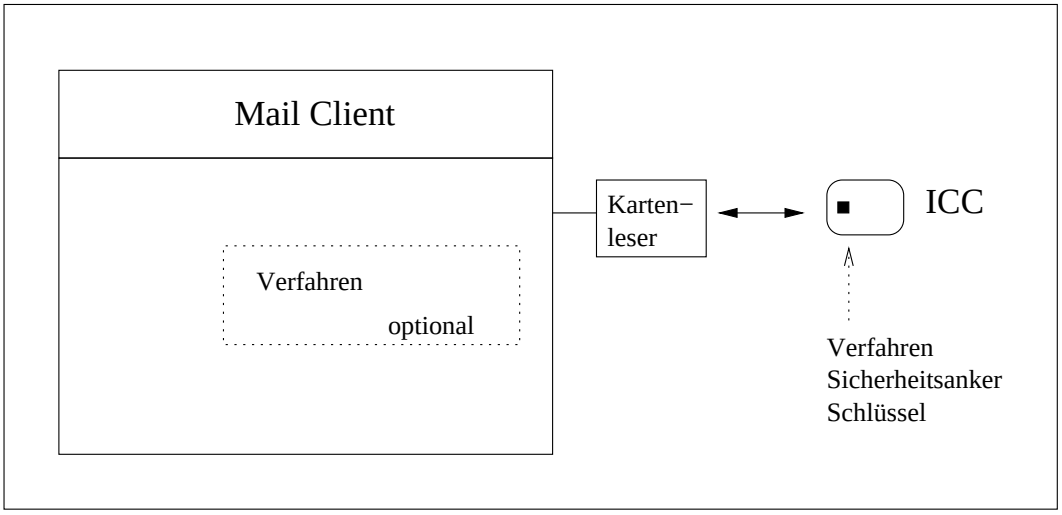

Abbildung 1.10: Variante e) des Modells: Client ohne PSE und mit einer ICC

Technik der Verschlüsselung (innerhalb einer PKI)

Verschlüsselungen gewährleisten Vertraulichkeit übertragener Daten. In diesem Abschnitt schildern Alice und Bob, wie sie innerhalb einer PKI, und ohne vorher einen geheimen Schlüssel ausgetauscht zu haben, Dokumente ver- und Kryptogramme entschlüsseln können.

Alice und Bob

Alice will Bob eine vertrauliche Nachricht D schicken. Angenommen, Alice und Bob verfügen beide über dasselbe symmetrische und dasselbe asymmetrische Verschlüsselungsverfahren, außerdem über einen dazugehörigen privaten Schlüssel und ein Zertifikat, das für "Key-Encipherment" vorgesehen ist. Key-Encipherment bedeutet, dass der öffentliche Schlüssel zum Schlüsselaustausch des Session Keys benutzt werden kann, welcher die eigentliche Verschlüsselung von D mit einem symmetrischen Verfahren effizient vornimmt.

1. Alice besorgt sich Bobs Zertifikat und entnimmt ihm Bobs öffentlichen Schlüssel $\mathrm{puK}_{\mathrm{Bob}}$.
2. Alice generiert einen zufälligen Session Key sK.
3. Alice verschlüsselt D mit dem symmetrischen Verfahren enc_sym und dem Session Key sK, und anschließend den Session Key sK mit dem asymmetrischen Verfahren enc_asym und Bobs öffentlichen Schlüssel puK: $D' = \mathrm{enc\_sym}(D, \mathrm{sK})$ und $\mathrm{sK}' = \mathrm{enc\_asym}(\mathrm{sK}, \mathrm{puK}_{\mathrm{Bob}})$.
4. Alice verschickt (D', sK').
5. Bob berechnet $\mathrm{sK} = \mathrm{dec\_asym}(\mathrm{sK}', \mathrm{prK}_{\mathrm{Bob}})$ und anschließend $D = \mathrm{dec\_sym}(N', \mathrm{sK})$.

Das für diesen Umgang nötige Equipment ist zum zuvor beschriebenen Modell samt seiner Varianten identisch. Die Standardprotokolle S/MIME und PKCS#7 unterstützen Verschlüsselungen.

Für Datei-Verschlüsselungen taugt ebenfalls die oben geschilderte Technik, wobei Alice eine Datei mit ihrem eigenen öffentlichen Schlüssel ver- und mit ihrem privaten Schlüssel wieder entschlüsselt.

Für lokale Datei-Verschlüsselungen ist allerdings eine Public-Key-Infrastruktur nicht zwingend notwendig. Stattdessen ist es effektiver, eine Datei mit einem symmetrischen Verfahren zu ver- und entschlüsseln, wobei der Secret Key aus einem Passwort abgeleitet werden kann. Zur Verschlüsselung von Daten, auf die eine Gruppe von Leuten Zugang haben soll oder um ein Key Recovery von privaten Schlüsseln bieten zu können, ist eine PKI geeignet. Dazu gibt es einige Programme, wie z. B. Pretty Good Privacy [PGP]. Die Idee ist, dass es ein Schlüsselpaar des „Projekts“ gibt, an dem eine Gruppe von Leuten arbeitet. Der Public Key des „Projekts“ ist öffentlich zugänglich und der Private Key wird jedem Mitglied der Gruppe verschlüsselt mitgeteilt, wozu ein Administrator benötigt wird. Eine Zusammenfassung zum Thema Gruppenverschlüsselung findet sich z. B. in [EcEG00].

Da das Prinzip der Datei-Verschlüsselung mit der zuvor in Abschnitt 1.3.2 vorgestellten Technik der Verschlüsselung identisch ist, wird auf das Thema Datei-Verschlüsselung in dieser Arbeit nicht näher eingegangen.

Technik der Authentisierung

Eine wichtige und vielbenutzte Technik mit Public-Key-Methoden ist die Authentisierung von Rechnern im Internet, welche aber auch bei realen Authentisierungen von Personen mit Chipkarten angewendet wird. Die Funktionsweise wird anhand von Alices und Bobs Rechnern dargestellt.

In einer **Client-Server-Architektur** wird als **Client** ein Softwareprogramm bezeichnet, der mit einem **Server** verbunden ist, welcher – ebenfalls ein Softwareprogramm – bestimmte Dienste bereitstellt. Mehrere Clients greifen auf einen Server zu. Fast alle Dienste im Internet basieren auf der Client-Server-Technologie, z. B. in Form von E-Mail- oder Web-Servern.

Alice und Bob
Alices Rechner – als der Client – will sich gegenüber Bobs Rechner – dem Server – authentisieren. Angenommen, Client und Server verfügen über dasselbe Signaturverfahren, den Sicherheitsanker und jeweils über einen eigenen privaten Schlüssel und ein Zertifikat, das für die Nutzung als “Digital Signature“ vorgesehen ist. Die Intention von Digital Signature ist die Nutzung von Schlüsseln und Zertifikaten in Authentisierungsprotokollen.

1. Der Client meldet sich beim Server, etwa mit einem “Hello!“

2. Der Server erzeugt eine zufällige Challenge RND und schickt RND an den Client.

3. Der Client signiert die Challenge RND und schickt die digitale Signatur S – als Response – zusammen mit seinem Zertifikat an den Server.

4. Der Server verifiziert das Zertifikat vom Client und anschließend mit dem ihm nun bekannten authentischen öffentlichen Schlüssel des Clients die Signatur S der Challenge.

5. Der Server validiert involvierte Zertifikate, indem er Sperrlisten anfordert und prüft, ob diese auch gültig sind. Alternativ kann er den Status eines Zertifikats abfragen.

6. Ist keines der involvierten Zertifikate gesperrt, sind die Signaturen korrekt und ist die korrekte Challenge RND signiert worden, kann der Server davon ausgehen, dass der im Zertifikat angegebene Client mit ihm kommuniziert. In Anbetracht von Man-in-the-Middle-Angriffen kann allerdings ein Angreifer dazwischen mithören. Deshalb wird meist im Laufe dieses Protokolls ein Sitzungsschlüssel (Session Key) zwischen Client und Server ausgehandelt und in die Signaturen integriert, mit dem sie ihre weitere Kommunikation mit einem symmetrischen Verfahren verschlüsseln, siehe auch Appendix A.8 auf Seite 211.

7. Um dem spezifischen Client nun den Zugang zu Dienstleistungen zu gestatten, hält der Server eine Datenbank vor oder im Zertifikat selbst sind bestimmte Attribute festgelegt.

Bei einer gegenseitigen Authentisierung übernimmt abwechselnd jeder der Teilnehmer die Rolle von Client und Server. Im Unterschied zur Signatur-Anwendung hat ein Client in einer Authentisierungs-Anwendung nach Ausgabe des Zertifikats im Allgemeinen keinen Kontakt mehr zum Trust Center.

Je nach Authentisierungs-Anwendung variieren Zertifikatsinhaber, Client und PSE:

- Bei einer **virtuellen Zugangskontrolle** authentisiert sich ein Rechner C gegenüber einem Rechner S. C und S verfügen jeweils über kryptographische Verfahren und eine integrierte

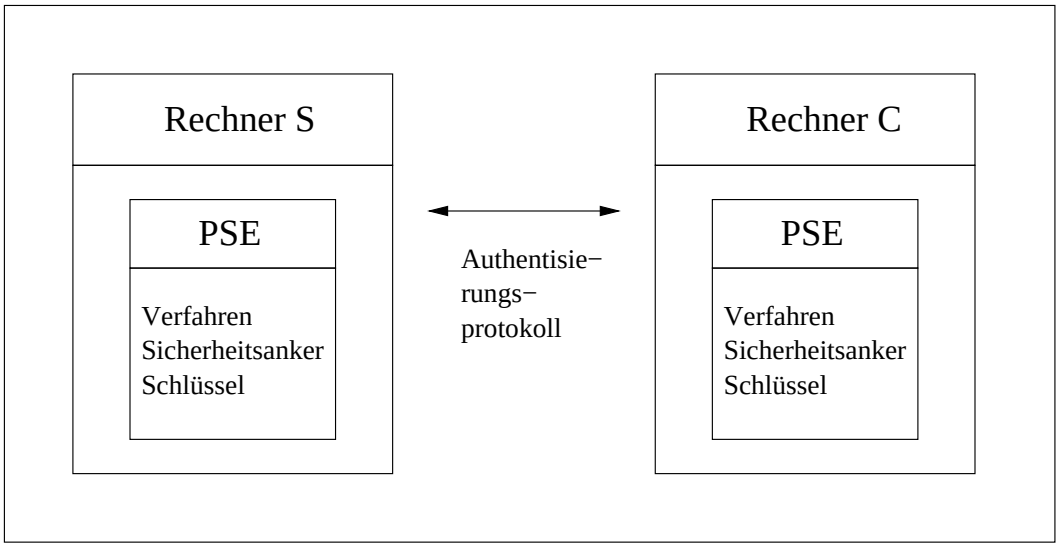

Abbildung 1.11: Virtuelle Zugangskontrolle zwischen zwei Rechnern

Software-PSE mit Schlüsseln und Sicherheitsankern. Zertifikatsinhaber ist der Rechner selbst oder der Benutzer des Rechners. S (als Server) und C (als Client) entsprechen jeweils dem Modell "Client mit PSE" auf Seite 23 in Abbildung 1.5. Abbildung 1.11 illustriert diese Situation.

- Bei einer **realen Zugangskontrolle** authentisiert sich eine Chipkarte gegenüber einem Zugangskontroll-Client mit Kartenleser. ICC und PSE des Zugangskontroll-Client verfügen jeweils über Verfahren, Sicherheitsanker und Schlüssel. Certificate Holder und Chipcard Holder ist eine natürliche Person, die sich an einem Tor zu einer Firma oder einer Tür zu einem Raum authentisieren muss. Diesem Szenario entspricht Variante d) auf Seite 26 in Abbildung 1.9. Abbildung 1.12 stellt dieses Szenario graphisch dar.

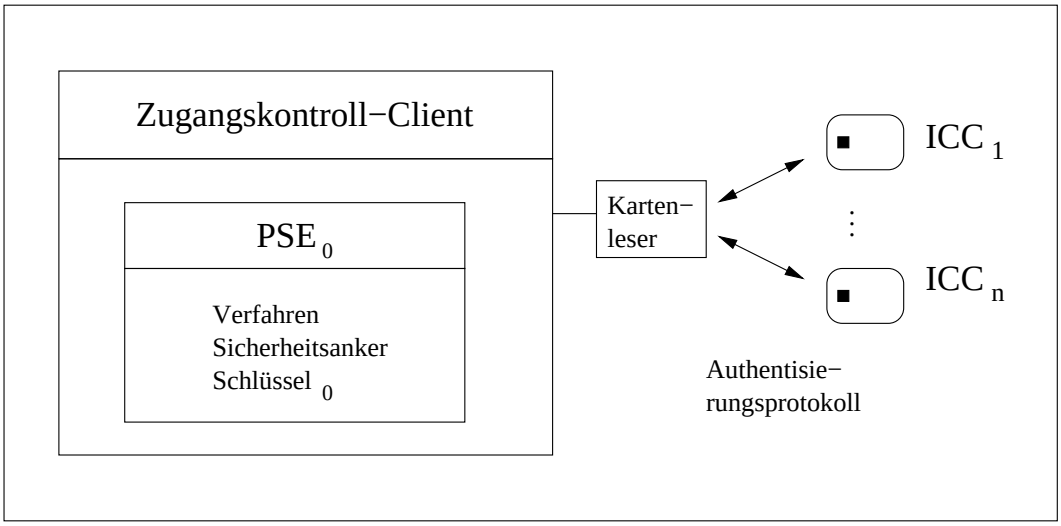

Abbildung 1.12: Reale Zugangskontrolle zwischen Chipkarte und Zugangskontroll-Client

Für Authentisierung ist Transport Layer Security (TLS) / Secure Socket Layer (SSL) ein Standard (siehe Appendix A.8 auf Seite 211).

Für reale Zugangskontrollen gibt es eine Spezifikation für einen als Chipkarte ausgelegten Dienstausweis: Die Office Identity Card (OIC) [OIC00].

Jede Anwendung hat ein individuelles Sicherheitsbedürfnis. Die Abbildung dieses Sicherheitsbedürfnisses auf eine konkrete Public-Key-Infrastruktur führt zu einer **Sicherheitsrichtlinie (Security Policy)**, in der festgelegt wird, zu welchen Teilen die zuvor beschriebenen Verfahren genutzt werden. Die Security Policy umfasst Nutzung digitaler Signaturen, Umfang der Verifikation der Zertifikate entlang der Zertifikatskette, Umfang der Pfadvalidierung, Nutzung von Verschlüsselungen oder Nutzung von Authentisierungen.

Als eine **sicherheitskritische Anwendung** wird eine Anwendung bezeichnet, die kryptographisch geeignete Verfahren, auf einer PSE sicher abgelegte Sicherheitsanker und private Schlüssel sowie einen Revokationsmechanismus – etwa mit CRL oder OCSP – nutzt.

1.4 Anwendungen

An Ideen, wie Public-Key-Infrastrukturen genutzt werden können, fehlt es nicht: E-Commerce, virtuelle Rathäuser, digitale Wahlen oder Homebanking sind einige Beispiele. Quasi alle Vorgänge, die bisher papiergebunden mit einer manuellen Unterschrift versehen wurden, wurden und werden geprüft, ob sie sich sinnvoll elektronisch realisieren und nutzen lassen.

Ein Teil dieser Vorschläge ist bereits realisiert worden. Zumeist hapert es dort, wo die digitale Signatur als Pendant zur handschriftlichen Unterschrift eingesetzt werden soll und die rechtliche Gleichstellung von „händischer“ und „elektronischer“ Unterschrift noch nicht gegeben ist. Die Bemühungen auf deutscher und europäischer Ebene, diese rechtliche Gleichstellung herzustellen, setzen auf hohe technische Forderungen, unter deren Nutzung eine Gleichstellung erreicht sein kann.

Da die Technik schon heute zur Verfügung steht, werden PKIs bereits dort eingesetzt, wo es keine rechtlichen Probleme gibt oder gelöst sind, wie z. B. in firmeninternen Abläufen oder der Kommunikation zwischen Unternehmen. In solchen Fällen werden manuelle und elektronische Signaturen durch Verträge gleichgesetzt.

Dieser Abschnitt gibt zunächst die jetzige juristische Situation und anschließend eine Auswahl heutiger Anwendungen wieder.

1.4.1 Rechtliche Rahmenbedingungen

Digitale Signaturen sind ein Äquivalent zur händischen Unterschrift. Wie verbindlich sind diese elektronischen Unterschriften? Die rechtlichen Bestimmungen zur Gleichstellung handschriftlicher Unterschriften und digitaler Signaturen hängen davon ab, ob sich Kommunikationspartner über entsprechende Richtlinien, Verordnungen und Verträge einigen oder ob über das Signaturgesetz eine Äquivalenz hergestellt wird. Dieser Abschnitt beleuchtet juristische Aspekte.

Richtlinien, Verordnungen und Verträge

Bei internen Arbeitsabläufen in Firmen regeln Firmenrichtlinien die Gleichstellung von digitalen Signaturen und händischen Unterschriften. Im Bereich des Business-to-Business, d. h. dem Handel zwischen Unternehmen, gelten Verträge, die den Umgang und die Verbindlichkeit von digitalen Signaturen regeln.

In Behörden gelten Verordnungen, die z. T. bereits an das elektronische Zeitalter angepasst wurden. Beispielsweise regelt der Abschnitt „Das maschinell geführte Grundbuch“ der Grundbuchordnung (GBO) die Bedingungen, unter denen das Grundbuch in maschineller Form geführt werden kann [GBO, Siebenter Abschnitt].

Signaturgesetz

Welchen Beweiswert hat eine digitale Signatur vor einem Gericht? Auf Grund des Prinzips der freien Beweiswürdigung kann ein Richter eine digitale Signatur anerkennen, wenn die gegebenen „Umstände“ dies nahe legen [DiSc01].

Sinn und Zweck des Signaturgesetz ist es, Rahmenbedingungen für elektronische Signaturen zu schaffen, um dadurch diese „Umstände“ eindeutig festzulegen. Seit 1997 gibt es das deutsche „Gesetz zur digitalen Signatur“ (SigG) [SigG97] und die dazugehörige Signaturverordnung (SigV)

[SigV97] sowie eine Begründung der Verordnung [SigV-B]. Vom Dezember 1999 ist die EU-Richtlinie „über gemeinschaftliche Rahmenbedingungen für elektronische Signaturen“ [EU99], die im Mai 2001 in nationales Recht, in Form des neuen Signaturgesetzes [SigG01], umgesetzt wurde. Im November 2001 wurde eine neue Signaturverordnung [SigV01] zum neuen Signaturgesetz verabschiedet.

Im Gesetz zur „digitalen Signatur“ ist stets von „elektronischen Signaturen“ die Rede, die es in drei verschiedenen Ausprägungen gibt. Zitat aus [SigG01, §2 Begriffsbestimmungen]:

„Im Sinne dieses Gesetzes sind

1. ‚elektronische Signaturen‘ Daten in elektronischer Form, die anderen elektronischen Daten beigefügt oder logisch mit ihnen verknüpft sind und die zur Authentifizierung dienen,
2. ‚fortgeschrittene elektronische Signaturen‘ elektronische Signaturen nach Nummer l, die
   (a) ausschließlich dem Signaturschlüsselinhaber zugeordnet sind,
   (b) die Identifizierung des Signaturschlüsselinhabers ermöglichen,
   (c) mit Mitteln erzeugt werden, die der Signaturschlüsselinhaber unter seiner alleinigen Kontrolle halten kann, und
   (d) mit den Daten, auf die sie sich beziehen, so verknüpft sind, dass eine nachträgliche Veränderung der Daten erkannt werden kann,
3. ‚qualifizierte elektronische Signaturen‘ elektronische Signaturen nach Nummer 2, die
   (a) auf einem zum Zeitpunkt ihrer Erzeugung gültigen qualifizierten Zertifikat beruhen und
   (b) mit einer sicheren Signaturerstellungseinheit erzeugt werden. [...]“

Welches Verhältnis haben eine „digitale Signatur“ nach Definition 1 auf Seite 3 und eine „elektronische Signatur“ nach SigG? Eine „elektronische Signatur“ kann – auch ganz ohne Kryptographie – der Name unter einer E-Mail sein oder – mit Kryptographie – ein Hashwert. Eine „fortgeschrittene elektronische Signatur“ ist das, was in Definition 1 unter dem Begriff „digitale Signatur“ definiert wurde. „Qualifizierte elektronische Signaturen“ sind „digitale Signaturen“, die das Kettenmodell (siehe Seite 22) und eine sichere Signaturerstellungseinheit nutzen. Eine sichere Signaturerstellungseinheit wird in [EU99] als “Secure Signature Creation Device (SSCD)“ bezeichnet. Ein Ansatz, ein SSCD zu realisieren, ist eine Chipkarte, die mit einem weiteren Gerät korrespondieren muss, weil „die Korrektheit der digitalen Signatur zuverlässig geprüft und zutreffend angezeigt“ [SigV01] werden muss. Das Problem sicherer Anzeigekomponenten wird hier nicht thematisiert; ein Ansatz wird im Rahmen von Vernet [Vernet] in Form des “Trusted Pocket Signers (TruPoSign)“ [TPS] verfolgt.

Abweichend zu den rechtlichen Bezeichnungen wird in dieser Dissertation der Begriff der „digitalen Signatur“ beibehalten.

Wie erwähnt, macht das Signaturgesetz keine Aussagen zur Gleichstellung von händischen Unterschriften und digitalen Signaturen. Diese Gleichstellung ist inzwischen durch neue Gesetze erreicht worden: „So wurden mit dem §126a BGB elektronische Dokumente, die mit einer qualifizierten elektronischen Signatur versehen sind, als ‚elektronische Form‘ definiert, die als Ersatz zur Schriftform eingesetzt werden kann. Für diese elektronische Form gilt nach §292a ZPO der Anschein der Echtheit“ [BPRS02]. D. h. eine qualifizierte digitale Signatur ermöglicht eine Beweiserleichterung.

1.4.2 Beispiele für Signatur-Anwendungen

Digitale Signaturen und Verschlüsselungen werden bereits vielfältig genutzt. Dieser Abschnitt definiert den Begriff einer Signatur-Anwendung, welche – grob gesprochen – das elektronische Pendant zum herkömmlichen Briefverkehr darstellt, und gibt zahlreiche Anwendungsbeispiele aus den Bereichen der PKI selbst, Intra-PKIs, E-Commerce und E-Government. Dabei ist unter „E-Commerce“ der über Datennetze abgewickelte Geschäftsverkehr zu verstehen, der im Bereich des Handels zwischen Unternehmen (Business-to-Business) und zwischen Unternehmen und Endkunden (Business-to-Customer) angewendet wird. „E-Government“ ist die Bezeichnung für die Kommunikation zwischen Behörden und Bürgern oder Unternehmen.

Definition 15
Eine **Signatur-Anwendung**[3] ist eine Anwendung der Technik der digitalen Signatur und optional der Technik der Verschlüsselungen, welche das Pendant zum papiergebundenen Briefverkehr in einer elektronischen Kommunikation darstellt. Die digitale Signatur ist das Pendant zur händischen Unterschrift und kann dieselben rechtlichen Verbindlichkeiten beinhalten, und die Verschlüsselung ist das digitale Pendant zum Verschließen eines Briefumschlags.

In welchen Bereichen wird eine solche Signatur-Anwendung heutzutage bereits genutzt und wo sind für die Zukunft Anwendungen geplant?

Aufbau der PKI

Zertifikate selbst stellen ein Beispiel für eine Signatur-Anwendung dar, denn eine Zertifizierungsinstanz verbürgt sich für den Inhalt des von ihr unterschriebenen, d. h. signierten, Zertifikats. Eine Zertifizierungsinstanz haftet nach dem neuen Signaturgesetz auch für diesen Inhalt, siehe [SigG01, §11].

Darüber hinaus werden innerhalb der PKI digitale Signaturen im Sinne einer Unterschrift bei den Sperrlisten (CRL), bei den Antworten auf Zertifikats-Status-Anfragen (OCSP) und bei den Zeitstempeln eingesetzt. Die CA bestätigt Revokationsinformationen und ein **Zeitstempeldienst (Time Stamping Service – TSS)** bestätigt mit einem **Zeitstempel**, dass ihm zu einem Zeitpunkt der Hashwert eines Dokuments vorgelegen hat. Der TSS wird von einer vertrauenswürdigen **Zeitstempelinstanz (Time Stamping Authority – TSA)** betrieben.

Intra-PKIs

Firmen- oder behördenintern werden Public-Key-Infrastrukturen als so genannte „Intra-PKIs“ schon heute genutzt. Gründe für die Umstellung von papiergebundenen auf elektronische Medien sind Zeit- und Kostenersparnisse.

Firmeninterne PKIs werden für die Kommunikation innerhalb eines Unternehmens und insbesondere zwischen verschiedenen Standorten oder für Mitarbeiter auf Reisen genutzt, und außerdem, um Arbeitsprozesse zu protokollieren und eine eindeutige Zuordnung vom Arbeitsprozess zum Durchführenden zu erhalten. Je nach Sicherheitsbedürfnis findet eine Validierung der digitalen Signatur in jedem Signatur- oder Verifikationsprozess oder nur im Streitfall statt. Es gibt auch auf symmetrischen Verfahren basierende unternehmensinterne Infrastrukturen, die hier jedoch nicht betrachtet werden.

[3] Der Begriff orientiert sich an [DINSig99] und hat sich etabliert, obwohl die Begriffe „unterschriebener Brief-Anwendung“ oder „Postanwendung“ vielleicht passender wären.

In einigen Grundbuchämtern Deutschlands existiert bereits das **elektronische Grundbuch**, oder die Einführung steht bevor. Neben dem Wunsch, die interne Bearbeitung zu beschleunigen, stehen zwei Gründe für die Umstellung von Papier auf die elektronische Form im Vordergrund: Grundbuchinformationen sollen an Personen mit berechtigtem Interesse vereinfacht nach außen gelangen können und vor allem soll die Papiermenge massiv reduziert werden, denn das Grundbuchamt in München beispielsweise muss etwa alle 10 Jahre anbauen.

„Personen mit berechtigem Interesse“, sind „Gerichte, Behörden, Notare, öffentlich bestellte Vermessungsingenieure, an dem Grundstück dinglich Berechtigte, eine von dinglich Berechtigten beauftragte Person oder Stelle und die Staatsbank Berlin“ [GBO, §133]. „An die Stelle der Abschrift tritt der Ausdruck und an die Stelle der beglaubigten Abschrift der amtliche Ausdruck. [...] Der amtliche Ausdruck ist als solcher zu bezeichnen und mit einem Dienstsiegel oder -stempel zu versehen; er steht einer beglaubigten Abschrift gleich“ [GBO, §131]. Derzeit haben die „einfachen“ Ausdrucke, egal ob im Grundbuchamt oder extern ausgedruckt, keine rechtliche Relevanz. Die amtlichen Ausdrucke werden nur im Grundbuchamt auf besonderem Papier ausgedruckt und mit einem „elektronischen Siegel“ versehen, hinter dem sich allerdings kein kryptographischer Prozess, sondern das Bild eines Stempels verbirgt. Der Grund, dass amtliche Ausdrucke nicht extern erstellt werden, liegt in den Gebühren, die anfallen und noch nicht elektronisch beglichen werden können.

Ein elektronisches Grundbuch setzt sich aus zwei Teilen zusammen:

1. “coded information“ – die Bestandteile, die bereits elektronisch erzeugt wurden
2. “non-coded information“ – die abgescannten und als Bild gespeicherten „alten“ papiergebundenen Grundbuchblätter

Das Anwendungsprogramm fügt diese beiden Teile so zusammen, dass dem Rechtspfleger auf dem Bildschirm im Hintergrund das eingescannte „alte“ Grundbuchblatt und darüber die nachträglich erzeugten „neuen“ elektronischen „coded information“ angezeigt werden.

Für die interne Bearbeitung des E-Grundbuchs existieren zwei gänzlich verschiedene Verfahren: Die Siemens-Lösung namens „Solum Star“ wird derzeit im Grundbuchamt Bremen und in Grundbuchämtern Hessens und Bayerns eingeführt und nutzt keine Public-Key-Infrastruktur. Die zweite existierende Lösung arbeitet mit Public-Key-Methoden, kommt von Debis und wird zum Beispiel im Justiz-Ministerium Baden-Württemberg eingesetzt.

Das Verfahren von Debis arbeitet wie folgt: Das Grundbuch wird dem Rechtspfleger elektronisch dargestellt. Zum Abschluss kann der Rechtspfleger einen ergänzten Grundbucheintrag signieren. Es wird RSA mit 1024 Bit langen Schlüsseln genutzt. Der private Schlüssel des Rechtspflegers befindet sich auf einer Smart Card. Im Signiervorgang werden Datensätze mit einer digitalen Signatur versehen. Daten und Signatur werden zum Anwendungs-Server transportiert. Dort werden die Signatur und entsprechende Zertifikate verifiziert und mittels Sperrlisten validiert. Bei positiver Prüfung wird das Grundbuchblatt zusammen mit dem Datum und einer digitalen Signatur des Servers auf zwei Magnetplatten abgespeichert. Daneben werden Protokolldaten erstellt. Die Sicherheit beruht neben der PKI-Technik auch auf einem Back Up, also auf einem physischen Abschotten des Servers und der Magnetplatten, auf einer Protokollierung von Zugriffen und einer regelmäßigen Prüfung von Signaturen.

In der **öffentlichen Verwaltung** soll die elektronische Kommunikation Papier, Zeit und damit Geld sparen. Beispielsweise wird in der Niedersächsischen Landesverwaltung ein **Kassensystem** eingesetzt, welches die digitale Signatur im Sinne einer Unterschrift nutzt. Seit etwa $1\frac{1}{2}$ Jahren

ist das System mit etwa 16 000 Teilnehmern in Betrieb und als Signaturalgorithmus wird RSA verwendet. Da das 4-Augen-Prinzip vorgeschrieben ist, bearbeiten stets zwei Mitarbeiter eine Auszahlung. Ein so genannter „Feststeller“ gibt Daten aus den auf Papier vorliegenden Rechnungsbegründungsunterlagen in das System ein und signiert den dabei entstehenden Datensatz nicht. Ein „Anordnungsbefugter“ prüft anschließend den Datensatz und gibt ihn unter Erzeugung einer digitalen Signatur frei. Diese Daten werden anschließend an den Applikations-Server geschickt, der die Signaturen verifiziert und Sperrlisten konsultiert. Sperrlisten werden mit jeder neuen Sperrung aktualisiert. Der Server generiert aus diesen Datensätzen elektronische Zahlungsanweisungen, die dann mit einem separaten Verschlüsselungsverfahren an die Bank gehen. Die digitalen Signaturen sind verbindlich, d. h. Sachbearbeiter sind für ihr Tun verantwortlich und regresspflichtig. Dieses Verfahren verkürzt die Bearbeitungszeit einer herkömmlichen Kassenanordnung, die von einer autorisierten Amtsperson handschriftlich unterschrieben werden musste, bevor sie auf den Dienstweg ging, von zehn auf ein bis zwei Tage, was gerade bei den überschuldeten öffentlichen Haushalten einen wichtigen Beitrag zur Kostendämpfung darstellt [IZN] [Siet01a].

Im **Bundesinnenministerium** [BMIa] werden in den Dienstausweis die Funktionen Lichtbildausweis, Gleitzeitausweis, Bezahlfunktion und Signaturkarte integriert. Im **Sphinx-Projekt** [BMIb] wurde der Kommunikationsstandard „MailTrusT“ [MTT] für die Bundesverwaltung für verschlüsselte und signierte Übermittlung von Daten entwickelt. Genutzt wird RSA mit RIPEMD-160, SHA-1 und MD5.

Im Rahmen des „ArchiSig“-Projekts [Arch] sollen in der **Staatskanzlei Niedersachsen** und im **Universitätsklinikum Heidelberg** nicht nur digitale Signaturen für die internen Arbeitsabläufe eingesetzt, sondern besonders ihre Eigenschaften hinsichtlich der Archivierung untersucht werden. Auf die Probleme, die aus der Aufbewahrung digitaler Signaturen über Jahre hinweg resultieren, wird im Kapitel 2 auf Seite 39 eingegangen.

E-Commerce zwischen Unternehmen

Im Business-to-Business-Bereich, kurz „B2B“, werden Public-Key-Techniken bereits eingesetzt, weil die elektronische gegenüber der papiergebundenen Kommunikation Zeit- und Geldvorteile in Form kürzerer Transport- und Bearbeitungszeiten und ersparter Porto- und Telefonkosten verspricht.

Identrus [Iden] ist ein Zusammenschluss diverser Banken, um E-Commerce bei Business-to-Business zu fördern. In der errichteten Public-Key-Infrastruktur stellt Identrus die RootCA zur Verfügung und jede Bank hat eine eigene CA. Es werden Standardzertifikate vom TC TrustCenter [TC] aus Hamburg genutzt, die mit RSA und 1024 Bit langen Schlüsseln arbeiten. Es werden Chipkarten, Zertifikats-Status-Anfragen zur Validierung, Software von Secude [Secude] und sichere E-Mails nach dem S/MIME-Standard eingesetzt. In Deutschland sind derzeit die Commerzbank, die HypoVereinsbank, die Dresdner und die Deutsche Bank beteiligt.

Drei Beispielanwendungen von Identrus: Frachtraum-Vermietung, Order Transaction Payment System und Business Direct.

In Singapur gibt es ein Pilotprojekt **Frachtraum-Vermietung**, bei dem Firmenkunden bis zu einer halben Stunde vor Abflug Frachtraum in Flugzeugen mieten können. Anmietung und Rechnung werden auf Basis von Identrus digital signiert und ersetzen die Telefon- oder Fax-Kommunikation.

Das **Order Transaction Payment System** der Deutschen Bank bietet den Unternehmenskunden die Möglichkeit, Akkreditive (internationale Zahlungsanweisungen oder -aufträge) an

ihre Bank in Form eines digital signierten Word-Dokuments zu schicken. Dazu wird das „.doc“-Dokument in ein PKCS#7-Format [PKCS7] mit Endung „.7“ verwandelt. Auf Seiten der Firma existieren CRL-Sperrlisten, die einmal pro Woche aktualisiert und vom jeweiligen Sachbearbeiter heruntergeladen werden. Im Verifizierungsvorgang bei der Bank wird bei Erhalt einer Signatur der Status des Zertifikats mittels OCSP-Validierung überprüft, dann wird vom Bildschirm ein Screen-Shot „geschossen“ und als Bild an die Datei angehängt und in einem Archiv abgelegt. Diese Vorgehensweise soll belegen, dass zum Zeitpunkt der Verifikation das Zertifikat gültig war. Zum Archiv existiert ein Back Up und der Zugriff wird protokolliert, weil das Handelsgesetzbuch eine Aufbewahrungsdauer von 10 Jahren verlangt.

Bei **Business Direct** der Deutschen Bank können mittelständische Firmen online Kontostände erfragen, Akkreditive eröffnen, Gelder anlegen oder Kredite beantragen. Identrus bietet auch hier eine PKI-Lösung an. Interessant ist, dass das zuvor genutzte Medium Telefon eine deutlich niedrigere Sicherheit bot, denn zur Authentisierung diente oft nur der Name. Die Aufträge an die Bank werden in dieser Anwendung mit einem “Utility Key“ signiert, der ohne PIN freigeschaltet wird. Einmal pro Stunde werden alle getätigten Aufträge noch einmal mit einem “Signature Key“ mit PIN-Freischaltung signiert, um Zeit zu sparen und dadurch Praktikabilität und Akzeptanz zu erhöhen.

e-Compete ist ein von „FairPay“ [FairPay] gefördertes Ausschreibungssystem. Die geltenden „Verdingungsordnungen für Bauleistungen“ [VOB] und „Verdingungsordnungen für Leistungen“ [VOL] sehen ein Einreichen von verschlüsselten und signierten Angeboten in elektronischer Form vor. e-Compete arbeitet momentan mit RSA und den Hashfunktionen MD5 und SHA-1.

Die **Bundesnotarkammer** [BNotK] will den Notaren Signaturkarten, Zertifikate und insbesondere Attributzertifikate für die Kommunikation zu Unternehmen zur Verfügung stellen. Die Bundesnotarkammer verfügt zu diesem Zweck auch bereits über ein zum alten Signaturgesetz akkreditiertes Trust Center.

ANX [ANX] ist ein Zusammenschluss von Herstellern und Zulieferern der Automobilindustrie. Alle Bestell- und Bezahlvorgänge werden PKI-basiert abgewickelt. Nach [Schu01] liegt die Kosteneinsparung pro Fahrzeug bei 70 US-Dollar.

Moderne **Frankiersysteme** nutzen bereits Public-Key-Kryptographie. Während in Deutschland die Deutsche Post mit ihrem „E-Post“-System erst beginnt, laufen in Amerika bereits seit einigen Jahren Frankiermaschinen, die mit kryptographischen Prozessen erzeugte Barcodes anstelle von Briefmarken und Stempeln auf Postsachen aufdrucken. Genutzt wird RSA mit SHA-1 mit 1024- bis 2048-Bit-Schlüsseln und ECDSA über $F(2^{163})$ mit 163 Bit langen Schlüsseln, was sich gerade aufgrund der engen Platzverhältnisse auf einer „Briefmarke“ anbietet [BlKr01] [Blei00].

E-Commerce zwischen Unternehmen und Privatleuten

Im Business-to-Customer oder Business-to-Consumer, kurz “B2C“, kommunizieren Unternehmen mit Privatleuten. Angebote in diesem Bereich unter Nutzung rechtsverbindlicher digitaler Signaturen gibt es noch wenige. Privatleute können sich allerdings heute schon das notwendige Equipment, in Form von Zertifikaten, Chipkarten und Software, besorgen, mit dem sie E-Mails digital signieren und verschlüsseln können – z. B. von der Deutschen Post [Post01] oder Deutschen Telekom [Tele01], die beide auf RSA setzen. Zur Zeit fehlt eine so genannte „Killerapplikation“, welche für die Bürger einen derartigen Vorteil verspricht, dass diese sich das Signatur-Equipment zulegen und die jährlichen Gebühren für die Zertifikate bezahlen würden.

Eine heutige Anwendung ist das **Homebanking Computer Interface (HBCI)** [HBCI], mit dem Bankkunden ihre Bankgeschäfte von zu Hause aus tätigen können. Die digitalen Signaturen,

die mit RSA erzeugt werden, sind über den geschlossenen Vertrag zwischen Bank und Kunde den händischen Unterschriften gleichgesetzt.

Im Gesundheitswesen werden bei der **Qualitätssicherung in der Nierenersatztherapie (Quasi-Niere)** [Quasi] Public-Key-Techniken eingesetzt, um Patientendaten vertraulich über offene Kommunikationsnetze zu transportieren und um Erklärungen der Patienten oder Befunde der Ärzte digital signieren zu können. Genutzt wird RSA.

Keine Beispiele für E-Commerce mit Public-Key-Techniken sind die Angebote von Versandhäusern oder Versicherungen, die das Internet zum Verkauf ihrer Produkte und Policen nutzen. Bei Versandhäusern wird die Sicherheit von Bestellungen mit traditionellen Verfahren durchgeführt, wie sie bei telefonischer Bestellung üblich sind, also etwa Bezahlung bei der ersten Bestellung per Nachnahme und danach per Rechnung und über Kennworte oder Kundennummern. Letztendlich treten Versandhäuser in Vorleistung und tragen das Risiko, falls Kunden nicht bezahlen oder bestellte Waren zurücksenden und Kosten entstehen. Ganz ähnlich handeln Versicherungen: Policen werden elektronisch bestellt, dann per Post zugeschickt und unterschrieben wieder zurückgesendet. Daneben gelten bei rein online abgeschlossenen Versicherungen, wie etwa der Kapitallebens- und Rentenversicherung der Allianz oder der Mannheimer Auslandsreise-Krankenversicherung, beglichene Überweisungen an die Versicherung als Nachweis.

E-Government zwischen Behörden und Bürgern

Im „virtuellen Rathaus“ sollen Bürger rund-um-die-Uhr ihre administrativen Besorgungen erledigen können.

Im September 2000 startete die deutsche Bundesregierung das 10-Punkte-Programm „Internet für alle“, in dem das Ziel gesetzt wurde, bis zum Jahr 2005 alle Dienstleistungen der Bundesverwaltung, die internetfähig sind, online anzubieten. Bundeskanzler Gerhard Schröder zufolge „wünschen sich bereits heute 69 Prozent der Bevölkerung, ihre Behördenangelegenheiten in direkter Kommunikation über das Internet erledigen zu können“ [Schr01]. Die E-Government-Initiative heißt „**BundOnline 2005**“. Details sind unter [Bund] verfügbar.

Derzeit sind z. B. Arbeits- oder BAföG-Amt online und verschiedene Steuererklärungen können mit der **elektronischen Steuererklärung ELSTER** [Elster] elektronisch übermittelt werden. Dabei wird eine mit SSL gesicherte Verbindung aufgebaut, in der 1024 Bit lange RSA-Schlüssel genutzt werden. Zur Authentisierung der Steuerzahler muss derzeit allerdings noch eine komprimierte Fassung der Steuererklärung ausgedruckt, unterschrieben und mit den sonstigen Belegen per Post an das Finanzamt geschickt werden. Um diesen Medienbruch zu vermeiden, wurde auf der CeBIT 2002 der Prototyp einer neuen Version von ELSTER vorgestellt, in der digitale Signaturen genutzt werden können. Eine Einführung in Bayern, Nordrhein-Westfalen, Niedersachsen und dem Saarland ist noch für 2002 geplant.

In Deutschland werden im Rahmen des Wettbewerbs **Media@Komm** Ideen gesammelt, welche Anwendungen Kommunen ihren Bürgern virtuell anbieten sollten, und in Pilotprojekten realisiert. Umgesetzt werden u. a. die „Anwendungsbündel“ Umzug und Wohnen, Studium, Heirat, Bau eines Hauses, Kauf eines Autos, Gesundheitswesen (Arztbrief), Schriftverkehr zwischen Rechtsanwälten/Notaren und dem Amtsgericht, die öffentliche Auftragsvergabe, elektronischer Zahlungsverkehr und die Kommunikation zu Behörden, Banken, Sparkassen und anderen Stellen [TZI].

Andere Länder gehen ähnliche Wege, wie z. B. Finnland [Finnland] mit dem als Chipkarte realisierten Personalausweis oder Österreich [Austria] mit der „Bürgerkarte“: „Die Bürgerkarte ist

die Sozialversicherungskarte, die durch die elektronische Signatur Ausweis zur Identifikation auf der Reise am Datenhighway wird“ [Bürger].

Ein weiteres Beispiel für E-Government sind **digitale Wahlen** im Internet, auch **E-Vote** oder **i-Vote** genannt. i-Vote ist der Name des Osnabrücker Pilotprojektes zur Wahl des Studentenparlaments [iVote]. Ob in Zukunft Sozial-, Kommunal-, Landtags- oder Bundestagswahlen im Internet abgehalten werden, ist allerdings noch ungewiss [Siet01b].

Wie beim Business-to-Customer zwischen Geschäften und Konsumenten fehlt derzeit für die Bürger der Anreiz, sich ein Zertifikat, entsprechende Software und Chipkarte zuzulegen, um ihre Behördengänge virtuell zu erledigen.

Das Sicherheitsniveau von Signatur-Anwendungen variiert von Anwendung zu Anwendung.

1.4.3 Beispiele für Authentisierungs-Anwendungen

Neben der Signatur-Anwendung wird die Authentisierungs-Anwendung in der Praxis häufig genutzt. Zunächst wird in diesem Abschnitt der Begriff der Authentisierungs-Anwendung exakt definiert und anschließend von zwei Anwendungsbeispielen berichtet.

Definition 16
Eine **Authentisierungs-Anwendung** ist eine Anwendung der digitalen Signatur in einem Authentisierungsprotokoll. Signiert werden insbesondere Zufallszahlen. Im Rahmen einer Authentisierung kann ein Session Key für die Verschlüsselung der nachfolgenden Kommunikation mit einem symmetrischen Verfahren ausgehandelt werden.

Authentisierungs-Anwendungen werden in virtuellen und realen Zugangskontrollen eingesetzt.

Virtuelle Zugangskontrolle

Virtuelle Zugangskontrollen sind Kontrollen zwischen Rechnern, wie z. B. in einer Client-Server-Architektur, in der ein Server Clients eine Dienstleistung zur Verfügung stellt, nachdem sich diese authentisiert haben. Virtual Private Networks (VPNs) sind ein Beispiel für diese Technik.

Es gibt sicherheitskritische Anwendungen, z. B. wenn Rechner von Filialen einer Firma über ein offenes Netz miteinander kommunizieren oder sich Außendienst-Mitarbeiter mit ihrem Notebook ins Firmennetz einwählen und Wirtschafts-Spionage ernst genommen wird. Daneben gibt es Anwendungen, die weniger sicherheitskritisch sind, z. B. wenn die gängigen Browser eine gesicherte Verbindung im Internet zu einem Server aufbauen wollen, zur Verifikation der Zertifikate allerdings über keine Sicherheitsanker verfügen und stattdessen den Nutzer fragen, ob dieser dem Zertifikat vertraut, oder wenn keine Sperrlisten oder OCSP-Dienste konsultiert werden.

Zugangskontrollen nutzen häufig **TLS/SSL**. TLS/SSL steht für **Transport Layer Security** bzw. **Secure Socket Layer**. SSL wurde ursprünglich von Netscape entwickelt und ist inzwischen von der Network Working Group unter dem Begriff TLS standardisiert worden. TLS nutzt Diffie-Hellman, DSA und RSA zur Authentisierung und zum Schlüsselaustausch sowie die symmetrischen Verfahren DES, Triple-DES und RC4 zur Verschlüsselung der Kommunikation. Für Details siehe Appendix A.8 auf Seite 211.

Reale Zugangskontrolle

Reale Zugangskontrollen sind Kontrollen, in denen „realen“ Menschen Zutritt zu einem Raum oder einem Gelände gewährt wird, nachdem sie sich über die in ihrer Chipkarte gespeicherten Schlüssel authentisieren konnten. Die **Office Identity Card (OIC)** [OIC00] ist eine Spezifikation eines als Chipkarte ausgelegten Dienstausweises, welche in einem Feldversuch als „digitaler Dienstausweis des Bundesministeriums des Innern“ erprobt wird [Fell01]. OIC ist offen für die Nutzung beliebiger Verfahren; RSA, DSA und ECDSA sind allerdings explizit spezifiziert.

1.5 Zusammenfassung

Public-Key-Infrastrukturen können in vielen Bereichen mit verschiedenen Sicherheitsbedürfnissen zur Anwendung kommen. Es gibt bereits einige PKIs und andere werden folgen. Der Trend zur Informationsgesellschaft, welche Public-Key-Infrastrukturen in immer mehr Bereichen einsetzt und auf eine funktionierende Technik angewiesen ist, ist unübersehbar.

Aktuelle Anwendungen nutzen zum überwiegenden Teil RSA als Signaturalgorithmus. Bei den Hashfunktionen werden MD5, SHA-1 und RIPEMD-160 genutzt.

Im Weiteren beschränken sich die Betrachtungen dieser Arbeit auf die folgenden beiden Klassen von Anwendungen:

- **Sicherheitskritische Signatur-Anwendung**
  Digitale Signaturen und Verschlüsselungen in einer elektronischen Kommunikation entsprechen händischen Unterschriften und dem „Briefverschließen“ in der papiergebundenen Kommunikation. Die Sicherheit beruht auf kryptographisch geeigneten Verfahren, die zusammen mit Sicherheitsankern und privaten Schlüsseln auf einer PSE abgelegt sind.

- **Sicherheitskritische Authentisierungs-Anwendung**
  In einem Authentisierungsprotokoll werden Zufallszahlen signiert und optional ein Session Key ausgehandelt für die symmetrische Verschlüsselung der nachfolgenden Kommunikation. Die kryptographisch geeigneten Verfahren sind zusammen mit den Sicherheitsankern sowie privaten Schlüsseln auf PSEs abgelegt.

Kapitel 2

Problemexposition

Public-Key-Infrastrukturen sind geeignet, die Authentizität, Integrität, Vertraulichkeit und Verbindlichkeit übertragener Daten zu gewährleisten. Allerdings nicht unbegrenzt lange. Während eine händische Unterschrift und ein versiegelter Brief über lange Zeit hinweg ihre Eigenschaften behalten können, hängen digitale Signaturen und Verschlüsselungen von der Sicherheit der kryptographischen Verfahren ab. Diese Kryptoverfahren sind zeitlich nur begrenzt geeignet, weil sie den fortschreitenden wissenschaftlichen Erkenntnissen und den Fortschritten bei der Rechenleistung ständig angepasst werden müssen.

Während auf stetige Fortschritte, die überschaubar und vorhersehbar sind, relativ gut reagiert werden kann, liegt ein wesentlich größeres Problem darin, dass es auch plötzliche große Fortschritte geben kann, weil die Verfahren der Public-Key-Kryptographie nicht beweisbar sicher sind. Es ist beispielsweise nicht auszuschließen, dass morgen ein genialer Mathematiker eine gute Idee hat, wie sich sehr große Zahlen effizient faktorisieren lassen, wodurch die RSA-Sicherheit hinfällig wäre. Das in der Einleitung geschilderte Beispiel der “Yescard“ zeigt eine ähnliche Situation.

Als Folge könnten kryptographische Verfahren und Schlüssel kompromittiert werden, was den Verlust der Authentizität und Integrität digitaler Signaturen oder den Verlust der Vertraulichkeit von Verschlüsselungen bedeuten könnte.

Um die Sicherheit in der PKI wiederherzustellen, wenn kryptographische Verfahren oder Schlüssel kompromittiert wurden, gibt es die Möglichkeit, die vom Schaden betroffenen Zertifikate zu sperren, so dass keine Signatur eines revozierten Zertifikats mehr akzeptiert und keine unsichere Verschlüsselung mehr durchgeführt zu werden braucht.

Es bleiben aber die folgenden Fragen offen:

- Wie können digitale Signaturen beim plötzlichen Auftreten eines Schadens, der diese Signaturen berührt, beweiskräftig bleiben? Was ist mit speziellen Signaturen auf Zeitstempeln, die besonders lange beweiskräftig bleiben sollen?
- Wirken die Revokationsmechanismen im Schadensfall noch? Welche Aussagekraft hat eine digital signierte Sperrliste oder OCSP-Antwort, wenn diese Signaturen vom Schaden betroffen sind?
- Wie können Verschlüsselungen sensibler Daten trotz kompromittiertem Verfahren oder Schlüssel geheim bleiben?

Und die wichtigste Frage:

- Wenn die Informationsgesellschaft in so vielen – und auch wichtigen – Bereichen auf Public-Key-Infrastrukturen setzt und sich von deren Funktionsfähigkeit abhängig macht, kann sie sich den Ausfall eines solchen Systems leisten?

In diesem Kapitel wird zunächst dargestellt, wie die Sicherheit kryptographischer Algorithmen bemessen wird, welche Algorithmen derzeit als kryptographisch geeignet angesehen werden, wie mit „alternden“ digitalen Signaturen umzugehen ist und welche unvorhersehbaren Entwicklungen es in der Vergangenheit gegeben hat. Anschließend werden die Probleme diskutiert, die auftreten würden, falls ein als sicher anerkanntes Kryptoverfahren unsicher werden sollte. Es werden die möglichen Schäden exakt aufgelistet sowie ihre Auswirkungen auf konkrete Anwendungen. Eine Analyse bereits etablierter Gegenmaßnahmen in heutigen Public-Key-Infrastrukturen führt zur Auflistung bislang ungelöster Probleme, die in dieser Dissertation gelöst werden.

2.1 Sicherheit kryptographischer Algorithmen

Es gibt keine beweisbar sicheren Signatur- oder Verschlüsselungsverfahren, die praktisch einsetzbar wären. Die einzig bekannte nachweisbar sichere Verschlüsselung ist nicht praktikabel: Beim **Vernam-One-Time-Pad-Verfahren** wird die Nachricht mit einem Schlüssel, der nur einmal benutzt wird und selbst so lang wie die Nachricht ist, durch Anwendung der XOR-Operation verschlüsselt. Das Problem des Schlüsselmanagements erklärt die fehlende Praktikabilität.

Statt Beweise gibt es Empfehlungen über derzeit kryptographisch geeignete Algorithmen mit ihren Parametern und Schlüssellängen, die sich am Stand von Forschung und Technik orientieren. Beispielsweise stellt das Bundesamt für Sicherheit in der Informationstechnik (BSI) [BSI] eine Übersicht über die Algorithmen und dazugehörigen Parameter zusammen, die zur Erzeugung von Signaturschlüsseln, zum Hashen zu signierender Daten oder zur Erzeugung und Prüfung digitaler Signaturen als geeignet anzusehen sind.

Zitat aus [BSI01]: „Um festzulegen, wie groß die Systemparameter bei diesen Verfahren gewählt werden müssen, um deren Sicherheit zu gewährleisten, müssen zum einen die besten heute bekannten Algorithmen zum Faktorisieren ganzer Zahlen bzw. zum Berechnen diskreter Logarithmen [...] betrachtet und zum anderen die Leistungsfähigkeit der heutigen Rechnertechnik berücksichtigt werden. Um eine Aussage über die Sicherheit für einen bestimmten zukünftigen Zeitraum zu machen, muss außerdem eine Prognose für die beiden genannten Aspekte zugrunde gelegt werden, vgl. [LeVe00]. Solche Prognosen sind nur für relativ kurze Zeiträume möglich (und können sich natürlich jederzeit aufgrund unvorhersehbarer dramatischer Entwicklungen als falsch erweisen).“

Dieser Abschnitt gliedert sich in vier Teile: Zunächst wird von den Aspekten berichtet, die für die Empfehlungen kryptographisch geeigneter Algorithmen zu berücksichtigen sind. Daraus folgt eine Übersicht über kryptographisch geeignete Verfahren. Anschließend wird beschrieben, wie mit digitalen Signaturen umzugehen ist, wenn ein verwendetes Verfahren seine Eignung zu verlieren droht. Abschnitt 2.1.4 beantwortet die Frage, weshalb es jederzeit passieren kann, dass diese Übersicht an neue Entwicklungen angepasst werden muss.

2.1.1 Kriterien für geeignete Kryptoalgorithmen

In der Kryptographie werden mathematische Basisprobleme ausgenutzt, die derzeit ungelöst sind. Für die Sicherheit von kryptographischen Anwendungen, die diese Probleme nutzen, ist es

wichtig, einerseits die mathematischen Hintergründe weiter zu analysieren und andererseits zu versuchen, die Probleme algorithmisch anzugehen. Dieser Abschnitt vermittelt einen Einblick, welche Methoden es gibt, große Zahlen zu faktorisieren oder diskrete Logarithmen in endlichen Körpern, auf elliptischen Kurven oder auf Zahlkörpern zu berechnen und welche Korrelationen sich zwischen den mathematischen Basisproblemen dadurch ergeben. Für eine Beurteilung der Sicherheit sind diese Überlegungen in Relation zur momentan verfügbaren Rechenleistung zu setzen. Details zu diesem Abschnitt finden sich z. B. bei [BuMa00], [Buch99], [Buch01] oder [Schn96].

Effizienz von Algorithmen

Die Effizienz eines Algorithmus zum Faktorisieren großer Zahlen oder Berechnen diskreter Logarithmen wird von seiner Laufzeit bestimmt. Ein Maß für die Laufzeit ist

$$L_n[u, v] = e^{v(\log n)^u (\log\log n)^{1-u}},$$

mit reellen Zahlen n, u, v und n größer als die Eulersche Konstante e.

Was sagt $L_n[u, v]$ über die Effizienz aus?

Ein Algorithmus mit Laufzeit $L_n[0, v] = (\log n)^v$ ist polynomiell, wobei v den Grad des Polynoms angibt, der die Laufzeit bestimmt. Polynomielle Algorithmen sind effizient.

Ein Algorithmus mit Laufzeit $L_n[1, v] = e^{v \log n}$ ist exponentiell. Die besten bekannten exponentiellen Algorithmen arbeiten nach der Brute-Force-Methode, d. h. alle Möglichkeiten durchzuprobieren, die aufgrund der Menge an Möglichkeiten ineffizient ist.

Ein Algorithmus mit Laufzeit $L_n[u, v]$ für $0 < u < 1$ heißt subexponentiell. Subexponentielle Algorithmen sind besser als exponentielle und schlechter als polynomielle.

Faktorisierungsproblem

Das Faktorisierungsproblem beschreibt die Schwierigkeit, zu einer Zahl $n = pq$ die Faktoren p und q zu ermitteln, $p, q \in \mathbb{N}$. RSA basiert auf dem Faktorisierungsproblem.

Angenommen, die Zahl n soll faktorisiert werden. Die schnellsten bekannten Faktorisierungsalgorithmen sind subexponentiell:

- Das Quadratische Sieb (QS) hat Laufzeit $L_n[1/2, 1 + o(1)]$ mit $\lim_{n\to\infty} o(1) = 0$.
- Die Elliptische-Kurven-Methode (ECM) hat Laufzeit $L_p[1/2, \sqrt{1/2} + o(1)]$, wobei p der kleinste Primfaktor von n ist. Hat p die Größenordnung von $\sqrt{n}$, so gilt $L_n[1/2, 1 + o(1)]$. Bis 1998 galt ECM als der schnellste Faktorisierungsalgorithmus.
- Das Zahlkörpersieb (NFS) hat Laufzeit $L_n[1/3, (64/9)^{1/3} + o(1)]$. Es wurde 1988 von John Pollard entdeckt.
- Für Zahlen spezieller Gestalt gibt es effizientere Algorithmen zum Faktorisieren, z. B. wurde 1999 die Zahl $(10^{211} - 1)/9$ mit einem speziellen Zahlkörpersieb (SNFS) faktorisiert.

Tabelle 2.1 gibt derzeitige Faktorisierungsrekorde an, wobei ein MIPS-Jahr die Zeit in Jahren angibt, die ein Rechner benötigt, der eine Million Instruktionen pro Sekunde berechnen kann – Million Instructions Per Second. Faktorisiert wurden die von RSA Security Inc. in einer Challenge veröffentlichten Zahlen n = RSA-k, wobei k die dezimale Länge von n angibt und n zwei Primfaktoren hat. Interessant ist RSA-155, die dezimal 155 Stellen hat und binär mit 512 Bit dargestellt werden kann, weil 512 Bit lange RSA-Schlüssel 1999 in vielen Anwendungen benutzt wurden.

Tabelle 2.1: Faktorisierungsrekorde

Zeitpunkt	n	Algorithmus	MIPS-Jahre
1991	RSA-100	Quadratisches Sieb	7
1992	RSA-110	Quadratisches Sieb	75
1993	RSA-120	Quadratisches Sieb	830
1994	RSA-129	Quadratisches Sieb	5000
1996	RSA-130	Zahlkörpersieb	500
1999	RSA-140	Zahlkörpersieb	2000
1999	RSA-155	Zahlkörpersieb	8000

Wie wird sich die Faktorisierung entwickeln? Andrew Odlyzko sagte 1999 in [Odly99] voraus, dass für die Faktorisierung einer 512 Bit lange Zahl $3*10^4$ MIPS-Jahre benötigt werden würden, für 768 Bits $2*10^8$ MIPS-Jahre, für 1024 Bits $3*10^{11}$ MIPS-Jahre, für 1280 Bits $1*10^{14}$ MIPS-Jahre, für 1536 Bits $3*10^{16}$ MIPS-Jahre und für 2048 Bits $3*10^{20}$ MIPS-Jahre.

DL-Problem in endlichen Körpern

Das DL-Problem ist das Problem, diskrete Logarithmen effizient zu berechnen. DSA oder ElGamal sind kryptographische Anwendungen dieses Problems.

Die Komplexität der Berechnung diskreter Logarithmen in $(\mathbb{Z}/p\mathbb{Z})^*$ korreliert mit dem Faktorisierungsproblem in der Weise, dass bessere Faktorisierungsalgorithmen eine schnellere DL-Berechnung liefern. Der Faktorisierung mit dem Quadratischen Sieb entspricht die COS-Methode von Don Coppersmith, Andrew Odlyzko und Richard Schroeppel zur DL-Berechnung. Auch das Zahlkörpersieb (NFS) kann für DL-Berechnung genutzt werden.

Die derzeit beste Methode, das DL-Problem in endlichen Körpern zu lösen, arbeitet mit Laufzeit $L_p[1/3, (64/9)^{1/3} + o(1)]$.

Zwei Rekorde:

Tabelle 2.2: Rekorde, das DL-Problem in endlichen Körpern zu lösen

Jahr	Länge von p	Algorithmus	Laufzeit
1996	85 Dezimalstellen	NFS	45 MIPS-Jahre
1998	90 Dezimalstellen	COS	4 Monate mit 18 Pentium-180-Rechnern

DL-Problem auf Zahlkörpern

Das mathematische Basisproblem, diskrete Logarithmen auf Zahlkörpern zu lösen, wird bei der DSA-Variante IQDSA ausgenutzt. IQ steht für Imaginär-Quadratische Zahlkörper.

Die Komplexität dieses Problems ist $L_\Delta[1/2, c + o(1)]$, wobei Δ die Diskriminante der Ordnung und c eine vom Körpergrad abhängige Konstante ist.

DL-Problem auf elliptischen Kurven

Das mathematische Basisproblem, diskrete Logarithmen auf elliptischen Kurven zu berechnen, wird in der Elliptischen-Kurven-Kryptographie (Elliptic Curve Cryptography – ECC) ausgenutzt. ECDSA und ECIES sind zwei Anwendungen der ECC.

Die Pollard-ρ-Methode ist der derzeit beste bekannte Algorithmus mit Laufzeit $\sim \sqrt{p}/r$, wobei p größter Teiler der Primordnung und r die Anzahl der Rechner ist, um das DL-Problem auf elliptischen Kurven zu berechnen.

Drei Rekorde:

Tabelle 2.3: Rekorde, das DL-Problem auf elliptischen Kurven zu lösen

Jahr	Körper/Bitgröße	Laufzeit
1998	ECCp-97	53 Tage auf 1288 Rechner
1999	ECC2-97	40 Tage auf 740 Rechner = 16 000 MIPS-Jahre
2000	ECC2K-108	mehr als für RSA-155 (8000 MIPS-Jahre)

Korrelationen zwischen den mathematischen Basisideen

Das Faktorisierungsproblem korreliert mit dem DL-Problem in endlichen Körpern, weil in beiden Problemen Zahlkörpersieb und Quadratisches Sieb zur Anwendung kommen können. Damit ist das DL-Problem in $(\mathbb{Z}/p\mathbb{Z})^*$, p eine Primzahl, nicht schwieriger zu lösen als das Faktorisierungsproblem für natürliche Zahlen, so dass RSA-Signaturalgorithmus und DSA sowie RSA-Verschlüsselung und ElGamal in einer Abhängigkeit stehen. Faktorisierungsproblem und DL-Problem auf Zahlkörpern stehen über das Quadratische Sieb in einer Beziehung, so dass RSA und IQDSA nicht als voneinander unabhängig anzusehen sind. Andrew Odlyzko nennt in [Odly99] den Grund für diese Abhängigkeiten: “There is a natural relation between multiplication and addition in a finite field, which is what makes the index calculus methods work in that setting“. Die angesprochenen Index-Calculus-Methoden bezeichnen eine besonders effiziente Methode zur Berechnung diskreter Logarithmen.

Zwischen Faktorisierungsproblem und DL-Problem auf elliptischen Kurven sind derzeit keine Korrelationen bekannt. Als Grund gibt Andrew Odlyzko die fehlende natürliche Beziehung zwischen Punkten einer elliptischen Kurve zu denen anderer Gruppen an: “There is no such natural relation between the group of points of an elliptic curve and another group, and this appears to be the main reason efficient discrete log methods for elliptic curves have not been found“ [Odly99]. Es ist also insbesondere keine Beeinflussung von RSA zu ECDSA bekannt.

Rechenleistung

Die derzeitigen Möglichkeiten bei den Algorithmen zum Faktorisieren großer Zahlen oder Berechnen diskreter Logarithmen müssen mit der derzeit verfügbaren Rechenleistung und ihrer voraussichtlichen Entwicklung verglichen werden, um festlegen zu können, bei welchen Parametern und Schlüssellängen die kryptographischen Algorithmen als sicher zu erachten sind.

Zur Berechnung der verfügbaren Rechenleistung beschrieb u. a. Andrew Odlyzko, dass 1994 RSA-129 mit einem Aufwand von 104 MIPS faktorisiert wurde, während damals im Internet insgesamt 3*107 MIPS zur Verfügung stand. Odlyzko führt als Beispiel Silicon Graphics an, die 1995 mit ihren 5 000 Angestellten und 10 000 Workstations 105 MIPS Leistung aufbringen konnten [Odly95].

Um die voraussichtliche Entwicklung der Rechenleistung abschätzen zu können, wird oft die Regel von Intel-Mitbegründer Gordon E. Moore zitiert, welche besagt, dass sich die Prozessorleistung alle 18 Monate verdoppelt [Moor65]; "Moores Law" stammt aus dem Jahre 1965 und hat auch heute noch seine Berechtigung. Odlyzko hat in [Odly95] und [Odly99] die unter Berücksichtigung dieser Regel verfügbare Computer-Leistung für die Faktorisierung von Zahlen vorhergesagt, wobei die Rechenleistung davon abhängt, ob ein Angriff im Geheimen stattfinden soll, oder ob mehrere Computer – etwa über das Internet – zusammengeschaltet werden; siehe Tabelle 2.4.

Tabelle 2.4: Verfügbare Computer-Leistung für Faktorisierung (in MIPS-Jahren)

Jahr	„Versteckter Angriff"	„Offenes Projekt"
2004	10^8	$2 * 10^9$
2014	10^{10} – 10^{11}	10^{11} – 10^{13}

2.1.2 Geeignete Kryptoalgorithmen

Aus verfügbarer Rechenleistung und Wissensstand zu mathematischen Algorithmen, um kryptographische Verfahren zu brechen, ergeben sich Kryptoalgorithmen, Parameter und Schlüssellängen, die für kryptographische Anwendungen geeignet sind. In diesem Abschnitt werden derzeit kryptographisch geeignete Verfahren aufgeführt.

In der Übersicht des BSI über geeignete Kryptoalgorithmen vom 5. Juli 2001 [BSI01] ist z. B. aufgeführt:

- Die 160-Bit Hashfunktionen SHA-1 und RIPEMD-160 sind bis mindestens Ende 2006 als kryptographisch geeignet eingestuft.
- RSA ist bis Ende 2006 mit einem Modulus n von mindestens 1024 Bit als sicher anzusehen; bis Ende 2006 werden dann aber 2048 Bit empfohlen (vgl. Bezeichnungen aus Beispiel zu Definition 3 auf Seite 6). Bei RSA sollten die Primfaktoren p und q von n ungefähr gleich groß sein, aber nicht zu dicht beieinander liegen: $\epsilon_1 < |\log_2(p) - \log_2(q)| < \epsilon_2$. Als Anhaltspunkte für die Werte ϵ_1 und ϵ_2 schlägt das BSI $\epsilon_1 \approx 0,5$ und $\epsilon_2 \approx 30$ vor. Die frühere Forderung, dass p und q starke Primzahlen sein müssen (d. h. $1 - p$ und $1 - q$ haben große Primfaktoren), wird aufgrund der heute bekannten besten Faktorisierungsalgorithmen nicht mehr aufgestellt.

- Zur Nutzung von RSA fordert das BSI, dass „der Hashwert [...] vor der Anwendung des geheimen Exponenten auf die Bitlänge des Moduls formatiert werden“ [BSI01] muss. Als geeignete Formatisierungsverfahren werden PKCS #1 v2.0 und ISO/IEC 9796-2 mit Zufallszahl genannt.

- Für DSA (vgl. Beispiel zu Definition 3 auf Seite 6) werden für p die gleichen Werte wie für n bei RSA und q mit 160 Bit bis Ende 2006 gefordert.

- Das BSI empfiehlt für DSA-Varianten auf elliptischen Kurven, wie z. B. ECDSA, über $E(F_p)$ bis Ende 2006 mindestens 192 Bit lange Werte für p; für q werden bis Ende 2005 160 Bit und bis Ende 2006 180 Bit empfohlen. Als Systemparameter werden eine elliptische Kurve E und ein Punkt P erzeugt, so dass folgende Bedingungen gelten: $\#E(F_p) = aq$ mit einer von p verschiedenen Primzahl q; $\mathrm{ord}\,(P) = q$; $r_0 := \min(r \mid q \text{ teilt } p^r - 1)$ ist groß, konkret $r_0 > 10^4$; die Klassenzahl der Hauptordnung, die zum Endomorphismenring von E gehört, ist mindestens 200.

- Bei DSA-Varianten über $E(F_{2^m})$ verändert sich nur ein Wert gegenüber $E(F_p)$: Bis Ende 2006 soll m mindestens 191 Bit lang sein. Als Systemparameter werden eine elliptische Kurve E und ein Punkt P erzeugt, so dass folgende Bedingungen gelten: m ist prim; $E(F_{2^m})$ ist nicht über F_2 definierbar (d. h. die j-Invariante der Kurve liegt nicht in F_2; $\#E(F_{2^m}) = aq$ mit q prim; $\mathrm{ord}\,(P) = q$; $r_0 := \min(r \mid q \text{ teilt } 2^{mr} - 1)$ ist groß, konkret $r_0 > 10^4$; die Klassenzahl der Hauptordnung, die zum Endomorphismenring von E gehört, ist mindestens 200.

- Für DSA und seine Varianten wird der Einsatz einer Hashfunktion gefordert.

- Das BSI gibt Kriterien für die Erzeugung von Zufallszahlen an, die bei der Erzeugung von Systemparametern für Signaturalgorithmen, für die Schlüsselgenerierung und bei DSA-ähnlichen Signaturalgorithmen bei jeder Generierung einer Signatur benötigt werden.

Die Empfehlungen des BSI für Verfahren und Parameter bis Ende 2007 sehen als wesentlichen Unterschied zur Empfehlung von 2001 ([BSI01]) für RSA und DSA ab 2007 einen Mindestwert von 2048 Bit Schlüssellänge vor [BSI02].

Auf EU-Ebene legt das Article 9 Committee (A9C) auf Grundlage der EU-Richtline für „gemeinschaftliche Rahmenbedingungen für elektronische Signaturen“ [EU99] Algorithmen und Parameter für sichere elektronische Signaturen fest [EU A9C01]. Die Werte ähneln denen des BSI.

Arjen Lenstra und Eric Verheul haben 2000 in ihrem Artikel “Selecting Cryptographic Key Sizes“ einen Ausblick über Sicherheit von kryptographischen Verfahren bis zum Jahr 2050 gegeben [LeVe00]. Den dortigen Tabellen kann entnommen werden, welche Schlüssellängen nach dem heutigen Wissensstand nötig wären, um bis zum Jahr 2050 etwa eine Signatur abzusichern. Weitere Aussagen zu voraussichtlichen Schlüssellängen finden sich u. a. bei Bruce Schneier [Schn96].

2.1.3 Re-Signieren

Dieser Abschnitt beschäftigt sich mit dem Re-Signieren digitaler Signaturen, welches notwendig sein kann, bevor die verwendeten kryptographischen Verfahren ihre Eignung verlieren.

Da Kryptoalgorithmen nur begrenzt kryptographisch geeignet sind und von Zeit zu Zeit neuen wissenschaftlichen Erkenntnissen und stärkeren Rechenleistungen angepasst werden müssen,

müssen digitale Signaturen für eine langfristige Haltbarkeit und damit Gleichsetzung zur händischen Unterschrift nachsigniert werden. Aus dem Signaturgesetz ergibt sich keine Rechtspflicht zum Nachsignierung; ein Nachsignieren „liegt aber grundsätzlich im Eigeninteresse eines jeden, der signierte Dokumente als Beweismittel aufbewahren will. [...] Eine solche Rechtspflicht kann sich aber indirekt aus vertraglichen Absprachen oder aus gesetzlichen Regelungen zur Aufbewahrung oder Dokumentation von Beweismitteln im Interesse der Allgemeinheit oder Dritter egeben“ [BPRS02].

Ein „Nachsignieren“ oder „Re-Signieren“ umfasst alle früheren digitalen Signaturen und Zeitstempel und kann von einer beliebigen Person mit einem erneuten Zeitstempel erzeugt werden. Hierzu drei Zitate:

Die Signaturverordnung zum alten SigG definierte die „erneute digitale Signatur“. Dieser Begriff ist in der neuen SigV [SigV01] zwar verschwunden, der Sachverhalt aber geblieben – wobei sich „digitale Signatur“ nun auf eine „qualifizierte elektronische Signatur“ bezieht: „Daten mit einer qualifizierten elektronischen Signatur sind nach §6 Abs. 1 Satz 2 des Signaturgesetzes neu zu signieren, wenn diese für längere Zeit in signierter Form benötigt werden, als die für ihre Erzeugung und Prüfung eingesetzten Algorithmen und zugehörigen Parameter als geeignet beurteilt sind. In diesem Falle sind die Daten vor dem Zeitpunkt des Ablaufs der Eignung der Algorithmen oder der zugehörigen Parameter mit einer neuen qualifizierten elektronischen Signatur zu versehen. Diese muss mit geeigneten neuen Algorithmen oder zugehörigen Parametern erfolgen, frühere Signaturen einschließen und einen qualifizierten Zeitstempel tragen“ [SigV01, §17].

[BSI-SigI99b, 5.2] konkretisiert dies dahingehend, dass „der Zeitstempel [...] als erneute digitale Signatur i. S. v. §18 SigV angesehen“ wird. Darüber hinaus wird in dieser Spezifikation festgehalten, dass „für denjenigen, der die digital signierten Daten über längere Zeit benötigt, die Obliegenheit [besteht], die Eignung der für die Generierung von Zeitstempeln verwendeten Signaturverfahren zu überwachen und ggf. eine erneute digitale Signatur zu erzeugen“ [BSI-SigI99b, 5.3.1].

In der Begründung zur Signaturverordnung ist der Vorgang der erneuten digitalen Signatur detaillierter beschrieben: „Wenn für digitale Signaturen eingesetzte Algorithmen und zugehörige Parameter – und dadurch die damit erzeugten digitalen Signaturen – infolge neuer wissenschaftlicher Erkenntnisse oder des technischen Fortschritts (z. B. schnellere Rechner) an Sicherheitswert verlieren, so ist vor Ablauf der Eignung der Algorithmen und zugehörigen Parameter eine neue digitale Signatur (mit neuen technischen Komponenten) erforderlich. Die Anwendung neuer technischer Komponenten für die Erzeugung neuer Signaturen ist dadurch sichergestellt, dass sich die Zertifizierungsstelle vor der Ausstellung eines Signaturschlüssel-Zertifikates von der Eignung der technischen Komponenten zu überzeugen hat (vgl. §5 Abs. 1) und der Gültigkeitszeitraum für Zertifikate den Zeitraum der Eignung nicht überschreiten darf (vgl. §7 Abs. 2).
Um zu verhindern, dass neue digitale Signaturen zu einem späteren Zeitpunkt (wenn der Sicherheitswert der früheren digitalen Signatur möglicherweise bereits so gering geworden ist, dass Fälschungen möglich sind) angebracht und zurückdatiert werden, ist für diese ein Zeitstempel erforderlich.
Damit frühere digitale Signaturen im Hinblick auf eventuelle spätere Fälschungsmöglichkeiten nicht bestritten werden können, müssen diese in die neue Signatur eingeschlossen und damit ‚konserviert‘ werden. Dabei genügt für eine beliebige Anzahl signierter Daten eine (übergreifende) neue digitale Signatur, die von einer beliebigen Person (z. B. Archivar) angebracht werden kann.
Unterbleibt bei einer vorhandenen digitalen Signatur mit Ablauf der Eignung der Algorithmen und zugehörigen Parameter nach §17 Abs. 2 eine erneute digitale Signatur, so verliert sie da-

mit die gesetzlich vorgegebene Sicherheit. Unabhängig davon kann sie noch über eine längere Zeit einen hohen Sicherheitswert behalten. Dessen Bewertung bleibt im Streitfalle dann jedoch Gerichten und Sachverständigen überlassen“ [SigV-B, §18].

Das Signaturgesetz von 2001 definiert „qualifizierte Zeitstempel“ als „elektronische Bescheinigungen eines Zertifizierungsdiensteanbieters [...] darüber, dass ihm bestimmte elektronische Daten zu einem bestimmten Zeitpunkt vorgelegen haben“ [SigG01, §2, Nummer 14].

2.1.4 Unvorhersehbare Entwicklungen

Wie lang sind diese – z. B. vom BSI oder A9C – veröffentlichten Empfehlungen über kryptographisch geeignete Algorithmen gültig? Und wie verlässlich sind die Empfehlungen? Das BSI erwähnt in [BSI01] ausdrücklich, dass die Prognosen über kryptographisch geeignete Verfahren „nur für relativ kurze Zeiträume möglich [sind] (und [...] sich natürlich jederzeit aufgrund unvorhersehbarer dramatischer Entwicklungen als falsch erweisen [können]).“ Das Beispiel der “Yescard“ wurde schon erwähnt. Ein Blick in die Vergangenheit zeigt, dass es plötzliche und unerwartete Fortschritte zum Faktorisierungs- oder DL-Problem oder fehlerhafte Implementierungen gegeben hat. Dieser Abschnitt schildert einige unvorhersehbare Entwicklungen aus der Vergangenheit und mögliche Ereignisse in der Zukunft.

Einer der ersten Fälle ist weniger unter kryptographischen als unter historischen Gesichtspunkten interessant: Bis 1732 wurde vermutet, alle Fermat-Zahlen $F_n = 2^{2^n} + 1$ seien Primzahlen, bis 1732 Euler feststellte, dass sich die fünfte Fermat-Zahl F_5 faktorisieren lässt [Buch01]:

Tabelle 2.5: Faktorisierung Fermat-Zahlen

n	$F_n = 2^{2^n} + 1$	Faktorisierung	„Finder“	Jahr
0	3	Primzahl		
1	5	Primzahl		
2	17	Primzahl		
3	257	Primzahl		
4	65537	Primzahl		
5	4294967297	641*6700417	Euler	1732
6	65 Bit	faktorisiert	Landry & Le Lasseur	1880
7	129 Bit	faktorisiert	Brillhard & Morrison	1970
8	257 Bit	faktorisiert	Brent & Pollard	1980
9	513 Bit	faktorisiert	Lenstra et. al.	1990

Das Zahlkörpersieb, welches die Faktorisierungstechnik entscheidend verbesserte, geht auf die Idee von John Pollard zurück (siehe Tabelle 2.1). Dieses Beispiel zeigt, wie die Idee eines einzelnen Mathematikers plötzlich bis dahin sichere Schlüssellängen unsicher erscheinen lässt.

Hans Dobbertin zeigte eine Kollision in der Hashfunktion MD4, d. h. er erzeugte zwei verschiedene Dokumente, die MD4 auf denselben Hashwert abbildete, und widerlegte damit die bis dato gemachte Annahme, MD4 sei kollisionsresistent [Dobb96].

Daniel Bleichenbacher präsentierte einen Angriff auf RSA mit PKCS#1 v1.5-Padding und SSL v3.0. Die Grundidee des Angriffs ist die folgende: Sei (n, e) ein öffentlicher RSA-Schlüssel und d der zugehörige private Schlüssel. Ziel ist die Signierung eines Textes c, d. h. $m \equiv c^d \bmod n$ ist

gesucht. Der Angreifer wählt eine Zahl s, berechnet $c' \equiv cs^e \bmod n$ und lässt c' vom „Opfer“ signieren, $c'^d \bmod n$. Damit erhält der Angreifer m, da er $c'^d = c^d s \equiv ms \bmod n$ und s kennt. Um diese Tatsache ausnutzen zu können, muss der Angreifer sein Opfer dazu bringen, ein Kryptogramm zu signieren, was das Opfer wahrscheinlich freiwillig nicht tun wird. Bleichenbacher nutzt deshalb eine Schwäche in PKCS#1 v1.5-Padding aus: Einem SSL-Server wird besagtes c' präsentiert, welcher dann zunächst mit seinem privaten Schlüssel d das Kryptogramm entschlüsselt, $c'^d \bmod n$, und anschließend prüft, ob das Ergebnis PKCS#1-konform ist, d. h. mit 00 02 beginnt. Bleichenbachers Idee beruht darauf, dass sich anhand der Reaktion des SSL-Servers ablesen lässt, ob c' eine gültige RSA-Verschlüsselung mit PKCS#1-Padding war oder nicht, d. h. der Angreifer weiß nun, ob ms mit 00 02 beginnt oder nicht. Abhängig von der Reaktion startet der Angreifer eine erneute Anfrage an den Server, so dass er schließlich eine große Anzahl von „Kryptogramm-Reaktion-Paaren“ vorliegen hat, aus denen sich mit statistischen Methoden m berechnen lässt. Der Angriff ist abhängig von den konkreten Implementierungen, die unterschiedliche Informationen herauslassen. Bleichenbacher erwähnt in [Blei98] Tests mit verschiedenen 512-Bit- und 1024-Bit-Schlüsseln, die zwischen 300 000 und 2 000 000 Ciphertexts benötigten, und Tests mit drei SSL-Servern, wobei bei zweien der Angriff erfolgreich funktionierte. Ein solcher Angriff, bei dem der Angreifer eine Anfrage zum Opfer schickt, eine Antwort erhält und eine von der Antwort abhängige erneute Anfrage startet, wird auch Adaptive Chosen Ciphertext Attack genannt.

Aktuell ist der Vorschlag von Daniel Bernstein, mit spezieller Hardware eine Beschleunigung von Faktorisierungsalgorithmen um den Faktor 3 zu erreichen. Diese Entwicklung wurde bei der Festlegung der geeigneten Kryptoalgorithmen bis zum Jahr 2007 berücksichtigt [BSI02].

Ein weiterer Grund für einen Schaden können Implementierungs- oder Organisationsfehler sein. Obwohl Anwendungen geprüft werden, belegen die immer wiederkehrenden Nachrichten von Updates und Patches, dass Implementierungsfehler nicht ausgeschlossen werden können. Ebenfalls nicht auszuschließen ist menschliches Fehlverhalten, obwohl organisatorische Maßnahmen dies zu verhindern versuchen.

Weitere mögliche Schäden können von neuen Angriffen auf bestehende Implementierungen herrühren. Etwa Angriffe auf Chipkarten, die Reaktionen auf gezielte Fehler mittels Differential Fault Analysis (DFA) [BiSh97] ausnutzen, die den Stromverbrauch durch Simple Power Analysis (SPA) oder Differential Power Analysis (DPA) [JaJK99] analysieren oder die Berechnungszeiten via Timing Attacks verwenden [Koch96]. Ein Ziel solcher Angriffe ist, ECDSA zu kompromittieren [RoSe01] [Seif02].

Eine weitere plötzliche Verbesserung wäre ein plötzlicher „Quantensprung“ bei der Rechenleistung, z. B. durch einen Quantencomputer. Dieser könnte durch seine massive Parallelität Probleme der Klasse NP effizient lösen, wodurch alle gängigen Public-Key-Kryptoverfahren unsicher werden würden. Beispielsweise hat Peter Shor 1994 einen Algorithmus für Quantencomputer entworfen, um Zahlen zu faktorisieren. [Stie01] berichtete Ende 2001, dass es gelungen sei, die Zahl 15 mit einem Quantencomputer zu faktorisieren. Ob und wann Quantencomputer die Kryptographie beeinflussen werden, ist derzeit unbekannt.

All diese Beispiele zeigen, dass Fortschritte in effizienten Algorithmen zum Faktorisieren großer Zahlen oder Berechnen diskreter Logarithmen oder Fehler in Implementierungen plötzlich und unerwartet aufgetreten sind. Für die Zukunft lassen sich weitere Schäden nicht ausschließen. In der Folge könnte ein solches Ereignis die Sicherheit der heutigen Public-Key-Kryptographie beschädigen, wie im nächsten Abschnitt ausführlich dargelegt wird.

2.2 Probleme

Algorithmen in der Public-Key-Kryptographie sind nicht beweisbar sicher. Es gibt nur Empfehlungen über geeignete Algorithmen, Parameter und Schlüssellängen. Diese Empfehlungen können sich plötzlich ändern. Mit den daraus resultierenden Problemen beschäftigt sich dieser Abschnitt. Zunächst werden mögliche Schäden exakt definiert und anschließend die Folgen für Public-Key-Infrastrukturen geschildert. Eine Analyse existierender Lösungsvorschläge führt zu vier offenen Problemen, die in dieser Dissertation gelöst werden.

2.2.1 Mögliche Schäden

Kryptographische und einsatzspezifische Komponenten – also Verfahren, Parameter und Schlüssel – können ihre kryptographische Eignung verlieren. In diesem Abschnitt wird definiert, wann ein Signatur- oder Verschlüsselungsalgorithmus, eine Hashfunktion, eine Formatierung, ein daraus zusammengesetztes Verfahren oder ein Schlüssel kryptographisch ungeeignet ist.

Schaden 1: Signaturalgorithmus (nach Definition 1 auf Seite 3) kryptographisch ungeeignet

Ein Signaturalgorithmus σ gilt als kryptographisch ungeeignet, wenn die in der Definition an ihn gestellten Anforderungen nicht mehr gegeben sind, d. h. wenn mindestens eine der folgenden Aussagen zutrifft:

- Es gibt einen effizienten Algorithmus, der einen privaten Schlüssel prK zu σ berechnet.
- Es gibt einen effizienten Algorithmus, der ohne Kenntnis des privaten Schlüssels prK eine Signatur (b', s') erzeugt, für die gilt: $\tau(b', s', \text{puK}) = \text{TRUE}$.

Mögliche Inputs der Algorithmen:

- der öffentliche Schlüssel puK zu prK
- eine Menge von Schlüsselpaaren (prK′, puK′)
- eine Menge von Bitstring-Signatur-Paaren $\{(b, s)\}$ mit $s = \sigma(b, \text{prK})$

Es ist zu unterscheiden, ob Schlüssel parameterunabhängig oder für bestimmte Parameter gebrochen werden, was z. B. eine bestimmte elliptische Kurve oder eine bestimmte Schlüssellänge sein kann. Die betroffene Menge von Schlüsseln M ist eine Teilmenge aller benutzter Schlüssel – insbesondere kann M alle oder nur einen Schlüssel beinhalten; für eine ein-elementige Menge M sei auf Schaden 7 verwiesen.

Zu bemerken ist, dass der Signaturalgorithmus RSA ohne Formatierung oder Hashfunktion aufgrund der multiplikativen Eigenschaft kryptographisch ungeeignet sein kann.

Schaden 2: Hashfunktion (nach Definition 2 auf Seite 4) kryptographisch ungeeignet

Eine Hashfunktion κ gilt als kryptographisch ungeeignet, wenn die in der Definition an sie gestellten Anforderungen nicht mehr gegeben sind, d. h. wenn mindestens eine der folgenden Aussagen zutrifft:

- Es gibt einen effizienten Algorithmus, der zu einem Hashwert y ein b' berechnet, so dass $y = \kappa(b')$ gilt.
- Es gibt einen effizienten Algorithmus, der zu einem Input b ein zweites b' findet, so dass $\kappa(b) = \kappa(b')$ gilt.
- Es gibt einen effizienten Algorithmus, der zwei Inputs b und b' findet, die auf denselben Hashwert $\kappa(b) = \kappa(b')$ abgebildet werden.

Es ist zu unterscheiden, ob nur ein einzelner Hashwert, Hashwerte einer bestimmten Struktur oder alle Hashwerte betroffen sind.

Schaden 3: Formatierung (nach Seite 6) kryptographisch ungeeignet

Eine Formatierung ϕ gilt als kryptographisch ungeeignet, wenn die an sie gestellte Anforderung nicht mehr gegeben ist, d. h. wenn durch ϕ die multiplikative Eigenschaft eines Signaturalgorithmus ausgenutzt werden oder die Randomisierung eines Verschlüsselungsalgorithmus fehlschlagen kann. Dieser Schaden schließt eine kryptographisch ungeeignete Redundanzfunktion ρ (nach Seite 6), als eine besondere Formatierung, mit ein.

Folgeschaden 4: Signaturverfahren (nach Definition 3 auf Seite 6) kompromittiert

Ein Signaturverfahren sign heißt kompromittiert, falls sich mindestens eine der Komponenten Signaturalgorithmus σ, Hashfunktion κ oder Formatierung ϕ als kryptographisch ungeeignet erweisen sollte.

Schaden 5: Verschlüsselungsalgorithmus (nach Definitionen 4 und 5 auf Seite 10) kryptographisch ungeeignet

Ein asymmetrischer bzw. symmetrischer Verschlüsselungsalgorithms η_asym resp. η_sym gilt als kryptographisch ungeeignet, wenn die in der Definition an ihn gestellten Anforderungen nicht mehr gegeben sind, d. h. wenn mindestens eine der folgenden Aussagen zutrifft:

- Es gibt einen effizienten Algorithmus, der einen privaten Schlüssel prK zu η_asym bzw. einen geheimen Schlüssel sK zu η_sym berechnet.
- Es gibt einen effizienten Algorithmus, der aus einem Kryptogramm $\delta\_asym(p, \mathrm{prK})$ bzw. $\delta\_sym(p, \mathrm{sK})$ den Klartext p berechnet.

Mögliche Inputs der Algorithmen:

- der öffentliche Schlüssel puK zu prK
- eine Menge von Schlüsselpaaren (prK′, puK′)

- eine Menge von Chiffretexten $\{\delta\_\text{asym}(p', \text{prK})\}$ bzw. $\{\delta\_\text{sym}(p', \text{sK})\}$
- eine Menge vermuteter Klartexte p, aufgrund von Kontext, Struktur o. ä.
- eine Menge von Klartext-Chiffretext-Paaren $\{(p'', c'')\}$ mit $c'' = \delta\_\text{asym}(p'', \text{prK})$ bzw. $c'' = \delta\_\text{sym}(p'', \text{sK})$

Wie bei Schaden 1 ist zu unterscheiden, ob Schlüssel parameterunabhängig oder nur für bestimmte Parameter gebrochen werden. Die betroffene Menge von Schlüsseln M kann eine Teilmenge aller benutzten Schlüssel sein – muss aber nicht. Für $\#M = 1$ sei auf Schaden 7 verwiesen.

Zu beachten ist, dass eine Verschlüsselung ohne Redundanz bei einem zu kleinen Klartext-Raum, d. h. zu wenigen möglichen Klartexten, kryptographisch ungeeignet sein kann.

Folgeschaden 6: Verschlüsselungsverfahren (nach Definition 6 auf Seite 12) kompromittiert

Ein Verschlüsselungsverfahren enc_sym bzw. enc_asym heißt kompromittiert, falls sich mindestens eine der Komponenten symmetrischer Verschlüsselungsalgorithmus η_sym, asymmetrischer Verschlüsselungsalgorithmus η_asym oder Formatierung ϕ als kryptographisch ungeeignet erweisen sollte.

Schaden 7: Ein Schlüssel kompromittiert

Ein privater Schlüssel prK bei einem asymmetrischen Verfahren bzw. ein geheimer Schlüssel sK bei einem symmetrischen Verfahren heißt kompromittiert, wenn er bekannt geworden ist. Betroffen sein kann

- ein einzelner privater Schlüssel eines Zertifikatsinhabers (CH-Key),
- der private Schlüssel der Wurzelzertifizierungsinstanz (RootCA-Key),
- der private Schlüssel einer Zertifizierungsinstanz (CA-Key),
- der private Schlüssel einer Zertifizierungsinstanz für Sperrauskünfte (CRL- oder OCSP-Key),
- der private Schlüssel einer Zeitstempelinstanz (TSA-Key) oder
- ein geheimer Schlüssel, mit dem eine Kommunikation verschlüsselt wird.

2.2.2 Folgen

In diesem Abschnitt werden die unmittelbaren Folgen für kryptographisch abgesicherte Daten diskutiert, falls einer der zuvor diskutierten Schadensfälle eingetreten sein sollte.

Angenommen, einer der zuvor beschriebenen möglichen Schäden tritt ein. Und weiter angenommen, der Schaden tritt plötzlich und unerwartet auf, so dass keine Möglichkeit mehr besteht zu reagieren, um etwa Daten auf anderen Medien zu sichern oder kryptographische Verfahren und Parameter anzupassen und ein Re-Signieren durchzuführen. Die Schäden können den Verlust der Integrität, der Authentizität und der Vertraulichkeit übermittelter Daten bewirken.

Verlust der Integrität

Angenommen, eine Hashfunktion κ wird als kryptographisch ungeeignet eingestuft. Die Folgen sind davon abhängig, wie effizient eine Kollision in κ herbeigeführt werden kann und wie sinnvoll zwei verschiedene Texte D und D^* für den Kontext sind, die auf denselben Hashwert abgebildet werden: $\kappa(D) = \kappa(D^*)$. Wenn eine Kollision effizient und sinnvoll erreicht werden kann, kann keine gesicherte Aussage über die Integrität signierter Daten mehr gemacht werden, weil u. U. zwei verschiedene Dokumente D und D^* dieselbe Signatur S tragen und sich die Frage stellt: Welches Dokument ist das ursprünglich signierte?

Darüber hinaus kann das Bekanntwerden einer einzigen solchen effizienten und sinnvollen Kollision ausreichen, die Beweiskraft digitaler Signaturen, die diese Hashfunktion nutzen, zu untergraben. Ein Beispiel schildert, was passieren könnte.

Alice und Bob
Alice und Bob haben einen Vertrag elektronisch geschlossenen, in dem Alice Bob zusichert, eine Ware W zu einem Preis P abzunehmen. Der Vertrag D wurde von Alice digital signiert – $S = \mathrm{sign}(D, \mathrm{prK}_{\mathrm{Alice}})$ – und das dabei benutzte Signaturverfahren sign enthält die nun kompromittierten Hashfunktion κ. Kurze Zeit später ist Alice mit dem Handel unzufrieden – aber aufgrund des Vertrages daran gebunden.
Nach Bekanntwerden, dass κ kryptographisch ungeeignet ist, weigert sich Alice, die Ware abzunehmen, worauf Bob einen Richter einschaltet. Die Argumentation von Alice ist, sie hätte nie einen Kontrakt mit Bob geschlossen und erklärt die vorliegende Signatur S dadurch, dass sie stets alle E-Mails – auch zu anderen PKI-Nutzern – digital signiert und dass Bob ihre Kommunikation abgehört und ein Dokument D^* mit Signatur S mitgeschnitten haben muss. Da κ kryptographisch ungeeignet ist, argumentiert Alice weiter, kann Bob zu D^* das Dokument D erzeugt haben, welches sie signiert haben soll, weil die Signatur S auch zu D passt.
Wem der Richter glauben würde, hängt davon ab, ob Alice das Dokument D^* vorweisen kann oder – falls sie ihre signierten E-Mails nicht archiviert – wie effektiv sich eine sinnvolle Kollision herbeiführen lässt. Da allerdings tatsächlich eine Kompromittierung von κ bekannt geworden ist, könnte der Richter Alice Glauben schenken.

Die Konsequenzen sind ungewiss, da bislang kein solcher Fall vor Gericht verhandelt wurde. Zumindest würde ein solcher Schaden die Rechtssicherheit empfindlich stören.

Betroffen sind in erster Linie digitale Signaturen in Signatur-Anwendungen, weil zu signierende Texte frei zu gestalten sind. Demgegenüber können Zertifikate, Sperrinformationen oder digitale Signaturen in Authentisierungs-Anwendungen aufgrund ihrer speziellen Form weniger betroffen sein. Das Kompromittieren einer Hashfunktion betrifft Signaturen unabhängig vom Erstellungszeitpunkt, d. h. sowohl vor einem Schaden erzeugte Signaturen als auch Signaturen, die erst nach einem Schaden entstehen. Es sind auch Signaturen betroffen, die mit einem Zeitstempel versehen wurden, weil Zeitstempel nur den Hashwert zeitstempeln.

Verlust der Authentizität

Angenommen, ein Signaturverfahren wird kompromittiert. Eine gesicherte Aussage über die Authentizität signierter Daten kann in diesem Fall nicht mehr gemacht werden, weil nicht mehr festzustellen ist, ob es sich bei einer fraglichen Signatur um eine authentische (d. h. vom rechtmäßigen Besitzer des privaten Schlüssels autorisierte) oder eine gefälschte Signatur handelt. Dabei ist der Erstellungszeitpunkt von Bedeutung: Denn vor einem Schaden erzeugte Signaturen bleiben gültig, sofern nachweisbar ist, dass sie tatsächlich vor einem Schaden erzeugt wurden und zwischenzeitlich nicht verändert werden konnten.

Die Frage ist also, wie sich der Erstellungszeitpunkt eindeutig feststellen lässt. Ein mitsigniertes Datum ist nicht ausreichend, weil es zurückdatiert sein kann. Heutzutage üblich ist ein von einem vertrauenswürdigen Zeitstempeldienst erzeugter Zeitstempel auf einem Dokument. Das Problem ist, dass auch Zeitstempel kryptographische Erzeugnisse und wie zuvor beschrieben ebenfalls vor Schäden nicht gefeit sind.

Betroffen sind alle Arten von Signaturen. Zunächst die für den Betrieb der PKI notwendigen Signaturen in Zertifikaten oder Sperrinformationen, dann Signaturen in Authentisierungs-Anwendungen, in denen solche Schäden einen unerlaubten Zutritt durch virtuelle oder reale Zugangskontrollen ermöglichen kann, oder digitale Signaturen in Signatur-Anwendungen, wo die Beweiskraft geschwächt werden kann, falls die digitalen Signaturen nicht zusätzlich gesichert sind. Beispielsweise werden in Banken Signaturen mit einem „Medien-Bruch“ in abgesicherten Archiven und durch Back Ups gesichert, wodurch die Beweiskraft einer digitalen Unterschrift nicht nur an der digitalen Signatur selbst hängt, sondern auch auf einer physikalischen Unversehrtheit basiert. Hier haben Sicherheitsbeauftragte aufgrund von Richtlinien den Auftrag, sich um die Sicherung der Daten zu kümmern.

Verlust der Vertraulichkeit

Angenommen, ein Verschlüsselungsverfahren wird kompromittiert. Eine gesicherte Aussage über die Verschwiegenheit übertragener Daten kann in diesem Fall nicht mehr gemacht werden, weil nicht festzustellen ist, ob ein bestimmtes Kryptogramm bereits von anderen Personen entschlüsselt wurde. Betroffen sind Kryptogramme in Signatur- und Authentisierungs-Anwendungen, die nur kurzfristig oder auf lange Sicht geheim gehalten bleiben sollen.

Maßnahmen, Daten rechtzeitig anderweitig zu sichern oder kryptographische Verfahren und Parameter anzupassen, sind nutzlos, weil einmal übertragene Daten von Angreifern mitgehört, abgespeichert und – nachdem die Verfahren schwach sind – entziffert werden können. Das Problem der Langzeit-Verschlüsselung wird in dieser Dissertation nicht thematisiert.

Darüber hinaus liegt der Fokus dieser Arbeit auf Verschlüsselungen, für die eine Public-Key-Infrastruktur benötigt wird: Die effektive Verschlüsselung von Daten erfolgt mit einem symmetrischen Verfahren und der dafür notwendige geheime Schlüssel wird mit einem asymmetrischen Verfahren verschlüsselt. Die PKI ist zwischen Teilnehmern nötig, die den geheimen Schlüssel zuvor nicht persönlich ausgetauscht haben. Verschlüsselungen von Daten oder Festplatten mit Passwörtern oder auf Chipkarten abgelegten Schlüsseln werden daher nicht betrachtet. Dieser Arbeit wird ferner zugrunde gelegt, dass verschlüsselt übertragene Nachrichten nach Empfang beim Empfänger zunächst ent- und – sofern eine erneute Datei- oder Festplatten-Verschlüsselung stattfinden soll – anschließend mit anderen Methoden neu verschlüsselt werden.

Heutige Reaktionen

Wenn ein Schaden bekannt wird, wird dieser zunächst von den entsprechenden Stellen, etwa dem BSI oder dem A9C, geprüft. A9C schreibt dazu: “The management activities to introduce new algorithms and their parameters and to delete algorithms and their parameters from the list need to respond to the following situation[s]: [...] (3.) In the case of new attacks the immediate need to remove an algorithm can arise. Through A9C, mechanisms shall be put in place that can react within [...] at most 1 month for case 3“ [EU A9C01, 4.1].

Ist ein Schaden von praktischer Relevanz, werden vom Schaden betroffene Zertifikate von den Trust Centern revoziert: Zertifikate werden nicht nur revoziert, falls sich Änderungen in den

Eintragungen ergeben oder der zertifizierte Schlüssel verloren geht (Verlust der zugehörigen PIN/Passwort), sondern auch, wenn Schlüssel kompromittiert werden oder kompromittiert erscheinen [Road00]. Eine Sperrung kann der Zertifikatsinhaber selbst, eine im Zertifikat vermerkte Person oder aber das Trust Center veranlassen, wie den Allgemeinen Geschäftsbedingungen der Signatur-Anwendungen von Deutscher Post oder Deutscher Telekom zu entnehmen ist: „Deutsche Post-Signtrust sperrt das ausgestellte Zertifikat auch, wenn die den angewendeten Verfahren zugrunde liegenden Algorithmen gebrochen wurden“ [Post01]. „§5 Sperrung von Zertifikaten – Die Deutsche Telekom sperrt – auch ohne entsprechenden Auftrag des Nutzers – das ausgestellte Zertifikat, wenn [...] die dem Signaturverfahren zugrunde liegenden Algorithmen gebrochen wurden [...]“ [Tele01]. Revozierte Zertifikate werden aus der Liste der gültigen Zertifikate entfernt und in Sperrlisten eingetragen.

Durch diese Maßnahme wird die Sicherheit der PKI wiederhergestellt, weil jeder Zertifikatsinhaber Signaturen und Zertifikate via CRL oder OCSP validieren und je nach Validierungsergebnis Signaturen akzeptieren oder Verschlüsselungen durchführen kann. Wie schnell eine Revokation Wirkung zeigt, hängt von der konkreten Anwendung und der dort definierten Security Policy ab, die den Umfang und den Einsatz der Validierung vorschreibt; beispielsweise Validierung der gesamten Zertifikatskette, Validierung bei Verschlüsselung, Validierung nur im Verifizier- oder auch im Signierprozess.

Übrigens bleiben beim deutschen Signaturgesetz wegen des Kettenmodells Zertifikate gültig, auch wenn das darüberliegende CA-Zertifikat revoziert wurde. Das bedeutet einerseits, dass Zertifikaten, denen vor der Revokation des CA-Zertifikats vertraut wurde, auch weiterhin Glauben geschenkt werden kann. Das bedeutet aber andererseits auch, dass wenn ein Angreifer den privaten Schlüssel der CA gebrochen hat und mit diesem Schlüssel der CA neue Zertifikate generiert und in Umlauf bringt, es keine Möglichkeit gibt, diese Zertifikate zu sperren. Entscheidend ist also wie im Abschnitt „Verlust der Authentizität“ auf Seite 52 der nachweisbare Erstellungszeitpunkt des Zertifikats, welcher sich nicht mit dem darin enthaltenen und mitsignierten Gültigkeitszeitraum belegen lässt. Aus diesem Grund kann das Kettenmodell problematisch werden.

2.2.3 Offene Probleme

Die derzeitige Herangehensweise, mit den – keinesfalls nur theoretischen – Gefahren moderner Informationssysteme umzugehen, ist nicht ausreichend. Es bleiben wichtige Probleme offen, die im Folgenden detailliert beschrieben werden.

Problem: Unzureichende Revokationsmechanismen

Angenommen, ein Angreifer kompromittiert den Zertifizierungsschlüssel einer CA. Dann kann der Angreifer neue Zertifikate für untergeordnete Zertifizierungsinstanzen oder Zertifikatsinhaber erzeugen. Bei der Revokation dieser Zertifikate können zwei Problemen auftreten:

1. Sperrlisten und Zertifikats-Status-Antworten sind von der CA – mit einem für diesen Zweck zertifizierten Schlüssel – digital signiert. Die Revokationsinformation ist allerdings wertlos, weil ein Angreifer im Schadensfall in der Lage ist, diese Information beliebig zu manipulieren, digital zu signieren und in offenen Netzen relativ leicht gegen die von der CA ausgestellte Original-Revokationsinformation auszutauschen.

2. In Sperrlisten werden die Zertifikatsseriennummern revozierter Zertifikate aufgeführt. Falls ein Angreifer mit einem kompromittierten CA-Schlüssel neue Zertifikate mit fiktiven Seriennummern erzeugt hat, gibt es für die CA keine Möglichkeit, diese Zertifikate – bevor ihre Fälschung auffällt – in einer Sperrliste aufzuführen. Anders sieht es bei Zertifikats-Status-Antworten aus, deren OCSP-Server als Datenbasis für die Antwort nicht nur Sperrlisten, sondern auch Positivlisten gültiger Zertifikate nutzen, weil hier ein für die CA unbekanntes Zertifikat als „nicht gültig“ gekennzeichnet werden würde.

Die Verbreitung der Information über kompromittierte Zertifizierungsschlüssel und ihre Folgen mittels “out-of-band“-Kommunikation – z. B. via Post, Zeitung, TV, Radio o. ä. – ist nicht für jede Anwendung und jeden Benutzerkreis praktikabel.

Dass Schlüssel einer Zertifizierungsinstanz und insbesondere der Wurzelzertifizierungsinstanz kompromittiert werden könnten, wird bei PKIX, einem Standard für Public-Key-Infrastrukturen, als „katastrophal“ eingeschätzt: “Compromise of a private key associated with a Root CA is catastrophic for users relying on that Root CA. If a Root CA’s private key is compromised, that CA’s Public Key Certificate (PKC) must be revoked and all PKCs subordinate to it must also be revoked“ [Road00, 3.5.6.2].

Eine weitere Frage betrifft die Praktikabilität der Sperrlisten. Heutige Systeme sind eher darauf ausgelegt, dass nur ein kleiner Prozentsatz der ausgegebenen Zertifikate jemals revoziert wird. Die Möglichkeit, dass u. U. sehr viele Zertifikate gesperrt werden müssen und eine Sperrliste dementsprechend viele Zertifikatsseriennummern enthalten müsste, was eine Sperrliste groß und unhandlich machen würde, wird nicht berücksichtigt. Eine einfache Lösung, das in der PKI-Hierachie oberste Zertifikat zu sperren und bei jeder Validierung stets die gesamte Zertifizierungskette zu betrachten, wird über das im Signaturgesetz SigG geforderte Kettenmodell (siehe Seite 22) verhindert, weil ein gesperrtes CA-Zertifikat nicht zu einer Sperrung von darunter liegenden Teilnehmer-Zertifikaten führt.

Falls eine Hashfunktion kompromittiert wird, die in Zertifikaten nicht aufgeführt ist, stellt sich die Frage, wie Zertifikatsinhaber vor dieser Hashfunktion gewarnt werden können. Die Revokation einer Hashfunktion kann zwar in der vom BSI veröffentlichten Liste der geeigneten Kryptoalgorithmen, nicht aber automatisch bei den CHs erfolgen.

Offenes Problem: Verlust der Verfügbarkeit von PKI-Anwendungen

Dieses Problem ist das Hauptproblem. Denn durch Revokation von Zertifikaten können digitale Signaturen und Verschlüsselungen nicht mehr die an sie gestellten Anforderungen erfüllen:

- Eine digitale Signatur, deren Zertifikat revoziert ist, kann nicht mehr als gültig validiert werden und sollte abgelehnt werden. Die Folge: Verlust der Beweisbarkeit digitaler Signaturen in Signatur-Anwendungen und Verlust der gesicherten Authentizität in Authentisierungs-Anwendungen.
- Eine Verschlüsselung von Daten mit dem öffentlichen Schlüssel des Empfängers, dessen Zertifikat revoziert ist, sollte vermieden werden, weil die Vertraulichkeit nicht mehr gewährleistet sein kann.

Je nach Umfang der von einem Schaden betroffenen Zertifikate schränkt dies die Funktionsfähigkeit der PKI und damit die diese PKI nutzenden Anwendungen ein.

PKIX hat diesen Punkt erkannt: „Until such time as the Root CA has been issued a new Public Key Certificate (PKC) and the Root CA issues PKCs to users relying upon it, users relying on that Root CA are cut off from the rest of the system, as there is no way to develop a valid certification path back to a trusted node“ [Road00, 3.5.6.2].

Die Wiederherstellung der PKI kann dann die Entwicklung und Produktion neuer Komponenten nach sich ziehen, was Zeit und Geld kosten würde. Werden Chipkarten eingesetzt, müssten diese u. U. erst bestellt und produziert werden, was gerade dann besonders lange dauern dürfte, wenn eine erhöhte Nachfrage nach Chipkarten einsetzt, weil mehrere PKIs neu aufgebaut werden müssten.

Darüber hinaus müssten Software, Schlüssel und Zertifikate neu ausgegeben werden. Die Zertifikatsinhaber müssten von der Authentizität der Software überzeugt werden und die Zertifizierungsinstanz müsste u. U. Zertifikatsinhaber neu zertifizieren. Auf Basis der bisherigen, nun kompromittierten Public-Key-Infrastruktur kann aber keine neue PKI aufgebaut werden. Die von PKIX vorgeschlagene “out-of-band“-Kommunikation, um Teilnehmer etwa via Tageszeitung über den Wechsel eines CA-Schlüssels zu informieren (siehe [Road00, 3.5.6.2]) oder die Echtheit von Software zu beweisen (etwa mittels eines abgedruckten Hashwertes), ist bei sicherheitsrelevanten Anwendungen nicht zu verantworten und darüber hinaus unpraktikabel, wenn Benutzer neue Software, neue Schlüssel und neue Sicherheitsanker „per Hand“ in ihr System („Vertrauen Sie diesem Zertifikat, so klicken Sie ‚Ja‘!“) einpflegen sollen.

Dieser Punkt betrifft alle Anwendungen und es gibt keine Ansätze zu einer Lösung.

Dieses Problem kann – in abgeschwächter Form – auch dann eintreten, wenn gar kein Schadensfall eingetreten ist, sondern sich die Empfehlungen zu geeigneten Kryptoalgorithmen mit ihren Parametern und Schlüssellängen ändern und dies ein Anpassen der Implementierung bedeuten würde. Zitat aus der Empfehlung vom 5. Juli 2001 [BSI01]: „Der Schritt von 1024 auf 2048 Modullänge für den RSA-Algorithmus stellt eine Erhöhung dar, die – je nach Implementierung – erhebliche Aufwände an die Realisierung in Hard- und Software stellt. Abweichend von der letztjährigen Fassung wurde deshalb [...] der Zeitraum, in dem die alten Parameterlängen verwendet werden können, auf Ende 2006 verlängert.“

Dies zeigt, dass selbst der „Normalfall“ – also die konzeptionell bedingte und wohlbekannte Tatsache, dass Anpassungen nötig ist – für einige Implementierungen sehr problematisch werden kann – vergleichbar dem Jahr-2000-Problem. Deshalb ist eine flexible Integration der Kryptokomponenten in eine Public-Key-Infrastruktur – so wie es die FlexiPKI realisiert – nötig und das Fail-Safe-Konzept kann auch im Nicht-Schadensfall dazu genutzt werden, neue Kryptoverfahren sicher und schnell im laufenden Betrieb der PKI zu verteilen.

Offenes Problem: Verlust der Verbindlichkeit digitaler Signaturen

Durch Kompromittierung des Signaturverfahrens oder privater Schlüssel können auch vor einem Schaden erzeugte digitale Signaturen ihre Beweiskraft (Authentizität und Integrität) verlieren.

Zeitstempel schützen nur unzureichend, weil auch sie von Schäden betroffen sein können. Ein Re-Signieren benötigt eine gewisse Vorlaufzeit und wirkt daher nicht, wenn ein Schaden plötzlich und unerwartet auftritt.

Die Beweiskraft digitaler Signaturen kann erhalten bleiben, wenn die Signaturen zusätzlich abgespeichert werden, die Beweiskraft also – wie in Banken oder dem Grundbuchamt – auch auf die Sicherheit eines Speichermediums in einer vertrauenswürdigen Instanz übergeht.

Offenes Problem: Verlust der Vertraulichkeit verschlüsselter Daten

Durch Kompromittierung des Verschlüsselungsverfahrens oder privater bzw. geheimer Schlüssel können verschlüsselt übertragene Daten entschlüsselt werden. Diese Daten können eine Nachricht selbst oder ein Session Key sein. Betroffen sind Verschlüsselungen in einer Signatur-Anwendung oder eine verschlüsselte Kommunikation in einer Authentisierungs-Anwendung.

Da das Abhören und Aufzeichnen von verschlüsselt übertragenen Daten über offene Netze nicht zu vermeiden ist und es ebenso unvermeidbar ist, dass diese verschlüsselten Daten von Unbefugten entschlüsselt werden können, wenn die benutzten Verfahren schwach geworden sind, liegt der Fokus in dieser Arbeit auf übertragenen Nachrichten, die relativ kurzfristig vertraulich bleiben sollen. Das bedeutet, dass für diese gerade beim Militär oder in Botschaften wichtigen Langzeit-Verschlüsselungen in dieser Arbeit keine Antwort gegeben werden kann.

2.3 Zusammenfassung

Die heutigen Signatur- und Verschlüsselungsverfahren sind nicht beweisbar sicher. Unter Berücksichtigung von Empfehlungen an Kryptoverfahren und Parameter, die sich an Fortschritten in der Computertechnik und wissenschaftlichen Erkenntnissen orientieren, kann eine für die Praxis hinreichende Sicherheit von Signatur- und Verschlüsselungsverfahren erreicht werden. Die Empfehlungen gelten für 5 – 6 Jahre.

Dennoch ist nicht auszuschließen, dass ein plötzliches und unerwartetes Ereignis für die Kompromittierung einer Klasse von Schlüsseln sorgt – wie die Beispiele aus der Vergangenheit auch belegen. Da PKIs zunehmend angewendet werden, können diese Schadensfälle schwerwiegende wirtschaftliche und gesellschaftliche Auswirkungen zur Folge haben.

Es gibt zwei Arten von Schäden, die sich durch ihre Schlüsselabhängigkeit unterscheiden:

1. Kompromittieren einer kryptographischen Komponente, wie Signaturalgorithmus, Verschlüsselungsalgorithmus, Hashfunktion oder Formatierung. Dieser Schaden betrifft alle Schlüssel zum jeweiligen Verfahren.

2. Kompromittieren von einsatzspezifischen Komponenten, wie Systemparametern oder einer Klasse von Schlüsseln zu einem Algorithmus. Diese Klasse von Schlüsseln ist eine Teilmenge aller Schlüssel und kann insbesondere ein einzelner Schlüssel sein.

Diese Schäden implizieren vier offene Probleme:

1. Verlust der Verfügbarkeit von PKI-Anwendungen

2. Verlust der Beweisbarkeit digitaler Signaturen

3. Verlust von praktikablen Revokationsmechanismen

4. Verlust der Vertraulichkeit verschlüsselter Daten

In dieser Arbeit wird zu jedem offenen Problem eine Lösung für sicherheitsrelevante Public-Key-Infrastrukturen präsentiert.

Kapitel 3

Fail-Safe-Konzept

Die Grundidee zur Lösung der offenen Probleme besteht darin, kryptographische und einsatzspezifische Komponenten so flexibel in eine Public-Key-Infrastruktur zu integrieren, dass ihr Hinzufügen und Löschen und damit ein Anpassen an aktuelle Sicherheitskriterien in der Produktivphase einer Anwendung möglich ist. Die flexible Public-Key-Infrastruktur „FlexiPKI“, die an der TU Darmstadt im Fachgebiet „Kryptographie, Computeralgebra“ derzeit entsteht, folgt dieser Philosophie (siehe [FlexiPKI] oder [BuRT00]). Die FlexiPKI ist in Java programmiert und kryptographische Komponenten werden dem System von so genannten Providern zur Verfügung gestellt. Wenn es nötig sein sollte, kryptographische Komponenten zu aktualisieren, so brauchen in der FlexiPKI nur die entsprechenden Provider hinzugefügt oder gelöscht zu werden, was einen Vorteil gegenüber PKIs bedeutet, welche die kryptographischen Komponenten statisch implementiert haben.

Durch eine flexible Gestaltung der Komponenten in einer PKI ist das Problem der Verfügbarkeit einer PKI-Anwendung im Schadensfall allerdings noch nicht gelöst, weil zwar neue kryptographische und einsatzspezifische Komponenten in die bestehende FlexiPKI integriert werden können, die Zertifizierungsinstanz diese neuen Komponenten und dazu passende neue Schlüssel und Zertifikate erst noch an die Zertifikatsinhaber auf sichere Weise verteilen muss. Da hierzu aufgrund des Schadens aber keine PKI zur Verfügung steht, welche dies “in-band“ – d. h. mit Mitteln einer PKI – durchführt, und da die konventionelle Registrierungs- und Verteilungsprozedur zu lange dauert, müssen schon vorher entsprechende Vorkehrungen getroffen werden.

Das Fail-Safe-Konzept beinhaltet deshalb über die Grundidee der flexiblen Komponenten-Integration hinaus die Idee, mehrere verschiedene Kryptoverfahren samt Schlüsseln und Zertifikaten in der PKI parallel bereitzustellen – also Nutzung „multipler Kryptographie“ in einer PKI. Unter der Annahme, dass auf verschiedenen mathematischen Basisproblemen basierende Kryptoverfahren und verschiedene Sicherheitsanker nicht gleichzeitig kompromittiert werden, stehen durch Einsatz multipler Kryptographie im Schadensfall weiterhin Elemente der Public-Key-Infrastruktur bereit. Dies sichert die Verfügbarkeit der PKI-Anwendung im Schadensfall. Darüber hinaus kann multiple Kryptographie dazu genutzt werden, die Beweisbarkeit digitaler Signaturen und die Vertraulichkeit verschlüsselter Nachrichten trotz kompromittierter Schlüssel zu garantieren. Diese Idee, multiple Kryptographie in der PKI bereitzustellen und somit die offenen Probleme zu lösen, soll nachhaltig – also auf längere Zeit – wirken, auch wenn Verfahren und Schlüssel kompromittiert werden. Aus diesem Grund und auch, um dynamisch auf neue Sicherheitskriterien reagieren zu können, sieht das Fail-Safe-Konzept das Hinzufügen und Löschen von kryptographischen und einsatzspezifischen Komponenten im laufenden Betrieb der

PKI-Anwendung vor. Eine besondere Herausforderung ist die Absicherung dieser Aktualisierung – besonders wenn im Schadensfall Verfahren und Schlüssel kompromittiert sind.

In diesem Kapitel wird die Idee des Fail-Safe-Konzepts für Public-Key-Infrastrukturen anhand einer besonderen PKI dargelegt – der „Fail-Safe-PKI“. Zunächst wird der Begriff des Fail-Safe-Konzepts allgemein definiert und anschließend das daraus abgeleitete, aus vier Bausteinen bestehende Konzept für PKIs beschrieben. Jeder Baustein löst eines der angesprochenen Probleme:

1. Nachhaltige Existenz multipler Kryptographie zur Sicherung der Verfügbarkeit
2. Multiple digitale Signaturen zur Bewahrung der Beweiskraft digitaler Signaturen
3. Erweiterte Revokationsmechanismen zum Erhalt der Funktionsfähigkeit der Revokationsmechanismen
4. Iterative Verschlüsselungen zur Gewährleistung der Vertraulichkeit von Verschlüsselungen

Die sinnvolle Nutzung dieser vier Elemente ist von der konkreten Anwendung abhängig, weshalb in diesem Kapitel auch individuelle Konfigurationsmöglichkeiten des Fail-Safe-Konzepts diskutiert werden. Überlegungen zu Interoperabilität und Performance untermauern die Praktikabilität dieses Konzepts. Eine Analyse verwandter Lösungsideen rundet dieses Kapitel ab.

3.1 Definition eines Fail-Safe-Konzepts

Der Begriff “fail-safe“ geht auf die 1975 in dem Artikel “The Protection of Information in Computer Systems“ von Jerome Saltzer und Michael Schroeder [SaSc75] verfassten Prinzipien eines Sicherheitssystems zurück. Ein sicherheitskritisches System ist “fail-safe“, wenn es beim Auftreten bestimmter Schäden („fail“) in einem sicheren Zustand bleibt oder in einen sicheren Zustand übergeht („safe“). Ein Beispiel aus [SaSc75]: “Fail-safe defaults: Base access decisions on permission rather than exclusion“, d. h. Entscheidungen zum Zugriff sollen eher durch eine Erlaubnis als über einen Ausschluss geregelt werden, weil im Fall, dass die Kontrolle der Erlaubnis ausfällt, die Sicherheit dadurch gewährleistet wird, dass überhaupt kein Zugriff möglich ist. Der Begriff “fail-safe“ ist auch in anderen Bereichen üblich. Ein nicht-informationstechnisches Beispiel sind die mechanischen Signalanlagen bei der Eisenbahn: Bricht der Draht eines Signals, so springt das Signal auf „Zughalt“.

Übersetzt in die Welt der Public-Key-Infrastrukturen würde dies bedeuten, dass z. B. bei Ausfall eines kryptographischen Verfahrens durch die Revokation der betroffenen Zertifikate ein sicherer Zustand hergestellt wird, weil fortan keine digitalen Signaturen mehr akzeptiert oder asymmetrische Verschlüsselungen durchgeführt werden. Eine PKI, die sich ausschließlich auf Revokation verlässt, ist noch nicht fail-safe, weil der Revokationsmechanismus durch den Schaden selbst ausfällt und weil digitale Signaturen und Verschlüsselungen auch im Schadensfall ihre Beweiskraft bzw. Vertraulichkeit behalten sollen. Darüber hinaus unterscheiden sich die Folgen einer eingeschränkten Verfügbarkeit deutlich: Beim Beispiel der Eisenbahn ist es u. U. akzeptabel, dass, wenn ein Signal ausfällt und ein Zug deshalb wartet, der Zug auf einen Wartungsdienst wartet oder ein rotes Signal nach Rücksprache mit der Leitzentrale missachtet, während das Warten, bis eine Public-Key-Infrastruktur wieder voll funktionstüchtig ist, zu gravierenden Einschränkungen in der Informationsgesellschaft führen kann. Im einen Fall sind viele Personen für ein paar Stunden betroffen – im anderen Fall sehr viele Personen weltweit für mehr als ein paar Stunden.

Das Fail-Safe-Konzept für Public-Key-Infrastrukturen bedeutet deshalb mehr:

1. Verfügbarkeit der Public-Key-Infrastruktur im Schadensfall
2. Beweiskräftige digitaler Signaturen im Schadensfall
3. Sichere Revokationsmechanismen im Schadensfall
4. Vertrauliche Verschlüsselungen im Schadensfall

Aus den Anforderungen, dass ein System im Schadensfall weiterhin seine Sicherheit und Verfügbarkeit gewährleistet, wird das Fail-Safe-Konzept wie folgt definiert:

Definition 17
Ein **Fail-Safe-Konzept** für ein technisches System stellt die Verfügbarkeit des Systems bei Auftreten bestimmter Schadensfälle sicher und ist durch drei Eigenschaften charakterisiert:

1. Redundanz der technischen Mittel (nach [Beck99]):
   Für die Erfüllung der geforderten Funktionen stehen mehr funktionsfähige Mittel zur Verfügung als nötig, so dass im Schadensfall die Funktion weiterhin erbracht wird. Je nach Anwendung wird die Redundanz als heiß, warm oder kalt bezeichnet:
   - Heiße Redundanz (aktive oder parallele Redundanz),
     bei der das zusätzliche technische Mittel ständig in Betrieb ist und der gleichen Beanspruchung wie die Primäreinheit unterliegt.
   - Warme Redundanz (leicht belastete Redundanz),
     bei der das Redundanzmittel bis zum Ausfall der arbeitenden Einheit oder bis zu seinem eigenen Ausfall einer kleineren Belastung ausgesetzt ist.
   - Kalte Redundanz (Standby-, unbelastete Redundanz),
     bei der das zusätzliche technische Mittel bis zum Ausfall der arbeitenden Einheit keiner Belastung ausgesetzt ist.
2. Unabhängigkeit der redundanten technischen Mittel:
   Die redundanten technischen Mittel sind in Hinblick auf die möglichen Schadensfälle in dem Sinn voneinander unabhängig, dass ein technisches Mittel vom Ausfall eines dazu unabhängigen Mittels nicht betroffen ist.
3. Nachhaltigkeit der Redundanz und der Unabhängigkeit:
   Redundante und unabhängige Mittel können im laufenden Betrieb aktualisiert werden – ohne das System zu unterbrechen. Dadurch kann das Fail-Safe-Konzept rekursiv erneuert werden, wenn redundante Mittel durch einen Schaden verbraucht sind. Diese Eigenschaft impliziert eine flexible Integration der technischen Mittel.

Bevor das Fail-Safe-Konzept auf Public-Key-Infrastrukturen angewendet wird, erläutern drei Beispiele das Konzept.

Beispiele (Fail-Safe-Konzept)

1. Speichermedien:
   Angenommen, Daten vom Laptop werden zur Sicherheit auf Disketten abgespeichert. Die Datensicherung erfolgt also durch die redundanten technischen Mittel „Festplatte“ und

„Diskette“. Die Art der Redundanz ist warm bis heiß, weil üblicherweise auf der Festplatte gearbeitet wird und nur die fertigen Daten anschließend auf Disketten abgelegt werden. Die Mittel sind bzgl. technischer Defekte voneinander unabhängig: Tritt ein Defekt auf einem Speichermedium auf, der die Daten vernichtet, so bleiben die Daten auf dem anderen Medium erhalten. Eine Nachhaltigkeit der Daten wird dadurch erreicht, dass – falls Daten eines Speichermediums verloren gehen – Daten vom redundanten Medium auf ein weiteres Medium kopiert werden können, so dass anschließend wieder Redundanz vorliegt.

2. Schlüssel:
   Angenommen, zwei Personen haben je einen Schlüssel zu einem Schloss. Die Schlüssel sind also redundant ausgelegt. Wenn die eine Person nur dann ihren Schlüssel nutzt, falls die andere Person ihren Schlüssel verloren oder vergessen hat, liegt kalte bis warme Redundanz vor. Wenn es sich bei dem Schloss um das Schloss zum Tresor einer Bank handelt, beide Schlüssel verschieden sind und nur beide Personen gleichzeitig nach dem 4-Augen-Prinzip den Tresor betreten dürfen, dann liegt heiße Redundanz vor. Die Nachhaltigkeit bedeutet hier, dass Schlüssel nachgekauft werden können, ohne das Schloss auszutauschen.

3. Autorisierung:
   Angenommen, die Autorisierung zur Erzeugung einer digitalen Signatur erfolgt über PIN oder ein biometrisches Verfahren, z. B. den Fingerprint. PIN und Fingerprint sind redundant und nebeneinander zu benutzen, also besteht eine warme bis heiße Redundanz. Die Verfahren sind unabhängig hinsichtlich des Sicherheitsniveaus: Während die Sicherheit des PIN-Verfahrens von der Anzahl der möglichen Kombinationen abhängt, wird die Sicherheit eines Fingerprints durch Parameter und die daraus resultierende Fehlerquote (False Acceptance Rate / False Rejection Rate) festgelegt. Sollte sich herausstellen, dass diese Parameter für die Sicherheit ungenügend sind und deshalb kein Fingerprint mehr zu benutzen ist, bleibt das PIN-Verfahren davon unberührt. Eine Nachhaltigkeit der redundanten und unabhängigen Autorisierungsmittel bedeutet, dass in das System ein neue Form der Autorisierung eingespielt werden kann, ohne das System neu aufzusetzen.

Das Fail-Safe-Konzept für Public-Key-Infrastrukturen ist das Folgende:

Die technischen Mittel in Public-Key-Infrastrukturen sind die kryptographischen Verfahren sowie Schlüssel und Zertifikate. Im Sinne einer Redundanz werden sie mehrfach vorhanden sein und eine Unabhängigkeit zueinander aufweisen, so dass davon auszugehen ist, dass Verfahren oder Schlüssel vom Kompromittieren des Redundanzsystems unberührt bleiben. Die Art der Redundanz (kalt, warm oder heiß) hängt von der jeweiligen Anwendung ab. Durch das Hinzufügen und Löschen von kryptographischen und einsatzspezifischen Komponenten im laufenden Betrieb der PKI ist die Nachhaltigkeit gegeben. Dieses Fail-Safe-Konzept für Public-Key-Infrastrukturen lässt sich im Hinblick auf die zu lösenden Probleme auf die vier Baustein „Nachhaltige Existenz multipler Kryptographie zur Sicherung der Verfügbarkeit“, „Multiple digitale Signaturen zur Bewahrung der Beweiskraft digitaler Signaturen“, „Erweiterte Revokationsmechanismen zum Erhalt der Funktionsfähigkeit der Revokationsmechanismen“ und „Iterative Verschlüsselungen zur Gewährleistung der Vertraulichkeit von Verschlüsselungen“ aufteilen, die in den nächsten Abschnitten diskutiert werden.

3.2 Nachhaltige Existenz multipler Kryptographie

Um die Verfügbarkeit einer Public-Key-Infrastruktur im Schadensfall gewährleisten zu können, stehen in einer PKI mehrere verschiedene kryptographische und einsatzspezifische Komponenten zur Verfügung, was mit „multipler Kryptographie“ bezeichnet wird. Um die Verfügbarkeit nachhaltig – d. h. auf längere Zeit – garantieren zu können, ist ein Aktualisieren von Komponenten im laufenden Betrieb der PKI vorgesehen. Dieser Abschnitt beschreibt zunächst die grundlegenden Annahmen zur multiplen Kryptographie und begründet, weshalb diese Annahmen sinnvoll sind. Daraufhin wird die durch die Integration von multipler Kryptographie veränderte Public-Key-Infrastruktur dargestellt und die Maßnahmen diskutiert, um Komponenten im laufenden Betrieb der PKI zu aktualisieren.

3.2.1 Multiple Kryptographie

Multiple Kryptographie bezeichnet das Vorhandensein mehrerer verschiedener kryptographischer und einsatzspezifischer Komponenten in einer PKI. An die Verschiedenartigkeit dieser Komponenten werden gewisse Anforderungen gestellt, die in diesem Abschnitt vorgestellt werden.

Die Public-Key-Infrastruktur, die dieses Fail-Safe-Konzept nutzen wird, trägt den Namen „Fail-Safe-PKI“. In einer Fail-Safe-PKI sind mehrere, voneinander unabhängige kryptographische und einsatzspezifische Komponenten integriert, d. h. mehrere Kryptoverfahren samt Schlüsseln und Zertifikaten. Mit anderen Worten: Die Fail-Safe-PKI enthält praktisch mehrere parallel zu nutzende „Teil-PKIs“ und damit multiple Kryptographie.

Dieser Idee liegt folgende Annahme zugrunde: Es wird angenommen, dass die Wahrscheinlichkeit sehr gering ist, die auf verschiedenen mathematischen Problemen beruhenden asymmetrischen Algorithmen würden durch einen Schaden gleichzeitig – also simultan – kompromittiert werden.

Diese Annahme wird begründet, wobei zu berücksichtigen ist, dass ein Schaden entweder dadurch auftritt, dass ein effizienter Lösungsalgorithmus entdeckt wird, der beispielweise in akzeptabler Zeit eine große Zahl faktorisiert oder einen diskreten Logarithmus berechnet, oder dass die Rechenleistung plötzlich deutlich verbessert wird:

- Algorithmische Fortschritte:
  Zwei asymmetrische Algorithmen, die auf verschiedenen mathematischen Basisproblemen beruhen, sind nicht zwingend von demselben Lösungsalgorithmus bzw. dessen Idee angreifbar, weil diese Lösungsidee speziell an ein jeweiliges Basisproblem angepasst ist. Beispielsweise konnten bislang die Methoden und Ideen zum Berechnen diskreter Logarithmen in endlichen Körpern nicht zur Berechnung diskreter Logarithmen auf elliptischen Kurven genutzt werden. Als Grund gibt Andrew Odlyzko die fehlende natürliche Beziehung zwischen Punkten einer elliptischen Kurve zu denen anderer Gruppen an [Odly99].

- Fortschritte in der Rechenleistung:
  Nach heutigem Kenntnisstand ist eine plötzliche Verbesserungen in der Computer-Leistung, wodurch zwei auf verschiedenen mathematischen Basisproblemen beruhenden Algorithmen simultan kompromittiert werden, nur mit einen Quantencomputer möglich. Es ist aber fraglich, ob und – wenn ja – wann es gelingen wird, einen Quantencomputer für derartige Berechnungen zu bauen und wie die Kryptographie im Zeitalter von Quantencomputern aussehen wird, weil dann u. U. Quantenkryptographie schon praktisch genutzt

wird. Es besteht also ein gewisses Restrisiko, aber solange auszuschließen ist, dass Quantencomputer in absehbarer Zeit möglich werden, ist anzunehmen, dass zwei Verfahren derzeit nicht simultan kompromittiert werden.

Aus diesen Gründen wird angenommen, dass es asymmetrische Algorithmen gibt, die keine Korrelation zueinander aufweisen. „Keine Korrelation“ zwischen zwei mathematischen Basisproblemen α^1 und α^2 bzw. Algorithmen bedeutet: Wenn ein Basisproblem (o. B. d. A. sei dies α^1) gelöst und dadurch ein darauf basierender Algorithmus gebrochen wird, bleibt α^2 davon unberührt, so dass ein auf α^2 basierender Algorithmus weiterhin als sicher angesehen werden kann. Wie in Abschnitt 2.1.1 auf Seite 43 erwähnt, ist z. B. keine Korrelation zwischen Faktorisierungsproblem und DL-Problem auf elliptischen Kurven oder zwischen DL-Problem in endlichen Körpern zu dem auf elliptischen Kurven bekannt. Deshalb sind nach derzeitigem Wissensstand die Signaturalgorithmen RSA und DSA jeweils zu ECDSA unabhängig sowie die Verschlüsselungsalgorithmen RSA und ElGamal zu ECIES. Solche Paare asymmetrischer Algorithmen sind deshalb als für das Fail-Safe-Konzept geeignet anzusehen.

Da asymmetrische Signatur- und Verschlüsselungsalgorithmen meist nicht direkt, sondern in Kombination mit Hashfunktion und Formatierung in Signatur- und Verschlüsselungsverfahren eingesetzt werden, muss es zusätzlich Hashfunktionen und Formatierungen geben, die jeweils voneinander unabhängig sind. Die Hashfunktionen SHA-1, RIPEMD-160 und MD5 nutzen unterschiedliche Initial-Werte, Shift-Operationen, Funktionen und Runden und sind daher als voneinander unabhängig zu betrachten. Ebenso sind die Formatierungen PKCS#1 und ISO 9796-2 aufgrund ihrer Konstruktion als voneinander unabhängig anzusehen. Bzgl. der Formatierung ist folgendes zu beachten: id steht für die identische Abbildung und wird für einen einheitlichen Aufbau von Signatur- und Verschlüsselungsverfahren bei den Verfahren pro forma eingesetzt, die gar keine Formatierung benötigen, wie z. B. DSA oder ECDSA. Deshalb sind zwei Verfahren, die voneinander unabhängige Signatur- oder Verschlüsselungsalgorithmen nutzen, auch dann als voneinander unabhängig anzusehen, falls beide id nutzen oder sie sich bei der Formatierung in der Weise unterscheiden, dass das eine Verfahren eine Formatierung nutzt und das andere nicht. Denn es besteht keine Korrelation: Wird die Formatierung gebrochen, ist nur das eine Verfahren davon betroffen, welches die Formatierung benötigt.

Zusammenfassend können beispielsweise die kryptographischen Primitiven

- RSA und ECDSA,
- DSA und ECDSA,
- SHA-1 und RIPEMD-160,
- SHA-1 und MD5,
- RSA und ECIES sowie
- ElGamal und ECIES

als voneinander unabhängig angesehen werden. Für die Unabhängigkeit von daraus zusammengesetzten Signatur- und Verschlüsselungsverfahren ist es wichtig, dass sie keine gemeinsame Komponente haben. Einige voneinander unabhängige Verfahren sind die folgenden:

- rsasignatureWithripemd160 (RSA mit RIPEMD-160 und PKCS#1-Formatierung) und ecdsa-with-SHA1 (ECDSA mit SHA-1)

- sigS_ISO9796-2rndWithsha1 (RSA mit SHA-1 und ISO 9796-2-Formatierung mit Zufallszahl) und ecSignWithripemd160 (ECDSA mit RIPEMD-160)
- RSA encryption (RSA mit PKCS#1-Randomisierung) und ECIES[1]

Für symmetrische Verschlüsselungsalgorithmen wird ähnlich argumentiert: Die symmetrischen Algorithmen Triple-DES, AES und IDEA beispielweise basieren auf völlig verschiedenen Ideen. Stellt sich heraus, dass ein Algorithmus kryptographisch ungeeignet ist, kann deshalb davon ausgegangen werden, dass die anderen Algorithmen weiterhin sicher sind. Dieser Sachverhalt überträgt sich auf die darauf aufbauenden Verschlüsselungsverfahren.

An die Sicherheitsanker – also die bei den Zertifikatsinhabern authentisch eingebrachten öffentlichen Schlüssel der Zertifizierungsinstanz – werden ebenfalls Annahmen gemacht: Es wird angenommen, dass es aufgrund von Vorkehrungen in der CA nicht möglich ist, zwei Schlüssel der Zertifizierungsinstanz simultan zu kompromittieren oder einen CA-Schlüssel zu einem Verfahren in dem Moment zu kompromittieren, in dem ein dazu unabhängiger Algorithmus ebenfalls kompromittiert wird.

Zusammenfassend liegt dem Fail-Safe-Konzept also die Annahme zugrunde, es gäbe mindestens zwei voneinander unabhängige Signatur- und Verschlüsselungsalgorithmen, Hashfunktionen und Formatierungen sowie zwei – untereinander und zum Kompromittieren eines Algorithmus – unabhängige Sicherheitsanker.

3.2.2 Einsatz multipler Kryptographie in einer PKI

Durch Einsatz der zuvor beschriebenen multiplen kryptographischen Komponenten spaltet sich eine Public-Key-Infrastruktur in zwei Teil-PKIs auf. In diesem Abschnitt wird die Konstruktion der Teil-PKIs beschrieben und erläutert, wie sie für die Verfügbarkeit der PKI im Schadensfall sorgen.

Die Teil-PKIs werden mit PKI^A und PKI^B bezeichnet und sind wie folgt konstruiert.

PKI^A

PKI^A wird durch seinen Signaturalgorithmus σ^A charakterisiert, der auf dem mathematischen Basisproblem α^A beruht. Zu σ^A gehören Schlüsselpaare ($\text{prK}^A_{\text{Instanz}}$, $\text{puK}^A_{\text{Instanz}}$) und Zertifikate $\text{cert}^A_{\text{Instanz}}$, wobei „Instanz“ für alle Beteiligten in der PKI steht – also Zertifizierungsinstanz, Zertifikatsinhaber usw. Die Zertifikate cert^A sind mit einem aus σ^A bestehenden Signaturverfahren sign^A von der Zertifizierungsinstanz signiert worden.

In der PKI^A können neben sign^A weitere Signaturverfahren $\text{sign}^{A'}$, $\text{sign}^{A''}$ usw. existieren, die stets σ^A nutzen und sich ausschließlich in Hashfunktion und Formatierung unterscheiden:
$\text{sign}^{A'} = \sigma^A \circ \phi^1 \circ \kappa^1$, $\text{sign}^{A''} = \sigma^A \circ \phi^2 \circ \kappa^2$, ..., für nicht zwingend verschiedene Hashfunktionen κ^1, κ^2 und Formatierungen ϕ^1, ϕ^2.

Darüber hinaus kann in PKI^A ein auf α^A basierender asymmetrischer Verschlüsselungsalgorithmus η_asym^A samt verschiedenen -verfahren enc_asym^A, $\text{enc\_asym}^{A'}$ usw. mit zugehörigen Schlüsselpaaren und Zertifikaten sowie ein symmetrischer Verschlüsselungsalgorithmus η_sym^A mit -verfahren enc_sym^A, $\text{enc\_sym}^{A'}$ usw. vorhanden sein. Alle Zertifikate in PKI^A sind mit sign^A signiert.

PKI^B

PKI^B wird durch seinen Signaturalgorithmus σ^B charakterisiert, der auf dem zu α^A unabhängigen mathematischen Basisproblem α^B beruht. Wie in PKI^A gehören zu σ^B Schlüsselpaare

[1] Eine Randomisierung kann im Rahmen der in ECIES enthaltenen symmetrischen Verschlüsselung erfolgen.

($\mathrm{prK}^B_{\mathrm{Instanz}}$, $\mathrm{puK}^B_{\mathrm{Instanz}}$) und Zertifikate $\mathrm{cert}^B_{\mathrm{Instanz}}$. Zertifikate cert^B sind mit dem Signaturverfahren sign^B von der Zertifizierungsinstanz signiert worden, wobei sign^B aus σ^B besteht und sign^B zu sign^A unabhängig ist. Insbesondere teilen die beiden Signaturverfahren weder Hashfunktion noch Formatierung.

Wie in PKI^A können in PKI^B neben sign^B weitere Signaturverfahren $\mathrm{sign}^{B'}$, $\mathrm{sign}^{B''}$ usw. existieren, die stets σ^B nutzen und sich ausschließlich in Hashfunktion und Formatierung unterscheiden: $\mathrm{sign}^{B'} = \sigma^B \circ \phi^1 \circ \kappa^1$, $\mathrm{sign}^{B''} = \sigma^B \circ \phi^2 \circ \kappa^2$, ..., für nicht zwingend verschiedene Hashfunktionen κ^1, κ^2 und Formatierungen ϕ^1, ϕ^2.

Darüber hinaus kann in PKI^B ein auf α^B basierender asymmetrischer Verschlüsselungsalgorithmus η_asym^B samt verschiedenen -verfahren $\mathrm{enc\_asym}^B$, $\mathrm{enc\_asym}^{B'}$ usw. mit zugehörigen Schlüsselpaaren und Zertifikaten sowie ein symmetrischer Verschlüsselungsalgorithmus η_sym^B mit -verfahren $\mathrm{enc\_sym}^B$, $\mathrm{enc\_sym}^{B'}$ usw. vorhanden sein. Zwischen η_sym^A und η_sym^B ist keine Korrelation bekannt. Alle Zertifikate in PKI^A sind mit sign^A signiert.

Tabelle 3.1 zeigt die wesentlichen Elemente von PKI^A und PKI^B im Überblick.

Tabelle 3.1: Komponenten von PKI^A und PKI^B

	PKI^A	PKI^B
Signaturalgorithmus, basiert auf math. Basisproblem	σ^A α^A	σ^B α^B
Schlüsselpaare	(prK^A, puK^A)	(prK^B, puK^B)
Zertifikate, signiert mit Signaturverfahren	cert^A sign^A	cert^B sign^B
Weitere Signaturverfahren	$\mathrm{sign}^{A'}$, $\mathrm{sign}^{A''}$, ...	$\mathrm{sign}^{B'}$, $\mathrm{sign}^{B''}$, ...
Asymm. Verschl'algorithmus, basiert auf math. Basisproblem	η_asym^A α^A	η_asym^B α^B
Schlüsselpaare	$(\mathrm{prK}'^A, \mathrm{puK}'^A)$	$(\mathrm{prK}'^B, \mathrm{puK}'^B)$
Zertifikate, signiert mit Signaturverfahren	cert'^A sign^A	cert'^B sign^B
Asymm. Verschlüsselungsverfahren	$\mathrm{enc\_asym}^A$, $\mathrm{enc\_asym}^{A'}$, ..	$\mathrm{enc\_asym}^B$, $\mathrm{enc\_asym}^{B'}$, ..
Symm. Verschl'algorithmus	η_sym^A	η_sym^B
Symm. Verschlüsselungsverfahren	$\mathrm{enc\_sym}^A$, $\mathrm{enc\_sym}^{A'}$, ...	$\mathrm{enc\_sym}^B$, $\mathrm{enc\_sym}^{B'}$, ...

Beispiel
PKI^A enthält RSA als Signaturalgorithmus. Damit ist das Faktorisierungsproblem das zugrunde liegende mathematische Basisproblem in PKI^A. Zertifikate werden mit dem Signaturverfahren rsasignatureWithripemd160 (RSA mit RIPEMD-160 und PKCS#1-Formatierung) signiert und darüber hinaus steht RSAwithMD5 und sigS_ISO9796-2Withsha1 (RSA mit SHA-1 und ISO 9796-2-Formatierung) für digitale Signaturen zur Verfügung.

In PKI^B wird das DL-Problem auf elliptischen Kurven zur Kryptographie angewendet. ecdsa-with-SHA1 (ECDSA mit SHA-1) ist das Signaturverfahren, das die Certification Authority zur Zertifizierung nutzt. Es ist zum in PKI^A für Zertifizierungen genutzten rsasignatureWithripemd160 unabhängig. Daneben stehen Nutzern von PKI^B die Verfahren ecSignWithripemd160 (ECDSA mit RIPEMD-160) und ECDSAwithSHA1 zur Verfügung.

In beiden Teil-PKIs kommen identische Hashfunktionen vor, aber die Zertifikate sind von voneinander unabhängigen Signaturverfahren signiert worden.

So konstruierte Teil-PKIs gewährleisten Verfügbarkeit im Schadensfall: Tritt ein Schaden ein, der – ohne Beschränkung der Allgemeinheit – PKI^A berührt, dann wird dieser Schaden von den zuständigen Stellen – etwa dem BSI – geprüft und bei praktischer Relevanz werden die betroffenen Zertifikate aus PKI^A von den Trust Centern revoziert. Da diese Zertifikate nicht mehr gültig sind, sollten involvierte digitale Signaturen abgelehnt und Verschlüsselungen mit involvierten Schlüsseln vermieden werden. Dies führt dazu, dass ein Teil von PKI^A nicht mehr funktioniert, wobei die Größe des nicht mehr funktionierenden Anteils von PKI^A schadensabhängig ist. Die Verfügbarkeit der Gesamt-PKI bleibt dennoch durch PKI^B gewahrt.

3.2.3 Nachhaltigkeit

Der Wechsel von einer Teil-PKI zu einer anderen soll nicht endlich oft möglich, sondern prinzipiell beliebig häufig durchführbar sein, weil die Zukunft – wie beschrieben – nicht vorhersehbar ist. Es macht deshalb wenig Sinn, bei der Personalisierung viele Teil-PKIs zu etablieren und bei Auftreten eines Schadens eine Teil-PKI nach der anderen zu nutzen. Stattdessen wird es in der Fail-Safe-PKI möglich sein, kryptographische und einsatzspezifische Komponenten auf sichere Weise im laufenden Betrieb zu aktualisieren. Die Sicherheit dieses Updates basiert auf der Existenz einer sicheren Teil-PKI, von der nach Voraussetzung zum Zeitpunkt der Aktualisierung mindestens eine vorhanden ist.

Das automatisierte Löschen einer Hashfunktion im System der Zertifikatsinhaber ist eine Lösung, um zu verhindern, dass CHs mit dieser kompromittierten Hashfunktion erzeugte Signaturen akzeptieren, weil sich Hashfunktionen mit den etablierten Methoden nicht revozieren lassen.

In dieser Arbeit liegt das Haupaugenmerk auf der Problematik, im Schadensfall kryptographische und einsatzspezifische Komponenten zu aktualisieren. Das Fail-Safe-Konzept ist allerdings so flexibel ausgelegt, dass auch das reine Hinzufügen oder Löschen von Verfahren, Schlüsseln und Sicherheitsankern – ohne dass ein Schaden vorliegt – durchgeführt werden kann.

In diesem Abschnitt wird diskutiert, wer die Aktualisierung initiiert, welches Protokoll zum Update benutzt wird, welche schadensabhängigen Maßnahmen ergriffen werden und wie der Ablauf zur Aktualisierung aussieht.

Der Update Service

Die Idee für den Update ist, dass für die Aktualisierung vorgesehene Zertifikatsinhaber eine vom Trust Center autorisierte Information zum Update von Komponenten erhalten. Dazu wird eine neue organisatorische Einheit namens **Update Service** gegründet, welcher entweder inner- oder außerhalb des Trust Centers angesiedelt ist:

- Update Service innerhalb des Trust Centers:
  Der Update Service ist innerhalb des Trust Centers neben RA, CA und DIR angesiedelt. Dieses Modell orientiert sich am Business-Modell von „Signtrust“ der Deutschen Post. Dort sind die Benutzer nicht Besitzer der Signatur-Chipkarten, sondern sie zahlen nur für die Nutzung der Dienstleistung. Signtrust wäre wahrscheinlich daran interessiert, seinen Kunden wieder eine funktionierende PKI zur Verfügung zu stellen. Auch in Unternehmens- oder Behörden-PKIs treten die Unternehmen bzw. Behörden selbst als Trust Center auf und haben ein ureigenes Interesse daran, dass die PKI funktioniert.

- Update Service außerhalb des Trust Centers:
  Falls die Dienstleistung des Update Services nicht direkt vom Trust Center übernommen wird, kann eine externe Firma die Verteilung übernehmen. In diesem Fall stellt die CA dieser Firma ein Attributzertifikat aus, welches besagt, dass diese Firma als Update Service die Update-Informationen autorisieren darf. Damit können die Zertifikatsinhaber die Aktualisierungs-Maßnahme über das Attributzertifikat und den Sicherheitsanker der CA verifizieren.

Dem Fail-Safe-Konzept liegt die Annahme zugrunde, dass zunächst eine vertrauenswürdige Instanz, wie etwa das BSI, neue Empfehlungen zu geeigneten Kryptoalgorithmen herausgibt, woraufhin die Trust Center Zertifikate revozieren, die vom Schaden tangiert werden. Weiter wird angenommen, dass der für die Aktualisierung zuständige Update Service vertrauenswürdig ist und nur einen sinnvollen Update durchführt.

Das Update Management Protocol (UMP)

Wie sieht das Protokoll zum Update von Komponenten aus? Dazu wird ein neues Protokoll definiert – das **Update Management Protocol (UMP)**.

Das Update Management Protocol besteht aus zwei Teilen: Der Update Service schickt ein **UpdateComponent** an alle von der Aktualisierungsmaßnahme involvierten Certificate Holder. Das UpdateComponent initiiert das Update Management Protocol. Im UpdateComponent ist im Wesentlichen codiert, welche kryptographischen und einsatzspezifischen Komponenten beim Zertifikatsinhaber deinstalliert und/oder installiert werden sollen. Wie in Abschnitt 1.3.2 auf Seite 23 diskutiert, kann das Equipment des CHs Clients, Software-PSEs und Chipkarten umfassen, so dass bei allen Komponenten ein notwendiger Update ausgeführt wird. Mit einem **UpdateComponentResponse** – dem zweite Teil des Update Management Protocols – bestätigt die PSE des CHs dem Update Service den erfolgreichen Update. In das UpdateComponentResponse werden neu erzeugte Schlüssel integriert, so dass der Update Service diese Schlüssel zur Zertifizierung an die Certification Authority weiterleitet (Abb. 3.1).

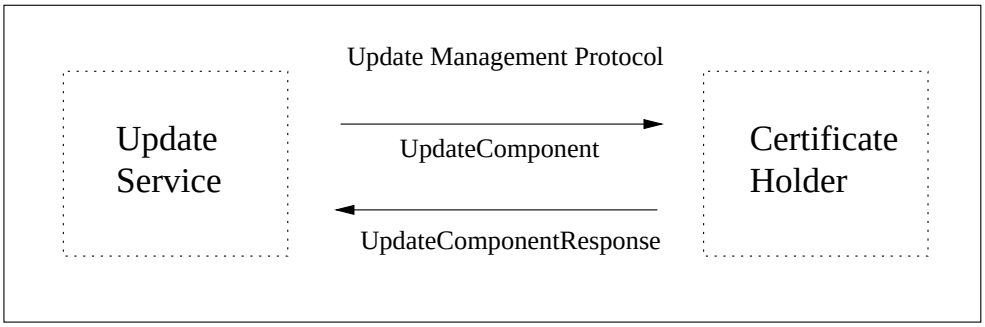

Abbildung 3.1: Aktualisieren von Komponenten (grob)

Zur Absicherung des UpdateComponents: Für das Deaktivieren einer Komponente, die in ein Signaturverfahren integriert ist, ist das UpdateComponent mit diesem Signaturverfahren digital zu signieren. Intention ist, dass dadurch ein Angreifer, der ein Verfahren „geknackt“ hat, schlimmstenfalls genau dieses Verfahren deaktivieren kann. Und für das Installieren einer neuen Komponente ist das UpdateComponent mit einem noch sicheren Verfahren zu signieren.

Nach Voraussetzung gibt es mindestens ein weiteres Signaturverfahren, welches vom Schaden nicht tangiert ist. Dieses Signaturverfahren ist nach Konstruktion der Fail-Safe-PKI in einer Teil-PKI integriert, die vom Schaden nicht berührt und funktionsfähig ist. Damit wird also das UpdateComponent mit zwei digitalen Signaturen abgesichert, deren Signaturverfahren voneinander unabhängig sind und keine gemeinsame kryptographische Komponente haben, und eine der Signaturen ist vom Schaden betroffen. Falls kein Signaturverfahren kompromittiert ist – z. B. bei ausschließlichem Hinzufügen von Komponenten oder dem Austausch eines symmetrischen Verfahrens –, wird das UpdateComponent dennoch aus Konsistenzgründen mit zwei voneinander unabhängigen Signaturverfahren digital signiert. Die Forderung, dass eine der Signaturen vom Schaden berührt ist, entfällt dann.

Zur Absicherung des UpdateComponentResponses genügt eine einfache, nicht revozierte digitale Signatur.

Entscheidend für die Sicherheit ist der Abschluss einer Updatemaßnahme. Ein Angriff verdeutlicht dies: Die Grundannahme für das Fail-Safe-Konzept ist, dass nur ein(!) Schaden zur Zeit auftritt; es also nicht möglich ist, dass etwa zwei als voneinander unabhängig angesehene Signaturalgorithmen simultan kompromittiert werden. Angenommen, zwei Schäden treten nacheinander ein: Zuerst wird sign^A und dann sign^B kompromittiert. In einer ersten Maßnahme wird PKI^A durch PKI^C ersetzt und in einer zweiten PKI^B durch PKI^D. Zum Zeitpunkt des ersten und des zweiten Schadens ist die Voraussetzung – es existiert jeweils nur ein(!) Schadensfall – erfüllt, so dass die Sicherheit des Updates gewährleistet ist. Der Angriff ist nun Folgender: Ein Angreifer verhindert das Ausführen eines Updates bei einem Zertifikatsinhaber, indem er das UpdateComponent abfängt. Durch die verfügbare PKI^B merkt dieser CH nicht, dass PKI^A nicht mehr funktionsfähig ist. Nachdem auch sign^B kompromittiert ist, kann der Angreifer diesem CH ein gefälschtes UpdateComponent schicken, durch welches z. B. ein neuer Algorithmus mit einer Hintertür zum Ausspionieren des privaten Schlüssels in sein System integriert werden soll. Die Verifikation eines solchen UpdateComponents kann positiv ausgehen, weil es mit zwei voneinander unabhängige Signaturverfahren – sign^A und sign^B – signiert ist, wodurch tatsächlich gefälschter Code beim CH installiert werden kann. Dieser mögliche Angriff zeigt, dass Schadensfälle disjunkt bearbeitet werden müssen. Aus diesem Grund ist die Bestätigung des UpdateComponents in Form des UpdateComponentResponses notwendig, durch das der Update Service über den gegenwärtigen Status des Updates bei jedem Zertifikatsinhaber informiert wird. Denn obiger Angriff greift nicht mehr, wenn der Update Service wüsste, dass das erste UpdateComponent den CH nie erreichte, weil das Trust Center die Zertifizierung neuer Schlüssel unterbinden und Zertifikate revozieren würde.

Schadensabhängige Maßnahmen

Welche Maßnahmen bei einem Schaden zu ergreifen und im Update Management Protocol durchzuführen sind, hängt vom Schaden ab. Schäden können unerwartet oder erwartet auftreten, wenn z. B. neue Empfehlungen zu Kryptoalgorithmen schon länger bekannt sind. Im Folgenden werden die schadensabhängigen Maßnahmen diskutiert:

1. Schaden: Signaturalgorithmus kryptographisch ungeeignet
   Maßnahmen:

   (a) Kryptographische Komponenten betreffend:
   Der Signaturalgorithmus braucht nicht ausgetauscht zu werden, falls durch Anpassen der Schlüssellänge der Schaden behoben werden kann und die Implementierung eine

veränderte Schlüssellänge zulässt. Falls hingegen zwar durch Anpassen der Schlüssellängen der Schaden behoben werden kann, die Implementierung aber eine veränderte Schlüssellänge nicht zulässt, oder falls ein Anpassen der Schlüssellängen nicht ausreicht, muss der Signaturalgorithmus aktualisiert werden. In diesem Fall wird die aktuelle Implementierung gelöscht und eine neue Implementierung installiert. Wird der Algorithmus getauscht, muss auch das Signaturverfahren getauscht werden.

(b) Einsatzspezifische Komponenten betreffend:
Schlüssel und Sicherheitsanker müssen ausgetauscht werden. Dazu generiert die Zertifizierungsinstanz ein neues Schlüsselpaar für sich selbst, zertifiziert es und verteilt diesen Sicherheitsanker als Schlüssel oder Selbstzertifikat. Die Zertifikatsinhaber erzeugen lokal ein neues Schlüsselpaar und lassen es von der Zertifizierungsinstanz zertifizieren. Anders als in der Personalisierungsphase ist es nun nicht möglich, dass die CA die Schlüssel für die CHs generiert und etwa verschlüsselt überträgt. Der Grund ist, dass die Sicherheit eines in diesem Prozess neu installierten Signaturalgorithmus samt Schlüsseln dann von der Sicherheit des Verschlüsselungsverfahrens abhängt, deren Sicherheit ebenfalls nicht beweisbar ist und welches – auch später irgendwann einmal – kompromittiert werden könnte. Intention ist, dass die im laufenden Betrieb neu installierten Verfahren und Schlüssel genauso sicher sind, wie die in der Personalisierungsphase installierten.

Es gibt aber Fälle, in denen auf ein Key Recovery nicht verzichtet werden kann, beispielsweise in Unternehmen. Dort soll der Zugriff auf verschlüsselte Daten auch bei Verlust des privaten Schlüssels möglich sein, wenn etwa ein Mitarbeiter erkrankt, verstirbt oder seine Chipkarte verliert. Da auch hier ein Übertragen des privaten Schlüssels von der CA zum CH nicht möglich ist, muss in diesen Fällen der neue private Schlüssel bei CA und CH synchron erzeugt werden: Das Kommando, einen neuen Schlüssel zu generieren, bewirkt beim Equipment des Zertifikatsinhabers, dass mit einem Pseudozufallszahlengenerator aus gewissen Daten ein neuer Schlüssel „zufällig“ erzeugt wird. Durch die Synchronisierung muss die CA in der Lage sein, aus dem gleichen Pseudozufallszahlengenerator und denselben Daten denselben privaten Schlüssel zu erzeugen.

Neue Zertifikate und Policies lassen sich über Verzeichnisdienste, das Certificate Management Protocol (CMP) oder E-Mail verteilen und von den Zertifikatsinhabern mittels Sicherheitsankern prüfen.

2. Schaden: Hashfunktion kryptographisch ungeeignet
Maßnahmen:

(a) Kryptographische Komponenten betreffend:
Die Hashfunktion muss ausgetauscht werden. Ist die Hashfunktion in das Signaturverfahren integriert, muss dieses ebenfalls ausgetauscht werden.

(b) Einsatzspezifische Komponenten betreffend:
Schlüssel sind nicht kompromittiert worden und müssen nicht neu generiert werden. Aber Zertifikate müssen erneuert werden, wenn Zertifikate selbst vom Schaden betroffen sind, d. h. wenn Signaturen der Zertifikate selbst unter Nutzung dieser Hashfunktion erzeugt wurden. Der Austausch von Zertifikaten gestaltet sich problemlos, da alte Zertifikate revoziert und neue vom Verzeichnisdienst zur Verfügung gestellt oder über etablierte Mechanismen wie E-Mail oder das Certificate Management Protocol (CMP) verteilt werden und mittels Sicherheitsanker verifiziert werden können.

Ebenso verhält es sich mit geänderten Policies: Sie werden zur Verfügung gestellt und sind verifizierbar.

3. Schaden: Formatierung kryptographisch ungeeignet
   Maßnahmen:
   (a) Kryptographische Komponenten betreffend:
   Die Formatierung muss ausgetauscht werden. Ist die Formatierung in Signatur- bzw. Verschlüsselungsverfahren integriert, muss dieses ausgetauscht werden.
   (b) Einsatzspezifische Komponenten betreffend:
   Schlüssel sind nicht kompromittiert worden und müssen nicht neu generiert werden. Aber Zertifikate müssen erneuert werden, wenn sie mittels dieser Formatierung signiert wurden. Der Austausch von Zertifikaten gestaltet sich problemlos, da der Verzeichnisdienst revozierte Zertifikate aufführt und neue Zertifikate verteilt werden und sich diese mittels Sicherheitsanker verifizieren lassen. Ganz analog verhalten sich geänderte Policies: Sie werden zur Verfügung gestellt und sind verifizierbar.

4. Schaden: Verschlüsselungsalgorithmus kompromittiert
   Maßnahmen:
   (a) Kryptographische Komponenten betreffend:
   Analog zu 1a müssen Verschlüsselungsalgorithmus und -verfahren ausgetauscht werden.
   (b) Einsatzspezifische Komponenten betreffend:
   Analog zu 1b.

5. Schaden: Schlüssel der Wurzelzertifizierungsinstanz (RootCA-Key) kompromittiert
   Maßnahmen:
   Der Wurzelzertifizierungsinstanz-Schlüssel als Sicherheitsanker muss ausgetauscht werden. Die RootCA generiert ein neues Schlüsselpaar und zertifiziert den öffentlichen Schlüssel in einem Selbstzertifikat. Je nachdem, ob der Sicherheitsanker als Schlüssel oder Zertifikat realisiert ist, muss dieser Schlüssel resp. dieses Zertifikat bei den Zertifikatsinhabern ausgetauscht werden.

6. Schaden: Zertifizierungsinstanz-Schlüssel (CA-Key) kompromittiert
   Maßnahmen:
   Der Zertifizierungsinstanz-Schlüssel muss ausgetauscht werden. Die CA generiert ein neues Schlüsselpaar und lässt den öffentlichen Schlüssel von der RootCA zertifizieren. Die Benutzer der PKI können das neue CA-Zertifikat vom Verzeichnisdienst beziehen oder die CA verschickt es zusätzlich.

7. Schaden: Ein Teilnehmer-Schlüssel (CH-Key) kompromittiert
   Maßnahmen:
   Der Schlüssel des Zertifikatsinhabers muss ausgetauscht werden. Dazu löscht der Zertifikatsinhaber sein altes Schlüsselpaar, generiert ein neues und lässt den öffentlichen Schlüssel von der Zertifizierungsinstanz zertifizieren. Wie in 1b argumentiert, ist es nicht möglich, den Schlüssel von der CA generieren zu lassen und verschlüsselt zu übertragen. Stattdessen kann ein Key Recovery durch das synchrone Generieren eines neuen Schlüssels mit einem Pseudozufallszahlengenerator aus identischen Daten in CA und PSE des CHs gelingen. Neue Zertifikate können vom Verzeichnisdienst bezogen oder via E-Mail oder CMP zugeschickt werden.

8. Schaden: Schlüssel der Zeitstempelinstanz (TSA-Key) oder der Schlüssel der CA für Sperrauskünfte (CRL- oder OCSP-Key) kompromittiert
   Maßnahmen:
   Analog zu 7.

Ist ein Schaden auf physikalischer und nicht logischer Ebene entstanden, also beispielsweise ein neuer physikalischer Angriff auf Chipkarten, so kann kein Austausch von Software Angriffe verhindern. Sind organisatorische Mängel Ursache für einen Schaden – etwa wenn RootCA-Keys in der CA kompromittiert werden –, so müssen diese Mängel ausgeräumt werden, bevor Gegenmaßnahmen erfolgen.

Der Ablauf bei einer Aktualisierung

Angenommen, das Trust Center möchte eine Aktualisierung von in der PKI verwendeten Komponenten durchführen, weil dies beispielsweise durch Auftritt eines Schadens und Prüfung des Schadens durch die entsprechenden Stellen – wie etwa das BSI – notwendig geworden ist. Es wird vorausgesetzt, dass die neuen kryptographischen Komponenten von entsprechenden Stellen – wie etwa dem TÜV – evaluiert wurden.

Der Ablauf zur Aktualisierung von Komponenten gliedert sich grob wie folgt:

Der Update Service des Trust Centers führt mittels des neuen Update Management Protocols (UMP) den Update bei allen involvierten Certificate Holdern durch, d. h. beim gesamten Equipment, das die CHs nutzen. In Abschnitt 1.3.2 auf Seite 23 wurde das Modell eines Clients mit Personal Security Environment (PSE) in verschiedenen Varianten beschrieben, weshalb der Update ebenfalls anhand des Modells und aller Varianten für alle möglichen Anwendungsszenarien betrachtet wird.

Initiiert wird die Aktualisierung, indem der Update Service ein UpdateComponent an alle involvierten CHs verschickt. Das UpdateComponent ist für alle CHs identisch. Die CHs können alle, einige oder nur ein einziger Zertifikatsinhaber sein. Das UpdateComponent enthält alle Informationen über zu deaktivierende und zu installierende Komponenten. Während ein Sicherheitsanker direkt im UpdateComponent übermittelt wird, wird der Code für eine kryptographische Komponente nicht direkt, sondern nur der Link zum Code angegeben. Auf diese Weise lässt sich der Umfang des UpdateComponents verringern, die Netzlast – wenn mehrere Benutzer den Code laden – zeitlich strecken und Code für verschiedene Systeme diversifizieren – indem jeder Client den angegebenen Link um seine Systemkonfiguration ergänzt und somit genau den Code lädt, der für sein System geeignet ist. Falls der Code – z. B. aus patentrechtlichen Gründen – nicht im Klartext übertragen werden soll, wird er verschlüsselt übermittelt.

Dabei sind verschiedene Management-Strukturen zu beachten: Bei einer Signatur-Anwendung kann der Update Service eine besondere, mit einem UpdateComponent versehene E-Mail verschicken, während bei einer Authentisierungs-Anwendung ein Umweg gegangen werden muss, weil der Update Service in einer solchen Anwendung i. A. nach Ausgabe von Chipkarte und Zertifikat keinen Kontakt mehr zum CH hat. Der Umweg verläuft hier über Revokationsinformationen, so dass einem Server in einer Authentisierungs-Anwendung die Aufgabe zukommt, an alle Clients das UpdateComponent weiterzuleiten.

Wenn der Client ein UpdateComponent erhalten hat, werden zunächst dort Client-relevante Aktionen ausgeführt. Anschließend reicht der Client ein besonderes Chipkarten-Kommando des UpdateComponents an eine mögliche Chipkarte weiter, wo ICC-relevante Aktionen ausgeführt werden.

In die Bestätigung, in der ein CH dem Update Service den erfolgreichen Update anzeigt, können in der PSE des CHs neu erzeugte öffentliche Schlüssel integriert werden, die der Update Service an die Certification Authority zur Zertifizierung weitergeleitet. Nach Abschluss des Updates und Installation neuer Zertifikate und Policies ist die Aktualisierung abgeschlossen.
Abbildung 3.2 illustriert den Ablauf bei einer Aktualisierung grob.

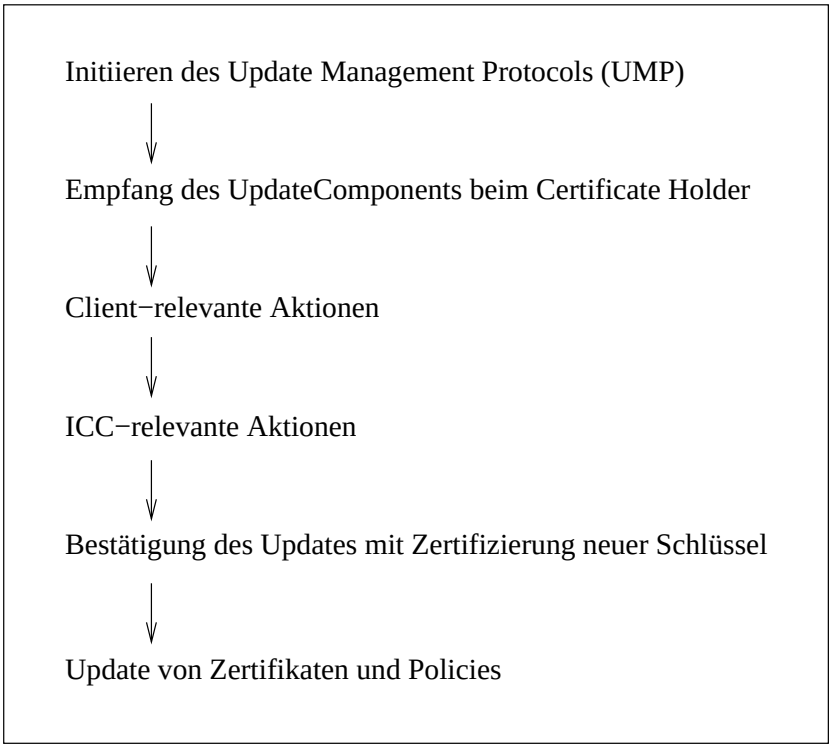

Abbildung 3.2: Ablauf bei einer Aktualisierung (grob)

Der Einsatz von multipler Kryptographie in einer PKI, um die Verfügbarkeit im Schadensfall gewährleisten zu können, und die Mechanismen zum Aktualisieren kryptographischer und einsatzspezifischer Komponenten sind sowohl für Signatur- als auch für Authentisierungs-Anwendungen sinnvoll.
Der Ablauf der Aktualisierung wird detailiert in Abschnitt 4.5 auf Seite 107 beschrieben.

Konsistenz

Wenn in einer PKI kryptographische und einsatzspezifische Komponenten aktualisiert werden, stellt sich die Frage nach der Konsistenz von Verfahren, Schlüsseln und Zertifikaten. Denn ein Update soll sich auf die gesamte PKI auswirken, Zertifikatsinhaber sollen während und nach einem Update sicher miteinander kommunizieren können und das Trust Center soll einen Überblick über die Aktualisierung behalten. Dieser Abschnitt begründet, weshalb der Zustand der PKI zu jedem Zeitpunkt „wohldefiniert“ ist, d. h. zu jeder Zeit die Sicherheit und Verfügbarkeit gewährleistet ist.
Diskutiert wird die Entwicklung einer Fail-Safe-PKI, in der PKI^A durch PKI^C ersetzt wird – wie in Graphik 3.3 dargestellt. Die Situation bei ausschließlichem Löschen bzw. Hinzufügen von Komponenten leitet sich aus dieser Diskussion ab.

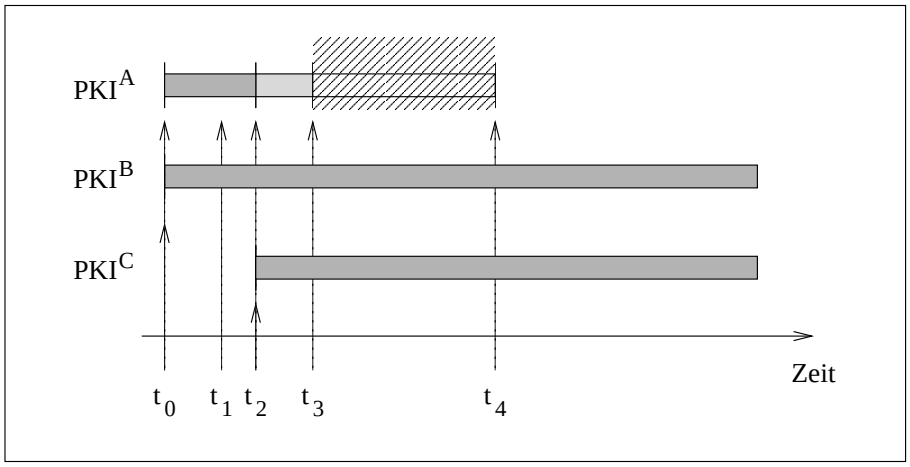

Abbildung 3.3: Konsistenz-Überlegungen

Zum Zeitpunkt t_0 stehen Verfahren, Schlüssel und Zertifikate für PKI^A und PKI^B bereit. Die von der Zertifizierungsinstanz ausgegebenen Zertifikate sind entscheidend: Durch sie erst sind beweiskräftige digitale Signaturen oder Verschlüsselungen möglich.

Zum Zeitpunkt t_1 tritt ein PKI^A berührender Schaden ein, so dass die CA zum Zeitpunkt t_2 alle Zertifikate zu PKI^A – cert^A – revoziert. Von nun an sollten keine mittels PKI^A signierten Dokumente mehr akzeptiert oder vertrauliche Dokumente über PKI^A verschlüsselt werden. Die Verfügbarkeit der Fail-Safe-PKI ist über PKI^B gewährleistet.

Ab t_3 wird der Austausch von PKI^A- durch PKI^C-Komponenten bei den CHs initiiert: Es werden Verfahren und Sicherheitsanker zu PKI^C installiert, neu generierte Schlüssel zur Zertifizierung an die CA geleitet, Zertifikate cert^C von der CA ausgegeben und sämtliche Komponenten aus PKI^A gelöscht. Zum Zeitpunkt t_4 ist der Austausch abgeschlossen, d. h. bei allen Teilnehmern ist PKI^A elliminiert und PKI^C installiert.

In der Zeit zwischen t_3 und t_4 gibt es in der PKI drei Arten von Zertifikatsinhabern: Es gibt Teilnehmer, die noch über PKI^A verfügen, Teilnehmer, die schon über Verfahren und Sicherheitsanker zu PKI^C besitzen, und Teilnehmer, die darüber hinaus schon auf einen zertifizierten Schlüssel zu PKI^C zurückgreifen können. Dabei ist zu beachten:

1. Die Verfügbarkeit der PKI ist über PKI^B gewährleistet.

2. Die noch vorhandenen Komponenten aus PKI^A sind aufgrund der revozierten Zertifikate nutzlos.

3. Eine Kommunikation mit Komponenten aus PKI^C ist erst dann möglich, wenn der Kommunikationspartner über ein entsprechendes Zertifikat verfügt.

Fazit: Zu jedem Zeitpunkt kann sich jeder Teilnehmer über das Verzeichnis der gültigen Zertifikate und die Sperrliste einen Überblick darüber verschaffen, welche Zertifikate es gibt und gültig sind und welche Teil-PKIs er benutzen kann.

3.3 Multiple digitale Signaturen

Wenn digitale Signaturen als elektronisches Pendant zur händischen Unterschrift genutzt werden, dann sollen sie über längere Zeit hinweg ihre Beweiskraft erhalten. In diesem Abschnitt wird dargestellt, wie durch Einsatz von multipler Kryptographie die Beweiskraft digitaler Signaturen über einen Schaden hinweg erhalten bleiben kann.

Eine Aktivität nach Eintritt eines Schadens ist zu spät. Deshalb müssen digitale Signaturen schon vor einem Schadensfall präpariert werden. Da multiple Kryptographie samt Schlüsseln und Zertifikaten in der Fail-Safe-PKI bereitsteht, können elektronische Dokumente multipel, d. h. parallel mit verschiedenen Signaturverfahren signiert werden.

Parallel bedeutet das Folgende: Angenommen, ein Dokument D sei digital zu signieren. Dann wird D zweimal separat mit den Signaturverfahren sign^A und sign^B und entsprechenden privaten Schlüsseln prK^A bzw. prK^B signiert:

$$(\mathrm{sign}^A(D, \mathrm{prK}^A),\ \mathrm{sign}^B(D, \mathrm{prK}^B))$$

Offensichtlich gewährleistet die multiple digitale Signatur den Beweiswert von D auch im Schadensfall: Wird PKI^A kompromittiert, bleibt die zweite Signatur bestehen; wird PKI^B kompromittiert, so bleibt die erste Signatur unangetastet.

Da in Signatur-Anwendungen – wie in Kapitel 1 beschrieben – für die Beweiskraft eines elektronischen Dokuments der Zeitstempel eines autorisierten Zeitstempeldienstes unumgänglich ist, ist der Zeitstempel ein sinnvoller Einsatzbereich für multiple digitale Signaturen. Dabei müssen zwei verschiedene Hashfunktionen benutzt werden, so dass sich ein multipler digitaler Zeitstempel wie folgt darstellt: Angenommen, ein Dokument D sei multipel zeitzustempeln. Dann wird D durch Anwendung zweier unabhängiger Hashfunktionen κ^A und κ^B zweimal gehasht: $h_A = \kappa^A(D)$ und $h_B = \kappa^B(D)$. Das Hashwert-Tupel (h_A, h_B) wird vom Zeitstempeldienst (TSA) zusammen mit der aktuellen Zeit t zweimal separat mit den Signaturverfahren sign^A und sign^B und den entsprechenden privaten Schlüsseln $\mathrm{prK}^A_{\mathrm{TSA}}$ bzw. $\mathrm{prK}^B_{\mathrm{TSA}}$ signiert:

$$(\mathrm{sign}^A((h_A, h_B) \,|\, t, \mathrm{prK}^A_{\mathrm{TSA}}),\ \mathrm{sign}^B((h_A, h_B) \,|\, t, \mathrm{prK}^B_{\mathrm{TSA}}))$$

Ein signiertes Dokument, welches aus beweistechnischen Gründen mit einem multiplen Zeitstempel versehen wird, braucht selbst nicht zwingend mit einer multiplen digitalen Signatur versehen zu sein, denn der Zeitstempeldienst dokumentiert mit seiner multiplen digitalen Signatur, dass ihm zum Zeitpunkt t das Dokument vorgelegen hat und damit vor Eintritt eines eventuellen Schadens erzeugt wurde. Diese Tatsache lässt sich auch im Schadensfall überprüfen.

Um nach einem Schaden wieder über eine multiple digitale Signatur verfügen zu können, kann das betreffende Dokument durch ein „Nachsignieren“ oder „Re-Signieren“ mit einer „erneuten digitalen Signatur“ [SigV97, §18] versehen werden. Durch die Existenz von mindestens einer noch sicheren Signatur ist nach einem Re-Signieren die durchgängige Beweisbarkeit gewährleistet. Ein Re-Signieren kann – wie in Abschnitt 2.1.3 auf Seite 45 ausgeführt – von einem Zeitstempeldienst ausgeführt werden, der in diesem Fall einen neuen Zeitstempel mit einer multiplen digitalen Signatur erzeugt, so dass das Dokument wieder multipel abgesichert ist.

Multiple digitale Signaturen können darüber hinaus auch die Sicherheit erhöhen, falls ein Schaden zwar eingetreten ist, aber von der Öffentlichkeit nicht bemerkt wurde. Eine multiple Signatur könnte ein Ausnutzen dieses Schadens unterbinden. Dazu wäre allerdings notwendig, im

Verifikationsprozess stets sämtliche vorhandenen digitalen Signaturen zu verifizieren und zu validieren, und ein Dokument nur dann zu akzeptieren, wenn alle Signaturen gültig verifiziert und validiert sind. Wird allerdings davon ausgegangen, dass ein Schaden nicht unbemerkt bleibt, ist dieses Vorgehen nicht nötig, weil dann die bestehenden Verfahren bis zur Revokation sicher sind. Intention ist deshalb, stets eine der Signaturen zu verifizieren und zu validieren und nur im Schadensfall – wenn die Signatur als nicht valid abgewiesen wird – auf eine der weiteren Signaturen zurückzugreifen und somit den Beweiswert der Signatur herzustellen.

Auf Revokationsmechanismen kann nach wie vor nicht verzichtet werden, weil auch andere Gründe – wie etwa die Beendigung eines Arbeitsverhältnisses oder der Verlust der Chipkarte samt PIN – zu einer Sperrung führen können.

Multiple digitale Signaturen sind in Signatur-Anwendungen sinnvoll. Eine Arbeitsteilung kann erfolgen, wenn der Zeitstempel multipel ausgeführt ist. Multiple digitale Signaturen machen in Authentisierungs-Anwendungen wenig Sinn, wenn die dort signierten Zufallszahlen nicht aufbewahrt werden, was meistens der Fall ist.

3.4 Erweiterte Revokationsmechanismen

In diesem Abschnitt wird eine Lösung präsentiert, um Revokationsinformationen im Schadensfall absichern zu können und um große Revokationslisten zu vermeiden.

Um Sperrlisten (Certificate Revocation Lists – CRLs) und Antworten auf Zertifikats-Status-Anfragen (Online Certificate Status Protocol Responses – OCSP Responses) vertrauen zu können, müssen sie mit multiplen digitalen Signaturen entsprechend Abschnitt 3.3 auf Seite 75 abgesichert werden und diese multiplen Signaturen können im Verifikationsvorgang erst dann akzeptiert werden, wenn alle Signaturen mathematisch korrekt sind. Eine Validierung via CRL oder OCSP ist nicht möglich.

Als weitere Ergänzung wird es in der Fail-Safe-PKI möglich sein, mit einem Eintrag einen ganzen Bereich von Zertifikatsseriennummern zu revozieren, um große Sperrlisten und die damit verbundene Unhandlichkeit zu vermeiden. Seriennummern werden von der CA vergeben und sind innerhalb einer CA einmalig vorhanden. Die Idee ist, dass die CA die Zertifikatsseriennummern folgendermaßen vergibt:

Angenommen, es gibt n Signaturverfahren, $n \in \mathbb{N}$, $n \geq 2$, welche die CA zur Zertifizierung nutzt. Diese Verfahren seien durchnummeriert, so dass $V = \{1, ..., n\}$ die Menge der Signaturverfahren darstellt. Mit jedem Signaturverfahren i, $i = 1, ..., n$, werden N_i, $N_i \in \mathbb{N}$, Zertifikate erstellt. Die Zertifikatsseriennummer für ein Zertifikat berechnet sich als

$$i * 10^k + n_i,$$

wobei n_i, $n_i \leq N_i$, eines der N_i mit Verfahren i zertifizierten Zertifikate und $k \in \mathbb{N}$ hinreichend groß gewählt ist, so dass $N_i < 10^k$ für alle i ist. Damit kennzeichnen die ersten Stellen der Seriennummer ein Signaturverfahren und die letzten $k-1$ Stellen ein Zertifikat, das mit diesem Signaturverfahren erzeugt wurde. Jedes Zertifikat hat damit innerhalb der CA eine einzigartige Seriennummer.

Zum Sperren aller Zertifikate, die das Signaturverfahren mit der Nummer j nutzen, wird „das“ Zertifikat

$$j * 10^k$$

gesperrt. Diese Sperrung ist so zu deuten, dass alle Zertifikate mit Nummer $n \in N$ gesperrt werden, wobei $N = \{j * 10^k + m \,|\, m = 1, ..., N_j\}$ ist.

Falls im Laufe der Zeit neue Signaturverfahren zur Zertifizierung genutzt werden, ist ein Anpassen möglich, indem V erweitert wird. k muss deshalb auch für mögliche Erweiterungen hinreichend groß gewählt werden.

Die Anwendung dieser Ideen ist sowohl für Signatur- als auch für Authentisierungs-Anwendungen sinnvoll.

3.5 Iterative Verschlüsselungen

In diesem Abschnitt wird eine Lösung zum Problem des Verlusts der Vertraulichkeit diskutiert.

Die Idee ist, Dokumente, die kurzfristig geheim bleiben sollen, in der Fail-Safe-PKI iterativ, d. h. mehrfach mit verschiedenen Verfahren zu verschlüsseln.

Iterativ bedeutet das Folgende: Angenommen, ein Dokument D sei zu verschlüsseln. Dann wird D mittels symmetrischem Verfahren enc_sym^A und einem zufällig gewählten Session Key sK^A verschlüsselt und sK^A mit einem asymmetrischen Verfahren enc_asym^A und dem öffentlichen Schlüssel des Empfängers puK^A in ein Kryptogramm verwandelt. Das so entstehende zusammengesetzte Kryptogramm

$$D^* = (\text{enc\_sym}^A(D, \text{sK}^A),\ \text{enc\_asym}^A(\text{sK}^A, \text{puK}^A))$$

ist das neue D bei einer erneuten Verschlüsselung mit PKI^B-Komponenten:

$$(\text{enc\_sym}^B(D^*, \text{sK}^B),\ \text{enc\_asym}^B(\text{sK}^B, \text{puK}^B))$$

Offensichtlich gewährleistet die iterative Verschlüsselung Vertraulichkeit von D auch im Schadensfall: Wird PKI^A kompromittiert, ist die „innere“ Verschlüsselung zwar unsicher, auf Grund der „äußeren“ Verschlüsselung mit den noch sicheren Komponenten aus PKI^B bleibt D vertraulich. Andersherum, wird PKI^B kompromittiert, bleibt die innere Verschlüsselung mit PKI^A intakt.

Da nach einem Schadensfall die kompromittierten Komponenten durch neue ersetzt werden, müssen Kryptogramme vor Deaktiveren des Entschlüsselungsverfahren ent- und nach Installation eines neuen Verfahrens und neuer Schlüssel erneut verschlüsselt werden.

Langfristige Verschlüsselungen werden aus den bereits erwähnten Gründen (siehe Abschnitt 2.2.3 auf Seite 54) in dieser Arbeit nicht erörtert.

3.6 Interoperabilität

Interoperabilität zwischen Teilnehmern der Fail-Safe-PKI und Teilnehmern, die dieses Fail-Safe-Konzept nicht nutzen, ist – auch aus Akzeptanz-Gründen – wichtig. Dieser Abschnitt beleuchtet einige Aspekte zur Interoperabilität.

Die Interoperabilität muss einerseits bzgl. kryptographischen Komponenten und andererseits bzgl. Protokollen betrachtet werden.

Interoperabilität der kryptographischen Komponenten:
Angenommen, CH_1 sei ein Zertifikatsinhaber in einer Fail-Safe-PKI, die aus zwei Teil-PKIs

PKI^A und PKI^B besteht. Seine CA unterstützt PKI^A und PKI^B. Angenommen, ein weiterer Zertifikatsinhaber CH_2 verfügt über kein Fail-Safe-Konzept, sondern eine „normale“ PKI – ohne Beschränkung der Allgemeinheit sei dies PKI^A. Dann können die beiden Teilnehmer CH_1 und CH_2 über PKI^A sicher miteinander kommunizieren. Die Interoperabilität ist gewährleistet.

Interoperabilität der Protokolle:
Der de facto Standard PKCS#7 sieht multiple digitale Signaturen und iterative Verschlüsselungen bereits vor (siehe Appendix A.6 auf Seite 206). Dabei hängt die Interoperabilität davon ab, ob zwei kommunizierende Anwendungen PKCS#7 identisch interpretiert haben und auch eine zweite Signatur verifizieren oder ein entschlüsseltes Kryptogramm erneut entschlüsseln.

Die neuen Protokolle, die in der Fail-Safe-PKI für multiple Zeitstempel, multiple Sperrlisten, multiple Zertifikats-Status-Antworten oder erweiterte Revokationsmechanismen genutzt werden, sind zu den bestehenden Protokollen nicht kompatibel. Um eine spätere Integration in die Standards zu erleichtern, basieren die neuen Protokolle auf den jeweiligen Standards und unterscheiden sich von ihnen so wenig wie möglich. Neue und standardisierte Protokolle können nebeneinander koexistent bestehen: Entweder entspricht die Anfrage nach einer „neuen“ Dienstleistung dem Standard und ist nur durch eine entsprechend neue Versionsnummer gekennzeichnet, so dass der Server aufgrund dieser Nummer entscheiden kann, ob eine einfach oder eine multipel signierte Antwort gewünscht ist, oder die Anfrage unterscheidet sich selber schon vom Standard, so dass für ein neues Protokoll ein anderer Server anzusprechen ist. Wenn beispielsweise CH_2 einen Zeitstempel anfordert, so erhält er eine ausschließlich mittels PKI^A signierte Antwort, die für ihn interpretierbar ist. CH_1 aber hat die Möglichkeit, bei der Anforderung nach einem Zeitstempel zwei Hashwerte zu übermitteln und einen Server anzusprechen, welcher im die Antwort multipel über PKI^A und PKI^B signiert.

Das neue Update Management Protocol ist nur innerhalb der Fail-Safe-PKI interpretierbar, so dass ein Teilnehmer einer Nicht-Fail-Safe-PKI es nicht verstehen und also ablehnen wird.

Das Fail-Safe-Konzept für Public-Key-Infrastrukturen ist zu Signaturgesetz und -verordnung in dem Sinne kompatibel, dass eine SigG-konforme PKI das Fail-Safe-Konzept unterstützen kann.

Es gibt heutzutage – ganz abseits von der Fail-Safe-Konzeption – große Probleme bzgl. Interoperabilität zwischen PKIs, die nicht dieselbe RootCA haben, wie es z. B. bei Firmen-PKIs oft der Fall ist. Zwei Lösungen sind Cross-Zertifikate und die Bridge-CA: Je zwei CAs zertifizieren sich gegenseitig über so genannte Cross-Zertifikate. Da die Anzahl der Cross-Zertifizierungen mit der Anzahl der CAs quadratisch zunimmt ($n^2 - n$ Cross-Zertifikate bei n CAs), ist die Bridge-CA entwickelt worden, welche als „Brücke“ zwischen CAs fungiert und mit $2n$ Zertifikaten bei n CAs auskommt. Beide Ideen sind in einer Fail-Safe-PKI möglich.

3.7 Performance und Kosten

Die Performance und die zusätzlichen Kosten des Fail-Safe-Konzepts sind für die Akzeptanz dieser Ideen wichtig. Globale Aussagen lassen sich nicht aufstellen, weil Performance und Kosten anwendungsabhängig sind. In diesem Abschnitt werden einige wichtige Aspekte geschildert, die bei Performance- und Kosten-Überlegungen zu bedenken sind.

Zunächst, welche Kosten fallen in einer Fail-Safe-PKI an? Erst einmal sind Aufwendungen für den Aufbau der PKI notwendig. Das Trust Center mit seiner Certification Authority, der Registration Authority und dem Verzeichnisdienst müssen räumlich und personell ausgestattet werden. Bei Realisierung des Fail-Safe-Konzepts ergibt sich hier kein Mehraufwand, weil Räume und Personal ohnehin zur Verfügung stehen.

Für den Betrieb der PKI ist eine Schlüssel- und Zertifikatserzeugung Voraussetzung. In einer Fail-Safe-PKI müssen doppelt soviele Schlüssel generiert und zertifiziert werden. Dieser Mehraufwand muss in einem Trust Center geleistet werden.

Bei der Verteilung der Schlüssel und Zertifikate ist kein Mehraufwand abzusehen, weil Schlüssel und Zertifikate ohnehin übergeben werden müssen. Im Falle von Chipkarten sollen sämtliche Schlüssel auf einer einzigen Chipkarte Platz finden, so dass nach-wie-vor nur eine Chipkarte übergeben werden muss.

Die Kosten der Software für das Fail-Safe-Konzept liegen nicht wesentlich höher als herkömmliche PKI-Software, weil nur in der Konzeption und ihrer Realisierung zusätzliche Funktionalitäten des Fail-Safe-Konzepts bedacht, implementiert und getestet werden müssen.

Welche Performance bietet eine Fail-Safe-PKI im täglichen Umgang mit Signaturen und Verschlüsselungen? Ganz klar: Das Fail-Safe-Konzept benötigt Aufwand! Eine Fail-Safe-PKI beansprucht gegenüber einer herkömmlichen PKI mehr Speicherplatz, mehr Rechenzeit und eine höhere Netzlast. Aber: Wie hoch dieser Aufwand ist, hängt individuell von der jeweiligen Anwendung und den dort realisierten vier Bausteinen des Fail-Safe-Konzepts „nachhaltige Existenz multipler Verfahren“, „multiple digitale Signaturen“, „erweiterte Revokationsmechanismen“ und „iterative Verschlüsselungen“ und dem eingesetzten Equipment ab:

- Eine Authentisierungs-Anwendung benötigt weder multiple digitale Signaturen noch iterative Verschlüsselungen. Es reicht aus, wenn multiple Verfahren in der PKI bereitstehen, um im Schadensfall einzuspringen, was einer kalten Redundanz entspricht. Demgegenüber sind multiple Signaturen und iterative Verschlüsselungen in einer Signatur-Anwendung durchaus sinnvoll, so dass hier von warmer Redundanz gesprochen werden kann. Funktionalitäten zum Update von Komponenten sind sowohl für Authentisierungs- als auch für Signatur-Anwendungen sinnvoll, kommen allerdings im „normalen“ Betrieb der PKI wahrscheinlich eher seltener zum Einsatz. In beiden Klassen von Anwendungen sind erweiterte Revokationsmechanismen sinnvoll.

- Sind Server oder Clients involviert, so sind ein erhöhter Speicherbedarf und umfangreichere Berechnungen u. U. tolerierbar, während Chipkarten an ihre Grenzen stoßen können. Andererseits können Aufgaben der Chipkarte, wie etwa das Ver- und Entschlüsseln von Dokumenten mit dem Session Key, in Arbeitsteilung auf den Client verlagert werden.

Gegen diese Nachteile müssen die Vorteile des Fail-Safe-Konzepts aufgerechnet werden: Fällt die Verfügbarkeit einer – auch wichtigen – Public-Key-Infrastruktur aus, bis neue Komponenten entwickelt, produziert und verteilt sind, können sehr hohe Kosten entstehen. Demgegenüber ist die Verfügbarkeit im Schadensfall in einer Fail-Safe-PKI gesichert. Die Performance des Austauschs kompromittierter Komponenten ist unkritisch, weil erstens die Verfügbarkeit der PKI gesichert ist und zweitens dieser Austausch deutlich besser als die Verteilung neuer Verfahren, Schlüssel und Zertifikate auf herkömmlichem Weg ist.

Für exakte Performance-Überlegungen gilt es nun abzuwägen, welches Maß an Sicherheit eine Anwendung braucht, wie stark eine Anwendung gegen die erwähnten offenen Probleme abgesichert werden muss und was diese Fail-Safe-Maßnahmen kosten.

3.8 Verwandte Themen

Die Problematik, dass Kryptoverfahren nicht beweisbar sicher sind und jederzeit gebrochen werden können, ist in der Literatur bereits thematisiert worden. Dieser Abschnitt gibt einen Überblick über in der Literatur vorgestellte Lösungen.

Das Konzept der multiplen Signaturen ist nicht neu. Beispielsweise wird in [FJPP95] eine solche Idee vorgestellt. Allerdings ist die Motivation für dieses Vorgehen eine andere: Die Idee ist, einerseits Vertrauen für individuelle Nutzer und andererseits Vertrauen staatlicher Regulierer in Einklang zu bringen. Das Vertrauen der Nutzer in dieses System soll sich nicht nur in einem beim privaten Benutzer oder der staatlichen Zentrale generierten Schlüssel äußern, sondern es sollen Schlüssel, die von mehreren Seiten erzeugt wurden, verwendet werden. Auf Verfahren und insbesondere die Nutzung verschiedener Verfahren wird nicht eingegangen. Petra Wohlmacher stellt in [Wohl00] und [Wohl01] dazu Multiple-Key-Verschlüsselungsverfahren, Multiple-Key-Parameter-Verschlüsselungsverfahren, multiples Verschlüsseln, Multiple-Key-Signaturverfahren, Multiple-Key-Parameter-Signaturverfahren und multiples Signieren und die entsprechenden Signaturformate vor. Es wird allerdings stets ein(!) Verfahren mit unterschiedlichen Schlüsseln benutzt, so dass diese Konzepte nur bei Kompromittierung einzelner Schlüssel und nicht mehrerer Schlüssel oder gar des Verfahrens schützen. In [Wohl01, 9.6.2] diskutiert Wohlmacher ein „Notfallmanagement“, das unabhängig von dem hier präsentierten Fail-Safe-Konzept entstanden ist, wie sie in [Wohl01, 10] erwähnt. Die Unterschiede liegen

1. in der Namenswahl mehrfach signierter digitaler Signaturen – die hier benutzten „multiplen digitalen Signaturen“ heißen dort „Poly-Signaturen mit parallelen Signaturen“ –,
2. in den Zertifikaten – hier beinhalten Zertifikate nur eine Signatur, so dass PKI^A und PKI^B möglichst getrennt sind, was Vorteile bei Revokation eines(!) Zertifikats verspricht, während dort Zertifikate mehrfache Signaturen tragen – und
3. im Update-Konzept, welches dort nicht vorhanden ist.

Die Idee des Austauschs greift Andreas Pfitzmann in [Pfit01] auf: “In der Signierkomponente könnten mehrere asymmetrische kryptographische Algorithmen implementiert werden, deren Sicherheit möglichst sogar auf unterschiedlichen mathematischen Problemen basiert und deren Sicherheit in unterschiedlicher Weise validiert oder gar bewiesen ist. Entsprechend könnten alle diese Algorithmen parallel verwendet werden, wenn es um Signaturen höchster Sicherheit geht. Eine solche Signatur ist nur dann gültig, wenn alle parallel erzeugten und zu prüfenden Signaturen gültig sind. Ggf. können einzelne Signatursysteme fließend ausgetauscht werden, so daß dieser parallele Betrieb in der Lage ist, sich an ‚neue Kryptotrends‘ anzupassen.“

Es gibt auch Konzepte, um die Beweisbarkeit digitaler Signaturen auch im Schadensfall zu gewährleisten – mittels so genannter Fail-Stop-Signaturen. Fail-Stop-Signaturen sind Signaturen, bei denen es zum öffentlichen Schlüssel mehrere private Schlüssel gibt. Es gibt also mehrere gültige Signaturen, aber nur eine, mit dem der Signierer signiert. Im Fall, dass das kryptographische Verfahren gebrochen oder der private Schlüssel kompromittiert wird, kann der behauptete Signierer einen Echtheitsbeweis antreten und zeigen, dass er eine andere gültige Signatur erzeugen würde [PePf97]. Ein Problem könnte sein, dass solche Fail-Stop-Signaturen die Verbindlichkeit digitaler Signaturen unterlaufen, denn es gibt mehrere korrekte Signaturen zu einem Dokument. Und der Signierer könnte selbst ein Interesse daran haben, seinen eigenen Schlüssel zu kompromittieren und zu behaupten, er hätte eine andere Signatur erzeugt.

Die Problematik der Zeitstempel, falls sich die Hashfunktion als kryptographisch schwach erweisen sollte, wurde in [KeSc01] im Rahmen des Opelix-Projekts [Opelix] aufgegriffen. Opelix steht für “Open Personalized Electronic Information Commerce System“. Das dort vorgestellte Konzept „verketteter Zeitstempel“ entstand unabhängig zum Fail-Safe-Konzept und wurde erstmals auf der EVA 2001-Konferenz im März 2000 vorgestellt.

Die Problematik, dass ein Schlüssel und insbesondere ein Wurzelzertifizierungsinstanz-Schlüssel kompromittiert werden kann und das dadurch ein Benutzer „vom Rest des Systems abgeschnitten“ [Road00, 3.5.6.2] wäre bis ein neuer Sicherheitsanker verteilt ist, wird in PKIX erwähnt [Road00]. Es war bislang aber keine Lösung zu diesem Problem entwickelt worden.

3.9 Zusammenfassung

Das Fail-Safe-Konzept für Public-Key-Infrastrukturen besteht darin, eine PKI zu konzipieren, die im Fall, dass kryptographische oder einsatzspezifische Komponenten kompromittiert werden,

- die Verfügbarkeit der PKI-Anwendung sichert,
- die Beweisbarkeit digitaler Signaturen gewährleistet,
- die Revokationsmechanismen ohne “out-of-band“-Methoden bereitstellt und
- die Vertraulichkeit von verschlüsselten Daten garantiert.

Erreicht wird dies durch die Integration mehrerer unabhängiger Kryptoverfahren samt Schlüsseln und Zertifikaten in alle Komponenten einer PKI, so dass multiple digitale Signaturen, iterative Verschlüsselungen und erweiterte Revokationsinformationen genutzt werden können. Eine wesentliche Eigenschaft des Fail-Safe-Konzepts ist die Möglichkeit, mit dem neuen Update Managment Protocol Komponenten in der PKI an neue Empfehlungen dynamisch und sicher anzupassen – auch wenn Verfahren oder Sicherheitsanker kompromittiert sind.

Dieses Konzept leitet sich aus der Definition eines allgemeinen Fail-Safe-Konzepts ab, das über die drei Eigenschaften „Redundanz der technischen Mittel“, „Unabhängigkeit der redundanten technischen Mittel“ und „Nachhaltigkeit der Redundanz und der Unabhängigkeit“ charakterisiert ist.

Überlegungen zur Performance und den zusätzlichen Kosten des Fail-Safe-Konzepts zeigen, dass globale Aussagen nicht gemacht werden können, weil der Aufwand von Anwendung zu Anwendung variiert und abhängig vom technischen Equipment sowie des geforderten Sicherheitsniveaus ist. Dennoch ist abzusehen, dass die zusätzlichen Kosten zum Betrieb der Fail-Safe-PKI – gerade im Hinblick auf den Sicherheitsgewinn – vernachlässigbar sind. Die für die Akzeptanz wichtige Interoperabilität zu PKIs, die kein Fail-Safe-Konzept nutzen, ist gegeben, und ebenso ist eine Konformität zum Signaturgesetz möglich.

Nachdem in diesem Kapitel Lösungen zu den vier Problemen im Fail-Safe-Konzept vorgestellt wurden, wird im nächsten Kapitel die Realisierung in einer PKI in allen Details diskutiert.

Kapitel 4

Fail-Safe-PKI

Das Fail-Safe-Konzept für Public-Key-Infrastrukturen besteht aus den vier Bausteinen: Nachhaltige Existenz multipler Kryptographie, multiple digitale Signaturen, erweiterte Revokationsmechanismen und iterative Verschlüsselungen. Deren Realisierung in einer Public-Key-Infrastruktur hängt von der konkreten Anwendung ab, welche ein Fail-Safe-Konzept nutzen will. Die Ideen, die realisiert werden können, sind,

- mehrere kryptographische Verfahren samt Schlüsseln und Zertifikaten in der PKI zu installieren – also mehrere unabhängige, parallel zu nutzende Teil-PKIs zu schaffen – und sicher hinzufügen und löschen zu können,
- Dokumente parallel multipel digital zu signieren,
- die Möglichkeit zu schaffen, mehrere Zertifikate mit einem Eintrag in der Sperrliste zu revozieren und
- Dokumente iterativ multipel zu verschlüsseln.

Diese im vorigen Kapitel 3 grob vorgestellten und auf ihre Praktikabilität hin überprüften Lösungsideen werden in diesem Kapitel in einer Public-Key-Infrastruktur exakt beschrieben. Die dabei entstehende „optimale“ PKI nutzt das komplette, aus vier Bausteinen bestehende Fail-Safe-Konzept. Diese Public-Key-Infrastruktur heißt Fail-Safe-PKI.

Dieses Kapitel gliedert sich folgendermaßen: Im ersten Abschnitt werden die Begriffe der Unabhängigkeit zweier kryptographischer Algorithmen und Verfahren mathematisch definiert und kryptographische Techniken wie einfache und multiple digitale Signaturen sowie einfache und iterative Verschlüsselungen erklärt. Anschließend wird der Aufbau der Fail-Safe-PKI dargelegt, der gegenüber einer herkömmlichen PKI über zusätzliche Verfahren, Schlüssel und Zertifikate verfügt. Die Fail-Safe-PKI wird anhand einer einstufigen PKI-Hierachie beschrieben; eine Verallgemeinerung wird in Abschnitt 4.6 auf Seite 153 beschrieben. Die Funktionalitäten in der Fail-Safe-PKI werden in den dann folgenden drei Abschnitten diskutiert, die sich mit den von Kapitel 1 bekannten normalen Funktionalitäten, mit den erweiterten Funktionalitäten – multiple digitale Signaturen und iterative Verschlüsselungen – und der Funktionalität zum Aktualisieren beschäftigen. Der Update von kryptographischen und einsatzspezifischen Komponenten wird in zehn Phasen detailliert in allen Komponenten der PKI beschrieben. Die gewählte Darstellungsweise ist mathematisch motiviert, um die Sicherheit des Updates beweisen zu können. Überlegungen zur Performance und den Kosten einer Fail-Safe-PKI runden dieses Kapitel ab.

Die Protokolle sind abstrakt notiert. Detaillierte Beschreibungen der Datenformate in der üblichen Syntax finden sich in den Appendizes A und B.

4.1 Kryptographische Verfahren

Eine Fail-Safe-PKI bietet über eine „normale“ PKI hinausgehende, besondere kryptographische Techniken: multiple digitale Signaturen und iterative Verschlüsselungen. Damit diese Techniken einen Mehrwert gegenüber herkömmlichen Signaturen und Verschlüsselungen bieten, müssen die verwendeten Verfahren voneinander unabhängig sein. In diesem Abschnitt wird der Begriff der Unabhängigkeit zweier Algorithmen definiert sowie ihr Einsatz in den kryptographischen Techniken digitale Signatur und Verschlüsselung beschrieben.

Für das Fail-Safe-Konzept sind voneinander unabhängige Algorithmen notwendig; in dem Sinne, dass für zwei Algorithmen a und b gilt: Wird a kompromittiert, bleibt b davon unberührt. Um diesen Sachverhalt in einer mathematischen Definition klar festzulegen, muss ein Umweg gegangen werden, weil es keine Beweise für die Unabhängigkeit von Algorithmen gibt – genauso wie es keine Beweise für die Sicherheit von Kryptoalgorithmen selbst gibt. Der Umweg besteht aus Matrizen, an denen abzulesen ist, welche kryptographischen Verfahren nach derzeitigem Wissensstand als voneinander unabhängig anzusehen sind. Die mathematischen Definitionen von „Unabhängigkeit“ beziehen sich dann auf die Einträge dieser Matrizen.

Zu den Matrizen: Wie in Abschnitt 2.1.1 auf Seite 43 und Abschnitt 3.2.1 auf Seite 63 argumentiert, gibt es auf unterschiedlichen mathematischen Basisproblemen beruhende Algorithmen, die nach derzeitigem Kenntnisstand keine Korrelation zueinander aufweisen, so dass die Wahrscheinlichkeit für das gleichzeitige Kompromittieren dieser Algorithmen sehr unwahrscheinlich ist. Dieses Wissen wird in einer Matrix ausgedrückt: Eine Korrelation zwischen zwei Algorithmen liegt genau dann vor, wenn der Schnittpunkt (entlang Spalte und Zeile) zweier Algorithmen mit einem „X“ gekennzeichnet ist. Ist also der Schnittpunkt zwischen den Algorithmen a und b leer, so ist derzeit davon auszugehen, dass b sicher bleibt, auch wenn ein mathematisches Problem gelöst und a kompromittiert wird.

Dabei ist zu beachten, dass die Einträge nicht beweisbar sind und – wie die Liste kryptographisch geeigneter Algorithmen des BSIs – ausschließlich eine Empfehlung darstellen. Die Einträge der Matrix können sich also jederzeit ändern. Daraus ergibt sich die Notwendigkeit an Mathematiker, Algorithmen auf Korrelationen zu analysieren und dies zu dokumentieren.

Tabelle 4.1 stellt Korrelationen zwischen asymmetrischen Algorithmen dar.

Tabelle 4.1: Bekannte Korrelationen („X“) zwischen asymmetrischen Algorithmen

	RSA	DSA	ECDSA	RSA Enc	ElGamal	ECIES
RSA	X	X		X	X	
DSA	X	X		X	X	
ECDSA			X			X
RSA Enc	X	X		X	X	
ElGamal	X	X		X	X	
ECIES			X			X

Analog werden Korrelationen zwischen weiteren kryptographischen Primitiven β^1 und β^2 – also symmetrischen Algorithmen, Hashfunktionen und Formatierungen – in einer Matrix dargestellt (siehe Tabelle 4.2). Die Einträge sind entsprechend der Argumentation in Abschnitt 3.2.1 auf Seite 63 gestaltet.

Tabelle 4.2: Bekannte Korrelationen („X“) zwischen weiteren kryptographischen Primitiven

	DES_3	AES	MD5	SHA-1	RIPEMD	PKCS#1	ISO 9796	id
DES_3	X							
AES		X						
MD5			X					
SHA-1				X				
RIPEMD					X			
PKCS#1						X		
ISO 9796							X	
id								X

Definition 18 formuliert auf Grundlage der Matrizen (Tab. 4.1 und 4.2) den Begriff der Unabhängigkeit für asymmetrische Algorithmen, symmetrische Verschlüsselungsalgorithmen, Hashfunktionen und Formatierungen mathematisch.

Definition 18
Es heißen

- je zwei Signaturalgorithmen σ^i und σ^j, $i, j \in \mathbb{N}$, $i \neq j$,
- je zwei asymmetrische Verschlüsselungsalgorithmen η_asym^i und η_asym^j, $i, j \in \mathbb{N}$, $i \neq j$,
- ein Signaturalgorithmus σ^i und ein asymmetrischer Verschlüsselungsalgorithmus η_asym^j, $i, j \in \mathbb{N}$,
- zwei Hashfunktionen κ^i und κ^j, $i, j \in \mathbb{N}$, $i \neq j$,
- zwei Formatierungen ϕ^i und ϕ^j, $i, j \in \mathbb{N}$, $i \neq j$, und
- zwei symmetrische Verschlüsselungsalgorithmen η_sym^i und η_sym^j, $i, j \in \mathbb{N}$, $i \neq j$,

voneinander unabhängig, wenn in der Matrix von Tabelle 4.1 bzw. 4.2 kein „X“ zwischen ihnen im Schnittpunkt von Spalte und Zeile steht.

Für aus diesen kryptographischen Primitiven zusammengesetzte Verfahren überträgt sich der Begriff der Unabhängigkeit:

- Zwei Signaturverfahren sign^m und sign^n sind voneinander unabhängig, $m, n \in \mathbb{N}$, $m \neq n$, wenn die kryptographischen Primitiven, aus denen die Signaturverfahren zusammengesetzt sind, jeweils voneinander unabhängig sind. Es gilt also für $\text{sign}^m = \sigma^{m_\sigma} \circ \phi^{m_\phi} \circ \kappa^{m_\kappa}$ und $\text{sign}^n = \sigma^{n_\sigma} \circ \phi^{n_\phi} \circ \kappa^{n_\kappa}$: sign^m und sign^n sind voneinander unabhängig, wenn σ^{m_σ} und σ^{n_σ}, κ^{m_κ} und κ^{n_κ} sowie ϕ^{m_ϕ} und ϕ^{n_ϕ} jeweils voneinander unabhängig sind, dabei können ϕ^{m_ϕ} und ϕ^{n_ϕ} die identische Abbildung id darstellen, falls die Signaturverfahren keine Formatierung benötigen.
- Zwei symmetrische Verschlüsselungsverfahren enc_sym^k und enc_sym^l sind voneinander unabhängig, $k, l \in \mathbb{N}$, $k \neq l$, wenn die kryptographischen Primitiven, aus denen die Verfahren zusammengesetzt sind, jeweils voneinander unabhängig sind.

- Zwei asymmetrische Verschlüsselungsverfahren enc_asym^r und enc_asym^s sind voneinander unabhängig, $r, s \in \mathbb{N}$, $r \neq s$, wenn die kryptographischen Primitiven, aus denen die Verfahren zusammengesetzt sind, jeweils voneinander unabhängig sind.

- Ein Signaturverfahren und ein asymmetrisches Verschlüsselungsverfahren sind voneinander unabhängig, wenn ihre Primitiven keine Korrelationen aufweisen.

Beispiel voneinander unabhängiger Verfahren
Das Signaturverfahren rsasignatureWithripemd160, welches sich aus RSA, RIPEMD-160 und PKCS#1 zusammensetzt, ist zu ecdsa-with-SHA1 unabhängig, das sich aus ECDSA, SHA-1 und identischer Abbildung id zusammensetzt. Weitere Beispiele finden sich in Abschnitt 3.2.1 auf Seite 63.

In einer Public-Key-Infrastruktur, die das Fail-Safe-Konzept nutzt, sind mindestens zwei voneinander unabhängige Signatur- und Verschlüsselungsalgorithmen eingesetzt. Durch die Kombination mit verschiedenen Hashfunktionen und Formatierungen entstehen mehrere Signatur- und Verschlüsselungsverfahren. Für multiple digitale Signaturen und iterative Verschlüsselungen sind – weil nicht bekannt ist, welcher Schadensfall in der Zukunft auftreten wird – nur Verfahren sinnvoll, die keine Komponente gemeinsam haben. Dazu wird der Begriff der „Fail-Safe-Konzept-geeigneten“ Verfahren eingeführt.

Definition 19
Sei $m_S \in \mathbb{N}$, $m_S \geq 2$. Eine Menge von Signaturverfahren $S = \{\text{sign}^i \,|\, i = 1, ..., m_S\}$ heißt **Fail-Safe-Konzept-geeignet**, wenn es zu jedem Signaturverfahren sign $\in S$ ein dazu unabhängiges sign$' \in S$ gibt.
Seien $m_{\text{E\_sym}}, m_{\text{E\_asym}} \in \mathbb{N}$, $m_{\text{E\_sym}} \geq 2$, $m_{\text{E\_asym}} \geq 2$. Eine Menge symmetrischer Verschlüsselungsverfahren E_sym $= \{\text{enc\_sym}^i \,|\, i = 1, ..., m_{\text{E\_sym}}\}$ bzw. eine Menge asymmetrischer Verschlüsselungsverfahren E_asym $= \{\text{enc\_asym}^i \,|\, i = 1, ..., m_{\text{E\_asym}}\}$ heißt **Fail-Safe-Konzept-geeignet**, wenn zu jedem Verschlüsselungsverfahren enc_sym $\in$ E_sym bzw. enc_asym $\in$ E_asym ein dazu unabhängiges enc_sym$' \in$ E_sym resp. enc_asym$' \in$ E_asym existiert.

Kryptographische Techniken in der Fail-Safe-PKI sind die schon aus Kapitel 1 bekannten digitalen Signaturen (nach Definition 3 auf Seite 6) und Verschlüsselungen (nach Definition 6 auf Seite 12), die hier jetzt um den Zusatz „einfach“ ergänzt werden.

Definition 20
Sei sign$^i \in S$, $i \in \{1, ..., m_S\}$, ein Fail-Safe-Konzept-geeignetes Signaturverfahren und prK^i ein zu diesem Verfahren gehörender privater Schlüssel. Aus einem Dokument D wird eine **einfache digitale Signatur**[1] erzeugt:

$$S^i = \text{sign}^i(D, \text{prK}^i)$$

Der Gebrauch der einfachen digitalen Signatur lässt sich mit dem **Signatur/Verifikations-Modell für eine einfache digitale Signatur** verdeutlichen:

[1] Eine „einfache digitale Signatur“ im Fail-Safe-Konzept ist zur in Definition 3 eingeführten „digitalen Signatur“ identisch.

Signaturverfahren	$\text{sign}^i \in S$ für ein $i \in \{1, ..., m_S\}$
Private Key	$\text{prK}^i_{\text{Signierer}}$
Public Key	$\text{puK}^i_{\text{Signierer}}$
Zu Signieren / Input	D
Einfache digitale Signatur	$S^i = \text{sign}^i(D, \text{prK}^i_{\text{Signierer}})$
Signiertes Dokument übertragen	$S^* = (D, S^i)$
Verifizieren	$\text{verify}^i(D, S^i, \text{puK}^i_{\text{Signierer}}) = \text{TRUE}$ oder FALSE (abhängig vom Verifikationsalgorithmus)

Definition 21
Sei $\text{enc\_sym}^i \in$ E_sym, $i \in \{1, ..., m_{\text{E\_sym}}\}$, ein symmetrisches Verschlüsselungsverfahren mit einem geheimen Schlüssel sK^i. Sei $\text{enc\_asym}^j \in$ E_asym, $j \in \{1, ..., m_{\text{E\_asym}}\}$, ein asymmetrisches Verschlüsselungsverfahren mit dem zugehörigen öffentlichen Schlüssel eines Empfängers $\text{puK}^j_{\text{Empfänger}}$. Ein Klartext P wird mit einer **einfachen symmetrischen Verschlüsselung**

$$C^i = \text{enc\_sym}^i(P, \text{sK}^i)$$

bzw. einer **einfachen asymmetrischen Verschlüsselung**

$$D^j = \text{enc\_asym}^j(P, \text{puK}^j_{\text{Empfänger}})$$

in ein Kryptogramm verwandelt[2].

Der Gebrauch von einfachen symmetrischen, einfachen asymmetrischen oder einfachen hybriden Verschlüsselungen lässt sich anhand der folgenden Modelle ablesen:

Ver-/Entschlüsselungs-Modell für eine einfache symmetrische Verschlüsselung (Die Kommunikationsteilnehmer müssen den geheimen Schlüssel anderweitig übertragen):

Symm. Verschlüsselungsverfahren	$\text{enc\_sym}^i \in$ E_sym für ein $i \in \{1, ..., m_{\text{E\_sym}}\}$
Secret Key	sK^i
Klartext	P
Einfaches Verschlüsseln	$C^i = \text{enc\_sym}^i(P, \text{sK}^i)$
Kryptogramm übertragen	C^i
Entschlüsseln	$P = \text{dec\_sym}^i(C^i, \text{sK}^i)$

Ver-/Entschlüsselungs-Modell für eine einfache asymmetrische Verschlüsselung:

Asymm. Verschlüsselungsverfahren	$\text{enc\_asym}^j \in$ E_asym für ein $j \in \{1, ..., m_{\text{E\_asym}}\}$
Private Key des Empfängers	$\text{prK}^j_{\text{Empfänger}}$
Public Key des Empfängers	$\text{puK}^j_{\text{Empfänger}}$
Klartext	P
Einfaches Verschlüsseln	$D^j = \text{enc\_asym}^j(P, \text{puK}^j_{\text{Empfänger}})$
Kryptogramm übertragen	D^j
Entschlüsseln	$P = \text{dec\_asym}^j(D^j, \text{prK}^j_{\text{Empfänger}})$

[2]Eine „einfache Verschlüsselung“ im Fail-Safe-Konzept ist zur in Definition 6 eingeführten „Verschlüsselung“ identisch.

Ver-/Entschlüsselungs-Modell für eine einfache hybride Verschlüsselung:

Symm. Verschlüsselungsverfahren	$\text{enc\_sym}^i \in \text{E\_sym}$ für ein $i \in \{1, ..., m_{\text{E\_sym}}\}$
Session Key (zufällig)	sK^i
Asymm. Verschlüsselungsverfahren	$\text{enc\_asym}^j \in \text{E\_asym}$ für ein $j \in \{1, ..., m_{\text{E\_asym}}\}$
Private Key des Empfängers	$\text{prK}^j_{\text{Empfänger}}$
Public Key des Empfängers	$\text{puK}^j_{\text{Empfänger}}$
Klartext	P
Einfache Verschlüsselung von P	$C^i = \text{enc\_sym}^i(P, \text{sK}^i)$
Einfache Verschlüsselung von sK^i	$D^j = \text{enc\_asym}^j(\text{sK}^i, \text{puK}^j_{\text{Empfänger}})$
Übertragen	(C^i, D^j)
Entschlüsselung von D^j	$\text{sK}^i = \text{dec\_asym}^j(D^j, \text{prK}^j_{\text{Empfänger}})$
Entschlüsselung von C^i	$P = \text{dec\_sym}^i(C^i, \text{sK}^i)$

Zur Vermeidung der offenen Probleme können multiple digitale Signaturen und iterative Verschlüsselungen genutzt werden. Sie nutzen jeweils mehrere kryptographische Verfahren, die voneinander unabhängig sind, um im Schadensfall die Beweiskraft einer digitalen Signatur oder Verschlüsselung weiterhin gewährleisten zu können.

Definition 22
Sei $S = \{\text{sign}^i \,|\, i \in 1, ..., m_S\}$ eine Menge Fail-Safe-Konzept-geeigneter Signaturverfahren mit zugehörigen privaten Schlüsseln $\{\text{prK}^i_{\text{Signierer}}\}$. Seien sign^{i_1} und sign^{i_2} zwei Signaturverfahren aus S, die voneinander unabhängig sind. Aus einem Dokument D wird eine **multiple digitale Signatur**

$$S^{i_1,i_2} = (S^{i_1},\, S^{i_1}) = (\text{sign}^{i_1}(D, \text{prK}^{i_1}_{\text{Signierer}}),\, \text{sign}^{i_2}(D, \text{prK}^{i_2}_{\text{Signierer}}))$$

generiert.

Sei $V \subseteq \{i_1, i_2\}$, $v = \#V$, $\#V \geq 1$. V ist die Indexmenge der Verfahren, die zur Verifikation herangezogen wird. Im Verifikationsprozess werden v Signaturen aus S^{i_1,i_2} mit den zugehörigen öffentlichen Schlüsseln $\text{puK}^V_{\text{Signierer}} \subseteq \{\text{puK}^{i_1}_{\text{Signierer}}, \text{puK}^{i_2}_{\text{Signierer}}\}$ verifiziert – also eine oder beide. Es müssen v Einzelsignaturen gültig sein, d. h. mit TRUE verifiziert werden, damit das Gesamtergebnis TRUE ist:

$$\text{verify}^V(D, S^{i_1,i_2}, \text{puK}^V_{\text{Signierer}}) = \text{TRUE},$$

wenn $\text{verify}^k(D, S^k, \text{puK}^k_{\text{Signierer}}) = \text{TRUE}$ für alle $k \in V$.

Die Konstruktion mit der Teilmenge V trägt dem Umstand Rechnung, dass im Verifikationsprozess nicht zwingend alle Signaturen verifiziert werden müssen, wie die Beispiele in Abschnitt 4.3 (Seite 98) und 4.4 (Seite 103) zeigen.

Das **Signatur/Verifikations-Modell für eine multiple digitale Signatur** ist das Folgende:

Fail-Safe-Konzept-geeignete Signaturverfahren	$S = \{\text{sign}^i \,|\, i = 1, ..., m_S\}$

Nutzende Signaturverfahren	$\{\text{sign}^{i_1}, \text{sign}^{i_2}\} \subseteq S$, sign^{i_1} und sign^{i_2} voneinander unabhängig
Private Keys	$\{\text{prK}^{i_1}_{\text{Signierer}}, \text{prK}^{i_2}_{\text{Signierer}}\}$
Public Keys	$\{\text{puK}^{i_1}_{\text{Signierer}}, \text{puK}^{i_2}_{\text{Signierer}}\}$
Zu Signieren / Input	D
Multiples Signieren	$S^{i_1,i_2} = (\text{sign}^{i_1}(D, \text{prK}^{i_1}_{\text{Signierer}}), \text{sign}^{i_2}(D, \text{prK}^{i_2}_{\text{Signierer}}))$
Signiertes Dokument	$S^* = (D, S^{i_1,i_2})$
Verifizieren mit	$V \subseteq \{i_1, i_2\}$, d. h. entweder wird die erste, die zweite oder beide Einzelsignaturen mit entsprechenden öffentlichen Schlüsseln verifiziert: 1. Fall: $V = \{i_1\}$ mit $\{\text{puK}^{i_1}_{\text{Signierer}}\}$ 2. Fall: $V = \{i_2\}$ mit $\{\text{puK}^{i_2}_{\text{Signierer}}\}$ 3. Fall: $V = \{i_1, i_2\}$ mit $\{\text{puK}^{i_1}_{\text{Signierer}}, \text{puK}^{i_2}_{\text{Signierer}}\}$
Verifikationsergebnis	$\text{verify}^V(D, S^{i_1,i_2}, \text{puK}^V_{\text{Signierer}}) = \text{TRUE}$, falls die v Einzelverifikationen TRUE sind, $v = \#V$

Definition 23
Sei $\text{E\_sym} = \{\text{enc\_sym}^i \mid i = 1, ..., m_{\text{E\_sym}}\}$ eine Menge symmetrischer Verschlüsselungsverfahren und sei $\text{E\_asym} = \{\text{enc\_asym}^i \mid i = 1, ..., m_{\text{E\_asym}}\}$ eine Menge asymmetrischer Verschlüsselungsverfahren, die Fail-Safe-Konzept-geeignet sind. Seien enc_sym^{i_1} und enc_sym^{i_2} zwei symmetrische Verschlüsselungsverfahren aus E_sym, die voneinander unabhängig sind – mit zugehörigen verschiedenen geheimen Schlüsseln sK^{i_1} und sK^{i_2}. Seien enc_asym^{i_1} und enc_asym^{i_2} zwei asymmetrische Verfahren aus E_asym, die voneinander unabhängig sind, und seien $\text{puK}^{i_1}_{\text{Empfänger}}$ und $\text{puK}^{i_2}_{\text{Empfänger}}$ zwei zugehörige verschiedene öffentliche Schlüssel des Empfängers. Aus einem Klartext P wird eine **iterative Verschlüsselung** wie folgt gebildet:

Bei einem symmetrischen Verschlüsselungsverfahren:

$$C_0 = P$$
$$C_1 = \text{enc\_sym}^{i_1}(C_0, \text{sK}^{i_1})$$
$$C_2 = \text{enc\_sym}^{i_2}(C_1, \text{sK}^{i_2})$$
C_2 ist die iterative Verschlüsselung von P

Bei einem asymmetrischen Verschlüsselungsverfahren:

$$C_0 = P$$
$$C_1 = \text{enc\_asym}^{i_1}(C_0, \text{puK}^{i_1}_{\text{Empfänger}})$$
$$C_2 = \text{enc\_asym}^{i_2}(C_1, \text{puK}^{i_2}_{\text{Empfänger}})$$
C_2 ist die iterative Verschlüsselung von P

Bei der **iterativen Entschlüsselung** beim Empfänger wird folgendermaßen vorgegangen:

Bei einem symmetrischen Verfahren:

C_2 ist die iterative Verschlüsselung von P
$$C_1 = \text{dec\_sym}^{i_2}(C_2, \text{sK}^{i_2})$$
$$C_0 = \text{dec\_sym}^{i_1}(C_1, \text{sK}^{i_1})$$
$$P = C_0$$

Bei einem asymmetrischen Verfahren:

$$
\begin{aligned}
&C_2 \text{ ist die iterative Verschlüsselung von } P\\
&C_1 = \text{dec\_asym}^{i_2}(C_2, \text{prK}^{i_2}_{\text{Empfänger}})\\
&C_0 = \text{dec\_asym}^{i_1}(C_1, \text{prK}^{i_1}_{\text{Empfänger}})\\
&P = C_0
\end{aligned}
$$

Eine iterative symmetrische Verschlüsselung bezeichnet Bruce Schneier in [Schn96, 15.7] als „Kaskadierung mehrerer Blockalgorithmen“.

Ver-/Entschlüsselungs-Modelle für iterative symmetrische und asymmetrische Verschlüsselungen sind die Folgenden:

Ver-/Entschlüsselungs-Modell für eine iterative symmetrische Verschlüsselung:

Fail-Safe-Konzept-geeignete symm. Verschlüsselungsverf.	$\text{E\_sym} = \{\text{enc\_sym}^i \mid i = 1, ..., m_{\text{E\_sym}}\}$
Nutzende symm. Verschl'verf.	$\{\text{enc\_sym}^{i_1}, \text{enc\_sym}^{i_2}\}$, voneinander unabhängig
Geheime Schlüssel	$\{\text{sK}^{i_1}, \text{sK}^{i_2}\}$
Klartext	$C_0 = P$
Iterativ verschlüsseln	$C_k = \text{enc\_sym}^{i_k}(C_{k-1}, \text{sK}^{i_k})$ für $k = 1, 2$
Übertragen	C_2
Entschlüsseln	$C_{2-k-1} = \text{dec\_sym}^{i_{2-k}}(C_{2-k}, \text{sK}^{i_{2-k}})$ für $k = 0, 1$
Klartext	$P = C_0$

Ver-/Entschlüsselungs-Modell für eine iterative asymmetrische Verschlüsselung:

Fail-Safe-Konzept-geeignete asymm. Verschlüsselungsverf.	$\text{E\_asym} = \{\text{enc\_asym}^j \mid j = 1, ..., m_{\text{E\_asym}}\}$
Nutzende asymm. Verschl'verf.	$\{\text{enc\_asym}^{j_1}, \text{enc\_asym}^{j_2}\}$, voneinander unabhängig
Private Schlüssel	$\{\text{prK}^{j_1}_{\text{Empfänger}}, \text{prK}^{j_2}_{\text{Empfänger}}\}$
Öffentliche Schlüssel	$\{\text{puK}^{j_1}_{\text{Empfänger}}, \text{puK}^{j_2}_{\text{Empfänger}}\}$
Klartext	$C_0 = P$
Iterativ verschlüsseln	$C_k = \text{enc\_asym}^{j_k}(C_{k-1}, \text{puK}^{j_k}_{\text{Empfänger}})$ für $k = 1, 2$
Übertragen	C_2
Entschlüsseln	$C_{2-k-1} = \text{dec\_asym}^{j_{2-k}}(C_{2-k}, \text{prK}^{j_{2-k}}_{\text{Empfänger}})$ für $k = 0, 1$
Klartext	$P = C_0$

Bei einer hybriden Verschlüsselung, bei der die eigentlich zu verschlüsselnde Nachricht mit einem symmetrischen Verfahren und der benutzte Session Key mit einem asymmetrischen Verfahren und dem Public Key des Empfängers verschlüsselt wird, kann sich das mehrfache Ausführen auf den symmetrischen Teil, den asymmetrischen Teil oder beide Teile beziehen. Die drei Möglichkeiten sind in den folgenden Modellen aufgeführt.

Ver-/Entschlüsselungs-Modell für eine iterative hybride Verschlüsselung: iterativ symmetrisch und iterativ asymmetrisch:

Fail-Safe-Konzept-geeignete symm. Verschlüsselungsverf.	$\text{E\_sym} = \{\text{enc\_sym}^i \mid i = 1, ..., m_{\text{E\_sym}}\}$

Nutzende symm. Verschl'verf.		$\{\text{enc\_sym}^{i_1}, \text{enc\_sym}^{i_2}\}$, voneinander unabhängig
Session Keys		$\{\text{sK}^{i_1}, \text{sK}^{i_2}\}$
Fail-Safe-Konzept-geeignete asymm. Verschlüsselungsverf.		$\text{E\_asym} = \{\text{enc\_asym}^{j} \mid j = 1, ..., m_{\text{E\_asym}}\}$
Nutzende asymm. Verschl'verf.		$\{\text{enc\_asym}^{j_1}, \text{enc\_asym}^{j_2}\}$, voneinander unabhängig
Private Schlüssel		$\{\text{prK}^{j_1}_{\text{Empfänger}}, \text{prK}^{j_2}_{\text{Empfänger}}\}$
Öffentliche Schlüssel		$\{\text{puK}^{j_1}_{\text{Empfänger}}, \text{puK}^{j_2}_{\text{Empfänger}}\}$
1.	Klartext	$C_0 = P$
	Session Key	sK^{i_1}
	Symm. verschlüsseln	$C'_1 = \text{enc\_sym}^{i_1}(C_0, \text{sK}^{i_1})$
	Asymm. verschlüsseln	$D_1 = \text{enc\_asym}^{j_1}(\text{sK}^{i_1}, \text{puK}^{j_1}_{\text{Empfänger}})$
	Kryptogramm	(C'_1, D_1)
2.	„Klartext“	$C_1 = (C'_1, D_1)$
	Session Key	sK^{i_2}
	Symm. verschlüsseln	$C'_2 = \text{enc\_sym}^{i_2}(C_1, \text{sK}^{i_2})$
	Asymm. verschlüsseln	$D_2 = \text{enc\_asym}^{j_2}(\text{sK}^{i_2}, \text{puK}^{j_2}_{\text{Empfänger}})$
	Kryptogramm	$C_2 = (C'_2, D_2)$
Übertragen wird		C_2
Iterativ entschlüsseln		
1.	Kryptogramm	$C_2 = (C'_2, D_2)$
	Asymm. entschlüsseln	$\text{sK}^{i_2} = \text{dec\_asym}^{j_2}(D_2, \text{prK}^{j_2}_{\text{Empfänger}})$
	Session Key ermittelt	sK^{i_2}
	Symm. entschlüsseln	$C_1 = \text{dec\_sym}^{i_2}(C'_2, \text{sK}^{i_2})$
	Entschlüsselt wurde	C_1
2.	Kryptogramm	$C_1 = (C'_1, D_1)$
	Asymm. entschlüsseln	$\text{sK}^{i_1} = \text{dec\_asym}^{j_1}(D_1, \text{prK}^{j_1}_{\text{Empfänger}})$
	Session Key ermittelt	sK^{i_1}
	Symm. entschlüsseln	$C_0 = \text{dec\_sym}^{i_1}(C'_1, \text{sK}^{i_1})$
	Entschlüsselt wurde	C_0
Entschlüsselt wurde		$P = C_0$

Ver-/Entschlüsselungs-Modell für eine iterative hybride Verschlüsselung: iterativ symmetrisch und einfach asymmetrisch:

Fail-Safe-Konzept-geeignete symm. Verschlüsselungsverf.	$\text{E\_sym} = \{\text{enc\_sym}^{i} \mid i = 1, ..., m_{\text{E\_sym}}\}$
Nutzende symm. Verschl'verf.	$\{\text{enc\_sym}^{i_1}, \text{enc\_sym}^{i_2}\}$, voneinander unabhängig
Session Keys	$\{\text{sK}^{i_1}, \text{sK}^{i_2}\}$
Fail-Safe-Konzept-geeignete asymm. Verschlüsselungsverf.	$\text{E\_asym} = \{\text{enc\_asym}^{j} \mid j = 1, ..., m_{\text{E\_asym}}\}$
Nutzendes asymm. Verschl'verf.	enc_asym^{j_1}

Privater Schlüssel	$\mathrm{prK}^{j_1}_{\text{Empfänger}}$
Öffentlicher Schlüssel	$\mathrm{puK}^{j_1}_{\text{Empfänger}}$
Klartext	P
Verschlüsselung von P	$C_2 = \text{enc\_sym}^{i_2}(\text{enc\_sym}^{i_1}(P, \mathrm{sK}^{i_1}), \mathrm{sK}^{i_2})$
Verschlüsselung von $(\mathrm{sK}^{i_1}, \mathrm{sK}^{i_2})$	$D_1 = \text{enc\_asym}^{j_1}((\mathrm{sK}^{i_1}, \mathrm{sK}^{i_2}), \mathrm{puK}^{j_1}_{\text{Empfänger}})$
Übertragen wird	(C_2, D_1)
Entschlüsselung von D_1	$(\mathrm{sK}^{i_1}, \mathrm{sK}^{i_2}) = \text{dec\_asym}^{j_1}(D_1, \mathrm{prK}^{j_1}_{\text{Empfänger}})$
Entschlüsselung von C_2	$P = \text{dec\_sym}^{i_1}(\text{dec\_sym}^{i_2}(C_2, \mathrm{sK}^{i_2}), \mathrm{sK}^{i_1})$

Ver-/Entschlüsselungs-Modell für eine iterative hybride Verschlüsselung: einfach symmetrisch und iterativ asymmetrisch:

Fail-Safe-Konzept-geeignete symm. Verschlüsselungsverf.	$\text{E\_sym} = \{\text{enc\_sym}^i \mid i = 1, ..., m_{\text{E\_sym}}\}$
Nutzendes symm. Verschl'verf.	enc_sym^{i_1}
Session Key	sK^{i_1}
Fail-Safe-Konzept-geeignete asymm. Verschlüsselungsverf.	$\text{E\_asym} = \{\text{enc\_asym}^j \mid j = 1, ..., m_{\text{E\_asym}}\}$
Nutzende asymm. Verschl'verf.	$\{\text{enc\_asym}^{j_1}, \text{enc\_asym}^{j_2}\}$, voneinander unabhängig
Private Schlüssel	$\{\mathrm{prK}^{j_1}_{\text{Empfänger}}, \mathrm{prK}^{j_2}_{\text{Empfänger}}\}$
Öffentliche Schlüssel	$\{\mathrm{puK}^{j_1}_{\text{Empfänger}}, \mathrm{puK}^{j_2}_{\text{Empfänger}}\}$
Klartext	P
Verschlüsselung von P	$C_1 = \text{enc\_sym}^{i_1}(P, \mathrm{sK}^{i_1})$
Verschlüsselung von sK^{i_1}	$D_2 = \text{enc\_asym}^{j_2}(\text{enc\_asym}^{j_1}(\mathrm{sK}^{i_1}, \mathrm{puK}^{j_1}_{\text{Empfänger}}), \mathrm{puK}^{j_2}_{\text{Empf.}})$
Übertragen wird	(C_1, D_2)
Entschlüsseln von D_2	$\mathrm{sK}^{i_1} = \text{dec\_asym}^{j_1}(\text{dec\_asym}^{j_2}(D_2, \mathrm{prK}^{j_2}_{\text{Empfänger}}), \mathrm{prK}^{j_1}_{\text{Empf.}})$
Entschlüsseln von C_1	$P = \text{dec\_sym}^{i_1}(C_1, \mathrm{sK}^{i_1})$

Anwendungsbeispiele von einfachen digitalen Signaturen und einfachen Verschlüsselungen finden sich in Abschnitt 4.3 auf Seite 98 und Beispiele für den Gebrauch der multiplen Techniken in Abschnitt 4.4 auf Seite 103. Zunächst wird der Aufbau der Fail-Safe-PKI beschrieben.

4.2 Aufbau der Fail-Safe-PKI

Nachdem geklärt ist, wann zwei Algorithmen voneinander unabhängig sind und wie zwei kryptographische Verfahren in digitalen Signaturen und Verschlüsselungen im Fail-Safe-Konzept genutzt werden können, wird in diesem Abschnitt der Frage nachgegangen, welche kryptographischen und einsatzspezifischen Komponenten in der Fail-Safe-PKI vorhanden sind und wie diese besondere Public-Key-Infrastruktur mit ihren Teil-PKIs aufgebaut ist.

Nach Definition 14 auf Seite 20 besteht eine Public-Key-Infrastruktur aus einer Menge von Hard- und Software, Beteiligten, Policies und Verfahren, um auf Public-Key-Kryptographie basierende Zertifikate zu erstellen, zu handhaben, zu verwalten und zu revozieren. Die Fail-Safe-PKI verfügt darüber hinaus über weitere Eigenschaften und Möglichkeiten:

Definition 24
Eine **Fail-Safe-PKI** ist eine PKI nach Definition 14 auf Seite 20 mit zusätzlicher Software, einer weiteren Beteiligten, ergänzten Policies und zusätzlichen Fail-Safe-Konzept-geeigneten Verfahren, um multiple digitale Signaturen erzeugen und verifizieren, iterative Verschlüsselungen bearbeiten, erweiterte Revokationsmechanismen managen und kryptographische und einsatzspezifische Komponenten sicher und im laufenden Betrieb aktualisieren zu können.

Die Fail-Safe-PKI wird anhand einer einstufigen Hierachie dargestellt, die aus einer Certification Authority (CA), welche auch die RootCA darstellt, und von ihr zertifizierten Certificate Holdern (CH) besteht (Abb. 4.1). Dieses Modell lässt sich auf beliebige PKI-Hierachien verallgemeinern – wie in Abschnitt 4.6 auf Seite 153 diskutiert.

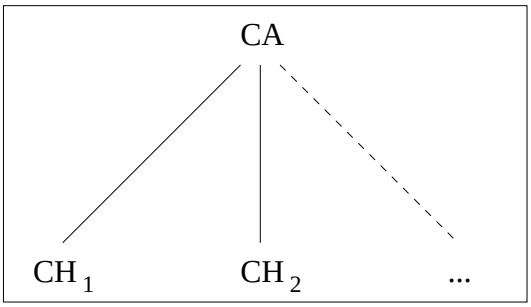

Abbildung 4.1: Fail-Safe-PKI als einstufige Hierachie

Das Trust Center der Fail-Safe-PKI vereinigt neben den üblichen Trust-Center-Dienstleistungen Zertifizierung (CA), Registrierung (RA), Revokationsdienst (CRL und OCSP), Verzeichnisservice (DIR) und Zeitstempelung (TSA) einen neu gegründeten Update Service, der den Update von kryptographischen und einsatzspezifischen Komponenten bei den CHs initiiert und als Ansprechpartner zur Verfügung steht. Exakte Aufgaben und Funktionalitäten des Update Service sind Thema in Abschnitt 4.5 auf Seite 107. In der Registration Authority der Fail-Safe-PKI kann ein „werdender“ Zertifikatsinhaber beantragen, dass er an der Fail-Safe-PKI teilnehmen möchte, worauf hin die Certification Authority entsprechende Schlüsselpaare in der PSE erzeugt und Zertifikate für den CH generiert, in denen je ein öffentlicher Schlüssel zertifiziert ist. Die Richtlinien (Policies) legen fest, wie ein Trust Center arbeitet. Insofern muss die Policy der Fail-Safe-PKI Informationen über das Fail-Safe-Konzept und die verwendeten Verfahren enthalten. Details zur Policy sind in Appendix A.9 auf Seite 215 zu finden.

Es wird unterstellt, dass es durch organisatorische Maßnahmen in der Wurzelzertifizierungsinstanz nicht möglich ist, mehrere RootCA-Keys, die als Sicherheitsanker eine besondere Bedeutung haben, gleichzeitig zu kompromittieren.

Diese Fail-Safe-PKI mitsamt Update von Komponenten wird für eine sicherheitskritische Anwendung gestaltet. Deshalb wird angenommen, dass kryptographische und einsatzspezifische Komponenten in Client und Chipkarte derart abgesichert sind, dass ein Update nur nach Setzen eines entsprechenden Security Status durch das Update Management Protocol des Fail-Safe-Konzepts möglich ist.

Nun zu den in der Fail-Safe-PKI eingesetzten kryptographischen Verfahren.

In der Fail-Safe-PKI gibt es folgende Mengen von Signaturalgorithmen, Hashfunktionen, Formatierungen, Signaturverfahren und asymmetrischen und symmetrischen Verschlüsselungsalgorithmen und -verfahren:

- $\Sigma = \{\sigma^i \mid i = 1, ..., m_\Sigma\}, m_\Sigma \in \mathbb{N}, m_\Sigma \geq 2$
- $H = \{\kappa^i \mid i = 1, ..., m_H\}, m_H \in \mathbb{N}, m_H \geq 2$
- $F = \{\phi^i \mid i = 1, ..., m_F\}, m_F \in \mathbb{N}, m_F \geq 2$
- $S = \{\text{sign}^i \mid i = 1, ..., m_S\}, m_S \in \mathbb{N}, m_S \geq 2$
- $\xi\_\text{asym} = \{\eta\_\text{asym}^i \mid i = 1, ..., m_{\xi\_\text{asym}}\}, m_{\xi\_\text{asym}} \in \mathbb{N}$
- $\xi\_\text{sym} = \{\eta\_\text{sym}^i \mid i = 1, ..., m_{\xi\_\text{sym}}\}, m_{\xi\_\text{sym}} \in \mathbb{N}$
- $\text{E\_asym} = \{\text{enc\_asym}^i \mid i = 1, ..., m_{\text{E\_asym}}\}, m_{\text{E\_asym}} \in \mathbb{N}$
- $\text{E\_sym} = \{\text{enc\_sym}^i \mid i = 1, ..., m_{\text{E\_sym}}\}, m_{\text{E\_sym}} \in \mathbb{N}$

Bei Anwendungen, die keine Verschlüsselung nutzen, sind Verschlüsselungsverfahren nicht nötig und deshalb wurde bei diesen Algorithmen und Verfahren auf die Forderung einer Mindestanzahl verzichtet. Anwendungen mit Verschlüsselungen müssen im Sinne des Fail-Safe-Gedankens allerdings für iterative Verschlüsselungen mehrere Algorithmen und Verfahren anbieten. Da Signaturverfahren stets für Zertifizierungen benötigt werden, sind in jeder Fail-Safe-PKI mindestens zwei Signaturverfahren eingesetzt.

Aus dieser Menge kryptographischer Komponenten werden Teil-PKIs gebildet. Es gibt insgesamt p Teil-PKIs

$$P = \{\text{PKI}^1, ..., \text{PKI}^r, \text{PKI}^{r+1}, ..., \text{PKI}^p\},$$

wobei $p \geq r \geq 2$ gilt.

Was sind die Größen r und p?

Zunächst zu den ersten r Teil-PKIs PKI^1, ..., PKI^r. Jede Teil-PKI PKI^i beinhaltet für $i = 1, ..., r$

- ein mathematisches Basisproblem α^i,
- einen daraus abgeleiteten Signaturalgorithmus σ^i,
- ein Signaturverfahren sign^i, welches insbesondere den Signaturalgorithmus σ^i neben Hashfunktion κ^i und Formatierung[3] ϕ^i enthält,
- Schlüsselpaare $(\text{prK}^i, \text{puK}^i)$ zu σ^i,
- Zertifikate cert^i, in dem puK^i veröffentlicht ist und das von der CA mit dem Signaturverfahren sign^i digital signiert wurde,
- Sicherheitsanker $\text{cert}^i_{\text{CA}}$ mit dem öffentlichen Schlüssel puK^i_{CA} der CA,
- mögliche weitere Hashfunktionen κ,
- mögliche weitere Formatierungen ϕ und
- mögliche weitere Signaturverfahren sign, die aus σ^i und weiteren Hashfunktionen und Formatierungen kombiniert werden.

[3] Da es mehrere Signaturverfahren geben kann, für die keine Formatierung notwendig ist, kann dieses ϕ^i für die identische Abbildung id mehrmals auch in anderen Teil-PKIs auftreten.

Zwei verschiedene Teil-PKIs PKI^i und PKI^j, $i, j \in \{1, ..., r\}$, unterscheiden sich aufgrund der unterschiedlichen mathematischen Basisprobleme in den Signaturalgorithmen σ^i und σ^j und den für Zertifizierungen genutzten Signaturverfahren sign^i und sign^j. Dennoch ist nicht auszuschließen, dass es in den beiden Teil-PKIs Gemeinsamkeiten in den kryptographischen Komponenten gibt, z. B. eine Hashfunktion oder Formatierung, die in zwei verschiedenen Signaturverfahren vorkommt.

Bzgl. Verschlüsselungen kann eine Teil-PKI PKI^i für $i = 1, ..., r$ über

- ein mathematisches Basisproblem α^i,
- einen daraus abgeleiteten asymmetrischen Verschlüsselungsalgorithmus η_asym^i,
- ein asymmetrisches Verschlüsselungsverfahren enc_asym^i, welches insbesondere den asymmetrischen Verschlüsselungsalgorithmus η_asym^i neben Formatierung ϕ^i enthält,
- Schlüsselpaare $(\text{prK}'^i, \text{puK}'^i)$ zu η_asym^i,
- Zertifikate cert'^i, in dem puK'^i veröffentlicht ist und das von der CA mit dem Signaturverfahren sign^i digital signiert wurde, wobei sign^i den Signaturalgorithmus σ^i verwendet, der auf dem gleichen mathematischen Basisproblem wie η_asym^i basiert,
- Sicherheitsanker $\text{cert}^i_{\text{CA}}$ mit dem öffentlichen Schlüssel der CA puK^i_{CA},
- weitere asymmetrische Verschlüsselungsverfahren enc_asym, die aus η_asym^i und anderen Formatierungen kombiniert werden,
- symmetrische Verschlüsselungsalgorithmen η_sym und
- symmetrische Verschlüsselungsverfahren enc_sym

verfügen.

Nun zu den optionalen weiteren Teil-PKIs PKI^{r+1}, ..., PKI^p, $p \geq r$.

Es kann aus Interoperabilitätsgründen notwendig sein, weitere Teil-PKIs einzurichten, die allerdings nicht unabhängig sind, sondern mit mindestens einer der ersten r Teil-PKIs korrelieren. Das ist z. B. dann der Fall, wenn weitere mathematische Basisprobleme genutzt werden, die zu einem mathematischen Problem Korrelationen aufweisen, welches in einer der ersten r Teil-PKIs Anwendung findet. Beispielsweise kann eine weitere Teil-PKI das DL-Problem in endlichen Körpern mit dem Signaturalgorithmus DSA kryptographisch ausnutzen, obwohl dieses mathematische Problem zum Faktorisierungsproblem und damit zu RSA korreliert. Es ist zu beachten, dass diese weiteren $p - r$ Teil-PKIs unter Fail-Safe-Gesichtspunkten keinen Sinn machen, weil auch sie im Schadensfall einer der ersten r Teil-PKIs mit betroffen wären und keinen Mehrwert bieten würden.

Beispiel von Kryptoverfahren in einer Fail-Safe-PKI

Die Fail-Safe-PKI in diesem Beispiel besteht für $p = r = 2$ aus zwei Teil-PKIs PKI^1 und PKI^2, in denen einmal das Faktorisierungs- und einmal das DL-Problem auf elliptischen Kurven als mathematische Basisprobleme genutzt werden. Durch zwei Signaturalgorithmen (RSA und ECDSA), zwei Hashfunktionen (RIPEMD-160 und SHA-1) und eine nur bei RSA notwendige Formatierung (PKCS#1 und id bei ECDSA) ergeben sich vier Signaturverfahren. Mit den offiziellen Distinguished Names der Object Identifier sind PKI^1 und PKI^2 folgendermaßen ausgestattet:

PKI^1 :

- Basisproblem $\alpha^1 \widehat{=}$ Faktorisierungsproblem
- Signaturalgorithmus $\sigma^1 \widehat{=}$ RSA
- Signaturverfahren $sign^1 \widehat{=}$ rsasignatureWithripemd160 (RSA mit RIPEMD-160 und PKCS#1)
- Schlüsselpaar (prK^1, puK^1) aus RSA-Schlüsseln
- Zertifikat $cert^1$, welches puK^1 zu σ^1 zertifiziert; Signatur mit $sign^1 \widehat{=}$ rsasignatureWithripemd160 (mit PKCS#1) erzeugt
- Sicherheitsanker $cert^1_{CA}$ mit dem öffentlichen Schlüssel der CA puK^1_{CA}
- Weiteres Signaturverfahren $sign^3 \widehat{=}$ sha1WithRSAENC (RSA mit SHA-1 und PKCS#1)
- asymmetrischer Verschlüsselungsalgorithmus $\eta\_asym^1 \widehat{=}$ id-ea-rsa (RSA Encryption)
- Schlüsselpaar (prK'^1, puK'^1) aus RSA-Schlüsseln
- Zertifikat $cert'^1$, welches puK'^1 zu η_asym^1 zertifiziert; Signatur mit $sign^1 \widehat{=}$ rsasignaturesWithripemd160 (mit PKCS#1) erzeugt
- Asymmetrisches Verschlüsselungsverfahren $enc\_asym^1 \widehat{=}$ RSA encryption (mit PKCS#1-Randomisierung)
- Symmetrisches Verschlüselungsverfahren $enc\_sym^1 \widehat{=}$ DES_3CBC_ISOpad (Triple-DES im CBC-Modus mit ISO-Padding)

PKI^2 :

- Basisproblem $\alpha^2 \widehat{=}$ DL-Problem auf elliptischen Kurven
- Signaturalgorithmus $\sigma^2 \widehat{=}$ ECDSA
- Signaturverfahren $sign^2 \widehat{=}$ ecdsa-with-SHA1 (ECDSA mit SHA-1)
- Schlüsselpaar (prK^2, puK^2) aus ECDSA-Schlüsseln
- Zertifikat $cert^2$, welches puK^2 zu σ^2 zertifiziert; Signatur mit $sign^2 \widehat{=}$ ecdsa-with-SHA1 (ECDSA mit SHA-1) erzeugt
- Sicherheitsanker $cert^2_{CA}$ mit dem öffentlichen Schlüssel der CA puK^2_{CA}
- Weiteres Signaturverfahren $sign^4 \widehat{=}$ ecSignWithripemd160 (ECDSA mit RIPEMD-160)
- Asymmetrischer Verschlüsselungsalgorithmus $\eta\_asym^2 \widehat{=}$ ECIES[4]
- Schlüsselpaar (prK'^2, puK'^2) aus ECIES-Schlüsseln
- Zertifikat $cert'^2$, welches puK'^2 zu η_asym^2 zertifiziert; Signatur mit $sign^2 \widehat{=}$ ecdsa-with-SHA1 erzeugt

[4]Eine Randomisierung kann mit der im Encryption Scheme enthaltenen symmetrischen Verschlüsselung erfolgen.

- Asymmetrisches Verschlüsselungsverfahren enc_asym$^2\widehat{=}$ ECIES
- Symmetrisches Verschlüsselungsverfahren enc_sym$^2\widehat{=}$ id-aes256-ECB (AES mit 256 Bit Schlüsseln im ECB-Modus)

In der Fail-Safe-PKI verfügen Certification Authority und Certificate Holder über sämtliche kryptographische (Signaturalgorithmus und -verfahren, Hashfunktion, Formatierung, Verschlüsselungsalgorithmus und -verfahren) und einsatzspezifische Komponenten (Schlüssel) der Teil-PKIs. Die CHs halten darüber hinaus den öffentlichen Schlüssel der CA als Sicherheitsanker vor und die CA pflegt das Verzeichnis (DIR), welches die gültigen Zertifikate

$$\{\text{cert}^i_{\text{CA}}, \text{cert}^i_{\text{CH}} \,|\, \text{alle CHs, alle } i = 1, ..., p\},$$

herkömmliche Sperrlisten

$$\text{CRL}^i, \text{ für } i \in \{1, ..., p\},$$

sowie multipel signierte Sperrlisten

$$\{\text{CRL}^{i,j} \,|\, \text{alle}\, i, j \in \{1, ..., p\},\ \text{so dass sign}^i \text{ und sign}^j \text{ voneinander unabhängig}\},$$

die in Abschnitt 4.4.3 auf Seite 104 detailliert beschrieben werden, führt.

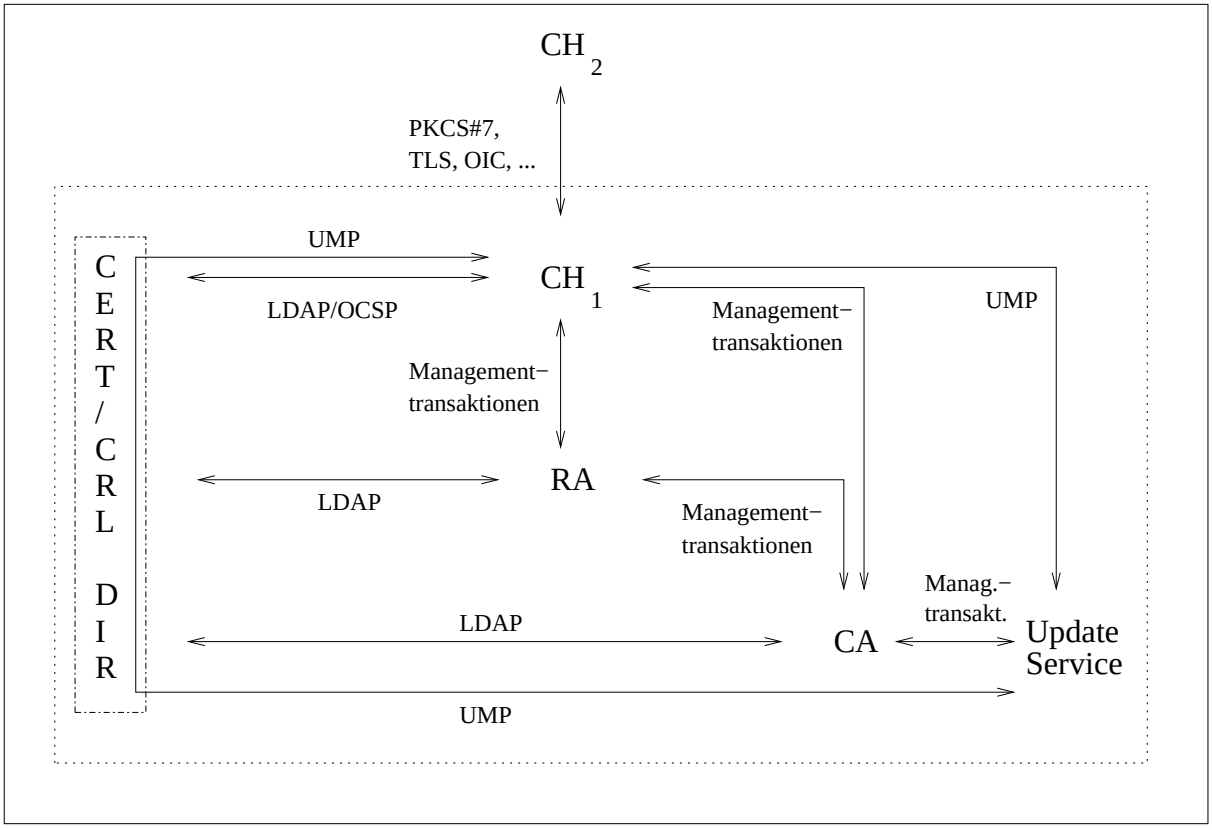

Abbildung 4.2: Fail-Safe-PKI-Struktur

Abbildung 4.2 zeigt eine Darstellung der Fail-Safe-PKI, die sich von der in Abschnitt 1.2 auf Seite 20 dargestellten PKI in der Existenz des Update Services sowie der nötigen Managementstrukturen von Update Service zu CA und CH unterscheidet. Diese Managementstrukturen verlaufen

über das neue Protokoll namens Update Management Protocol (UMP), das samt Transportwegen in Abschnitt 4.5 auf Seite 107 detailliert diskutiert wird.

Welche Funktionalitäten bietet eine Fail-Safe-PKI? Zunächst einmal solche, die auch in herkömmlichen PKIs vorkommen, wie einfache digitale Signaturen und einfache Verschlüsselungen, dann die multiplen Techniken und schließlich – als Highlight – die Möglichkeit, kryptographische und einsatzspezifische Komponenten im laufenden Betrieb der PKI sicher auszutauschen. Die folgenden drei Abschnitte geben über diese Funktionalitäten im Detail Auskunft.

4.3 Normale Funktionalität

Einfache digitale Signaturen und einfache Verschlüsselungen finden in einer Fail-Safe-PKI nach wie vor Verwendung, z. B. in Zertifikaten, Sperrlisten, Zertifikats-Status-Anfragen, Zeitstempeln, Policies, Dokumenten, E-Mails oder Authentisierungsprotokollen, um interoperabel zu CHs anderer PKIs zu sein und weil nicht jede Anwendung jede Komponente des Fail-Safe-Konzepts nutzen muss. Dieser Abschnitt diskutiert diese Anwendungen.

Im Sinne des Signatur/Verifikations-Modells (von Seite 86) und der Ver-/Entschlüsselungs-Modelle (ab Seite 87) werden in der Fail-Safe-PKI folgende Techniken zur Verfügung gestellt:

- Zertifikate

  In einem Zertifikat $\text{cert}^i_{\text{CH}}$, $i = 1, ..., p$, bestätigt die CA einen Zertitifikatsinhalt D. Ein Zertifikat ist mit einer einfachen digitalen Signatur versehen. D variiert in Abhängigkeit der Art des Zertifikats:

  1. In Signatur-Anwendungen zur digitalen Signatur enthält D:
     (a) *Version* – Versionsnummer
     (b) *Serial Number* – Seriennummer
     (c) *Signature Algorithm* – Signaturverfahren sign^i, das die CA für dieses Zertifikat benutzte
     (d) *Issuer* – Name der CA
     (e) *Validity* – Gültigkeitszeitraum
     (f) *Subject* – Name des CHs
     (g) *Subject Public Key Info* – Öffentlicher Schlüssel $\text{puK}^i_{\text{Zertifikatsinhaber}}$ zu Signaturalgorithmus σ^i; zum Signieren können verschiedene Signaturverfahren genutzt werden, die σ^i enthalten
     (h) *Extensions* – Verwendungszweck (Key Usage) des Schlüssels: "Non-Repudiation" für Nicht-Abstreitbarkeit, optional
     (i) *Signature Algorithm* – Signaturverfahren sign^i, mit dem dieses Zertifikat von der CA signiert wurde; sign^i enthält σ^i
     (j) *Signature Value* – Digitale Signatur
  2. In Signatur-Anwendungen zum Verschlüsseln enthält D abweichend zu 1:
     In 1g wird der Verschlüsselungsalgorithmus η_asym^i und der öffentlicher Schlüssel $\text{puK}^{\eta i}_{\text{Zertifikatsinhaber}}$ dazu angegeben.
     In 1h wird als Key Usage "Key Encipherment" für Schlüsseltransport vermerkt. σ^i und η_asym^i basieren auf demselben mathematischen Problem.

3. In Authentisierungs-Anwendungen enthält D abweichend zu 1:
   In 1h wird als Key Usage “Digital Signature“ vermerkt, was die Verwendung in Authentisierungsprotokollen vorsieht.
4. In Zertifikaten, in denen Schlüssel der CA zur Verifikation von Zertifikaten signiert werden – also die Sicherheitsanker –, enthält D abweichend zu 1:
   In 1h wird als Key Usage “Key-Cert-Sign“ vermerkt.
5. In Zertifikaten, in denen Schlüssel der CA zur Verifikation von CRL-Sperrlisten signiert werden, enthält D abweichend zu 1:
   In 1h wird als Key Usage “CRL-Sign“ vermerkt.
6. In Zertifikaten, in denen Schlüssel der CA zur Verifikation von OCSP-Antworten signiert werden, enthält D abweichend zu 1:
   In 1h wird als Extended Key Usage “OCSP-Sign“ vermerkt.

Das Signatur-/Verifikationsmodell für Zertifikate ist das Folgende:

Signaturverfahren	sign^i für $i \in \{1, ..., p\}$
Schlüssel	prK^i_{CA} und puK^i_{CA} zu Key Usage “Key-Cert-Sign“
Input D	Zertifikats-Inhalt für einen CH mit einem öffentlichen Schlüssel puK^i_{CH} zum Signaturalgorithmus σ^i und weiteren Daten
Signierer	CA
S^*	$\text{cert}^i_{\text{CH}}$
Verifizieren	$\text{verify}^i(S^*, \text{puK}^i_{\text{CA}})$

cert^i ist mit sign^i und prK^i_{CA} signiert, so dass ein solches Zertifikat mit dem Trust Anchor puK^i_{CA} aus dem Zertifikat $\text{cert}^i_{\text{CA}}$ verifiziert werden kann. Es ist als Signaturverfahren für Zertifizierungen nur sign^i zulässig, $i \in \{1, ..., p\}$. Zertifikate gibt es in jeder der p Teil-PKIs. An die Gültigkeitszeiträume der Zertifikate in verschiedenen Teil-PKIs werden keine Forderungen gestellt: Sie können – gerade am Anfang – identisch sein, müssen aber nicht.

Ein Zertifikat ist kein Beispiel für eine multiple Signatur. Die Verfahren, die die CA zur Erzeugung eines Zertifikats nutzt und die im Zertifikat zur Benutzung aufgeführt sind, basieren auf demselben mathematischen Problem, d. h. sign^i zertifiziert Schlüssel zu σ^i oder η_asym^i und diese Algorithmen nutzen dieselben mathematischen Basisprobleme wie das Signaturverfahren. Intention ist die strikte Trennung der Teil-PKIs bzgl. der für Zertifizierung benutzten Signaturverfahren und der für ein Signaturalgorithmus bestimmten Schlüssel.

Ein de facto Standard für Zertifikate ist X.509, der in Appendix A.1 auf Seite 187 beschrieben ist.

- Sperrliste

  In einer Sperrliste sind alle von der CA revozierten Zertifikatsnummern vermerkt.

Signaturverfahren	sign^i für $i \in \{1, ..., m_S\}$
Schlüssel	prK^i_{CA} und puK^i_{CA} zu Key Usage “CRL-Sign“
Input D	Menge revozierter Zertifikatsnummern
Signierer	CA
S^*	Sperrliste CRL^i
Verifizieren	$\text{verify}^i(S^*, \text{puK}^i_{\text{CA}})$

Zur Absicherung einer Sperrliste kann ein beliebiges der m_S Signaturverfahren genutzt werden. Ein de facto Standard für eine Sperrliste ist Certificate Revocation List (CRL), der in Appendix A.3 auf Seite 192 beschrieben ist.

- Zertifikats-Status-Anfrage und -Antwort

  Der gegenwärtige Status eines Zertifikats (gültig oder revoziert?) kann bei einem Server erfragt werden, wobei die Anfrage optional signiert sein kann:

Signaturverfahren	sign^i für $i \in \{1, ..., m_S\}$, optional zu nutzen
Schlüssel	prK^i_{CH} und puK^i_{CH}, optional zu nutzen
Input D	„Ist Zertifikat mit Nummer n gültig?“
Signierer/Requester	CH, optional zu signieren
S^*	Zertifikats-Status-Anfrage, optional signiert
Verifizieren	$\text{verify}^i(S^*, \text{puK}^i_{\text{CH}})$, optional

  Die Antwort auf eine Zertifikats-Status-Anfrage ist die Folgende:

Signaturverfahren	sign^i für $i \in \{1, ..., m_S\}$
Schlüssel	prK^i_{CA} und puK^i_{CA} zu Key Usage “OCSP-Sign“
Input D	„Zertifikat mit Nummer n ist gültig, ungültig oder unbekannt“, wenn die Antwort erfolgreich war; andernfalls wird eine Fehlermeldung übermittelt
Signierer	CA, falls die CA diesen Dienst anbietet, sonst ein Dienstleister
S^*	Zertifikats-Status-Antwort
Verifizieren	$\text{verify}^i(S^*, \text{puK}^i_{\text{CA}})$

  Ein de facto Standard ist das Online Certificate Status Protocols (OCSP), das die beiden Teile OCSP Request und OCSP Response anbietet. Appendix A.4 auf Seite 194 stellt OCSP dar.

- Zeitstempel

  Für die Beweisbarkeit digitaler Signaturen kann der Erstellungszeitpunkt entscheidend sein. Dazu gibt es Zeitstempel, mit dem eine vertrauenswürdige Instanz (Time Stamping Authority – TSA) bezeugt, das ihr ein Dokument bzw. dessen Hashwert zu einem Zeitpunkt vorgelegen hat.

Signaturverfahren	sign^i für $i \in \{1, ..., m_S\}$
Schlüssel	$\text{prK}^i_{\text{TSA}}$ und $\text{puK}^i_{\text{TSA}}$ zu Key Usage “Non-Repudiation“
Input von Benutzer an Zeitstempelinstanz	Hashwert $h = \kappa^j(D^*)$ eines Dokuments D^*, mit einer Hashfunktion κ^j generiert
Input D	$h \mid$ Zeit t
Signierer	TSA
S^*	Zeitstempel
Aufzubewahren	Dokument D^* und Zeitstempel
Verifizieren	$\text{verify}^i(S^*, \text{puK}^i_{\text{TSA}})$
Gehört S^* zu D^*?	Prüfen, ob $\kappa^j(D^*) = h$ für das h aus Zeitstempel S^* gilt.

Das Time Stamp Protocol (TSP) ist in Appendix A.5 auf Seite 200 im Detail beschrieben.

- Policy

  Die Policy einer Certification Authority beschreibt Regelungen ihrer Arbeitsweise.

Signaturverfahren	sign^i für $i \in \{1, ..., m_S\}$
Schlüssel	prK^i_{CA} und puK^i_{CA} zu Key Usage "Non-Repudiation"
Input D	Policy-Inhalt
Signierer	CA
S^*	signierte Policy
Verifizieren	$\text{verify}^i(S^*, \text{puK}^i_{\text{CA}})$

  Policies sind weder im Inhalt noch im Signaturformat standardisiert. Ein Vorschlag, was eine Policy enthalten sollte, ist in Appendix A.9 auf Seite 215 dargelegt.

- Dokument/E-Mail

  Dokumente können digital signiert und/oder mit einer hybriden Verschlüsselung gegen unbefugtes Mitlesen geschützt werden. Dasselbe gilt für eine beliebte Form der elektronischen Kommunikation – der E-Mail.

Signaturverfahren	sign^i für $i \in \{1, ..., m_S\}$
Signierer	CH_1
Schlüssel des Signierers	$\text{prK}^i_{\text{CH}_1}$ und $\text{puK}^i_{\text{CH}_1}$ zu Key Usage "Non-Repudiation"
Input D	Text
S^*	signiertes Dokument bzw. signierte E-Mail
Asymm. Verschl'verfahren	enc_asym^j für $j \in \{1, ..., m_{\text{E\_asym}}\}$
Empfänger	CH_2
Schlüssel des Empfängers	$\text{prK}^j_{\text{CH}_2}$ und $\text{puK}^j_{\text{CH}_2}$ zu Key Usage "Key Encipherment"
Klartext	S^*
Symm. Verschl'verfahren	enc_sym^k für $k \in \{1, ..., m_{\text{E\_sym}}\}$
Session Key	sK^k
Übertragen	mit symmetrischem Verfahren und sK^k verschlüsseltes S^* und mit asymmetrischem Verfahren und $\text{puK}^j_{\text{CH}_2}$ verschlüsselter Session Key sK^k
Entschlüsseln	mit asymmetrischem Verfahren und $\text{prK}^j_{\text{CH}_2}$ entschlüsselter Session Key, mit welchem mit dem symmetrischen Verfahren letztlich der Klartext S^* – das signierte Dokument resp. die signierte E-Mail – extrahiert wird
Verifizieren der Signatur	$\text{verify}^i(S^*, \text{puK}^i_{\text{CH}_1})$

  Ein de facto Standard, Dokumente zu signieren und/oder hybrid zu verschlüsseln und als E-Mail zu verschicken, ist PKCS#7 (Appendix A.6 auf Seite 206).

- Authentisierung

  In einer Authentisierung versucht sich ein Client gegenüber einem Server zu authentisieren. Der Client sendet einen Request – etwa ein "Hello!" – an den Server, der daraufhin eine Zufallszahl RND zurücksendet. Das Response des Clients verläuft wie folgt:

Signaturverfahren	sign^i für $i \in \{1, ..., m_S\}$
Signierer	Client
Schlüssel des Signierers	$\text{prK}^i_{\text{Client}}$ und $\text{puK}^i_{\text{Client}}$ zu Key Usage "Digital Signature"
Input D	RND
S^*	signiertes RND

  Im Authentisierungsprotokoll kann ein Session Key für die Verschlüsselung der nachfolgenden Kommunikation ausgehandelt werden. Dazu gibt es Schlüsselaustausch-Protokolle wie Diffie-Hellman. Oder es wird ein asymmetrisches Verschlüsselungsverfahren genutzt:

  Der Client generiert einen Schlüssel sK^j zufällig und sendet diesen – verschlüsselt mit einem asymmetrischen Verschlüsselungsverfahren und dem öffentlichen Schlüssel des Servers – zusammen mit der Challenge RND und signiert an den Server. Das Modell sieht wie folgt aus:

Asymm. Verschlüsselungsverf.	enc_asym^j für $j \in \{1, ..., m_{\text{E\_asym}}\}$
Öffentlicher Schlüssel	$\text{puK}^j_{\text{Server}}$ zu Key Usage "Key Encipherment"
Session Key	sK^i
Verschlüsseln	$C = \text{enc\_asym}^j(\text{sK}^i, \text{puK}^j_{\text{Server}})$
Input D	(C, RND)
Übertragen	S^*
Verifizieren	$\text{verify}^i(S^*, \text{puK}^i_{\text{Client}})$ und prüfen, ob RND aus S^* gleich gesendetem RND
Entschlüsseln	$\text{sK}^i = \text{dec\_asym}^j(C, \text{prK}^j_{\text{Server}})$

  Damit kann die anschließende Kommunikation mit einem symmetrischen Verschlüsselungsverfahren und dem Session Key sK^j effizient verschlüsselt werden – nach diesem Ver-/Entschlüsselungs-Modell:

Symm. Verschlüsselungsverf.	enc_sym^i für $i = \{1, ..., m_{\text{E\_sym}}\}$
Session Key	sK^i
Klartext	P
Iterativ verschlüsseln	$C = \text{enc\_sym}^i(P, \text{sK}^i)$
Übertragen	C
Entschlüsseln	$P = \text{dec\_sym}^i(C, \text{sK}^i)$

  Ein de facto Standard für Authentisierungen ist TLS/SSL (Appendix A.8 auf Seite 211).

Verifikation und Validierung erfolgen wie in Abschnitt 1.3.2 auf Seite 27 beschrieben und in Abhängigkeit des Sicherheitsniveaus der dedizierten Anwendung.

In der Fail-Safe-PKI wird das Schalenmodell (siehe Abschnitt 2 auf Seite 22) favorisiert, weil es dadurch ermöglicht wird, mehrere Zertifikate durch Revokation eines Zertifikats zu sperren,

weil dieses Zertifikat weit oberhalb im Verifikationspfad angesiedelt ist und sich seine Sperrung auf alle darunter liegenden Zertifikate auswirkt. Somit könnte mit einem einzelnen Eintrag eine Menge von Zertifikaten revoziert werden. Da das Signaturgesetz SigG abweichend das Kettenmodell vorschreibt, wird in der Fail-Safe-PKI das Schalenmodell nicht vorgeschrieben. Um dennoch große Sperrlisten zu vermeiden, wurde in Abschnitt 3.4 auf Seite 76 eine Möglichkeit geschaffen, mit einem einzelnen, besonders aufgebauten Sperreintrag einen Bereich von Zertifikatsnummern zu revozieren.

4.4 Erweiterte Funktionalität

Das Besondere einer Fail-Safe-PKI ist die Möglichkeit, multiple digitale Signaturen und iterative Verschlüsselungen zu nutzen. Sinnvolle Einsatzbereiche sind multipel signierte Dokumente, E-Mails, Sperrlisten, Zertifikats-Status-Antworten oder Zeitstempel sowie iterativ veschlüsselte Dokumente, E-Mails oder Kommunikationen. Ein Zertifikat fehlt bewusst in dieser Auflistung, weil das Fail-Safe-Konzept keine Vermischung von Algorithmen, die auf verschiedenen Basisproblemen basieren, in einem Zertifikat vorsieht. Dieser Abschnitt widmet sich diesen neuen Techniken.

4.4.1 Multipel signiertes Dokument/E-Mail

Für die Beweiskraft digitaler Signaturen ist es sinnvoll, wenn sie multipel ausgeführt werden, um im Schadensfall, wenn eine der beiden Signaturen kompromittiert sein sollte, weiterhin aussagekräftig zu sein. Anwendungsbeispiele sind Dokumente oder E-Mails. Das Signatur/Verifikations-Modell entsprechend dem Modell auf Seite 88 ist das Folgende:

Signaturverfahren	$\text{sign}^{i_1}, \text{sign}^{i_2} \in S$, voneinander unabhängig
Schlüsselpaare	$(\text{prK}_{\text{CH}}^{i_1}, \text{puK}_{\text{CH}}^{i_1})$, $(\text{prK}_{\text{CH}}^{i_2}, \text{puK}_{\text{CH}}^{i_2})$
Input D	Text
Signierer	CH
S^*	multipel signiertes Dokument/E-Mail (D, S^{i_1,i_2}), optional CH-Zertifikate
Verifizieren	Eine(!) Signatur zu verifizieren und zu validieren, reicht normalerweise, wenn kein Zertifikat revoziert ist, aus, d. h. $V \subseteq \{i_1, i_2\}$, $\#V = v = 1$: $\text{verify}^V(D, S^{i_1,i_2}, \{\text{puK}_{\text{CH}}^V\}) = \text{TRUE}$, wenn für alle $k \in V$ gilt: $\text{verify}^k(D, \text{sign}^k(D, \text{prK}_{\text{CH}}^k), \text{puK}_{\text{CH}}^k) = \text{TRUE}$ Falls das Verifikatsergebnis FALSE ist, muss die Signatur abgelehnt werden.
Validieren	Validieren ist notwendig, um sicherzustellen, dass das entsprechende Zertifikat zur Signatur gültig ist. Dazu wird das Zertifikat des CHs validiert: Ist es gültig, so ist die Signatur korrekt. Andernfalls, d. h. wenn das Zertifikat des CHs bereits revoziert wurde, wird eine zweite Signatur aus S^* verifiziert und validiert. Ist diese mathematisch korrekt und das Zertifikat nicht revoziert, so ist die Signatur gültig.

Für Archivierungszwecke sollte das Dokument mit einem Zeitstempel versehen und nachsigniert werden. Der genaue Vorgang wird in Abschnitt 4.5.12 auf Seite 151 erklärt.

Der de facto Standard PKCS#7 gestattet es bereits, Dokumente multipel zu signieren und als E-Mail zu verschicken. In Appendix A.6 auf Seite 206 ist die Syntax notiert.

4.4.2 Multipel signierte Policy

Eine besonderes, multipel signiertes Dokument ist die Policy. Eine Policy kann vorschreiben, dass in einer Verifikation multiple digitale Signaturen erwartet und verifiziert werden müssen. Da ein Austauschen der Policy – insbesondere wenn sie vom Client automatisch ausgewertet wird – sicherheitskritisch sein kann, kann es in Abhängigkeit des Sicherheitsniveaus bei einer konkreten Anwendung sinnvoll sein, die Policy multipel zu signieren und alle Signaturen zu verifizieren.

Signaturverfahren	$\text{sign}^{i_1}, \text{sign}^{i_2} \in S$, voneinander unabhängig
Schlüsselpaare	$(\text{prK}_{\text{CA}}^{i_1}, \text{puK}_{\text{CA}}^{i_1})$, $(\text{prK}_{\text{CA}}^{i_2}, \text{puK}_{\text{CA}}^{i_2})$
Input D	Policy-Inhalt
Signierer	CA
S^*	signierte Policy (D, S^{i_1,i_2})
Verifizieren	Es werden v Signaturen aus $V \subseteq \{i_1, i_2\}$ verifiziert und validiert: $\text{verify}^V(D, S^{i_1,i_2}, \{\text{puK}_{\text{CA}}^V\}) = \text{TRUE}$, wenn für $k \in V$ gilt: $\text{verify}^k(D, \text{sign}^k(D, \text{prK}_{\text{CA}}^k), \text{puK}_{\text{CA}}^k) = \text{TRUE}$ Falls das Verifikatsergebnis FALSE ist, muss die Policy abgelehnt werden.
Validieren	Validieren ist notwendig, um sicherzustellen, dass das Zertifikat der CA gültig ist.

Policies sind weder im Inhalt noch im Signaturformat standardisiert. Ein Vorschlag, was eine Policy enthalten sollte und wie die Forderung nach multiplen Verfahren formuliert werden kann, ist in Appendix A.9 auf Seite 218 dargelegt.

4.4.3 Multipel signierte Sperrliste

Eine Sperrliste sollte multipel signiert sein, damit die Sperrliste bei Kompromittierung einer ihrer Signaturen weiterhin ihre Authentizität und Integrität behält. Im Sinne des Signatur-Verifikations-Modells von Seite 88 gilt:

Signaturverfahren	$\text{sign}^{i_1}, \text{sign}^{i_2} \in S$, voneinander unabhängig
Schlüsselpaare	$(\text{prK}_{\text{CA}}^{i_1}, \text{puK}_{\text{CA}}^{i_1})$, $(\text{prK}_{\text{CA}}^{i_2}, \text{puK}_{\text{CA}}^{i_2})$
Input D	Menge revozierter Zertifikatsnummern. Eine besondere Zertifikatsnummer kann nach Abschnitt 3.4 auf Seite 76 einen Bereich von Nummern implizieren.
Signierer	CA
S^*	multipel signierte Sperrliste $\text{CRL}^{i_1,i_2} = (D, S^{i_1,i_2})$
Verifizieren	$v = 2$, d. h. beide Signaturen müssen verifiziert werden. Die Sperrliste ist gültig, wenn $\text{sign}^{i_1}(D, \text{prK}_{\text{CA}}^{i_1})$ und $\text{sign}^{i_2}(D, \text{prK}_{\text{CA}}^{i_2})$ mathematisch korrekt sind.

Validieren	Validieren, ob das Zertifikat zur Sperrliste revoziert ist, findet nicht statt.

In Appendix A.3 auf Seite 193 wird die entsprechende Syntax der erweiterten Sperrliste CRL^{i_1,i_2} definiert.

4.4.4 Multipel signierte Zertifikats-Status-Antwort

Zertifikats-Status-Anfragen müssen nicht multipel signiert werden, weil die Anfragen nicht aufgehoben werden. Eine entsprechende Antwort sollte hingegen stets multipel signiert sein, damit die Zertifikats-Status-Antwort bei Kompromittierung einer ihrer Signaturen weiterhin ihre Authentizität und Integrität behält. Im Sinne des Signatur-Verifikations-Modells von Seite 88 gilt:

Signaturverfahren	$\mathrm{sign}^{i_1}, \mathrm{sign}^{i_2} \in S$, voneinander unabhängig
Schlüsselpaare	$(\mathrm{prK}_{\mathrm{CA}}^{i_1}, \mathrm{puK}_{\mathrm{CA}}^{i_1})$, $(\mathrm{prK}_{\mathrm{CA}}^{i_2}, \mathrm{puK}_{\mathrm{CA}}^{i_2})$
Input D	„Zertifikat mit Nummer n ist gültig, ungültig oder unbekannt“, wenn die Antwort erfolgreich war; andernfalls wird eine Fehlermeldung übermittelt.
Signierer	CA, falls die CA diesen Dienst anbietet, oder ein Dienstleister
S^*	multipel signierte Zertifikats-Status-Antwort (D, S^{i_1,i_2})
Verifizieren	$v = 2$, d. h. beide Signaturen müssen verifiziert werden. Die Zertifikats-Status-Antwort ist gültig, wenn $\mathrm{sign}^{i_1}(D, \mathrm{prK}_{\mathrm{CA}}^{i_1})$ und $\mathrm{sign}^{i_2}(D, \mathrm{prK}_{\mathrm{CA}}^{i_2})$ mathematisch korrekt sind.
Validieren	Validieren, ob das Zertifikat zur Status-Antwort revoziert ist, findet nicht statt.

Diese Ideen werden in den de facto Standard Online Certificate Status Protocol (OCSP) integriert. Dazu wird eine neue Version dieses Standards geschaffen: Mit der neuen Version v11 wird OCSP für die Fail-Safe-PKI bezeichnet. In Appendix A.4 auf Seite 197 wird eine entsprechende Syntax für das erweiterte OCSPv11 definiert.

Eine Implementierung von OCSPv11 ist im Rahmen dieser Dissertation entstanden und wird in Abschnitt 6.2 auf Seite 178 beschrieben.

4.4.5 Multipel signierte Zeitstempel

Für die Beweiskraft eines digital signierten Dokuments ist sein Erstellungszeitpunkt entscheidend. Um die Beweiskraft im Falle eines Schadens aufrecht erhalten zu können, sollten Zeitstempel multipel signiert werden.

Signaturverfahren	$\mathrm{sign}^{i_1}, \mathrm{sign}^{i_2} \in S$, voneinander unabhängig
Schlüsselpaare	$(\mathrm{prK}_{\mathrm{TSA}}^{i_1}, \mathrm{puK}_{\mathrm{TSA}}^{i_1})$, $(\mathrm{prK}_{\mathrm{TSA}}^{i_2}, \mathrm{puK}_{\mathrm{TSA}}^{i_2})$
Input von Benutzer an TSA	Tupel von Hashwerten $(h_1, h_2) = (\kappa^1(D^*), \kappa^2(D^*))$ für ein Dokument D^* und zwei voneinander unabhängige Hashfunktionen κ^1 und κ^2
Input D	$(h_1, h_2) \mid$ Zeit t
Signierer	Zeitstempelinstanz (TSA), optional Zertifikate der TSA
S^*	multipel signierter Zeitstempel (D, S^{i_1,i_2})

Aufzubewahren	Dokument D^* und Zeitstempel S^*
Verifizieren	Eine(!) Signatur zu verifizieren und zu validieren, reicht normalerweise, wenn kein Zertifikat revoziert ist, aus, d. h. $V \subseteq \{i_1, i_2\}$, $\#V = v = 1$: $\text{verify}^V(D, S^{i_1,i_2}, \{\text{puK}^V_{\text{TSA}}\}) = \text{TRUE}$, wenn für alle $k \in V$ gilt: $\text{verify}^k((h_1, h_2) \mid t, \text{sign}^k((h_1, h_2) \mid t, \text{prK}^k_{\text{TSA}}), \text{puK}^k_{\text{TSA}}) = \text{TRUE}$ Falls das Verifikatsergebnis FALSE ist, muss der Zeitstempel abgelehnt werden.
Validieren	Validieren ist notwendig, um sicherzustellen, dass das entsprechende Zertifikat des Zeitstempels gültig ist. Dazu wird das Zertifikat des TSA validiert: Ist es noch gültig, so ist der Zeitstempel korrekt. Andernfalls, d. h. wenn das Zertifikat des Zeitstempels bereits revoziert wurde, wird die zweite Signatur aus S^* verifiziert und validiert. Ist diese mathematisch korrekt und das Zertifikat nicht revoziert, so ist der Zeitstempel gültig.
Gehört S^* zu D^*?	Prüfen, ob $\kappa^1(D^*) = h_1$ oder $\kappa^2(D^*) = h_2$.

Bevor die Gültigkeit des Zertifikats des Zeitstempels endet, sollte nachsigniert werden. Der genaue Vorgang des Nachsignierens wird in Abschnitt 4.5.12 auf Seite 151 erklärt.

Diese Ideen werden in der neuen Version v11 des de facto Standards Time Stamp Protocol (TSP) realisiert. Die Erweiterung des Time Stamp Protocols zu Version v11 (TSPv11) ist in Appendix A.5 auf Seite 202 im Detail beschrieben.

Eine Implementierung ist im Rahmen dieser Dissertation entstanden und in Abschnitt 6.3 auf Seite 181 dargestellt.

4.4.6 Iterativ verschlüsseltes Dokument/E-Mail

Kurzfristig geheim zu haltende Informationen können mit einem iterativen hybriden Verschlüsselungsverfahren verschlüsselt werden, so dass das sensible Dokument oder die sensible E-Mail auch dann vertraulich bleibt, wenn eines der verwendeten Verfahren kompromittiert werden sollte.

Das Ver-/Entschlüsselungs-Modell (nach dem Modell auf Seite 90) für die iterative hybride Verschlüsselung ist das Folgende:

Symm. Verschl'verf.	$\text{enc\_sym}^{i_1}, \text{enc\_sym}^{i_2} \in \text{E\_sym}$, voneinander unabhängig
Session Keys	$\{\text{sK}^{i_1}, \text{sK}^{i_2}\}$
Asymm. Verschl'verf.	$\text{enc\_asym}^{j_1}, \text{enc\_asym}^{j_2} \in \text{E\_asym}$, voneinander unabhängig
Private Schlüssel	$\{\text{prK}^{j_1}_{\text{Empfänger}}, \text{prK}^{j_2}_{\text{Empfänger}}\}$
Öffentliche Schlüssel	$\{\text{puK}^{j_1}_{\text{Empfänger}}, \text{puK}^{j_2}_{\text{Empfänger}}\}$
Klartext	P
1. Kryptogramm	C_1, das aus symm. verschlüsseltem P und asymm. verschlüsseltem sK^{i_1} besteht
2. Kryptogramm	C_2, das aus symm. verschlüsseltem C_1 und asymm. verschlüsseltem sK^{i_2} besteht
Übertragen	C_2
1. Entschlüsseln	C_1 aus C_2 mit sK^{i_2}, zuvor entschlüsselt mit $\text{prK}^{j_2}_{\text{Empfänger}}$
2. Entschlüsseln	P aus C_1 mit sK^{i_1}, zuvor entschlüsselt mit $\text{prK}^{j_1}_{\text{Empfänger}}$

PKCS#7 gestattet es bereits heute, Dokumente iterativ zu verschlüsseln und als E-Mail zu versenden. In Appendix A.6 auf Seite 206 findet sich die Syntax.

4.4.7 Iterative Authentisierung

In Authentisierungen – wie in Abschnitt 1.3.2 auf Seite 27 beschrieben – werden digitale Signaturen verwendet. Für ein iteratives Ausführen der digitalen Signatur innerhalb der Authentisierung sei auf Abschnitt 4.4.1 (Seite 103) verwiesen. Darüber hinaus können im Rahmen dieses Authentisierungsprozesses zwei Session Keys ausgehandelt werden, um die nachfolgende Kommunikation mit zwei iterativen symmetrischen Verfahren abzusichern.

Wird zum Schlüsselaustausch – noch innerhalb des Authentisierungsprozesses – ein iteratives asymmetrisches Verschlüsselungsverfahren eingesetzt, so sieht das Ver-/Entschlüsselungs-Modell wie folgt aus:

Asymm. Verschl'verf.	$\text{enc\_asym}^{j_1}, \text{enc\_asym}^{j_2} \in \text{E\_asym}$, voneinander unabhängig
Private Schlüssel	$\{\text{prK}^{j_1}_{\text{Empfänger}}, \text{prK}^{j_2}_{\text{Empfänger}}\}$
Öffentliche Schlüssel	$\{\text{puK}^{j_1}_{\text{Empfänger}}, \text{puK}^{j_2}_{\text{Empfänger}}\}$
Session Keys	$(\text{sK}^{i_1}, \text{sK}^{i_2})$
Iterativ verschlüsseln	$C_2 = \text{enc\_asym}^{j_2}(\text{enc\_asym}^{j_1}((\text{sK}^{i_1}, \text{sK}^{i_2}), \text{puK}^{j_1}_{\text{Empf.}}), \text{puK}^{j_2}_{\text{Empf.}})$
Übertragen	C_2
Entschlüsseln	$(\text{sK}^{i_1}, \text{sK}^{i_2}) = \text{dec\_asym}^{j_1}(\text{dec\_asym}^{j_1}(C_2, \text{prK}^{j_1}_{\text{Empf.}}), \text{prK}^{j_1}_{\text{Empf.}})$

Ist ein Paar von Session Keys, $(\text{sK}^{i_1}, \text{sK}^{i_2})$, übertragen worden, kann die nachfolgende Kommunikation mit einem iterativen symmetrischen Verfahren abgesichert werden – nach diesem Ver-/Entschlüsselungs-Modell:

Symm. Verschl'verf.	$\text{enc\_sym}^{i_1}, \text{enc\_sym}^{i_2} \in \text{E\_sym}$, voneinander unabhängig
Session Keys	$(\text{sK}^{i_1}, \text{sK}^{i_2})$
Klartext	P
Iterativ verschlüsseln	$C_2 = \text{enc\_sym}^{i_2}(\text{enc\_sym}^{i_1}(P, \text{sK}^{i_1}), \text{sK}^{i_2})$
Übertragen	C_2
Entschlüsseln	$P = \text{dec\_sym}^{i_1}(\text{dec\_sym}^{i_1}(C_2, \text{sK}^{i_1}), \text{sK}^{i_1})$

Nachdem die „gebräuchlichen“ Funktionalitäten, d. h. die Funktionen, die im täglichen Umgang in einer Fail-Safe-PKI vorkommen, beschrieben wurden, wird dem Rest dieses Kapitels der Fall gewidmet, der – hoffentlich – seltener vorkommt: Der Schadensfall. Ein Kryptoalgorithmus oder ein Schlüssel wird kompromittiert. Es wird gezeigt, wie Verfahren, Sicherheitsanker und Schlüssel in der Fail-Safe-PKI sicher ausgetauscht werden können.

4.5 Funktionalität im Schadensfall

Eine Besonderheit der Fail-Safe-PKI ist, dass kryptographische und einsatzspezifische Komponenten auf sichere Weise und im laufenden Betrieb hinzugefügt und gelöscht werden können. Als mögliche Gründe wurden in Abschnitt 3.2.3 auf Seite 69 verschiedene Schadensfälle diskutiert

und aufgelistet, wobei unter „Schadensfall“ nicht zwingend ein unerwartetes Ereignis zu verstehen ist, sondern auch schon länger bekannte neue Empfehlungen zu kryptographisch geeigneten Algorithmen, falls deren Realisierung eine neue Implementierung erforderlich macht.

In diesem Abschnitt wird der Ablauf zum Aktualisieren der PKI detailliert für die zwei Klassen von Anwendungen (nach Abschnitt 1.5 auf Seite 38) beschrieben sowie die Sicherheit dieser Aktion begründet. Zugrunde gelegt wird ein Schadensfall, welcher den Austausch von kryptographischen und einsatzspezifischen Komponenten erforderlich macht. Die Fälle, in denen Komponenten nur hinzugefügt oder nur gelöscht werden, sind in der Betrachtung des Austauschs von Komponenten enthalten – es werden dann „leere“ Komponenten gelöscht bzw. installiert.

Der Ablauf zum Austausch kryptographischer und einsatzspezifischer Komponenten gliedert sich in zehn Phasen. Um die Übersicht zu verbessern, wird als erstes der Ablauf kurz erläutert und in Abbildung 4.3 illustriert.

In Phase 0 – sozusagen als Auslöser – tritt ein Schaden auf, woraufhin in Phase 1 nach Prüfung des Schadens betroffene Zertifikate revoziert werden. Bis hierher unterscheidet sich der Ablauf innerhalb der Fail-Safe-PKI nicht von dem in einer herkömmlichen PKI. Allerdings treten in einer Fail-Safe-PKI vier bislang offene Probleme nicht mehr auf: Die Verfügbarkeit der PKI bleibt erhalten, die Beweisbarkeit multipler digitaler Signaturen und die Vertraulichkeit von iterativen Verschlüsselungen bleiben gewährleistet und Revokationsmechanismen bleiben intakt.

Nun beginnt der eigentliche Austausch. Der Update Service als vertrauenswürdige Instanz des Trust Centers führt mittels des neuen Update Management Protocols (UMP) den Austausch bei allen involvierten Certificate Holdern durch, d. h. beim gesamten Equipment, das die CHs nutzen. In Abschnitt 1.3.2 auf Seite 22 wurde das Modell eines Clients mit PSE in verschiedenen Varianten beschrieben. Deshalb wird der Austausch ebenfalls anhand des Modells und aller Varianten betrachtet.

Initiiert wird der Austausch in Phase 2, indem ein(!) UpdateComponent des Update Services, das für alle involvierten CHs identisch ist, verschickt wird – dies können alle, einige oder nur ein Zertifikatsinhaber sein. Das UpdateComponent enthält alle Informationen über zu deaktivierende und zu installierende Komponenten. Während ein Sicherheitsanker direkt im UpdateComponent übermittelt wird, wird der Code für eine kryptographische Komponente nicht direkt, sondern nur der Link zum Code angegeben. Auf diese Weise lässt sich der Umfang des UpdateComponents verringern, die Netzlast, wenn mehrere Benutzer den Code laden, zeitlich strecken und Code für verschiedene Systeme diversifizieren, so dass sich jeder Client unter dem angegebenen Link genau den Code lädt, der für sein System geeignet ist. UpdateComponent und Code sind vom Update Service so abgesichert, dass Authentizität und Integrität sowie beim UpdateComponent zusätzlich Aktualität und korrekte Adressierung gewährleistet sind.

Für den Transport des UpdateComponents sind verschiedene Managementstrukturen zu beachten: Bei einer Signatur-Anwendung kann der Update Service eine besondere, mit dem UpdateComponent versehene E-Mail verschicken, während bei einer Authentisierungs-Anwendung ein Umweg gegangen werden muss, weil der Update Service in einer solchen Anwendung i. A. nach Ausgabe von Chipkarte oder Zertifikat keinen Kontakt mehr zum Client hat. Der Umweg verläuft über Sperrinformationen, so dass einem Server in einer Authentisierungs-Anwendung die Aufgabe zukommt, an alle Clients das UpdateComponent weiterzuleiten. Die verschiedenen Varianten werden in Phase 3 besprochen.

Wenn der Client ein UpdateComponent erhalten hat, werden nach einer besonderen Verifikation zunächst dort Client-relevante Aktionen ausgeführt (in Phase 4). Anschließend reicht der Client ein besonderes Chipkarten-Kommando des UpdateComponents an eine mögliche Chipkarte

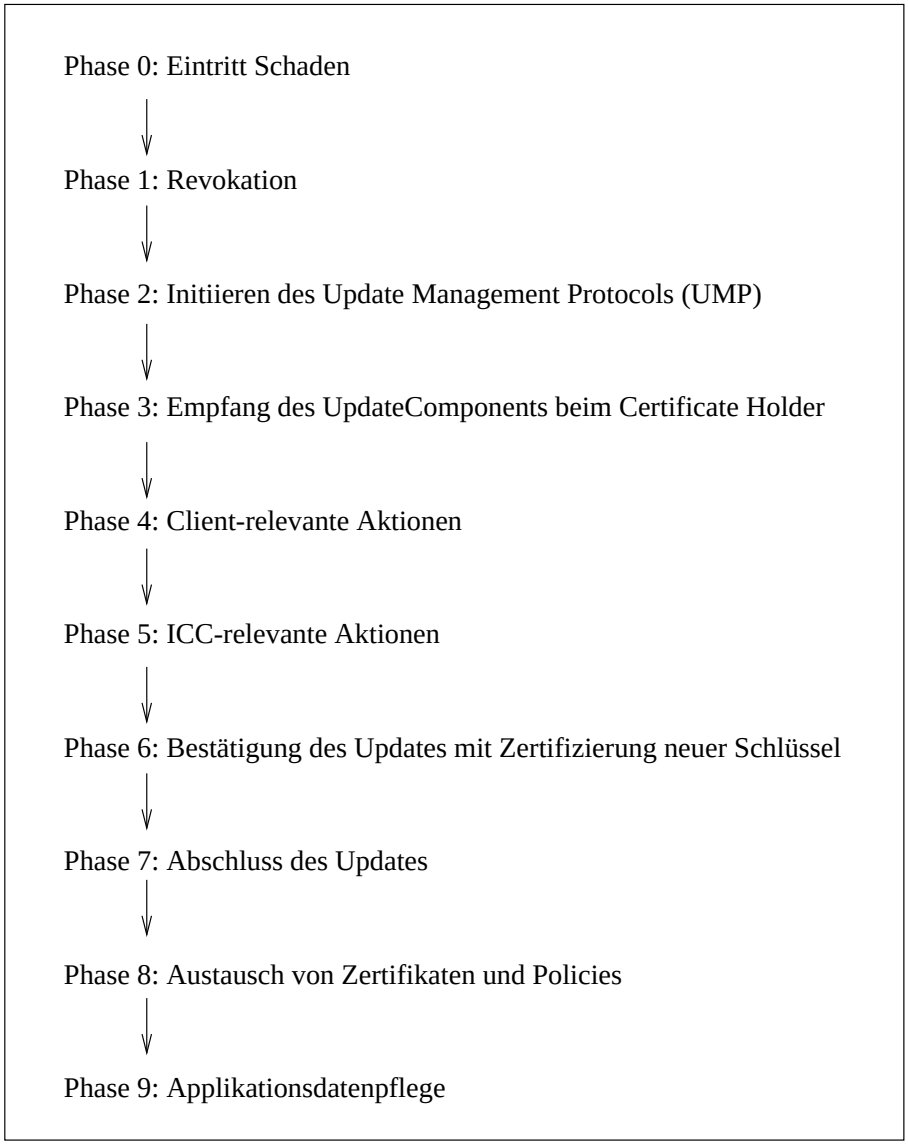

Abbildung 4.3: Ablauf im Schadensfall

weiter, wo nach Verifikation des Kommandos ICC-relevante Aktionen (in Phase 5) ausgeführt werden.

Einen erfolgreichen Update bestätigt die PSE des CHs, indem sie über den Client ein multipel signiertes UpdateComponentResponse an ein Update Service in Phase 6 zurückschickt. Der Update Service kann dadurch für jeden Zertifikatsinhaber einen abgeschlossenen Austausch registrieren und damit verhindern, dass Schadensfälle, die zeitlich nacheinander auftreten, durch eine Verzögerung dennoch gleichzeitig stattfinden und die Sicherheitsmaßnahmen des UMPs unterlaufen würden – wie in Abschnitt 3.2.3 auf Seite 69 erläutert. Wenn Schlüssel neu zu generieren sind, müssen in der PSE des CHs neu erzeugte Schlüssel an die Certification Authority zur Zertifizierung weitergeleitet werden. Der Transport neu generierter Schlüssel wird in das UpdateComponentResponse integriert.

Anschließend wird in Phase 7 der Austausch abgeschlossen. Neue Zertifikate und Policies können

in Phase 8 installiert und in Phase 9 schließlich digitale Signaturen nachsigniert werden.

Oberstes Gestaltungsprinzip des Austauschs ist neben der Sicherheit die Fehlertoleranz. Aus diesem Grund werden neue Verfahren und Sicherheitsanker zunächst neben den alten installiert und erst dann gelöscht, wenn die neuen Komponenten erfolgreich installiert wurden. Auf diese Weise wird gewährleistet, dass bei einer Störung weiterhin Verfahren und Sicherheitsanker bereitstehen und ein erneuter Austausch möglich ist. Darüber hinaus wird in jedem Abschnitt eine Diskussion verschiedener, möglicher Störungen im Ablauf geführt und die Reaktionen der Fail-Safe-PKI beschrieben.

Nun zur detaillierten Beschreibung.

4.5.1 Phase 0: Eintritt Schaden

Das für einen Austausch initiale Ereignis ist ein Schadensfall. Mögliche Schäden, auf die das Fail-Safe-Konzept reagieren kann, werden in diesem Abschnitt zusammengefasst.

Angenommen, ein Schaden tritt ein. Nach Abschnitt 2.2.1 auf Seite 49 werden folgende Schäden und Folgeschäden berücksichtigt:

1. Signaturalgorithmus σ^k (Definition 1 auf Seite 3) kompromittiert, $k \in \{1, ..., p\}$
   (a) unabhängig von Schlüssellängen, z. B. weil das mathematische Basisproblem α^k kryptographisch nicht mehr geeignet ist
   (b) abhängig von Schlüssellängen
2. Hashfunktion κ^k (Definition 2 auf Seite 4) kryptographisch ungeeignet, $k \in \{1, ..., m_H\}$
   (a) unabhängig vom Input
   (b) abhängig vom Input, z. B. für Dokumente einer bestimmten Struktur oder für nur ein spezielles Dokument
3. Formatierung ϕ^k (Definition auf Seite 6) kryptographisch ungeeignet, $k \in \{1, ..., m_F\}$
4. als Folge Signaturverfahren sign^k (Definition 3 auf Seite 6), welches sich aus σ^k, κ^k oder ϕ^k zusammensetzt, kompromittiert, $k \in \{1, ..., p\}$
5. als Folge Zertifikate cert^k ohne Beweiskraft, die Schlüssel zu σ^k zertifizieren oder mit sign^k signiert sind, $k \in \{1, ..., p\}$
6. als Folge weitere Signaturverfahren sign^l kompromittiert, die κ^k oder ϕ^k enthalten, $l \in \{p + 1, ..., m_S\}$
7. Verschlüsselungsalgorithmus η_asym^k bzw. η_sym^k (Definitionen 4 und 5 auf Seite 10) kompromittiert, $k \in \{1, ..., m_{\xi\_\text{asym}}\}$ resp. $k \in \{1, ..., m_{\xi\_\text{sym}}\}$
   (a) unabhängig von Schlüssellängen, z. B. weil das mathematische Basisproblem α^k kryptographisch nicht mehr geeignet ist
   (b) abhängig von Schlüssellängen
8. als Folge Zertifikate cert'^k nutzlos, die Schlüssel zu η_asym^k zertifizieren, $k \in \{1, ..., m_{\xi\_\text{asym}}\}$

9. als Folge Verschlüsselungsverfahren enc_asym^k bzw. enc_sym^k (Definition 6 auf Seite 12), welches sich aus $\eta\text{\_asym}^k$ bzw. $\eta\text{\_sym}^k$ oder ϕ^k zusammensetzt, kompromittiert, $k \in \{1, ..., m_{\text{E\_asym}}\}$ resp. $k \in \{1, ..., m_{\text{E\_sym}}\}$
10. als Folge weitere Verschlüsselungsverfahren enc_asym^l bzw. enc_sym^l kompromittiert, die ϕ^k enthalten, $l \in \{p+1, ..., m_{\text{E\_asym}}\}$ resp. $l \in \{p+1, ..., m_{\text{E\_sym}}\}$
11. Schlüssel der Wurzelzertifizierungsinstanz (RootCA-Key) $\text{prK}^k_{\text{RootCA}}$ kompromittiert, welcher der Sicherheitsanker ist, $k \in \{1, ..., p\}$
12. als Folge das Selbstzertifikat $\text{cert}^k_{\text{RootCA}}$ ohne Beweiskraft und somit auch sämtliche von der RootCA zertifizierten Zertifikate cert^k, $k \in \{1, ..., p\}$
13. Zertifizierungsinstanz-Schlüssel (CA-Key) prK^k_{CA} kompromittiert, $k \in \{1, ..., p\}$
14. als Folge das Zertifikat $\text{cert}^k_{\text{CA}}$ ohne Beweiskraft und somit auch sämtliche von der CA zertifizierten Zertifikate cert^k, $k \in \{1, ..., p\}$
15. Teilnehmer-Schlüssel (CH-Key) prK^k_{CH} kompromittiert, $k \in \{1, ..., p\}$
    (a) alle Schlüssel, unabhängig von der Länge
    (b) einige Schlüssel, abhängig von der Länge
    (c) ein einzelner Schlüssel
16. als Folge das Zertifikat $\text{cert}^k_{\text{CH}}$ ohne Beweiskraft, $k \in \{1, ..., p\}$
17. Schlüssel der Zeitstempelinstanz (TSA-Key) $\text{prK}^k_{\text{TSA}}$ oder Schlüssel der CA für Sperrauskünfte (CRL- oder OCSP-Key) prK^k_{CA} kompromittiert, $k \in \{1, ..., p\}$
18. als Folge das Zertifikat $\text{cert}^k_{\text{TSA}}$ oder $\text{cert}^k_{\text{CA}}$ ohne Beweiskraft, $k \in \{1, ..., p\}$
19. ein geheimer Session Key sK^k, der eine Kommunikation sichert, kompromittiert, $k \in \{1, ..., m_{\text{E\_sym}}\}$

Durch die Konstruktion der p Teil-PKIs, $P = \{\text{PKI}^1, ..., \text{PKI}^r, \text{PKI}^{r+1}, ..., \text{PKI}^p\}$ – siehe Abschnitt 4.2 auf Seite 93 –, ist durch einen Schaden höchstens eine der ersten r Teil-PKIs kompromittiert, weil die für Zertifizierungen eingesetzten Signaturverfahren in den ersten r Teil-PKIs paarweise voneinander unabhängig sind und nach Voraussetzung zwei voneinander unabhängige kryptographische Algorithmen oder zwei CA-Schlüssel nicht simultan kompromittiert werden. In diesem Fall ist PKI^k die kompromittierte Teil-PKI. Dennoch kann ein Schaden weitere Schäden in den ersten r Teil-PKIs implizieren, z. B. wenn eine kompromittierte Hashfunktion in mehreren Signaturverfahren in verschiedenen Teil-PKIs eingesetzt wird. Wichtig ist, dass durch die Wahl der für Zertifizierungen genutzten Signaturverfahren die Zertifikate in diesen Teil-PKIs davon nicht betroffen sind, so dass diese Teil-PKIs weiterhin verfügbar sind. Im Gegensatz zu den restlichen $p - r$ Teil-PKIs: Sie können vom Schaden, den PKI^k berührt, tangiert werden, weil die für Zertifizierungen genutzten Verfahren in diesen Teil-PKIs nicht zwingend zu den Verfahren aus den ersten p Teil-PKIs unabhängig sind. Bei einer solchen, kompromittierten Teil-PKI entspricht die Vorgehensweise zum Austausch von Komponenten der im Folgenden beschriebenen Vorgehensweise für die Teil-PKI PKI^k.

Ausgangspunkt von Phase 0 ist, dass ein Schaden bekannt wird. Bei einem unveröffentlichten Schaden kann demzufolge keine Gegenmaßnahme eingeleitet werden. Darüber, wie lange Schäden

unveröffentlicht bleiben, kann spekuliert werden. Dabei sind beispielsweise folgende Aspekte zu berücksichtigen: Mathematiker forschen weltweit an kryptographischen Algorithmen und es gibt die Meinung, dass sich eine neue und für die Kryptographie relevante Idee über Fachartikel und -konferenzen rasch verbreiten würde. Außerdem würde ein erfolgreicher Angriff irgendjemandem auffallen, so dass sich nach Auftreten mehrerer Angriffe die Meinung durchsetzen würde, dass es offensichtlich doch Schäden gibt, deren Existenz zuvor abgestritten wurde. Auf der anderen Seite ranken sich seit Jahren sehr viele Spekulationen um die Fähigkeiten der amerikanischen National Security Agency (NSA), so dass keine verlässlichen Angaben gemacht werden können, wie lange Angriffe unveröffentlicht bleiben.

Nach Bekanntwerden eines Schadens werden davon betroffene Zertifikate revoziert.

4.5.2 Phase 1: Revokation

Was passiert in der Revokationsphase und welche Eigenschaften der Fail-Safe-PKI funktionieren weiterhin uneingeschränkt? Dieser Abschnitt beschäftigt sich mit diesen Fragen.

Alle vom Schaden betroffenen Zertifikate aus PKI^k werden revoziert und in einfach signierten Sperrlisten CRL^i und multipel signierten Sperrlisten $\mathrm{CRL}^{i,j}$, $i, j \in \{1, ..., m_S\}$, veröffentlicht, wobei die benutzten Signaturverfahren sign^i und sign^j voneinander unabhängig sind. Signaturen, die mit einem Schlüssel prK^k, dessen Zertifikat cert^k revoziert ist, erzeugt wurden, dürfen nicht mehr akzeptiert werden.

Falls der Schaden einen ganzen Bereich von Zertifikatsnummern betrifft, kann der entsprechende Bereich durch eine einzige Zertifikatsnummer gesperrt werden, wie in Abschnitt 3.4 auf Seite 76 ausgeführt.

Nach Voraussetzung einer Fail-Safe-PKI existiert mindestens ein $u \in \{1, ..., r\}$, so dass Signaturverfahren sign^u samt Schlüsseln $(\mathrm{prK}^u, \mathrm{puK}^u)$ und Zertifikaten cert^u aus PKI^u nicht kompromittiert sind.

Folgende bislang offene Probleme sind jetzt gelöst:

1. Verfügbarkeit des Systems ist gewährleistet, weil mindestens PKI^u weiter funktioniert.
2. Sperrlisten und OCSP-Antworten kann weiterhin vertraut werden, weil mindestens eine Teil-PKI sicher ist und stets zwei Signaturen zur Verifikation vorgeschrieben sind. Sperrlisten bleiben handhabbar, weil ihr Umfang durch die Möglichkeit, durch einen Eintrag mehrere Zertifikatsnummern implizit zu sperren, nicht zu stark zunimmt.
3. Beweiskraft von mit PKI^k und PKI^u multipel signierten Dokumenten bleibt erhalten, weil die Signatur aus PKI^u die Beweiskraft des Dokuments gewährleistet. Statt PKI^u-Signatur kann auch eine andere, durch den Schaden nicht kompromittierte Signatur genutzt werden.
4. Vertraulichkeit von mit PKI^k und PKI^u iterativ verschlüsselten Dokumenten oder Nachrichten bleibt gewahrt, weil die Verschlüsselung, die mit PKI^u erzeugt wurde, nicht effizient gebrochen werden kann. Statt PKI^u-Verschlüsselung kann auch eine andere, durch den Schaden nicht kompromittierte Verschlüsselung genutzt werden.

Die Sicherheit der PKI ist durch Revokation hergestellt, d. h. jeder CH hat die Möglichkeit eine Signatur abzulehnen oder auf eine Verschlüsselung zu verzichten, falls das zugehörige Zertifikat revoziert ist.

Die Besonderheit der Fail-Safe-PKI – der Austausch kompromittierter Komponenten – schließt sich der Revokationsphase an.

4.5.3 Phase 2: Initiieren des Update Management Protocols (UMP)

In diesem Abschnitt wird dargestellt, wie der Update Service auf einen Schaden reagiert. Zunächst werden die schadensabhängigen Gegenmaßnahmen sowie ihre Auswirkungen auf bestehende und neue Teil-PKIs diskutiert. Anschließend wird das neue Update Management Protocol mit seinen Sicherheitseigenschaften vorgestellt und verschiedene Kommunikationswege diskutiert, um den Austausch von kompromittierten Komponenten bei allen betroffenen Certificate Holdern der Fail-Safe-PKI einzuleiten.

Der Update Service als vertrauenswürdige Instanz des Trust Centers darf den Austausch von kryptographischen und einsatzspezifischen Komponenten mittels Update Management Protocol initiieren. Dabei wird angenommen, dass Schäden disjunkt auftreten und sich zwei Aktualisierungsaktionen folglich nicht überschneiden.

Entsprechend der in Abschnitt 3.2.3 auf Seite 69 diskutierten möglichen Gegenmaßnahmen kann folgendes durchgeführt werden:

1. Signaturalgorithmus σ^k gegen σ^n austauschen
2. Hashfunktion κ^k gegen κ^n austauschen
3. Formatierung ϕ^k gegen ϕ^n austauschen
4. als Folge Signaturverfahren sign^k gegen sign^n austauschen
5. als Folge weitere Signaturverfahren gegen solche austauschen, die statt σ^k nun σ^n, statt κ^k nun κ^n oder statt ϕ^k nun ϕ^n enthalten
6. Verschlüsselungsalgorithmus η_asym^k gegen η_asym^n bzw. η_sym^k gegen η_sym^n austauschen
7. als Folge Verschlüsselungsverfahren enc_asym^k gegen enc_asym^n bzw. enc_sym^k gegen enc_sym^n austauschen
8. als Folge weitere Verschlüsselungsverfahren gegen solche austauschen, die statt η_asym^k nun η_asym^n bzw. η_sym^k nun η_sym^n oder statt ϕ^k nun ϕ^n enthalten
9. Sicherheitsanker puK_{CA}^k gegen puK_{CA}^n bzw. als Zertifikat $\text{cert}_{\text{CA}}^k$ gegen $\text{cert}_{\text{CA}}^n$ austauschen
10. CH-Schlüsselpaar $(\text{prK}_{\text{CH}}^k, \text{puK}_{\text{CH}}^k)$ gegen ein neu zu generierendes Paar $(\text{prK}_{\text{CH}}^n, \text{puK}_{\text{CH}}^n)$ austauschen; dies gilt auch für besondere CHs wie etwa den Zeitstempeldienst

Wie bereits erwähnt, beinhalten obige Maßnahmen auch das ausschließliche Deaktivieren oder Hinzufügen von kryptographischen oder einsatzspezifischen Komponenten.

An die neuen, mit n gekennzeichneten Komponenten werden die Anforderungen gestellt, dass eine neue kryptographische Primitive β^n unabhängig zur kompromittierten Primitive β^k ist und die Fail-Safe-PKI nun wieder aus mindestens zwei voneinander unabhängigen Teil-PKIs besteht.

Sind Kryptoverfahren auszuwechseln, wird eine neue Teil-PKI PKI^n für ein $n \in \mathbb{N}$ „gegründet“, was bei ausschließlichem Wechsel von Schlüsseln oder Sicherheitsankern nicht nötig ist. PKI^n enthält: ein mathematisches Basisproblem α^n, einen darauf basierenden Signaturalgorithmus σ^n, ein Signaturverfahren sign^n, welches insbesondere den Signaturalgorithmus σ^n neben Hashfunktion κ^n und Formatierung ϕ^n enthält, Schlüsselpaare $(\text{prK}^n, \text{puK}^n)$ zu σ^n, Zertifikate cert^n,

in dem puK^n veröffentlicht ist und das von der CA mit dem Signaturverfahren sign^n digital signiert wurde, Sicherheitsanker $\text{cert}^n_{\text{CA}}$ mit dem öffentlichen Schlüssel der CA puK^n_{CA}, mögliche weitere Hashfunktionen κ, mögliche weitere Formatierungen ϕ und mögliche weitere Signaturverfahren sign, die aus σ^n und weiteren Hashfunktionen und Formatierungen kombiniert werden.

Bzgl. Verschlüsselungen verfügt PKI^n zusätzlich über einen asymmetrischen Verschlüsselungsalgorithmus η_asym^i, ein asymmetrisches Verschlüsselungsverfahren enc_asym^i, welches insbesondere den asymmetrischen Verschlüsselungsalgorithmus η_asym^i neben Formatierung ϕ enthält, Schlüsselpaare $(\text{prK}'^i, \text{puK}'^i)$ zu η_asym^i, Zertifikate cert'^i, in dem puK'^i veröffentlicht ist und das von der CA mit dem Signaturverfahren sign^i digital signiert wurde, wobei sign^i den Signaturalgorithmus σ^i verwendet, der auf dem gleichen mathematischen Basisproblem wie η_asym^i basiert, weitere asymmetrische Verschlüsselungsverfahren enc_asym, die aus η_asym^i und anderen Formatierungen kombiniert werden, symmetrische Verschlüsselungsalgorithmen η_sym und symmetrische Verschlüsselungsverfahren enc_sym.

Worin unterscheidet sich PKI^n von PKI^k? Nur durch die kompromittierte Komponente (statt β^k nun β^n) und ein davon tangiertes Signatur- oder Verschlüsselungsverfahren (etwa statt sign^k nun sign^n). Weitere Primitiven aus PKI^n und PKI^k können aber identisch sein. Beispielweise bleiben die Hashfunktionen und Formatierungen in PKI^n unverändert, wenn ein Signaturalgorithmus σ^n den Algorithmus σ^k ablöst.

Der Austausch erstreckt sich nicht nur auf die kompromittierte Teil-PKI PKI^k, sondern auch auf sämtliche Teil-PKIs, bei denen Komponenten aktualisiert werden müssen. Dabei ist folgendes zu beachten: Nach Abschnitt 4.2 auf Seite 93 sind die Teil-PKIs wie folgt angeordnet: $P = \{\text{PKI}^1, ..., \text{PKI}^r, \text{PKI}^{r+1}, ..., \text{PKI}^p\}$. Durch die Wahl der für Zertifizierungen genutzten Signaturverfahren ist nur eine(!) der ersten r Teil-PKIs – PKI^k – und u. U. weitere der letzten $p-r$ Teil-PKIs kompromittiert. Für jede kompromittierte Teil-PKI müssen u. U. Schlüssel neu generiert und Zertifikate neu ausgestellt werden, so dass diese Teil-PKIs jeweils durch eine separate Maßnahme aktualisiert werden. Die restlichen Teil-PKIs, bei denen nur kryptographische Primitive und keine Schlüssel ausgetauscht werden, werden durch keine eigene Maßnahme aktualisiert, sondern im Zuge der Aktualisierung für PKI^k.

Es wird unterstellt, das Trust Center habe ein Interesse daran, seinen „Kunden“ – also den Certificate Holdern – die volle Funktionalität der PKI nachhaltig zur Verfügung stellen zu wollen, und deshalb den Update durch den Update Service durchführen zu lassen. Dazu wird angenommen, der Update Service verfüge über entsprechende Daten der Certificate Holder, um sie ansprechen zu können. In einer Signatur-Anwendung kennt es die E-Mail-Adressen seiner Kunden, die z. B. oftmals in Zertifikaten enthalten sind. In einer Authentisierungs-Anwendung kann davon ausgegangen werden, dass eine Firma Listen mit ausgegebenen Chipkarten und entsprechenden -nummern führt und pflegt.

Das Protokoll, welches der Update Service zum Austausch nutzt, ist das neue Update Management Protocol (UMP). Welche Informationen das UMP enthält, hängt vom jeweiligen Schaden ab. Der erste Teil des Update Management Protocols, mit dem der Update Service den Austausch einleitet, heißt **UpdateComponent**. Ein UpdateComponent kann(!) folgende Informationen und Kommandos enthalten:

- Deaktivieren aller kryptographischen Komponenten, die auf dem mathematischen Basisproblem α^k basieren oder mit der kryptographischen Primitive β^k – z. B. Signaturalgorithmus σ^k, Verschlüsselungsalgorithmus η^k, Hashfunktion κ^k oder Formatierung ϕ^k – zusammenhängen!

Steht beispielsweise α^k für das Faktorisierungsproblem, so werden durch diesen Befehl alle Signatur- (σ^k) und Verschlüsselungsalgorithmen (η_asym^k) sowie alle Signatur- (sign^k) und Verschlüsselungsverfahren (enc_asym^k) deaktiviert, die auf diesem Problem basieren. Wird die Hashfunktion κ^k kompromittiert, muss nicht nur κ^k gelöscht werden, sondern auch alle diese Hashfunktion nutzenden Signaturverfahren sign.
Die Abhängigkeiten werden über die Registry des Fail-Safe-Konzepts festgestellt – siehe Abschnitt 4.5.5 auf Seite 127.

- Laden der neuen Fail-Safe-Konzept-geeigneten kryptographischen Komponenten σ^n, κ^n, ϕ^n, sign^n, η_asym^n oder enc_asym^n über einen Link! Sowohl Link als auch der dort erhältliche Code sind abgesichert – siehe Abschnitt 4.5.5 auf Seite 123.

- Löschen des Sicherheitsankers puK_{CA}^k bzw. dessen (Selbst-)Zertifikat $\text{cert}_{\text{CA}}^k$! Falls mehrere Sicherheitsanker statt nur einem installiert sind: Löschen der Sicherheitsanker $\text{puK}_{\text{CA}}^{k_1}$, ..., $\text{puK}_{\text{CA}}^{k_m}$ bzw. deren Zertifikate $\text{cert}_{\text{CA}}^{k_1}$, ...,$\text{cert}_{\text{CA}}^{k_m}$!

- Installieren des neuen Sicherheitsankers puK_{CA}^n bzw. dessen (Selbst-)Zertifikat $\text{cert}_{\text{CA}}^n$! Falls mehrere Sicherheitsanker statt nur einem zu installieren sind: Installieren neuer Sicherheitsanker $\text{puK}_{\text{CA}}^{n_1}$, ..., $\text{puK}_{\text{CA}}^{n_m}$ bzw. deren Zertifikate $\text{cert}_{\text{CA}}^{n_1}$, ..., $\text{cert}_{\text{CA}}^{n_m}$!

- Löschen der eigenen Schlüsselpaare $(\text{puK}_{\text{CH}}^{k_1}, \text{prK}_{\text{CH}}^{k_1})$, ..., $(\text{puK}_{\text{CH}}^{k_m}, \text{prK}_{\text{CH}}^{k_m})$, die

  - zum Algorithmus σ^k, η_asym^k oder η_sym^k sowie
  - zum Verwendungszweck Key Usage gehören und
  - die Schlüssellänge Key Length haben!

  Algorithmus und Verwendungszweck sind nötig, um bei einem(!) CH einen Schlüssel ansprechen zu können. Durch Algorithmus und Key Usage ist dieser eindeutig bestimmt. Ist eine zu kurze Schlüssellänge Grund für den Schaden, kann eine kritische Schlüssellänge angegeben werden. Ist die aktuell benutzte Schlüssellänge unter der kritischen Marke, so muss ein neuer Schlüssel mit der geforderten Länge erzeugt werden. Zu den Schlüsseln gehörende gespeicherte Zertifikate müssen gelöscht werden.

- Kommando zum Generieren neuer eigener Schlüsselpaare $(\text{puK}_{\text{CH}}^{n_1}, \text{prK}_{\text{CH}}^{n_1})$, ..., $(\text{puK}_{\text{CH}}^{n_m}, \text{prK}_{\text{CH}}^{n_m})$, die

  - zum Algorithmus σ^k, η_asym^k oder η_sym^k sowie
  - zum Verwendungszweck Key Usage gehören und
  - die Schlüssellänge Key Length haben!

Die Trennung des Austauschs in die Einzelkommandos „Deaktivieren“ und „Laden“ erlaubt ein ausschließliches Hinzufügen bzw. Löschen von Komponenten.

Die ersten vier Aktionen – Löschen und Installieren von Verfahren und Sicherheitsankern – treffen auf alle CHs zu, die diese Verfahren oder Sicherheitsanker nutzen.

Ob die beiden letzten Aktionen – Löschen und Installieren von Schlüsseln – auf alle Zertifikatsinhaber zutreffen, hängt vom Schaden ab. Ist nur ein einzelner Certificate Holder involviert, so muss dieser CH exakt adressiert werden, weshalb das UpdateComponent folgende Informationen enthalten kann:

- Zähler (MessageID) zur Verhinderung von Replay-Attacken
- Identität des Zertifikatsinhabers (`subject` aus Zertifikat) als Adressierung

Darüber hinaus können drei weitere Informationen im UpdateComponent enthalten sein:

- Optional einen für einen Benutzer lesbarer Text, um ihm Informationen zum Austausch mitzuteilen
- UMP-ID, die diese Austauschmaßnahme kennzeichnet, um die Verwaltung, Verarbeitung und Zuordnung zu erleichtern
- Absender und Empfänger der Bestätigung des UpdateComponents

Der – von einer unabhängigen Stelle evaluierte – Code mit neuen kryptographischen Komponenten, der in so genannten Providern dem System zur Verfügung gestellt wird, wird im Update Management Protocol nicht mitgeliefert, sondern nur ein Link auf eine URL. Dieser URL – z. B. http oder ftp – hängt der Client auf eine festgelegte Konvention Informationen zu seiner eigenen Systemkonfiguration an. Durch die Systeminformationen, die etwa Betriebssystem, Art des Clients (insbesondere ob von mehreren Benutzern genutzt), Versionsnummer und alle installierten Provider enthält, erhält der Client genau den Code, den er für sein System benötigt.

Wenn also als Link eine http- oder ftp-URL angegeben wird, wie beispielsweise

`http://www.cdc.informatik.tu-darmstadt.de/download/load.php` oder

`ftp://cdc.informatik.tu-darmstadt.de`,

dann könnte für das Betriebssystem Windows NT und den Standard-Provider der Download über

`http://www.cdc.informatik.tu-darmstadt.de/download/load.php?version=Win-NT&provider=Standard` oder `ftp://cdc.informatik.tu-darmstadt.de/NT/Standard/datei.jar`,

erfolgen. Diese Vorgehensweise lässt sich beliebig ausweiten; sie bietet drei Vorteile:

1. Der Umfang des UpdateComponents wird möglichst klein gehalten, was wichtig sein kann, wenn der Update Service viele Zertifikatsinhaber mit dem UpdateComponent versorgt.
2. Netzlast und Belastung des Servers, der für den Download benutzt wird, werden möglichst klein gehalten, weil die Benutzer ihren Code zeitlich versetzt ziehen werden.
3. Der Code für verschiedene Systeme kann diversifiziert werden, so dass jeder Client selbst dafür verantwortlich ist, sich den Code zu ziehen, der für sein System geeignet ist. Die Alternative, dass der Update Service dieses Wissen hat und jedem Client ein für sein System spezifisches UpdateComponent schickt, kann bei vielen Benutzern weniger praktikabel sein.

Falls der Code nicht im Klartext übertragen werden kann, weil beispielsweise Patente geschützt werden sollen, wird der Code verschlüsselt übertragen. Dazu ist dann in den übertragenen Bits angegeben, dass der Code verschlüsselt ist und mit welchem Verfahren und Schlüssel er zu decodieren ist. Da eine für jeden CH individuelle Verschlüsselung mit den öffentlichen Schlüsseln der CHs zu aufwendig ist, sollte in diesem Fall ein spezieller geheimer Schlüssel in der PSE für solche Zwecke abgelegt sein. Da der Code in dieser Arbeit nicht näher spezifiziert wird, wird er

als eine Folge von Bits angesehen, deren Interpretation dem Client oder der Chipkarte überlassen bleibt.

Wie wird ein UpdateComponent abgesichert?

Das UpdateComponent wird von der CA multipel signiert, so dass die CHs die Signaturen mit ihren Sicherheitsankern verifizieren können. Die multiple digitale Signatur setzt sich aus sign^k aus der kompromittierten PKI^k und einem nach Voraussetzung sicheren Verfahren sign^u zusammen. Der Grund ist folgender: Die Idee ist, dass zum Deaktivieren eines Signaturalgorithmus, einer Hashfunktion oder einer Formatierung sowie eines tangierten Signaturverfahrens genau dieses Signaturverfahren zur Absicherung der Deaktivierungsaktion genutzt wird. Ein Angreifer, der ein Verfahren kompromittiert hat, kann somit schlimmstenfalls genau dieses Verfahren deaktivieren. Da zum Absichern der Installation neuer Verfahren ein sicheres Signaturverfahren benötigt wird, muss ein UpdateComponent also multipel signiert werden. Das gleiche gilt für den Austausch eines Sicherheitsankers zu σ^k oder eines asymmetrischen Verschlüsselungsalgorithmus und -verfahrens, das zu σ^k verwandt ist. Im Detail bedeutet dies:

1. Austausch einer kryptographischen Primitive aus sign^k:
   UpdateComponent mit sign^k und sign^u absichern.

2. Austausch einer kryptographischen Primitive, die in keinem Signaturverfahren in keiner Teil-PKI vorkommt, z. B. ein symmetrischer Verschlüsselungsalgorithmus:
   UpdateComponent aus Konsistenzgründen ebenfalls mit zwei – beliebigen – Signaturverfahren absichern.

3. Austausch eines Sicherheitsankers zu σ^k:
   UpdateComponent mit sign^k, welches σ^k enthält, und sign^u absichern.

4. Austausch eines eigenen Schlüsselpaares $(\text{prK}^k, \text{puK}^k)$ zum Signaturalgorithmus σ^k:
   UpdateComponent mit sign^k, welches σ^k enthält, und sign^u absichern.

5. Ausschließliches Deaktivieren einer Primitive aus sign^k:
   Analog zu 1.

6. Ausschließliches Deaktivieren einer Primitive, die in keinem Signaturverfahren vorkommt:
   Analog zu 2.

7. Ausschließliches Deaktivieren eines Sicherheitsankers zu σ^k:
   Analog zu 3.

8. Ausschließliches Deaktivieren eines eigenen Schlüsselpaares zu σ^k:
   Analog zu 4.

9. Ausschließliches Installieren (Hinzufügen) einer Primitive:
   UpdateComponent aus Konsistenzgründen ebenfalls mit zwei – beliebigen – Signaturverfahren absichern.

10. Ausschließliches Installieren (Hinzufügen) eines Sicherheitsankers zu σ^n:
    UpdateComponent mit sign^n, welches σ^n enthält, und einem dazu unabhängigen sign^m absichern.

11. Ausschließliches Installieren (Hinzufügen) eines eigenen Schlüsselpaares zu σ^n:
    UpdateComponent mit sign^n, welches σ^n enthält, und einem dazu unabhängigen sign^m absichern.

Falls die CA dieses UpdateComponent nicht selbst signiert, sondern der Update Service, muss der Update Service von der CA mit zwei speziellen Attributzertifikaten – je eines für PKI^k und PKI^u – für UMP autorisiert werden. Ein Attributzertifikat enthält dazu einen weiteren Eintrag als Extension und ist dem UpdateComponent zusammen mit dem zugehörigen Public-Key-Zertifikat – weil Attributzertifikate keinen Public Key enthalten – beigefügt. Eine solche Extension ist in Appendix A.2 auf Seite 191 nachzulesen.

Ein UpdateComponent enthält damit zusammenfassend folgende Informationen:

UpdateComponent:

- Freier Text
- UMP-ID
- Update Service, Adresse für Bestätigung
- Identität des Zertifikatsinhabers (`subject` aus Zertifikat)
- MessageID
- Zu löschen:
  - Alle Algorithmen zum mathematischen Basisproblem α^k oder zur kryptographischen Primitive β^k
  - Sicherheitsanker zum Algorithmus σ^k
  - Eigene Schlüssel zum Algorithmus σ^k oder η_asym^k, zum angegebenen Verwendungszweck (Key Usage) oder zur angegebenen kritischen Schlüssellänge (Key Length)
- Zu installieren:
  - Algorithmen über Link
  - Sicherheitsanker zum Algorithmus σ^n
  - Eigene Schlüssel zum Algorithmus σ^n oder η_asym^k, zum angegebenen Verwendungszweck (Key Usage) oder zur angegebenen kritischen Schlüssellänge (Key Length)
- Signatur mit $sign^k$ erzeugt, optional mit Zertifikat und Attributzertifikat
- Signatur mit $sign^u$ erzeugt, optional mit Zertifikat und Attributzertifikat

Der Update Service initiiert mit dem UpdateComponent das Update Management Protocol, indem er es an die betroffenen CHs sendet:

- Bei einer Signatur-Anwendung sendet der Update Service das UpdateComponent direkt an die E-Mail-Adressen betroffener Zertifikatsinhaber.
- Bei einer Authentisierungs-Anwendung integriert der Update Service das UpdateComponent in Sperrlisten und Zertifikats-Status-Antworten. Da der CH i. A. nach Ausgabe der Zertifikate keinen Kontakt mehr zum Trust Center hat, muss der Server in einer Client-Server-Authentisierung oder der Zugangskontroll-Client in einer realen Zugangskontrolle die Übermittlung von UpdateComponents übernehmen. Da bei einer Anwendung, die das

Fail-Safe-Konzept nutzt, ein erhöhtes Sicherheitsbedürfnis vorauszusetzen ist, wird angenommen, dass Server und Zugangskontroll-Clients regelmäßig eine Sperrliste abrufen oder den Status eines Zertifikats via OCSP erfragen.
In diesem Fall kann es vorkommen, dass Sperrlisten oder OCSP-Antworten mehrere UpdateComponents enthalten. In Appendix A.3 und A.4 auf den Seiten 193 und 197 ist die entsprechende Syntax nachzulesen.

Die Austauschinformationen können zweimal im UpdateComponent integriert werden: für Software-PSEs und Clients in der für Protokolle üblichen ASN.1-Notation und für ICCs als spezielles Smart-Card-Kommando. ASN.1 steht für Abstract Syntax Notation One. Details finden sich in Appendix B auf Seite 219.

Der Update Service hat den Austausch nun initiiert.

4.5.4 Phase 3: Empfang des UpdateComponents beim Certificate Holder

Dieser Abschnitt behandelt, auf welche Weise ein Certificate Holder ein UpdateComponent empfängt und welche Aktionen anschließend ablaufen. Der Empfang und das beteiligte Equipment variieren in den drei Anwendungen, die in dieser Arbeit stets betrachtet werden – Signatur-Anwendung sowie virtuelle und reale Authentisierungs-Anwendung –, weshalb der Empfang des UpdateComponents in allen drei Varianten ausführlich vorgestellt wird.

Um die Frage zu klären, wie ein Certificate Holder ein UpdateComponent empfängt und welches Equipment beteiligt ist, werden drei Anwendungen betrachtet:

1. In einer Signatur-Anwendung ist der Client ein Mail Client, welcher das UpdateComponent in dem Moment empfängt, in dem der Benutzer die vom Update Service geschickte Mail vom Mail Server abruft. Ist eine Chipkarte involviert, wird ein dem UpdateComponent entsprechendes Chipkarten-Kommando UPDATE COMPONENT COMMAND an die Chipkarte gesendet (Abb. 4.4). Ein Vorschlag, wie dieses UPDATE COMPONENT COMMAND aussehen kann, ist in Appendix B.2 auf Seite 223 abgedruckt.

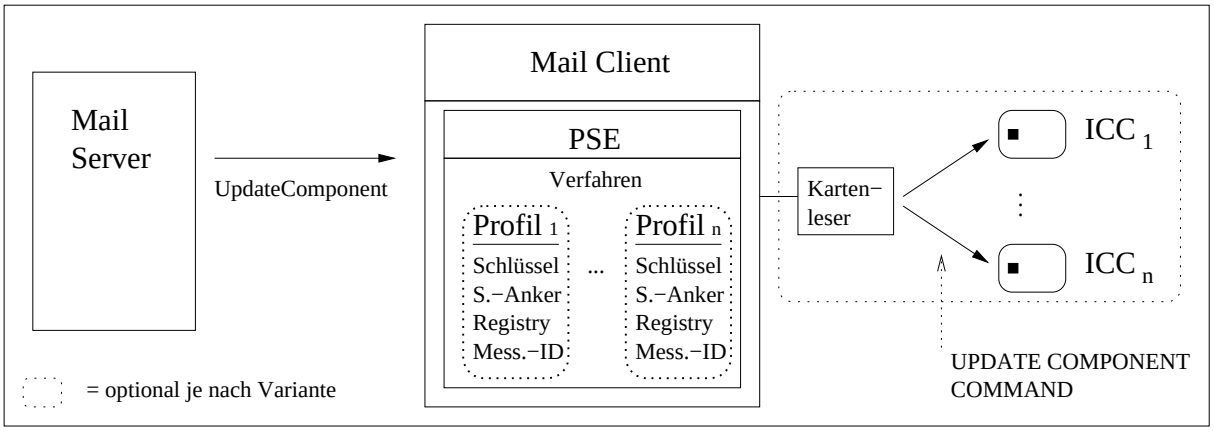

Abbildung 4.4: Empfang des UpdateComponents in einer Signatur-Anwendung

2. In einer virtuellen Zugangskontrolle gibt es zwei Clients, die UpdateComponent empfangen: Rechner S und C, die in Abschnitt 1.3.2 auf Seite 28 eingeführt wurden. Rechner

S, der als Server Daten oder Dienstleistungen zur Verfügung stellt, verfügt über Revokationsinformationen, die er etwa über CRL oder OCSP erhält. In einer Fail-Safe-PKI können diese Revokationsinformationen UpdateComponents enthalten. S empfängt also über diesen Weg das UpdateComponent. Will sich C gegenüber S authentisieren, so stößt C eine Authentisierung an – etwa mit einem "Hello!". S kann C dann das UpdateComponent innerhalb des Authentisierungsprotokolls übertragen. (Abb. 4.5). Ein entsprechender Vorschlag für eine Integration von UMP in das Authentisierungsprotokoll TLS/SSL findet sich im Appendix A.8 auf Seite 214.

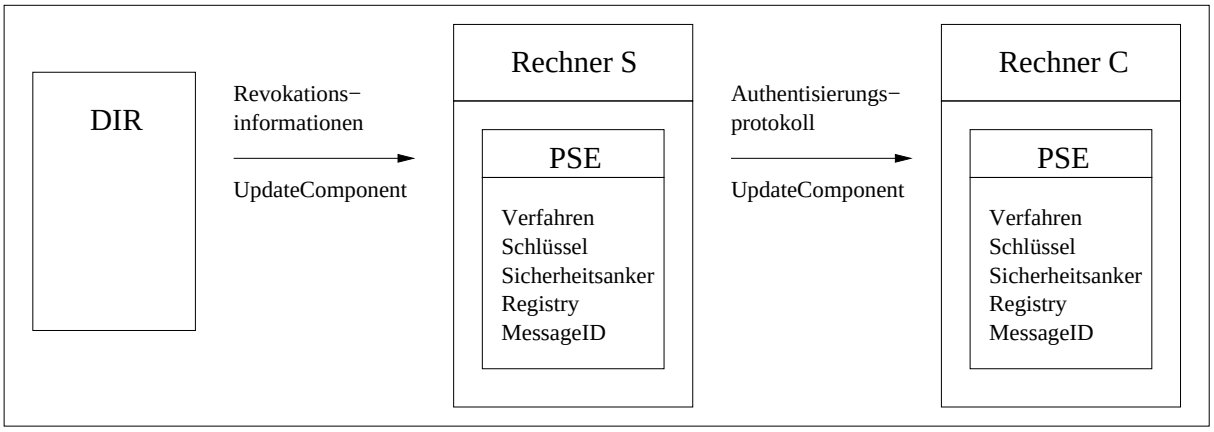

Abbildung 4.5: Empfang des UpdateComponents in einer virtuellen Zugangskontrolle

3. In einer realen Zugangskontrolle verhält es sich ähnlich zur virtuellen: Der Client – ein Zugangskontroll-Client – mit Kartenleser überwacht einen Zugang. Zugangsberechtigte Personen verfügen über entsprechende Chipkarten. Der Client erhält selbst das UpdateComponent zusammen mit Revokationsinformationen. Will sich eine Person mit ihrer Chipkarte authentisieren, so kann der Client ein dem UpdateComponent entsprechendes Chipkarten-Kommando UPDATE COMPONENT COMMAND an die Chipkarte innerhalb des Authentisierungsprotokolls übertragen. (Abb. 4.6). Ein Vorschlag für dieses UPDATE COMPONENT COMMAND ist in Appendix B.2 auf Seite 223 nachzulesen.

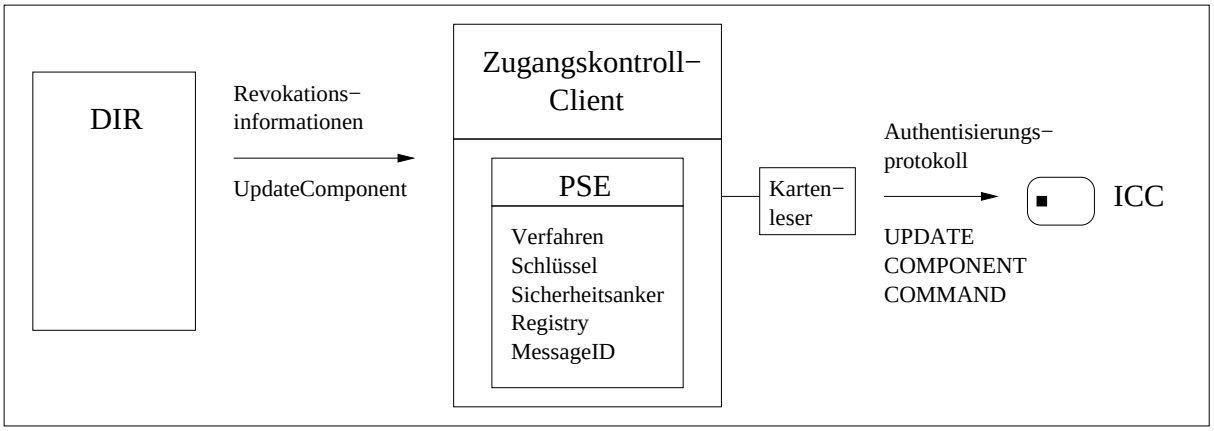

Abbildung 4.6: Empfang des UpdateComponents in einer realen Zugangskontrolle

Wie den vorangegangenen drei Abbildungen zu entnehmen ist, enthalten Software-PSEs in den Clients und Chipkarten neben kryptographischen Verfahren, Schlüsseln und Sicherheitsankern auch MessageID und Registry (siehe Abschnitt 4.5.5 auf Seite 127). Sind mehrere Profile für mehrere Benutzer in einer Software-PSE angelegt worden, so existieren Schlüssel, Sicherheitsanker, MessageID und Registry in jedem Profil für jeden Zertifikatsinhaber individuell.

Nach Empfang des UpdateComponents beim Client – also Mail Client, Rechner S und Rechner C oder Zugangskontroll-Client – verläuft der Ablauf wie folgt:

1. Der Client erkennt das UpdateComponent.
2. Der Client verifiziert eine(!) der Signaturen, um bei Denial-of-Service-Angriffen den Update frühzeitig abzubrechen. Den dazu nötigen Sicherheitsanker findet er in der eigenen PSE oder – falls der Client selbst über keine PSE verfügt – auf einer Chipkarte.
3. Ist diese Signatur korrekt, springt der Client in einen Update Modus zu diesem UpdateComponent, speichert diesen Zustand, das UpdateComponent mit seiner UMP-ID und fährt fort. Andernfalls beendet der Client den Update.
4. Der Client interpretiert das UpdateComponent über die Versionsnummer und verzweigt in eine entsprechende Unterroutine, wenn er diese Version unterstützt, sonst endet der Austausch.
5. Der Client kann einem Benutzer einen optionalen, im UpdateComponent-Feld "Freetext" enthaltenen Text anzeigen. Dieser Text informiert den Benutzer über die bevorstehende Aktion und bietet ihm die Option fortzufahren oder den Austausch später durchzuführen. Ihm sollten auch die Konsequenzen verdeutlicht werden, nämlich, dass er nach einem Austausch an ihn gesendete Kryptogramme u. U. nicht mehr entschlüsseln kann. Den Mail Server als „Tresor" zu benutzen, wird hier – wie in Abschnitt 1.3.2 auf Seite 27 erläutert – nicht thematisiert. Dem Benutzer wird empfohlen, E-Mails, die an ihn verschlüsselt übertragen wurden, vor dem Austausch zu entschlüsseln und unverschlüsselt abzulegen. Nach einem Austausch kann ein Benutzer u. U. auch keine Signaturen mehr verifizieren, was allerdings schon durch Revokation entsprechender Zertifikate unterbunden sein sollte.
6. Fortfahren in Phase 4.

Abbildung 4.7 illustriert den Empfang des UpdateComponents.
Der in Abbildungen abgebildete Kasten "Exit" bedeutet stets, dass der Programmablauf endet.

Was passiert, wenn der geplante Ablauf in Phase 3 nicht wie geplant, sondern durch Störungen unterbrochen wird? Anhand des Ablaufs werden mögliche Störungen und ihre Folgen diskutiert:

1. Was passiert, wenn ein Client kein UpdateComponent empfängt?
   Die Verfügbarkeit bleibt durch PKI^u gewährleistet. Über die ausbleibende Bestätigung mittels UpdateComponentResponse erfährt der Update Service, wenn ein UpdateComponent einen Zertifikatsinhaber nicht erreicht haben sollte. Darüber hinaus kann ein Certificate Holder in einer Signatur-Anwendung via E-Mail sein Trust Center informieren.
2. Was ist, wenn in Phase 3 der Client ein UpdateComponent nicht erkennt, die Signatur negativ verifiziert wird oder die Version nicht unterstützt wird, so dass das System im Zustand "Exit" endet?
   Die Verfügbarkeit bleibt – wie oben – gewährleistet.

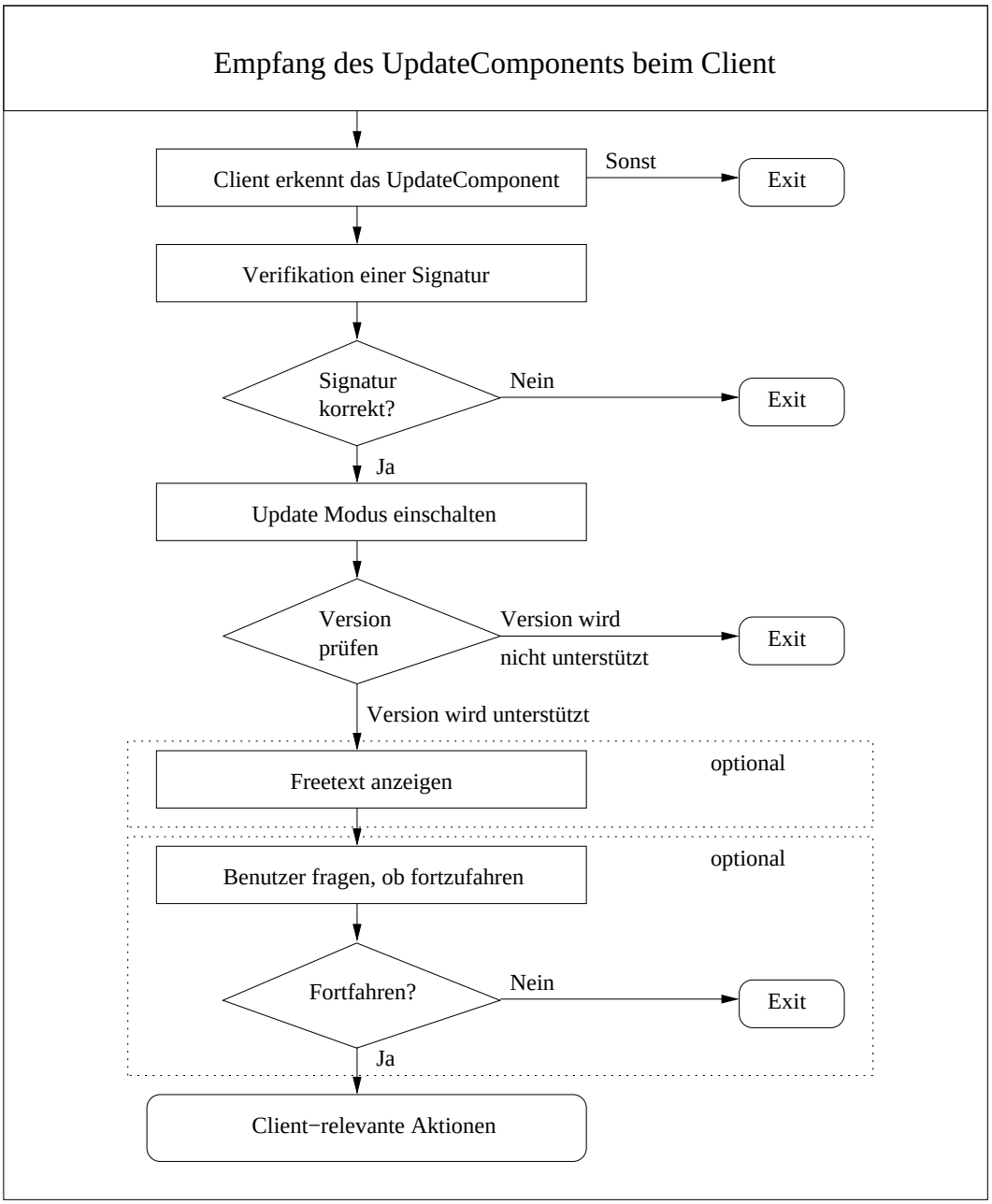

Abbildung 4.7: Phase 3: Empfang des UpdateComponents beim Client

3. Was passiert, wenn der Client nicht in den Update Modus schaltet, z. B. weil der Client falsch konfiguriert ist?
   An Clients, den mehrere Benutzer nutzen – wie etwa im Internet-Café –, könnte dies dazu führen, dass beim ersten Benutzer der Austausch vollständig abläuft, bei allen folgenden aber nicht. Die Benutzer müssten die Administratoren dieses Clients über den Fehler informieren, die dann den Fehler manuell beheben müssen.

4. Was, wenn kein Freetext angezeigt wird?
   Möglicherweise autorisiert der Benutzer daraufhin den Austausch nicht.

5. Was passiert, wenn der Benutzer den Austausch nicht bestätigt?
   Dann wird kein Austausch durchgeführt. Der Certificate Holder muss mit Problemen in der Verfügbarkeit rechnen, wenn die Redundanz-PKI PKI^u ebenfalls ausfallen sollte.

6. Wie reagiert der Ablauf in Phase 3 auf einen Stromausfall oder Rechnerabsturz?
   In diesem Fall kann ein erneuter Anlauf versucht werden.

Bevor mit Client-relevanten Aktionen fortgefahren wird, wird die Sicherheit des UpdateComponents thematisiert.

4.5.5 Sicherheit des UpdateComponents

Angenommen, das UMP initiierende UpdateComponent ist bei der PSE eines Certificate Holders angekommen – also bei der Software-PSE in Mail Client, Rechner S, Rechner C oder Zugangskontroll-Client oder bei einer Chipkarte. Die Verifikation des UpdateComponents unterscheidet sich deutlich von den bisher betrachteten Verifikationen von Signaturen, weil ein UpdateComponent als ein Kommando nicht nur authentisch und integer, sondern auch richtig adressiert und aktuell sein muss. Dadurch werden existenzielle Fälschungen, gefälschte Kommandos, korrekte Kommandos bei falschen Certificate Holdern und Replay-Attacken verhindert.

In diesem Abschnitt wird die Sicherheit des UpdateComponents in fünf Teilen analysiert. Zunächst wird das UMP-Signatur/UMP-Verifikations-Modell für UpdateComponents dargestellt und anschließend für die Sicherheitsbetrachtungen genutzt. In diesen Betrachtungen wird argumentiert, weshalb ein derart abgesichertes UpdateComponent die bei Kommandos üblichen Sicherheitsziele Authentizität, Integrität, korrekte Adressierung und Aktualität erfüllt. Ein für die Sicherheit wesentliches Hilfsmittel ist die Registry des Fail-Safe-Konzepts, die im dritten Teil erläutert wird. Die vom Modell abweichende Realität – Verfahren werden in Providern zusammengefasst und es werden Provider statt einzelne Verfahren ausgetauscht – wird im vierten Teil betrachtet. Und schließlich werden mit dieser Konstruktion vereitelte Angriffe diskutiert.

UMP-Signatur/UMP-Verifikations-Modell

Das Update Management Protocol (UMP) wird auf besondere Weise signiert und auf besondere Weise verifiziert.

Ein UpdateComponent wird mit einer **UMP-Signatur** abgesichert. Eine UMP-Signatur besteht aus zwei digitalen Signaturen, die von zwei voneinander unabhängigen Signaturverfahren erzeugt worden sind, die keine kryptographische Primitive gemeinsam haben, und eine der beiden Signaturen muss mit einem Signaturverfahren aus der vom Schaden kompromittierten Teil-PKI erzeugt worden sein. In Abschnitt 4.5.3 auf Seite 117 werden alle möglichen Fälle diskutiert.

Diese beiden Maßnahmen spiegeln sich in der für ein Akzeptieren eines UpdateComponents notwendigen Verifikation wider – der **UMP-Verifikation**. Die UMP-Verifikation besteht aus festen und variablen Sicherheitsbedingungen, so genannten **Security Conditions**, die für eine positive UMP-Verifikation erfüllt sein müssen. Die festen Security Conditions überwachen, ob das Format eingehalten wird und ob zwei Signaturen vorhanden und mathematisch korrekt sind, während die variablen Sicherheitsbedingungen für die Einhaltung der zuvor geschilderten Maßnahmen sorgen: Benutzung zweier voneinander unabhängiger Signaturverfahren, von denen eines vom Schaden tangiert ist. Die Variabilität ist nötig, weil sich diese Fakten durch neue Erkenntnisse oder neue Algorithmen ändern können. Aus diesem Grund werden die variablen Security Conditions über eine Registry definiert, die auf Seite 127 im Detail erläutert wird.

Das **UMP-Signatur/UMP-Verifikations-Modell** sieht nun wie folgt aus:

Signaturverfahren	$\text{sign}^k, \text{sign}^u \in S$, voneinander unabhängig
Input D = UC	UpdateComponent (UC) kann folgende Einträge enthalten – mindestens jedoch einen: – Deaktivieren aller kryptographischer Verfahren zum mathematischen Basisproblem α^k oder zur kryptographischen Primitive β^k – Installieren der unter der angegebenen URL befindlichen neuen Kryptoverfahren – Löschen der Sicherheitsanker $\text{puK}_{\text{CA}}^{k_1}$, ..., $\text{puK}_{\text{CA}}^{k_m}$ bzw. deren Zertifikate $\text{cert}_{\text{CA}}^{k_1}$, ..., $\text{cert}_{\text{CA}}^{k_m}$ – Installieren von neuen Sicherheitsankern $\text{puK}_{\text{CA}}^{n_1}$, ..., $\text{puK}_{\text{CA}}^{n_m}$ bzw. deren Zertifikate $\text{cert}_{\text{CA}}^{n_1}$, ..., $\text{cert}_{\text{CA}}^{n_m}$ – Löschen der eigenen Schlüsselpaare $(\text{puK}_{\text{CH}}^{k_1}, \text{prK}_{\text{CH}}^{k_1})$, ..., $(\text{puK}_{\text{CH}}^{k_m}, \text{prK}_{\text{CH}}^{k_m})$ mit Angabe von Algorithmus und optional Verwendungszweck und Schlüssellänge – Kommando zum Generieren eigener neuer Schlüsselpaare $(\text{puK}_{\text{CH}}^{n_1}, \text{prK}_{\text{CH}}^{n_1})$, ..., $(\text{puK}_{\text{CH}}^{n_m}, \text{prK}_{\text{CH}}^{n_m})$ geben – unter Angabe von Algorithmus und optional Verwendungszweck und Schlüssellänge – Löschen der alten Registry Reg^{alt} – Speichern der neuen Registry Reg^{neu} – MessageID – UMP-ID – Update Service, Adresse für Bestätigung – Adressierung: Identität des Empfängers
Signierer	CA
UMP-Signatur	$S^k = \text{sign}^k(D, \text{prK}_{\text{CA}}^k)$ $S^u = \text{sign}^u(D, \text{prK}_{\text{CA}}^u)$ $S^* = (D, S^k, S^u)$
S^*	Multipel signiertes UpdateComponent
UMP-Verifikation	$\text{ump\_verify}(D, S^k, S^u)$ = TRUE, falls gilt: Feste Security Conditions: - Format korrekt - Zwei Signaturen vorhanden - $\text{verify}(D, S^k, \text{puK}_{\text{CA}}^k)$ = TRUE - $\text{verify}(D, S^u, \text{puK}_{\text{CA}}^u)$ = TRUE Variable Security Conditions: - sign^k, sign^u sind ein für dieses UpdateComponent geeignetes Paar von Signaturverfahren $\text{ump\_verify}(D, S^k, S^u)$ = FALSE, sonst

Falls die CA nicht für die UMP-Signatur verantwortlich ist, sondern der Update Service als ein vom Trust Center beauftragter Dienstleister mit Attributzertifikaten $\text{att\_cert}_{\text{Update Service}}^k$ und $\text{att\_cert}_{\text{Update Service}}^u$ der CA, welche ihn für diese spezielle Aufgabe autorisiert, und Public-Key-Zertifikaten $\text{cert}_{\text{Update Service}}^k$ und $\text{cert}_{\text{Update Service}}^u$, dann gilt folgendes UMP-Signatur/UMP-Verifikations-Modell; dabei ist das Attributzertifikat mit einer speziellen Extension (siehe Appendix A.2 auf Seite 191) für genau diese Aufgabe ausgestellt:

Input D = UC	wie oben
Signierer	Von der CA autorisierter Update Service
UMP-Signatur	$S^k = \text{sign}^k(D, \text{prK}^k_{\text{Update Service}})$ $S^u = \text{sign}^u(D, \text{prK}^u_{\text{Update Service}})$ $S^* = (D, S^k, S^u)$
S^*	Multipel signiertes UpdateComponent
Attributzertifikate	$\text{att\_cert}^k_{\text{Update Service}}$, $\text{att\_cert}^u_{\text{Update Service}}$
Zertifikate	$\text{cert}^k_{\text{Update Service}}$, $\text{cert}^u_{\text{Update Service}}$
UMP-Verifikation	$\text{ump\_verify}(D, S^k, S^u)$ = TRUE, falls gilt: Feste Security Conditions: - Format korrekt - Zwei Signaturen vorhanden - $\text{verify}(D, S^k, \text{puK}^k_{\text{Update Service}})$ = TRUE - $\text{verify}(\text{cert}^k_{\text{Up. Ser.}}$ mit $\text{puK}^k_{\text{Up. Ser.}}, \text{cert}^k_{\text{Up. Ser.}}$-Signatur, $\text{puK}^k_{\text{CA}})$ =TRUE - $\text{verify}(\text{att\_cert}^k_{\text{Up. Ser.}}$-Inhalt, $\text{att\_cert}^k_{\text{Up. Ser.}}$-Signatur, $\text{puK}^k_{\text{CA}})$ =TRUE - att_cert^k autorisiert den Update Service, D zu signieren - $\text{verify}(D, S^u, \text{puK}^u_{\text{Update Service}})$ = TRUE - $\text{verify}(\text{cert}^u_{\text{Up. Ser.}}$ mit $\text{puK}^u_{\text{Up. Ser.}}, \text{cert}^u_{\text{Up. Ser.}}$-Signatur, $\text{puK}^u_{\text{CA}})$ =TRUE - $\text{verify}(\text{att\_cert}^u_{\text{Up. Ser.}}$-Inhalt, $\text{att\_cert}^u_{\text{Up. Ser.}}$-Signatur, $\text{puK}^u_{\text{CA}})$ =TRUE - att_cert^u autorisiert den Update Service, D zu signieren Variable Security Conditions: - sign^k, sign^u sind ein für dieses UpdateComponent geeignetes Paar von Signaturverfahren $\text{ump\_verify}(D, S^k, S^u)$ = FALSE, sonst

Eine ASN.1-Notation für diese neue Datenstruktur ist in Appendix B.1 auf Seite 219 nachzulesen. Für das Chipkarten-Kommando UPDATE COMPONENT COMMAND sei auf Appendix B.2 auf Seite 223 verwiesen.

Der Code mit den neuen Verfahren, welchen der Client über einen Link herunterlädt, ist mit denselben beiden Verfahren signiert, die auch das UpdateComponent absichern, so dass insbesondere Authentizität und Integrität des Codes gewährleistet sind.

Die Sicherheit des UpdateComponents

Wie kann sich der Empfänger über die Sicherheit eines empfangenen UpdateComponents sicher sein? Entsprechend den Anforderungen an Sicherheit von Chipkarten-Kommandos müssen

- Authentizität,
- Integrität,
- korrekte Adressierung und
- Aktualität

gegeben sein.

Diese vier Eigenschaften sind beim UpdateComponent gegeben:

Die Authentizität und Integrität eines UpdateComponents mit seinen beiden Signaturen S^k und S^u ist dadurch gegeben, dass nach Voraussetzung höchstens eine der beiden Fail-Safe-Konzept-geeigneten Signaturverfahren sign^k und sign^u kompromittiert und daher mindestens eine weiterhin sicher ist.

Die Adressierung eines UpdateComponents an einen bestimmten Certificate Holder kann notwendig sein, wenn ausschließlich sein Schlüssel kompromittiert wurde und er deshalb ein neues Schlüsselpaar generieren soll. Ein entsprechendes UpdateComponent enthält das Kommando, ein neues Schlüsselpaar zu erzeugen und dem Update Service zurückzuschicken. Um den betreffenden Schlüssel beim Certificate Holder genau spezifizieren zu können, wird der zugehörige Algorithmus und der Verwendungszweck (Key Usage) im UpdateComponent eingetragen. Wenn eine Klasse von Schlüsseln kompromittiert sein sollte, die von der Schlüssellänge unabhängig ist – ein Schlüssel dieser Klasse also auf den ersten Blick nicht erkennbar ist –, dann muss jeder Certificate Holder, der einen betroffenen Schlüssel nutzt, ein für ihn individuelles UpdateComponent erhalten. Eine explizite Adressierung entfällt in folgenden Fällen:

- Wenn eine Klasse von Schlüsseln kompromittiert sein sollte, weil die Schlüssellänge zu kurz ist, kann im UpdateComponent eine kritische Schlüssellänge eingestellt werden – die maximale ungenügende Länge; beispielweise 1023, wenn 1024 Bit gefordert wird. Eine Adressierung entfällt, weil Benutzer, welche die kritische Schlüssellänge bereits beachten, ein solches UpdateComponent verwerfen werden.
- Wenn ein Sicherheitsanker kompromittiert sein sollte, entfällt eine Adressierung, weil das UpdateComponent mit dem Sicherheitsanker zu verifizieren ist. Certificate Holder, die diesen Sicherheitsanker nicht nutzen, werden ein negatives Ergebnis der UMP-Verifikation erhalten und dieses UpdateComponent folglich nicht ausführen.
- Wenn ein kryptographisches Verfahren kompromittiert sein sollte, entfällt ebenfalls eine Adressierung, weil das UpdateComponent mit diesem Verfahren zu verifizieren ist und daher die UMP-Verifikation bei den Certificate Holdern, die dieses Verfahren nicht nutzen, mit FALSE ausgehen wird und sie folglich dieses UpdateComponent nicht ausführen werden.

Zur Aktualität: Wird ein kryptographisches Verfahren oder ein Sicherheitsanker im Rahmen des Update Management Protocols durch ein UpdateComponent ausgetauscht, ist ein erneutes Ausführen dieses UpdateComponents – als Replay-Attacke – nicht möglich, weil die UMP-Verifikation dieses UpdateComponents ab dem zweiten Mal mit FALSE ausgehen wird. Der Grund ist, dass die dazu notwendigen kryptographischen Verfahren oder Sicherheitsanker beim ersten Ausführen dieses UpdateComponents bereits entfernt wurden. Um zu verhindern, dass ein UpdateComponent, welches den Austausch des privaten Schlüssels bei einem bestimmten Certificate Holder durchführt, erneut zur Ausführung kommt, wird ein Zähler eingesetzt. In der PSE des Certificate Holders ist für diesen Zertifikatsinhaber eine MessageID gespeichert, die zu Anfang den Wert 0 hat. In das UpdateComponent kann ebenfalls eine MessageID integriert werden, die für den Certificate Holder individuell ist, d. h. das Trust Center bzw. der Update Service muss sich die MessageID von jedem CH merken. In der Verifikation des UpdateComponents wird verglichen, ob eine übertragene MessageID größer als die in der PSE gespeicherte ID ist. Nur wenn dies zutrifft, wird der Austausch durchgeführt und die alte MessageID durch die neue ersetzt. Es wird angenommen, dass ein einfacher Zähler ausreicht, weil ein Überlauf praktisch nicht vorkommen wird: Bei einer 32-Bit-Rechner-Architektur liegt der Überlauf bei $2^{32} - 1$. Dieser Wert würde bei einem Update pro Sekunde erst nach circa 136 Jahren eintreten.

Die Registry des Fail-Safe-Konzepts

Entscheidend für die Sicherheit des UpdateComponents ist, dass S^k und S^u mit zwei voneinander unabhängigen Signaturverfahren ($\text{sign}^k, \text{sign}^u$) erzeugt wurden, dass eines der Signaturverfahren aus der kompromittierten PKI^k stammt und – vor allem – dass die Software-PSE des Clients oder die Chipkarte dies feststellen kann.

Zu diesem Zweck gibt es in jeder Software-PSE eines jeden Clients und jeder Chipkarte eine **Registry**[5] **des Fail-Safe-Konzepts**, in der zu jeder kryptographischen Primitive und jedem Verfahren festgelegt wird, welche Paare von Signaturverfahren im UpdateComponent präsentiert worden sein müssen, um diese Primitive oder dieses Verfahren zu deaktivieren – also die variablen Security Conditions. Darüber hinaus sind in der Registry die Beziehungen von mathematischen Basisproblemen, kryptographischen Primitiven und Verfahren festgehalten, so dass die PSE anhand der Registry feststellen kann, welche kryptographischen Komponenten von der Lösung eines Basisproblems oder der Kompromittierung einer Primitive betroffen sind. Für das ausschließliche Installieren von neuen kryptographischen Verfahren wird ebenfalls eine Security Condition in Form von zwei voneinander unabhängigen Signaturverfahren definiert – freilich ohne die Notwendigkeit, dass ein „kompromittiertes“ Verfahren darunter sein muss.

Die Zusammenstellung der Informationen für diese Registry kann nicht automatisiert werden, stattdessen müssen entsprechende Aussagen zu den mathematischen Basisproblemen und Kryptoalgorithmen nach bestem Wissen im Trust Center zusammengestellt werden, siehe auch Abschnitte 2.1.1, 3.2.1 und 4.1 auf den Seiten 43, 63 und 84.

Die Registry ist sicherheitskritisch. Sie muss daher sicher in der PSE der Certificate Holder abgelegt werden. Da die Registry an neue Verfahren oder neue Erkenntnisse angepasst werden können muss, wird sie dynamisch in der PSE verankert und über das UpdateComponent ggf. aktualisiert. In diesem Fall wird die neue Registry Reg^{neu} multipel signiert übertragen. Bei Multi User Clients, den mehrere Zertifikatsinhaber nutzen, ist der mögliche Wechsel der Registry der Grund dafür, die Registry in den Profilen eines jeden Certificate Holders abzulegen, so dass die verschiedenen Nutzer auch während dieser Austausch-Phase die für sie korrekte Registry nutzen.

Die Registry ist keine(!) Aufstellung der bei einem CH vorhandenen Verfahren. Die Registry ist im Prinzip – außer, während eine Registry ausgetauscht wird – für alle CHs gleich und beinhaltet u. U. auch Verfahren, die ein dedizierter CH nicht hat.

Im Folgenden wird die Registry mit den variablen Security Conditions im Detail erläutert:

Es gibt in der Registry

- m_A mathematische Basisprobleme α^1, ..., α^{m_A}, $m_A \geq 2$, die in
- m_Σ Signaturalgorithmen σ^1, ..., σ^{m_Σ}, $m_\Sigma \geq r$, und
- m_{ξ_asym} asymmetrischen Verschlüsselungsalgorithmen, η_asym^1, ..., $\eta\_\text{asym}^{m_{\xi\_\text{asym}}}$, $m_{\xi\_\text{asym}} \geq 2$,

kryptographisch ausgenutzt werden. Weiter enthält die Registry die folgenden kryptographischen Primitiven:

- m_H Hashfunktionen κ^1, ..., κ^{m_H}, $m_H \geq 2$
- m_F Formatierungen ϕ^1, ..., ϕ^{m_F}, $m_F \geq 2$

[5] Die Registry des Fail-Safe-Konzepts ist nicht mit der Registry aus der Microsoft-Welt zu verwechseln.

- m_{ξ_sym} symmetrische Verschlüsselungsalgorithmen η_sym^1, ..., η_sym$^{m_{\xi_\text{sym}}}$, $m_{\xi\_\text{sym}} \geq 2$

Daraus kombiniert ergeben sich

- m_S Signaturverfahren sign^1, ..., sign^{m_S},
- $m_{\text{E\_asym}}$ asymmetrische Verschlüsselungsverfahren enc_asym^1, ..., enc_asym$^{m_{\text{E_asym}}}$ und
- $m_{\text{E\_sym}}$ symmetrische Verschlüsselungsverfahren enc_sym^1, ..., enc_sym$^{m_{\text{E_sym}}}$.

Die Registry ist mit diesen Bezeichnungen wie folgt aufgebaut:

ID	Basisproblem	Basis für Primitiven	Security Conditions
1	α^1	σ^{i_1}, ..., η_asym^{i_2}, ...	(sign^{j_1}, sign^{j_2}) OR ...
$\vdots$	$\vdots$	$\vdots$	$\vdots$
m_A	α^{m_A}	σ^{i_3}, ..., η_asym^{i_4}, ...	(sign^{j_3}, sign^{j_4}) OR ...

ID	Primitive	Enthalten in Verfahren	Security Conditions
$m_A + 1$	σ^1	sign^{i_5}, ..., enc_asym^{i_6}, ...	(sign^{j_5}, sign^{j_6}) OR ...
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$u + 1$	κ^1	sign^{i_7}, ..., enc_asym^{i_8}, ...	(sign^{j_7}, sign^{j_8}) OR ...
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$u' + 1$	ϕ^1	sign^{i_9}, ..., enc_asym$^{i_{10}}$, ...	(sign^{j_9}, sign$^{j_{10}}$) OR ...
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$v + 1$	η_asym^1	sign$^{i_{11}}$, ..., enc_asym$^{i_{12}}$, ...	(sign$^{j_{11}}$, sign$^{j_{12}}$) OR ...
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$v' + 1$	η_sym^1	sign$^{i_{13}}$, ..., enc_asym$^{i_{14}}$, ...	(sign$^{j_{13}}$, sign$^{j_{14}}$) OR ...
$\vdots$	$\vdots$	$\vdots$	$\vdots$
	Hinzufügen neuer Komponenten	-	(sign$^{j_{15}}$, sign$^{j_{16}}$) OR ...

ID	Zusammengesetzte Verfahren
$w + 1$	sign^1
$\vdots$	$\vdots$
$w' + t$	enc_asym^1
$\vdots$	$\vdots$
$w'' + t$	enc_sym^1
$\vdots$	$\vdots$

Dabei sind sämtliche Indizies natürliche Zahlen: $i_1, ..., i_{14}$, $j_1, ..., j_{16}$ stehen stellvertretend für die entsprechenden Primitiven und Verfahren und der Identifier ID nummeriert Basisprobleme, kryptographische Primitiven und Verfahren durch, um eine Zuordnung zu ermöglichen. Die

Security Conditions enthalten mindestens ein Paar (a, b) aus Signaturverfahren, und jeweils ein Element – o. B. d. A. sei dies a – basiert auf dem mathematischen Basisproblem oder der kryptographischen Primitive, während das andere Element – b – dieses Problem oder diese Primitive gerade nicht enthält.

Ein Beispiel verdeutlicht den Aufbau der Registry.

Beispiel Registry
Das Beispiel von Seite 95 wird fortgeführt. Die Registry hat folgende Gestalt:

ID	Basisproblem	Basis für Primitiven	Security Conditions
1	$\alpha^1 \widehat{=}$ Faktor'problem	σ^1, η_asym^1	(13, 14) OR (15,16)
2	$\alpha^2 \widehat{=}$ DLP (EC)	σ^2, η_asym^2	(14, 13) OR (16,15)

ID	Primitive	Enthalten in Verfahren	Security Conditions
3	$\sigma^1 \widehat{=}$ RSA	sign^1, sign^3	(13, 14) OR (15,16)
4	$\sigma^2 \widehat{=}$ ECDSA	sign^2, sign^4	(14, 13) OR (16,15)
5	η_asym$^1 \widehat{=}$ id-ea-rsa	enc_asym^1	(13, 14) OR (15,16)
6	η_asym$^2 \widehat{=}$ ECIES	enc_asym^2	(14, 13) OR (16,15)
7	$\kappa^1 \widehat{=}$ RIPEMD-160	sign^1, sign^4	(13, 14) OR (16,15)
8	$\kappa^2 \widehat{=}$ SHA-1	sign^2, sign^3	(14, 13) OR (15,16)
9	$\phi^1 \widehat{=}$ PKCS#1	sign^1, sign^3, enc_asym^1	(13, 14) OR (15,16)
10	$\phi^2 \widehat{=}$ id	sign^2, sign^4, enc_asym^2, enc_sym^1, enc_sym^2	-
11	η_sym$^1 \widehat{=}$ DES_3	enc_sym^1	(13, 14) OR (15,16)
12	η_sym$^2 \widehat{=}$ AES	enc_sym^2	(13, 14) OR (15,16)
	Hinzufügen neuer Komponenten	-	(13, 14) OR (15,16)

ID	Zusammengesetzte Verfahren
13	sign$^1 \widehat{=}$ rsasignatureWithripemd160 (mit PKCS#1)
14	sign$^2 \widehat{=}$ ecdsa-with-SHA1
15	sign$^3 \widehat{=}$ sha1WithRSAENC (mit PKCS#1)
16	sign$^4 \widehat{=}$ ecSignWithripemd160
17	enc_asym$^1 \widehat{=}$ RSA encryption (mit PKCS#1)
18	enc_asym$^2 \widehat{=}$ ECIES
19	enc_sym$^1 \widehat{=}$ DES_3CBC_ISOpad
20	enc_sym$^2 \widehat{=}$ id-aes256-ECB

Betrachtungen bzgl. Modell vs. Realität

Bis hierher wurde die Sicherheit der Idee, kryptographische Komponenten durch das Update Management Protocol zu ersetzen, in einer mathematisch motivierten Weise nachgewiesen – allerdings unter der Voraussetzung, dass die einzelnen Komponenten separat auszutauschen sind. Die Realität kann sich von diesem Modell unterscheiden: Die FlexiPKI [FlexiPKI] beispielsweise

nutzt das **Provider-Konzept**, bei dem in einem Provider mehrere kryptographische Verfahren vereinigt sind und mehrere Provider zum Einsatz kommen; d. h. es gibt n Provider P_1, ..., P_n mit

$$P_i \subseteq \{\sigma^1, ..., \sigma^{i_1}, \kappa^1, ..., \kappa^{i_2}, \phi^1, ..., \phi^{i_3}, \eta\_\text{sym}^1, ..., \eta\_\text{sym}^{i_4}, \eta\_\text{asym}^1, ..., \eta\_\text{asym}^{i_5}, \text{sign}^1, ..., \text{sign}^{i_6}, \text{enc\_sym}^1, ..., \text{enc\_sym}^{i_7}, \text{enc\_asym}^1, ..., \text{enc\_asym}^{i_8} \mid i_1, ..., i_8 \in \mathbb{N}\}$$

für $i = 1, ..., n$.

Beispiel
Der in der FlexiPKI eingesetzte `FlexiCoreProvider` enthält beispielsweise RSA, ElGamal, DSA, MD5withRSA, SHA1withRSA, RIPEMD160withRSA, DES, IDEA, MARS, SERPENT, TWOFISH, RIJNDAEL, SaferPlus, MD4, MD5, SHA-1, RIPEMD-128 und RIPEMD-160. Der `FlexiECProvider` enthält z. B. ECDSA, ECElGamal und SHA1withECDSA.

Der Austausch einzelner kryptographischer Verfahren aus einem Provider erfolgt in der Weise, dass ein neuer Provider zur Verfügung gestellt und installiert wird, der sich nur in dem ersetzten Verfahren vom alten Provider unterscheidet: Sei P_alt ein Provider, der über m kryptographische Komponenten component^i, $i = 1, ..., m$, $m \in \mathbb{N}$, $m > 0$, verfügt:

$$P_\text{alt} = \{\text{component}^i \mid i = 1, ..., m\}$$

Angenommen, nur component^k wird kompromittiert. Dann wird ein neuer Provider P_neu aufgebaut, der statt component^k ein neues component^{m+1} enthält:

$$P_\text{neu} = \{\text{component}^i \mid i = 1, ..., k-1, k+1, ..., m+1\}$$

Für den Übergang, während ein Client im Update Modus alte und neue Verfahren vorhält, gibt es einen weiteren Provider:

$$P_\text{Update\_Modus} = \{\text{component}^i \mid i = 1, ..., m+1\}$$

Der Übergangs-Provider $P_\text{Update\_Modus}$ ist wichtig, falls ein Client mehrmals ein UpdateComponent für verschiedene Certificate Holder empfängt und die alten Verfahren für die UMP-Verifikation nötig sind – beispielsweise bei einem Mail Client im Rathaus oder Internet-Café.

Gegenüberstellung einiger möglicher Angriffe und ihrer Abwehrmaßnahmen

Ein nach dieser Beschreibung konstruiertes und abgesichertes UpdateComponent vereitelt beispielsweise folgende Angriffe:

- Ein Angreifer kompromittiert den Signaturalgorithmus σ^k und fälscht dadurch ein UpdateComponent: Beispielsweise entfernt er ein noch sicheres Signaturverfahren sign^u und ersetzt es durch ein eigenes Signaturverfahren sign^n, in das er Hintertüren eingebaut hat. Dieses Vorgehen wird dadurch verhindert, dass das UpdateComponent mit zwei Signaturen abgesichert ist, deren Signaturalgorithmen voneinander unabhängig sind und die – nach Voraussetzung – nicht gleichzeitig kompromittiert werden können. Die Unabhängigkeit der beiden Signaturen wird in der PSE über die Registry gewährleistet. Die Registry ist in der PSE sicher verwahrt und, falls eine neue Registry übertragen wurde, multipel signiert.

- Ein Angreifer kompromittiert κ^k und fälscht ein UpdateComponent. Dieses Vorgehen wird dadurch verhindert, dass das UpdateComponent mit zwei Signaturen abgesichert ist, deren Signaturverfahren zwei voneinander unabhängige Hashfunktionen aufweisen und die – nach Voraussetzung – nicht simultan kompromittiert werden können. Die Unabhängigkeit der beiden Signaturen bei Kompromittierung der Hashfunktion wird in der PSE über die Registry gewährleistet.

- Ein Angreifer kompromittiert den Sicherheitsanker $\mathrm{prK}_{\mathrm{CA}}^{k}$ und fälscht ein UpdateComponent. Dieses Vorgehen wird – wie oben auch – dadurch vereitelt, dass der Angreifer zwei Sicherheitsanker hätte kompromittieren müssen, was nach Voraussetzung nicht simultan möglich ist.

- Ein Angreifer hört ein UpdateComponent vom Update Service zu Alice mit und speichert es. Er will es später erneut Alice zuschicken. Ist in das UpdateComponent eine MessageID integriert, wird der Replay-Angriff dadurch verhindert, dass die übertragene MessageID beim zweiten Mal nicht größer als die lokal bei Alice gespeicherte ist und daher Alice den Austausch nicht ausführen wird. Ist in das UpdateComponent keine MessageID integriert, wird der Replay-Angriff entweder daran scheitern, dass kryptographische Komponenten oder Sicherheitsanker bei Alices PSE bereits ausgetauscht wurden, so dass die UMP-Verifikation mit FALSE ausgehen und Alices PSE das UpdateComponent ablehnen wird, oder dass Alice bereits hinreichend lange Schlüssel nutzt und deswegen das UpdateComponent abweisen wird.

- Ein Angreifer hört ein UpdateComponent vom Update Service zu Alice mit und speichert es. Er schickt es Bob zu. Ist in das UpdateComponent eine Identität zur korrekten Adressierung integriert, wird der Angriff dadurch verhindert, dass Bobs PSE sich als nicht adressiert fühlt und daher UMP nicht ausführen wird. Enthält das UpdateComponent keine Adressierung, gibt es zwei Fälle: Entweder hat Bob die kryptographischen Komponenten oder Sicherheitsanker nicht oder nicht mehr (weil er UMP schon ausgeführt hat), dann wird die UMP-Verifikation mit FALSE ausgehen, oder er hat UpdateComponent noch nicht ausgeführt und tut es jetzt, weil es auch für ihn bestimmt ist.

- Ein Angreifer hat keine Chance, sicherheitskritische Komponenten wie Kryptoverfahren, als Schlüssel oder (Selbst-)Zertifikat realisierte Sicherheitsanker, MessageID oder Registry zu manipulieren, um auf diesem Wege ein manipuliertes UpdateComponent einzuspielen, weil diese Komponenten in der PSE des Clients sicher verwahrt werden, zu der der Angreifer bei Anwendungen mit hoher Sicherheit keinen Zugriff hat. Falls innerhalb eines UpdateComponents eine neue Registry $\mathrm{Reg}^{\mathrm{neu}}$ übertragen wird, so wird sie mit der alten Registry $\mathrm{Reg}^{\mathrm{alt}}$ abgesichert, die für die UMP-Verifikation dieses UpdateComponents nötig ist und welche bis zu ihrem Austausch sicher ist.

- Ein Angreifer kann zwar durch das Verhindern der Ausführung zweier verschiedener UpdateComponents, die zwei voneinander unabhängige Schadensfälle bearbeiten und die – entsprechend der Grundvoraussetzung des Fail-Safe-Konzepts – zeitlich versetzt initiiert wurden, einem Zertifikatsinhaber ein gefälschtes UpdateComponent zuschicken, dessen UMP-Verifikation mit TRUE ausgehen kann. Aber durch die fehlende Bestätigung des CHs, dass das erste UpdateComponent ausgeführt wurde, würde die Certification Authority diesem CH keinen neuen Schlüssel zertifizieren, wodurch der Angreifer keinen Nutzen aus dem Angriff ziehen kann. Die Bestätigung via UpdateComponentResponse ist Thema in Abschnitt 4.5.9 auf Seite 147.

Ein Angreifer kann allerdings, sofern er sich zur PSE eines Zertifikatsinhabers Zugang verschafft hat, ein UpdateComponent, dass auf diesen CH zutrifft, zur Ausführung bringen, so dass Verfahren und Sicherheitsanker ausgetauscht und neue Schlüssel generiert werden können. Für das Ausführen des UpdateComponents ist keine Autorisierung des Benutzers vorgesehen. Der Angreifer kann allerdings keinen Nutzen aus diesem Angriff ziehen: Der Angreifer ist an privaten Schlüsseln des „Opfers“ interessiert und er hat ohne PIN weder Zugriff auf die in der PSE abgelegten privaten Schlüssel noch kann er neu generierte Schlüssel für seine Zwecke nutzen, weil diese Schlüssel nur mit von der CA ausgestellten Zertifikaten einen Nutzen für ihn haben. Diese Zertifikate werden über das UpdateComponentResponse beantragt. Das UpdateComponentResponse ist mit einem noch sicheren, nicht revozierten Schlüssel signiert, auf den der Angreifer ohne PIN keinen Zugriff hat. Darüber hinaus kann der Zertifikatsinhaber seine Zertifikate sperren lassen, wenn er den Angriff bemerkt, was auch dazu führt, dass die CA keine auf diesem Zertifikat basierende Zertifizierung neuer Schlüssel durchführt. Siehe auch Abschnitt 4.5.9 auf Seite 147 zur Sicherheit des UpdateComponentResponses.

Nach der Diskussion der Sicherheit des UpdateComponents wird im Ablauf im Schadensfall fortgefahren: Auf den Empfang des UpdateComponents beim Certificate Holder folgen Client-relevanten Aktionen.

4.5.6 Phase 4: Client-relevante Aktionen

Das UpdateComponent ist beim Certificate Holder angekommen. Wo genau beim CH, wurde in Phase 3 auf Seite 119 erklärt: Ein Client empfängt ein UpdateComponent und dieser “Client“ stellt in einer Signatur-Anwendung einen Mail Client, in einer realen Zugangskontrolle einen Zugangskontroll-Client und in einer virtuellen Zugangskontrolle die beiden Rechner S und C dar. Mit “Client“ ist im Folgenden also Mail Client, Rechner S, Rechner C oder Zugangskontroll-Client gemeint – Chipkarten folgen in Phase 5. In diesem Abschnitt wird der exakte Ablauf im Client in zwei Teilen beschrieben: Zunächst werden die Sicherheitsmerkmale des UpdateComponents geprüft und anschließend der Austausch im Client ausgeführt. Diese Phase 4 findet nur dann statt, wenn im UpdateComponent Client-relevanten Aktionen enthalten sind.

Der Verlauf im Client, nachdem ein UpdateComponent empfangen wurde:

1. Der Client prüft, ob eine Adressierung vorhanden ist. Ist sie vorhanden, und ist sie zur eigenen Identität identisch, wird fortgefahren. Andernfalls bricht der Update im Client ab.

2. Der Client verifiziert, ob eine MessageID vorhanden ist. Ist sie vorhanden, prüft der Client ob die übertragene MessageID größer als die lokal in der PSE des Clients abgespeicherte ID ist. Ist dies der Fall, fährt der Client mit dem Ablauf fort, sonst wird abgebrochen.

3. Der Client führt eine UMP-Verifikation durch:

   (a) Ist das Format korrekt?

   (b) Sind zwei Signaturen vorhanden?

   (c) Sind die beiden Signaturen für dieses UpdateComponent geeignet? Dazu wird die in der PSE des Clients gespeicherte Registry hinzugezogen.

   (d) Sind beide Signaturen mathematisch korrekt? Zur Verifikation werden die in der PSE des Clients abgelegten Sicherheitsanker herangezogen, falls das UpdateComponent von der CA signiert wurde.

(e) Falls der Update Service anstelle der CA das UpdateComponent signiert hat, werden zusätzlich zwei Public-Key-Zertifikate und zwei Attributzertifikate mit dem Sicherheitsanker auf ihre mathematische Korrektheit geprüft. Mit den öffentlichen Schlüsseln der beiden Public-Key-Zertifikate werden die beiden Signaturen des UpdateComponents verifiziert. Darüber hinaus wird geprüft, ob in den Attributzertifikaten eingetragen ist, dass der Update Service für UMP autorisiert ist.

Die UMP-Verifikation ist genau dann TRUE, wenn alle vier bzw. fünf Punkte TRUE sind.

4. Ist die UMP-Verifikation TRUE, wird ein entsprechender Security Status gesetzt, der ein Ausführen des UMPs im Client zulässt. Bei FALSE verlässt der Client den Programmablauf.

5. Es folgt die Phase „Ausführen im Client“.

Der Ablauf bis hierher ist in Abbildung 4.8 graphisch dargestellt.

Was passiert, wenn der geplante Ablauf der Client-relevanten Aktionen in Phase 4 nicht wie geplant, sondern durch eine Störung – etwa einen Stromausfall o. ä. – unterbrochen wird? Es kann ein erneuter Versuch unternommen werden, weil die UMP-Verifikation bei Wiederholung genauso wie beim ersten Mal ausgehen wird.

Unter „Ausführen im Client“ ist folgender Ablauf vorgesehen:

1. Enthält das UpdateComponent Informationen zum Installieren von Algorithmen, so zieht der Client den Code über den im UpdateComponent angegebenen Link, die er um Informationen zu seiner Systemkonfiguration vervollständigt. Unter der URL findet der Client ein multipel signiertes Datenpaket mit einer Menge von Providern, das er lokal abspeichert. Die Verifikation der multiplen Signaturen beinhaltet zwei Prüfungen:

   (a) Sind beide Signaturen mathematisch korrekt?

   (b) Wurden für die beiden Signaturen die gleichen zwei, voneinander unabhängigen Signaturverfahren wie für der UMP-Verifikation genutzt?

   Sind beide Verifikationen TRUE, wird fortgefahren: Ist in der Menge der enthaltenen Provider der Übergangs-Provider `Provider_neu_Update_Modus` enthalten, so wird dieser installiert, andernfalls der enthaltene Provider `Provider_neu`. Wenn der Client exklusiv von nur einem Zertifikatsinhaber genutzt wird, wurde diese Information in die URL mit hinein codiert, so dass dieser CH nur einen `Provider_neu` erhält. Bei Multi User Clients sind die beschriebenen zwei Provider übermittelt worden. Sind Provider verschlüsselt übertragen worden, werden sie zuvor mit einem für diese Zwecke geheimen Schlüssel entschlüsselt. Das Deinstallieren findet hier noch nicht statt!

2. Enthält das UpdateComponent einen neuen Sicherheitsanker, so wird dieser in der PSE des Clients installiert – genauer im Profil des Zertifikatsinhabers, falls Profile mehrerer Zertifikatsinhaber angelegt sind. Ein Löschen des Sicherheitsankers findet hier noch nicht statt!

3. Enthält das UpdateComponent das Kommando, einen neuen Schlüssel zu generieren, so wird für den Zertifikatsinhaber in der Software-PSE des Clients ein neues Schlüsselpaar generiert. Der private Schlüssel wird im geheimen, nicht zugänglichen Speicherbereich abgelegt und der öffentliche Schlüssel in einem zugänglichen Bereich.

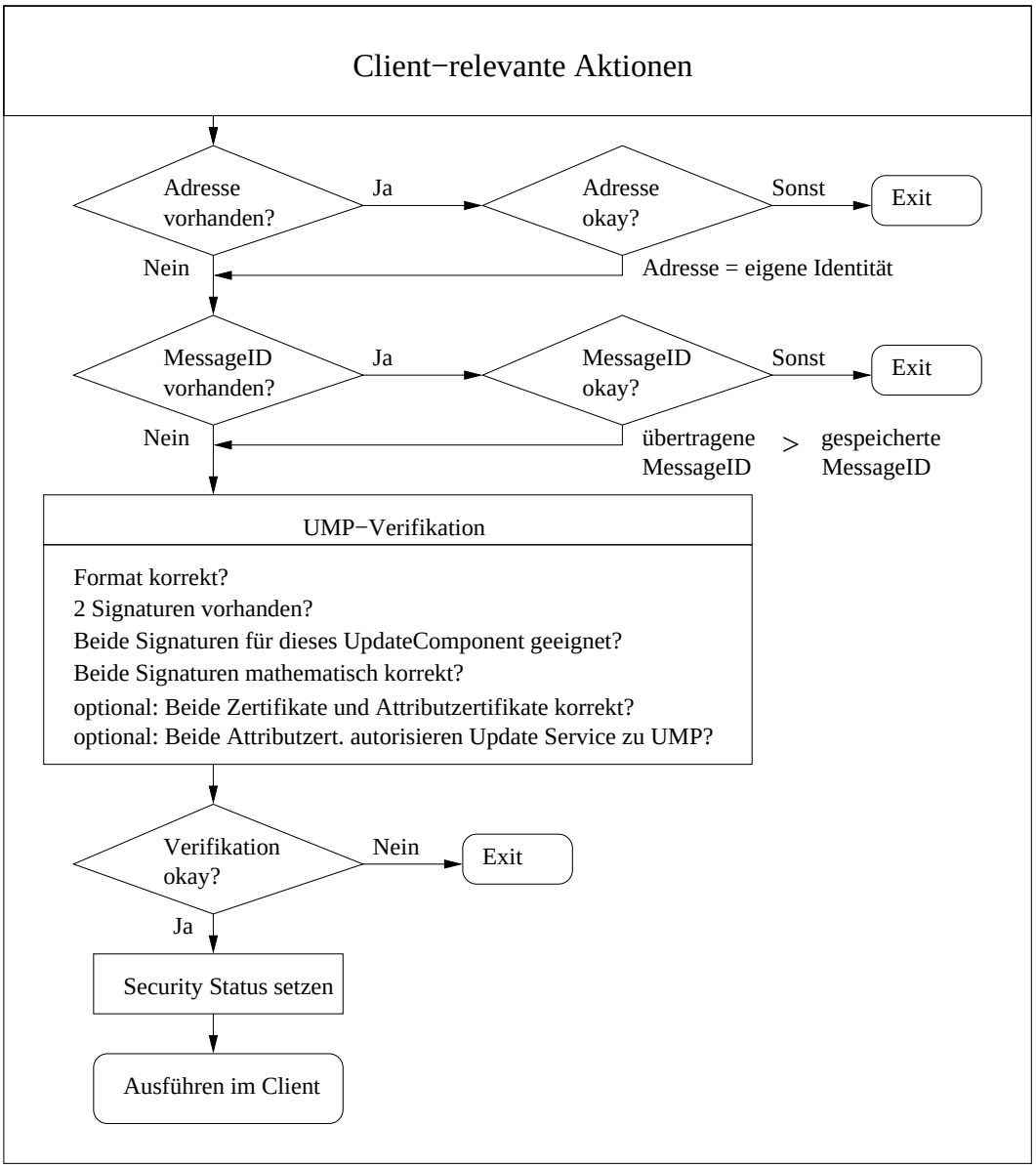

Abbildung 4.8: Phase 4: Client-relevante Aktionen

4. Enthält das UpdateComponent eine neue Registry Reg^{neu}, so wird diese in der PSE des Clients zusätzlich installiert; genauer im Profil des Zertifikatsinhabers, falls Profile mehrerer Zertifikatsinhaber angelegt sind. Der Client muss sich merken, dass bis zum Löschen der alten Registry Reg^{alt}, diese die aktuelle Registry ist. Ein Löschen der alten Registry findet an dieser Stelle noch nicht statt!

5. Ist das Installieren erfolgreich abgeschlossen, werden in der PSE des Clients zu löschende Provider entfernt und im Profil des Zertifikatsinhabers zu löschende Komponenten – wie Schlüsselpaare, Sicherheitsanker und die Registry – deaktiviert, die neue Registry aktiviert und eine im UpdateComponent übertragene MessageID anstelle der gespeicherten MessageID abgelegt.

Die alten Verfahren sind über den Übergangs-Provider `Provider_neu_Update_Modus` weiterhin verfügbar. Der Grund ist, dass in einer Signatur-Anwendung, in der mehrere Benutzer einen Client nutzen – etwa in einem Internet-Café –, dieser Client die UpdateComponents der nachfolgenden Benutzer auch noch verifizieren können muss, wozu er die alten Verfahren benötigt.
Dieser Prozess muss atomar ausgeführt werden, d. h. ganz oder gar nicht.

6. Fortsetzung in Phase 5.

Abbildung 4.9 illustriert das Ausführen im Client.

Wie schon in Punkt 1b auf Seite 70 argumentiert, gibt es Fälle, in denen auf ein Key Recovery nicht verzichtet werden kann und in denen die CA ebenfalls die privaten Schlüssel der Zertifikatsinhaber kennen muss. Da ein Übertragen des privaten Schlüssels nicht möglich ist, müssen CH und CA separat den selben Schlüssel generieren. Die notwendige Synchronisierung wird dadurch erreicht, dass beide über den gleichen Pseudozufallszahlengenerator und die gleichen Daten verfügen und somit den gleichen privaten Schlüssel erzeugen können. Durch die Zertifizierung in Phase 6 findet eine Kontrolle statt.

Welche PIN hat ein neu generierter Schlüssel? Die Idee ist, dass das 0-PIN-Verfahren eingesetzt wird: Bei der ersten Benutzung des Schlüssels lautet die PIN “0000“ und das System verlangt vom Benutzer eine Änderung der PIN.

Wie reagiert das System auf eine Störung während des Austauschs in Phase 4? In welchem Zustand verharrt das System bei Stromausfall oder Rechnerabsturz? Kann ein erneuter Austausch durchgeführt werden oder sind die für die UMP-Verifikation nötigen Verfahren und Sicherheitsanker bereits gelöscht oder die neue MessageID bereits gesetzt? Der Ablauf ist so gestaltet, dass zunächst nur neue Provider, neue Sicherheitsanker und eine neue, noch nicht aktivierte Registry installiert werden, ohne die alten Komponenten zu deinstallieren. Ebenso wird ein neues Schlüsselpaar nur generiert – das alte nicht gelöscht. Erst wenn die Installation erfolgreich abgeschlossen ist, werden atomar alte Provider, Schlüssel, Sicherheitsanker und Registry gelöscht, die neue Registry aktiviert und die MessageID aktualisiert. Eine Unterbrechung zu einem beliebigen Zeitpunkt vor der Vollendung führt dazu, dass bei einem erneuten Versuch des Austauschs die UMP-Verifikation stets dasselbe Ergebnis liefern wird, weil alte Komponenten erst dann gelöscht werden, wenn die neuen Komponenten erfolgreich installiert wurden. Falls in einem UpdateComponent, das zum Schutz vor Replay-Attacken mit einer MessageID versehen ist, nur das Kommando zum Generieren eines Schlüsselpaares gegeben wird, kann bei einer Störung, die direkt nach Löschen der alten Schlüsselpaare und vor Aktualisieren der MessageID auftritt, eine Wiederholung gelingen. Auf eine Störung nach der atomaren Aktion muss für den betreffenden Zertifikatsinhaber u. U. ein neues UpdateComponent mit neuer UMP-Absicherung und neuer MessageID gesendet werden.

Anschließend folgen Chipkarten-relevante Aktivitäten.

4.5.7 Phase 5: ICC-relevante Aktionen

Der Austausch von Komponenten erstreckt sich auch auf Chipkarten in einer Signatur-Anwendung oder einer realen Zugangskontrolle. Zu diesem Zweck befindet sich im UpdateComponent, das der Client empfängt, ein Link, unter dem der Client das für eine konkrete Chipkarte geeignete UPDATE COMPONENT COMMAND findet. Dieser Abschnitt behandelt, wie der Client

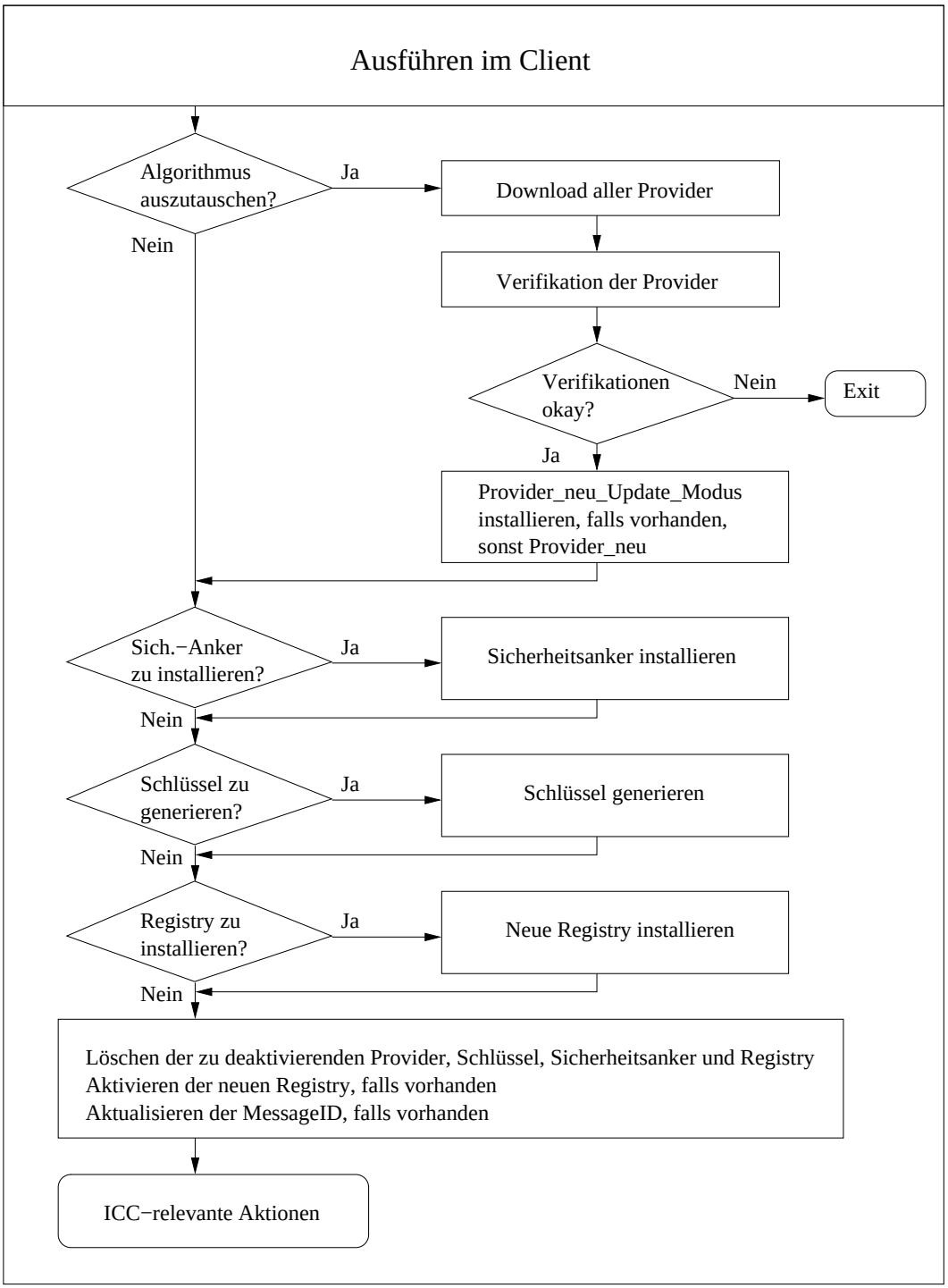

Abbildung 4.9: Phase 4: Ausführen im Client

das UpdateComponent entsprechende Chipkarten-Kommando UPDATE COMPONENT COMMAND erhält und es zur Chipkarte schickt und welche Verifikationen und Aktionen die Chipkarte ausführt. Dabei steht die Chipkarte (Integrated Circuit Card – ICC) – wie in Abschnitt 1.3.1 auf Seite 18 erwähnt – als Synonym für jegliche Hardware-PSEs. Diese Phase 5 findet nur

dann statt, wenn Chipkarten involviert sind.

Angenommen, der Client ist in einer Signatur- oder realen Authentisierungs-Anwendung so konfiguriert, dass er Chipkarten bearbeitet und also einen im UpdateComponent enthaltenen Link für Chipkarten-spezifische Informationen auswertet. Wie stößt der Client ICC-relevante Aktionen an? Abbildung 4.10 illustriert, dass

1. der Client verifiziert, ob im für den Client interpretierbaren, ASN.1-codierten UpdateComponent eine Adressierung vorhanden ist. Ist dem so, überprüft der Client, ob die Adressierung auf die aktuelle Chipkarte zutrifft. Stimmen Adresse und Identität der Karte überein, wird fortgefahren; andernfalls bricht der Programmablauf ab.

2. Der Client lädt unter dem im UpdateComponent angegebenen Link zusammen mit den Systeminformationen der Chipkarte die ICC-relevanten Informationen herunter, die für diese Chipkarte geeignet sind. Diese ICC-relevanten Informationen sind als Chipkarten-Kommando codiert. Ein Vorschlag für dieses UPDATE COMPONENT COMMAND ist in Appendix B.2 auf Seite 223 beschrieben.

3. Die ICC-relevanten Informationen werden anschließend an die Chipkarte gesendet, wo

4. mit „Verifizieren und Ausführen in der Chipkarte“ fortgefahren wird.

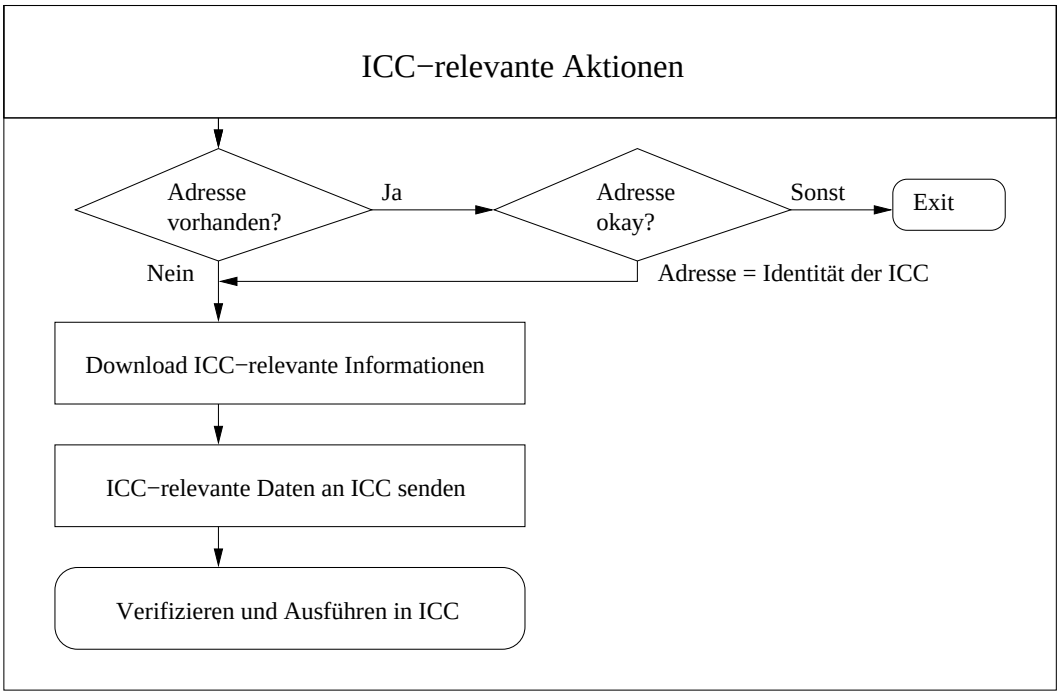

Abbildung 4.10: Phase 5: ICC-relevante Aktionen

Falls in Phase 5 eine Störung auftreten sollte, während der Client ICC-relevanten Informationen lädt und an die Chipkarte sendet, kann eine Wiederholung stattfinden.

Ist das UPDATE COMPONENT COMMAND bei der Chipkarte angekommen, werden folgende Schritte durchgeführt, die sich am Ablauf im Client aus Phase 4 orientieren:

1. Die Chipkarte prüft, ob eine Adressierung vorhanden ist. Ist sie vorhanden, und ist sie zur eigenen Identität identisch, wird fortgefahren, ansonsten wird abgebrochen.
2. Die Chipkarte verifiziert, ob eine MessageID vorhanden ist. Ist sie vorhanden, prüft die Chipkarte, ob die übertragene MessageID größer als die lokal abgespeicherte ID ist. Nur wenn dies der Fall ist, fährt die Chipkarte mit dem Ablauf fort.
3. Die Chipkarte führt eine UMP-Verifikation des UPDATE COMPONENT COMMANDs durch:
   (a) Ist das Format korrekt?
   (b) Sind zwei Signaturen vorhanden?
   (c) Sind die beiden Signaturen für dieses UPDATE COMPONENT COMMAND geeignet? Dazu wird die in der Chipkarte gespeicherte Registry hinzugezogen.
   (d) Sind beide Signaturen mathematisch korrekt? Zur Verifikation werden die in der Chipkarte abgelegten Sicherheitsanker herangezogen, falls das UPDATE COMPONENT COMMAND von der CA signiert wurde.
   (e) Falls der Update Service anstelle der CA das UPDATE COMPONENT COMMAND signiert hat, werden zusätzlich zwei Public-Key-Zertifikate und zwei Attributzertifikate mit dem Sicherheitsanker auf ihre mathematische Korrektheit geprüft. Mit den öffentlichen Schlüsseln der beiden Public-Key-Zertifikate werden die beiden Signaturen des UPDATE COMPONENT COMMANDs verifiziert. Darüber hinaus wird geprüft, ob in den Attributzertifikaten eingetragen ist, dass der Update Service für UMP autorisiert ist. Diese Zertifikate müssen für Chipkarten interpretierbar sein und sollten deshalb als Card Verifiable (CV)-Zertifikat ausgelegt sein, siehe A.2 auf Seite 191.

   Die UMP-Verifikation ist genau dann TRUE, wenn alle vier bzw. fünf Punkte TRUE sind.
4. Ist die UMP-Verifikation TRUE, wird in der Chipkarte der Security Status gesetzt, der den Update von Code, Schlüsseln und Zertifikaten zulässt.
5. Es folgt die Phase „Ausführen in ICC“.

Dieser Ablauf ist in Abbildung 4.11 graphisch dargestellt.

Die nun folgende Phase „Ausführen in ICC“ verläuft analog zur Phase „Ausführen im Client“ auf Seite 133:

1. Enthält das UPDATE COMPONENT COMMAND Informationen zum Installieren von Algorithmen, so werden die mitgelieferten Provider installiert. Falls der Code verschlüsselt ist, wird der zuvor mit einem für diese Zwecke auf der Chipkarte abgespeicherten geheimen Schlüssel entschlüsselt.
2. Enthält das UPDATE COMPONENT COMMAND einen neuen Sicherheitsanker, so wird dieser in der Chipkarte installiert. Ein Löschen des Sicherheitsankers findet hier noch nicht statt!
3. Enthält das UPDATE COMPONENT COMMAND das Kommando, einen neuen Schlüssel zu generieren, so wird in der Chipkarte ein neues Schlüsselpaar generiert. Der private Schlüssel wird im geheimen, nicht zugänglichen Speicherbereich abgelegt und der öffentliche Schlüssel in einem zugänglichen Bereich.

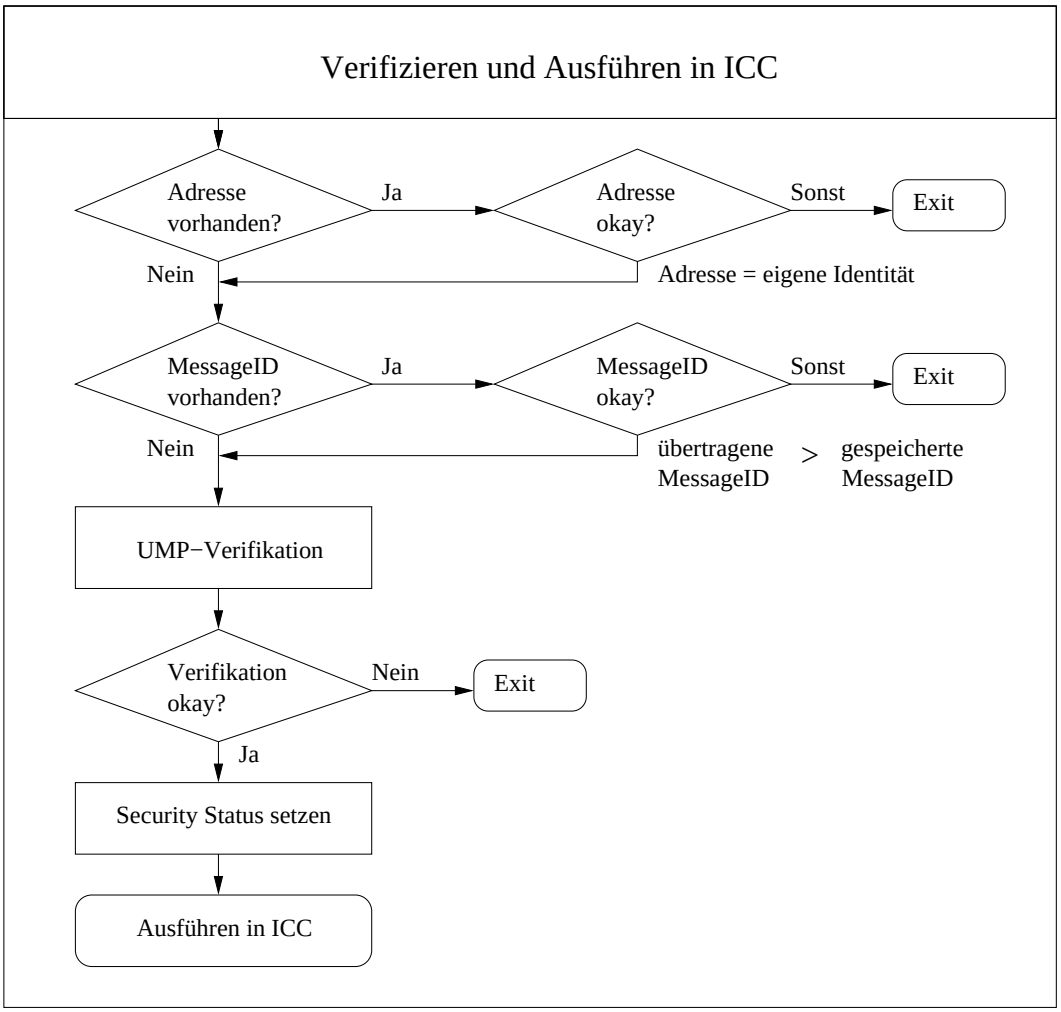

Abbildung 4.11: Phase 5: Verifizieren und Ausführen in ICC

4. Enthält das UPDATE COMPONENT COMMAND eine neue Registry, so wird diese in der Chipkarte installiert, aber noch nicht aktiviert. Ein Löschen der alten Registry findet hier noch nicht statt!

5. Ist das Installieren erfolgreich abgeschlossen, werden zu löschende Provider, Sicherheitsanker, Schlüsselpaare und/oder die alte Registry nun gelöscht, die neue Registry aktiviert und eine im UPDATE COMPONENT COMMAND übertragene MessageID anstelle der in der Chipkarte gespeicherten MessageID abgelegt. Dieser Prozess muss atomar, d. h. entweder vollständig oder gar nicht, ausgeführt werden.

6. Fortsetzung in Phase 6.

Graphisch lässt sich das Verhalten der Chipkarte zum Update wie in Abbildung 4.12 dargestellt verdeutlichen.

Analog zum Client ist es möglich, das „Generieren“ eines privaten Schlüssels in der Chipkarte so zu gestalten, dass über eine Synchronisierung ein Key Recovery möglich ist; siehe Abschnitt

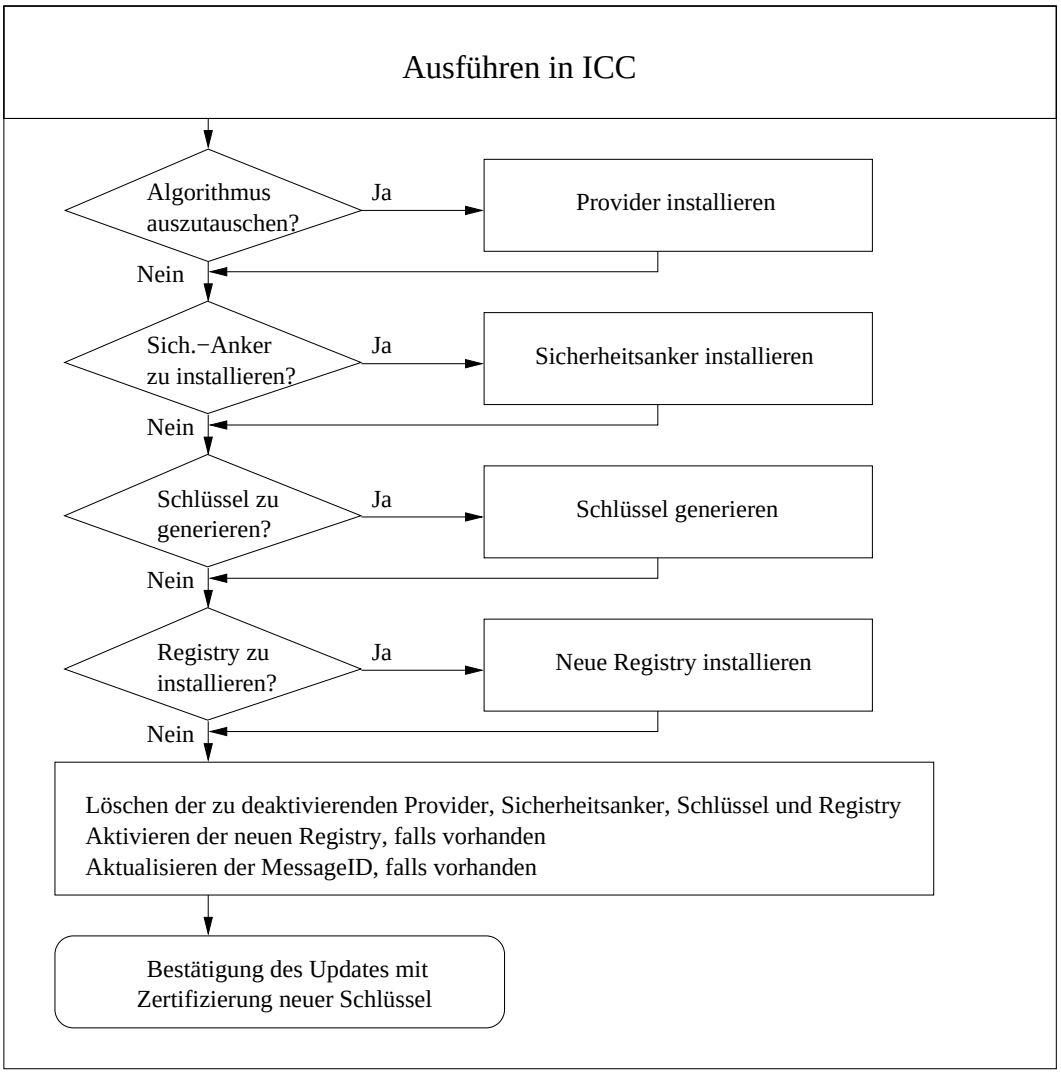

Abbildung 4.12: Phase 5: Ausführen in ICC

4.5.6 auf Seite 135.

Die PIN eines neu generierter Schlüssels wird auf “0000“ eingestellt und ist bei der ersten Benutzung zu ändern (0-PIN-Verfahren).

Was passiert wenn beim Verifizieren und Ausführen in der Chipkarte in Phase 5 eine Störung stattfindet– etwa durch Stromausfall oder Ziehen der Chipkarte aus dem Kartenleser? Wie schon in Phase 4 argumentiert, wird nach einer Störung in Phase 5 ein erneuter Versuch gelingen, denn aufgrund der Konstruktion werden alte Komponenten erst dann deinstalliert, wenn neue erfolgreich installiert sind.

An Phase 5 schließt sich Phase 6 mit der Bestätigung und der Zertifizierung neuer Schlüssel an.

4.5.8 Phase 6: Bestätigung des Updates mit Zertifizierung neuer Schlüssel

Der Certificate Holder bestätigt das erfolgreiche Austauschen von Komponenten dem Update Service und überträgt neu generierte öffentliche Schlüssel, so dass der Update Service sie zur

Zertifizierung an die Certification Authority weiterleiten kann. Falls der Austausch nicht erfolgreich durchgeführt werden konnte, wird Phase 6 nicht durchlaufen und keine Bestätigung ausgestellt. Der exakte Ablauf wird in diesem Abschnitt in vier Teilen erläutert: Zunächst wird der Kommunikationsweg der Bestätigung vom CH zum Update Service für die drei Anwendungen – Signatur-Anwendung sowie virtuelle und reale Authentisierungs-Anwendung – konkretisiert. Anschließend wird der exakte Inhalt der Bestätigung sowie seine Absicherung vorgestellt. Daran schließt sich die Darstellung des Ablaufs an, so dass neu generierte Schlüssel letztlich in der CA ankommen.

Zunächst: Welche Szenarien werden betrachtet und welches Equipment ist beteiligt? Entsprechend zum Empfang des UpdateComponents in Phase 3 auf Seite 119 werden drei Anwendungen betrachtet:

1. Signatur-Anwendung mit Mail Client (und optionalem PSE) und optionaler Chipkarte:
   Ein Mail Server übermittelt via E-Mail das UpdateComponent an den Mail Client, wo das UpdateComponent ausgeführt wird, sofern Verfahren, Sicherheitsanker oder Zertifikate in einer PSE im Client vorhanden sind. Optional wird einer Chipkarte das UpdateComponent entsprechende UPDATE COMPONENT COMMAND zugeschickt, so dass auch dort der Austausch ausgeführt werden kann.

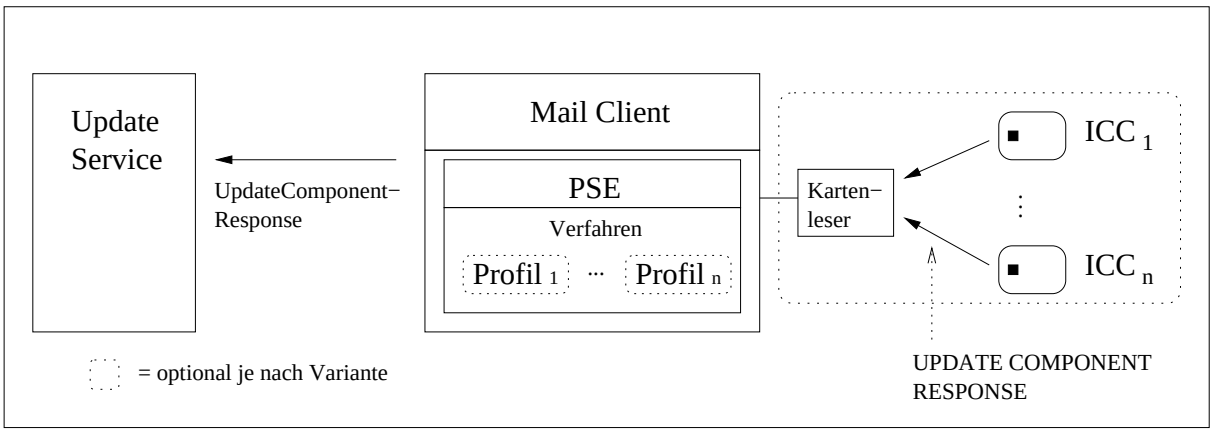

Abbildung 4.13: Bestätigung des UpdateComponents in einer Signatur-Anwendung

Je nachdem, ob Komponenten bei Client, Chipkarte oder beiden ausgetauscht wurden, gibt es für eine Bestätigung drei Möglichkeiten:

(a) Der CH verfügt ausschließlich über eine Chipkarte, die er an einem für ihn anonymen Mail Client nutzt:
Die Chipkarte sendet als Bestätigung ein UPDATE COMPONENT RESPONSE an den Mail Client. Der Client erzeugt ein UpdateComponentResponse, in das er ausschließlich das RESPONSE der Karte integriert, und schickt es via E-Mail an den Update Service.

(b) Der CH verfügt über eine Chipkarte und einen auf ihn personalisierten Mail Client, in dem ebenfalls Komponenten ausgetauscht wurden:
Die Chipkarte sendet als Bestätigung ein UPDATE COMPONENT RESPONSE an den Mail Client. Der Client, der selber einen Austausch durchgeführt hat, ergänzt

das RESPONSE der ICC um seine eigene Bestätigung, erzeugt daraus ein UpdateComponentResponse und schickt es via E-Mail an den Update Service.

(c) Der CH verfügt ausschließlich über einen auf ihn personalisierten Mail Client mit Software-PSE:
Der Mail Client erzeugt aus seiner Bestätigung ein UpdateComponentResponse und schickt es via E-Mail an den Update Service.

Pro Bestätigung eines CHs wird ein(!) UpdateComponentResponse generiert. Falls der CH über Client und Chipkarte verfügt und zwei Updates durchgeführt hat, enthält das UpdateComponentResponse zwei Bestätigungen – einmal die des Clients direkt im UpdateComponentResponse und darin integriert das UPDATE COMPONENT RESPONSE der Chipkarte. Abbildung 4.13 illustriert dieses Szenario.

2. Virtuelle Zugangskontrolle mit den beiden Rechnern S und C:
Ein Rechner S, der als Server fungiert, erhält über Revokationsinformationen das UpdateComponent. S führt das UpdateComponent zunächst bei sich selbst aus und versucht anschließend, es in jeder folgenden Authentisierung an Rechner C – als Clients – abzusetzen. S schickt eine Bestätigung für sich selbst und eine Bestätigung für jeden C als UpdateComponentResponse via E-Mail direkt an den Update Service. Abbildung 4.14 zeigt dieses Szenario graphisch.

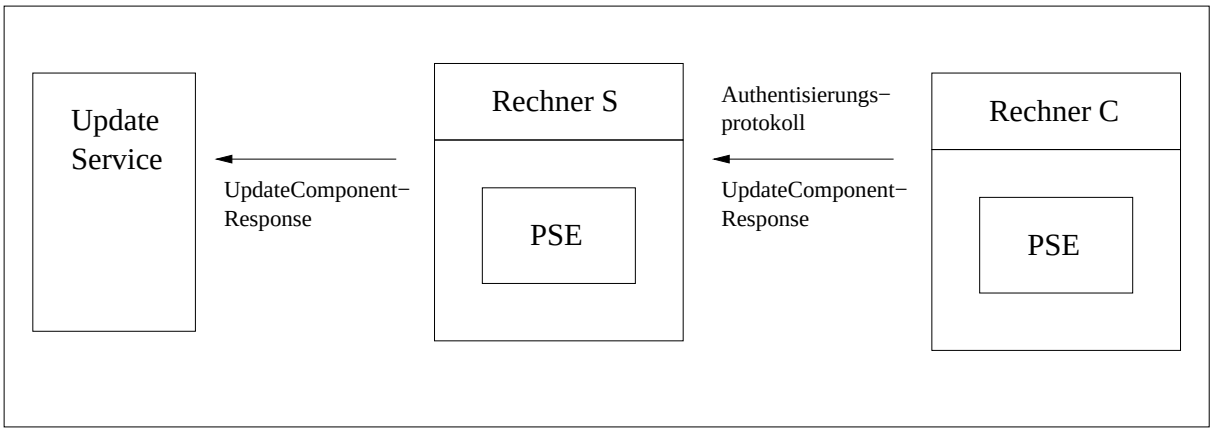

Abbildung 4.14: Bestätigung des UpdateComponents in einer virtuellen Zugangskontrolle

3. Reale Zugangskontrolle mit Zugangskontroll-Client (und Software-PSE) und Chipkarten:
Der Zugangskontroll-Client erhält das UpdateComponent über Revokationsinformationen. Der Client führt das UpdateComponent zunächst bei sich selbst aus und versucht anschließend, das UPDATE COMPONENT COMMAND in jedem folgenden Authentisierungsprotokoll an Chipkarten abzusetzen. Die Bestätigung des Zugangskontroll-Clients sowie die UPDATE COMPONENT RESPONSEs einer jeden Chipkarte schickt der Client in je einem UpdateComponentResponse via E-Mail direkt an den Update Service. In Abbildung 4.15 ist diese Situation dargestellt.

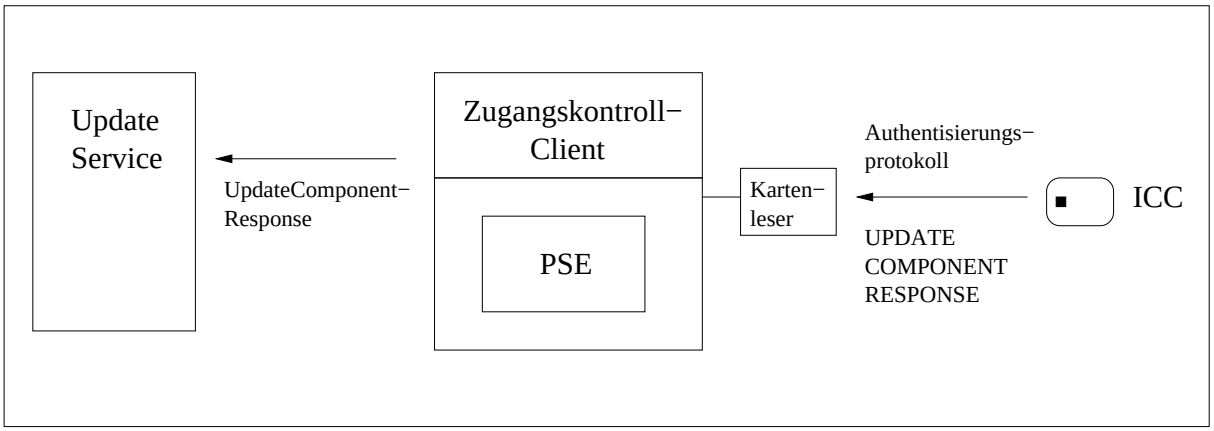

Abbildung 4.15: Bestätigung des UpdateComponents in einer realen Zugangskontrolle

Was enthält das UpdateComponentResponse? Das UpdateComponentResponse hat zwei Funktionen

1. als Bestätigung für das erfolgreiche Ausführen eines UpdateComponents beim Certificate Holder und
2. als Transportmittel für beim Certificate Holder neu generierte und von der Certification Authority zu zertifizierende Schlüssel.

Für die Bestätigung wird in das UpdateComponentResponse bzw. in das UPDATE COMPONENT RESPONSE stets die Identität des Zertifikatsinhabers und die UMP-ID des UpdateComponents bzw. UPDATE COMPONENT COMMANDs integriert; dabei stammt die Identität aus dem `subject`-Eintrag des Zertifikats. Die UMP-ID stellt die Verbindung zwischen UMP-Request und -Response her. Die MessageID ist beigefügt, sofern das UpdateComponent sie enthielt.

Darüber hinaus können optional neu generierte Schlüssel zur Zertifizierung transportiert werden. Dabei ist folgendes zu beachten: Es können – je nach Anforderung im UpdateComponent bzw. UPDATE COMPONENT COMMAND – beliebig viele Schlüssel generiert werden. Allerdings wird für die folgenden Betrachtungen von nur einem Schlüssel ausgegangen, weil sich die Überlegungen sukzessiv auf weitere Schlüssel anwenden lassen. Die PSE generiert also auf die Anforderung hin ein neues Schlüsselpaar $(\mathrm{prK}^m_{\mathrm{CH}}, \mathrm{puK}^m_{\mathrm{CH}})$ zum Algorithmus σ^m oder η_asym^m, wobei m für einen neu installierten Algorithmus oder einen noch sicheren Algorithmus steht. Die PSE legt den privaten Schlüssel $\mathrm{prK}^m_{\mathrm{CH}}$ in einem unzugänglichen Speicherbereich ab, so dass nur die PSE beim Erzeugen einer digitalen Signatur oder beim Entschlüsseln eines Kryptogramms darauf zugreifen kann, und den öffentlichen Schlüssel $\mathrm{puK}^m_{\mathrm{CH}}$ in einen zugänglichen Speicher.

Im Falle einer Chipkarte ist jedoch folgendes zu beachten: In einer Signatur-Anwendung kann es aus rechtlichen oder beweistechnischen Gründen notwendig sein zu beweisen, dass eine Signatur von einer Chipkarte und nicht von einer Software-PSE erzeugt wurde. Daher muss sich bei dem hier vorgestellten Austausch des privaten Schlüssels die Zertifizierungsinstanz davon überzeugen können, dass der neu generierte Schlüssel wirklich in der Chipkarte erzeugt wurde. Es soll verhindert werden, dass ein Schlüssel statt in der Chipkarte in der Software-PSE generiert, in ein UpdateComponentResponse integriert und von der Chipkarte (zweimal) signiert wird. Diese Tatsache kann auch in Authentisierungs-Anwendungen wichtig sein. Deshalb ist es

in diesen Fällen notwendig, eine Information mit dem neuen Schlüssel zu verknüpfen, die ausschließlich die Chipkarte erzeugen kann. Die Lösung ist ein Geheimnis g_{ICC} – ein hinreichend langer Schlüssel –, welches ausschließlich Zertifizierungsinstanz und Chipkarte kennen. Um die Sicherheit zu erhöhen, bestimmt die CA pro Chipkarte ein Geheimnis g_{ICC}, merkt sich dieses Geheimnis zu jeder Chipkarte und bringt es während der Personalisierung in die Chipkarte ein. g_{ICC} wird im UpdateComponentResponse nicht im Klartext übertragen, sondern an die zu signierenden Informationen konkateniert und in die Signatur integriert. Aufgrund der Hashfunktion des Signaturverfahrens ist es nicht möglich, g_{ICC} aus der Signatur heraus zu berechnen.

Ein UpdateComponentResponse, das der Client zum Update Service schickt, enthält damit zusammenfassend folgende Informationen:

UpdateComponentResponse:

- UPDATE COMPONENT RESPONSE der Chipkarte, optional, bestehend aus:
  - Identität des Zertifikatsinhabers
  - UMP-ID
  - MessageID, optional
  - $\mathrm{puK}^m_{\mathrm{CH}}$, optional, für in der Chipkarte erzeugte Schlüssel
- Bestätigung des Clients, optional, bestehend aus:
  - Identität des Zertifikatsinhabers
  - UMP-ID
  - MessageID, optional
  - $\mathrm{puK}^m_{\mathrm{CH}}$, optional, für im Client erzeugte Schlüssel

Wie ist das UpdateComponentResponse bzw. das UPDATE COMPONENT RESPONSE der Chipkarte abgesichert? Es reicht eine einfache digitale Signatur mit einem Signaturverfahren, dessen CH-Zertifikat nicht revoziert ist – z. B. sign^u –, weil auf dieser Grundlage die Certification Authority ein neues Zertifikat ausstellen kann. Zur Überprüfung, ob der CH auch den privaten Schlüssel zum übertragenen öffentlichen Schlüssel kennt, wird eine zweite Signatur erzeugt. Diese zweite Signatur ist mit einem zu σ^m bzw. η_asym^m verwandten Signaturverfahren sign^m erzeugt worden. Aus Konsistenzgründen soll das UpdateComponentResponse – auch wenn kein Schlüssel und somit keine Bestätigung, dass der private Schlüssel zum öffentlichen bekannt ist – stets mit zwei Signaturverfahren multipel signiert werden. Falls dafür mehrere Signaturverfahren zur Wahl stehen, sollten voneinander unabhängige gewählt werden. Im Falle einer Chipkarte signiert diese zusätzlich das Geheimnis g_{ICC} mit, so dass die folgende multiple Signatur das UpdateComponentResponse absichert:

$$\begin{aligned}&(\mathrm{sign}^m(\mathrm{CH} \,|\, \mathrm{UMP-ID} \,|\, \mathrm{MessageID} \,|\, \mathrm{puK}^m_{\mathrm{CH}} \,[\,|\, g_{\mathrm{ICC}}\,]\,,\ \mathrm{prK}^m_{\mathrm{CH}}),\\ &\ \ \mathrm{sign}^u(\mathrm{CH} \,|\, \mathrm{UMP-ID} \,|\, \mathrm{MessageID} \,|\, \mathrm{puK}^m_{\mathrm{CH}} \,[\,|\, g_{\mathrm{ICC}}\,]\,,\ \mathrm{prK}^u_{\mathrm{CH}}))\end{aligned}$$

Der Ablauf in Phase 6 verläuft wie folgt:

1. Falls eine Chipkarte involviert ist:
   Eine Chipkarte antwortet auf ein COMMAND stets mit einem RESPONSE. Hat der Client

– also Mail- oder Zugangskontroll-Client – ein UPDATE COMPONENT COMMAND an die Chipkarte abgesetzt, sendet sie ein UPDATE COMPONENT RESPONSE zurück, mit dem sie das erfolgreiche Ausführen des Kommandos signalisiert und – optional – neu erzeugte öffentliche Schlüssel übermittelt. Das UPDATE COMPONENT RESPONSE beinhaltet:

- Identität des Zertifikatsinhabers CH
- UMP-Kennung UMP-ID
- MessageID, optional
- neu generierte öffentliche Schlüssel $\mathrm{puK}^{m}_{\mathrm{CH}}$, optional
- zwei Signaturen, welche obige Daten zuzüglich g_{ICC} einschließen und mit zwei Signaturverfahren erzeugt wurden, wozu der Benutzer seine PIN präsentieren muss

2. Falls bei einem Client durch das UpdateComponent Komponenten ausgetauscht wurden:
   Ein Client – also Zugangskontroll-Client, Mail Client oder Rechner S oder C –, bei dem Komponenten ausgetauscht wurden, erzeugt ein UpdateComponentResponse und lässt es von der PSE, auf die er Zugriff hat – also der eigenen Software-PSE oder einer Chipkarte –, zweimal signieren. Dazu muss der Benutzer seine PIN präsentieren. In das UpdateComponentResponse ist integriert:

   - optional ein von einer Chipkarte übermitteltes UPDATE COMPONENT RESPONSE
   - Identität des Zertifikatsinhabers CH
   - UMP-ID
   - MessageID, optional
   - optional in der Software-PSE des Clients neu generierte Schlüssel
   - zwei Signaturen, die obige Daten einschließen und mit zwei Signaturverfahren erzeugt wurden

3. Falls ein Client ein Rechner C einer virtuellen Zugangskontrolle ist:
   Der Client hat keinen Kontakt zum Update Service. C transportiert dieses UpdateComponentResponse-Datenpaket während einer Authentisierung zum Rechner S, so dass S den Weitertransport zum Update Service übernehmen kann. Siehe Appendix A.8 auf Seite 214.

4. Ein Client – also Mail Client, Zugangskontroll-Client oder Rechner S – speichert dieses UpdateComponentResponse bei sich unter der UMP-ID für den Fall einer Wiederholung ab, falls es auf dem Weg zum Update Service nicht ankommt, und sendet es via PKCS#7 an den Update Service, siehe Appendix A.6 auf Seite 209, dessen Adresse im UpdateComponent enthalten ist.

5. Der Update Service verifiziert das UpdateComponentResponse dahingehend, ob der CH ein UpdateComponent erhalten hat und ob eine optional vorhandene MessageID korrekt ist.

6. Ist das Response korrekt, leitet der Update Service das UpdateComponentResponse an die Certification Authority weiter.
   Die Art und Weise, wie die beiden Bereiche Update Service und Certification Authority innerhalb des Trust Centers kommunizieren und absichern, wird hier nicht thematisiert.

7. Die CA verifiziert

(a) die Signatur, deren Zertifikat nicht revoziert wurde, um die Identität des CHs zu garantieren und

(b) die Signatur zum übertragenen Schlüssel, um zu prüfen, ob der CH über den zugehörigen privaten Schlüssel verfügt.

Sind beide Verifikationen korrekt und kam dieser Request vom Update Service, so kann die CA davon ausgehen, dass bei diesem CH ein Austausch der privaten Schlüssel wirklich nötig wurde, dass die neuen Schlüssel tatsächlich vom CH stammen und – falls Chipkarten involviert sind – die neuen Schlüssel in einer Chipkarte erzeugt wurden. Nach einer Prüfung auf kryptographische Eignung und Dublettenfreiheit zertifiziert die CA die neuen Schlüssel als

$$\text{cert}^m_{\text{CH}} = \text{sign}^m(\text{Zertifikatsinhalt mit puK}^m_{\text{CH}}, \text{prK}^m_{\text{CA}}).$$

8. Die CA veröffentlicht die neuen Zertifikate im Verzeichnis der gültigen Zertifikate (DIR), sofern der CH ihrer Veröffentlichung zugestimmt hat.

9. Dem Certificate Holder werden seine neuen Zertifikate entweder per E-Mail zugestellt oder er kann sie sich selbst vom Verzeichnis über LDAP oder das Certificate Management Protocol (CMP) besorgen und mit dem Sicherheitsanker verifizieren. Im Rahmen des Update Management Protocols werden Zertifikate jedoch nicht zugestellt. Der genaue Ablauf ist in Phase 9 (Abschnitt 4.5.12 auf Seite 151) beschrieben.

10. Falls die Certification Authority einen neuen Schlüssel nicht zertifiziert, z. B. weil er schon einmal vergeben wurde oder als kryptographisch ungeeignet gilt, schickt die CA über den Update Service dem betreffenden CH ein erneutes UpdateComponent, welches ausschließlich den Austausch des Schlüssels initiiert. Die UMP-Verifikation dieses UpdateComponents wird mit TRUE ausgehen, weil die PSE des Certificate Holders über zwei sichere Verfahren samt Sicherheitsankern verfügt (die u. U. zuvor ausgetauscht wurden).

11. Fortsetzung in Phase 7.

Den räumlichen Verlauf der Zertifizierung mit dem UpdateComponentResponse vom CH zur CA stellt Abbildung 4.16 dar; der dort abgebildete “Client“ kann je nach Anwendung ein Mail Client, ein Zugangskontroll-Client oder ein Rechner S sein.

Was passiert, wenn durch eine Störung im Ablauf in Phase 6 das UpdateComponentResponse nicht beim Update Service ankommt? Der Update Service würde dem Certificate Holder das UpdateComponent erneut zuschicken, entweder weil der CH sein fehlendes neues Zertifikat beim Trust Center gemeldet hat oder weil dem Update Service die Bestätigung fehlt. Der Client würde dieses UpdateComponentResponse, das er zu der UMP-ID abgespeichert hatte, erneut an den Update Service schicken.

Die exakte ASN.1-Notation des UpdateComponentResponses ist in Appendix B.1 auf Seite 222 nachzulesen. Für die Antwort der Chipkarte, das UPDATE COMPONENT RESPONSE, sei auf Appendix B.2 auf Seite 223 verwiesen.

Ein de facto Standard für die Übermittlung eines öffentlichen Schlüssels zum Zertifizieren ist Certification Request Syntax Standard (PKCS#10) [PKCS10]. PKCS#10 beschreibt ein Datenformat, in das Name, öffentlicher Schlüssel und Signaturalgorithmus sowie eine Signatur mit diesem Schlüssel integriert werden. Wie PKCS#10 abgesichert wird, ist nicht Bestandteil dieses Standards. Es wird allerdings PKCS#7 zur Absicherung genannt. Da in der Fail-Safe-PKI

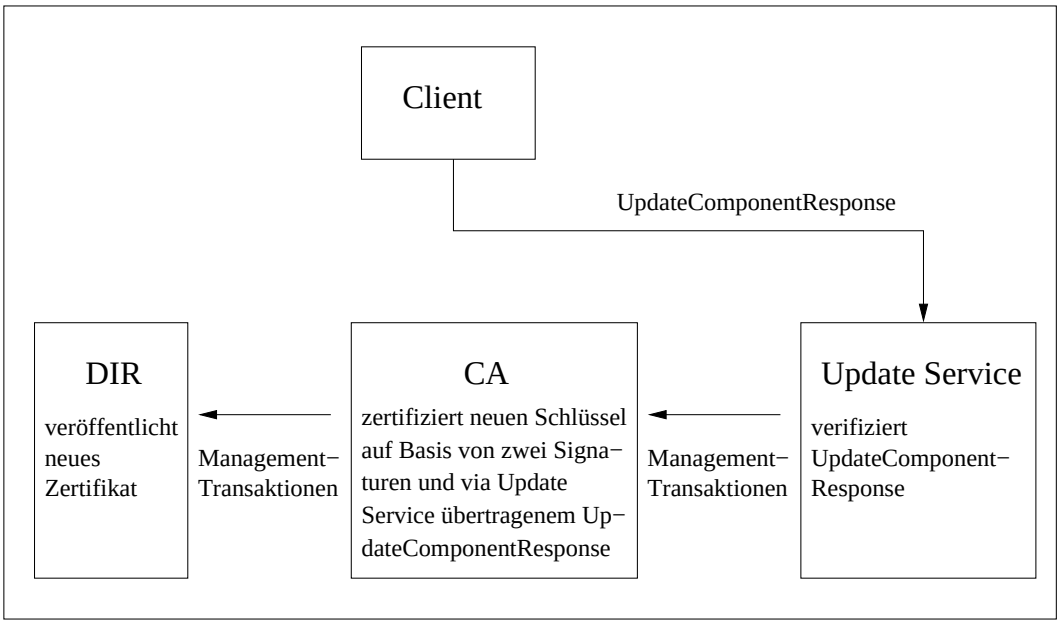

Abbildung 4.16: Phase 6: Bestätigung des UpdateComponents

die MessageID eine unverzichtbare Information für das UpdateComponentResponse ist, ist ein neues Datenformat definiert worden, das sich allerdings an PKCS#10 orientiert.

Nachdem der Client neu zu generierende Schlüssel an den Update Service geschickt hat, kann der Client den Update abschließen. Doch zuvor wird die Sicherheit des UpdateComponentResponses begründet.

4.5.9 Sicherheit des UpdateComponentResponses

In diesem Abschnitt wird die Sicherheit des UpdateComponentResponses diskutiert. Dazu werden anhand des Signatur/Verifikations-Modells für das UpdateComponentResponse die Sicherheitsziele Authentizität, Integrität, Aktualität und Identität geprüft.

Das **Signatur/Verifikations-Modell für UpdateComponentResponse** hat folgende Form:

Signaturverfahren	$\text{sign}^m, \text{sign}^u \in S$, voneinander unabhängig
Input D = UCR	UpdateComponentResponse (UCR) enthält folgende Einträge: - UPDATE COMPONENT RESPONSE der ICC des CHs, optional - Identität des CHs - UMP-ID - MessageID, optional - puK^m_{CH}, optional g_{ICC} ist in UCR nicht enthalten(!)
Signierer	CH
Signatur	$S^m = \text{sign}^m(\text{CH} \,\|\, \text{UMP} - \text{ID} \,\|\, \text{MessageID} \,\|\, \text{puK}^m_{\text{CH}} [\,|\, g_{\text{ICC}}\,],\, \text{prK}^m_{\text{CH}})$ $S^u = \text{sign}^u(\text{CH} \,\|\, \text{UMP} - \text{ID} \,\|\, \text{MessageID} \,\|\, \text{puK}^m_{\text{CH}} [\,|\, g_{\text{ICC}}\,],\, \text{prK}^u_{\text{CH}})$ mit optionalem g_{ICC} $S^* = \{D, S^m, S^u\}$
S^*	Multipel signiertes UpdateComponentResponse

Verifikation	verify$(D\,[\|g_{\mathrm{ICC}}], S^m, S^u)$ = TRUE, falls gilt: - verify$(D\,[\|g_{\mathrm{ICC}}], S^m, \mathrm{puK}^m_{\mathrm{CH}})$ = TRUE - verify$(D\,[\|g_{\mathrm{ICC}}], S^u, \mathrm{puK}^u_{\mathrm{CH}})$ = TRUE, wobei die CA das Geheimnis g_{ICC} kennt - Update Service hat diesen CH mit dieser MessageID aufgefordert, einen neuen Schlüssel zu generieren ump_verify$(D\,[\|g_{\mathrm{ICC}}], S^m, S^u)$ = FALSE, sonst

Die Sicherheitskriterien hinsichtlich Authentizität, Integrität, Aktualität und Identität sind erfüllt:

- Die erste Signatur bestätigt, dass der CH den privaten Schlüssel zum übertragenen öffentlichen Schlüssel besitzt.
- Die zweite Signatur, dessen Zertifikat nicht revoziert ist, bestätigt die Authentizität und Integrität des UpdateComponentResponses sowie die korrekte Identität.
- Die Aktualität des UpdateComponentResponses wird durch Verwendung einer MessageID gewährleistet, die eine Replay-Attacke verhindern würde, oder weil nur diejenigen Certificate Holder ein UpdateComponentResponse schicken, die ein UpdateComponent erhalten haben und ausführen konnten. Es wird unterstellt, der Update Service merkt sich in diesem Fall, wer vom Schaden betroffen ist und wem ein UpdateComponent geschickt wurde und vergleicht diese Daten mit eingehenden UpdateComponentResponses, so dass auch eine Replay-Attacke auffallen und die erneute Bearbeitung eines eingehenden Responses verhindert werden würde.
- Darüber hinaus bestätigen mit Geheimnis g_{ICC} versehende Signaturen, dass die Chipkarte den neuen Schlüssel generiert hat.

Wie in Abschnitt 4.5.5 auf Seite 123 erwähnt, gibt es zur Ausführung des UpdateComponents keine Autorisierung des Zertifikatsinhabers, so dass ein Angreifer, der sich Zugang zur PSE eines Opfers verschafft hat, Verfahren und Sicherheitsanker austauschen und neue Schlüssel generieren kann. Allerdings hat er keine Chance, ohne PIN an private Schlüssel zu gelangen, zu denen ein Zertifikat existiert: Der private Schlüssel des noch sicheren Verfahrens $\mathrm{prK}^u_{\mathrm{CH}}$ ist mit einer PIN geschützt und das UpdateComponentResponse, auf dessen Grundlage ein neuer privater Schlüssel $\mathrm{prK}^m_{\mathrm{CH}}$ zertifiziert werden soll, ist ebenfalls mit $\mathrm{prK}^u_{\mathrm{CH}}$ signiert, was die Eingabe einer PIN erfordert. Der Angreifer kann über die 0-PIN den neuen Schlüssel $\mathrm{prK}^m_{\mathrm{CH}}$ benutzen und die PIN ändern, aber da kein Zertifikat existiert, wird eine solche Signatur keinen Schaden anrichten. Außerdem wird die CA den neuen Schlüssel nicht zertifizieren, falls der Zertifikatsinhaber sein Zertifikat hat revozieren lassen.

Soviel zum Nachweis, dass durch das UpdateComponentResponse ein von einer PSE neu generierter Schlüssel sicher zertifiziert werden kann.

4.5.10 Phase 7: Abschluss des Updates

Wozu ist dieser Abschluss nötig? Der Abschluss kann z. B. bei Signatur-Anwendungen sinnvoll sein, wenn mehrere Zertifikatsinhaber mit ihren Chipkarten einen gemeinsamen Multi User Client mit mehreren Profilen nutzen – etwa im Rathaus oder Internet-Café. Verhindert werden soll, dass der erste Benutzer einen Austausch durchführt und der Client anschließend bei allen folgenden

Benutzern einen Austausch verweigert, weil der Client durch den Austausch von Providern beim ersten Aktualisieren nun benötigte Verfahren nicht mehr hat. Verfahren sind in der PSE im Gegensatz zu Schlüsseln, Sicherheitsankern, Registries und MessageIDs in den Profilen einmalig vorhanden. Aus diesem Grund muss ein Update beim Client abgeschlossen werden, wovon dieser Abschnitt handelt.

Ob ein Update aktiv und noch nicht abgeschossen ist, ist am Update Modus abzulesen. Dieser wird in Phase 3 auf Seite 119 eingeschaltet.

In Phase 4 auf Seite 132 wird eine Menge von Providern geladen, die bei Multi User Clients den endgültigen neuen Provider `Provider_neu` und zusätzlich den Übergangs-Provider `Provider_neu_Update_Modus` enthält. Letzterer enthält nicht nur die neuen Verfahren, sondern auch die alten, so dass in der Update-Phase – währenddessen der Update Modus also eingeschaltet ist – über `Provider_neu_Update_Modus` alle Certificate Holder mit dem Client arbeiten können, unabhängig davon, ob sie den Austausch bereits durchgeführt haben oder nicht.

Der Client verlässt den Update Modus, wenn der Austausch vollständig bei allen Benutzern, die den Client nutzen, durchgeführt wurde, und beendet den Austausch, indem nun letztlich der Provider `Provider_neu` installiert und der Übergangs-Provider `Provider_neu_Update_Modus` gelöscht wird. Diese Aktion muss atomar ausgeführt werden. In dem Fall, dass der Client exklusiv von nur einem Certificate Holder genutzt wird, wie im Modell von Abbildung 1.5 auf Seite 23 dargestellt, ist nur ein(!) Provider (`Provider_neu`) in Phase 4 installiert worden, so dass in dieser Phase 7 nur der Update Modus ausgeschaltet wird.

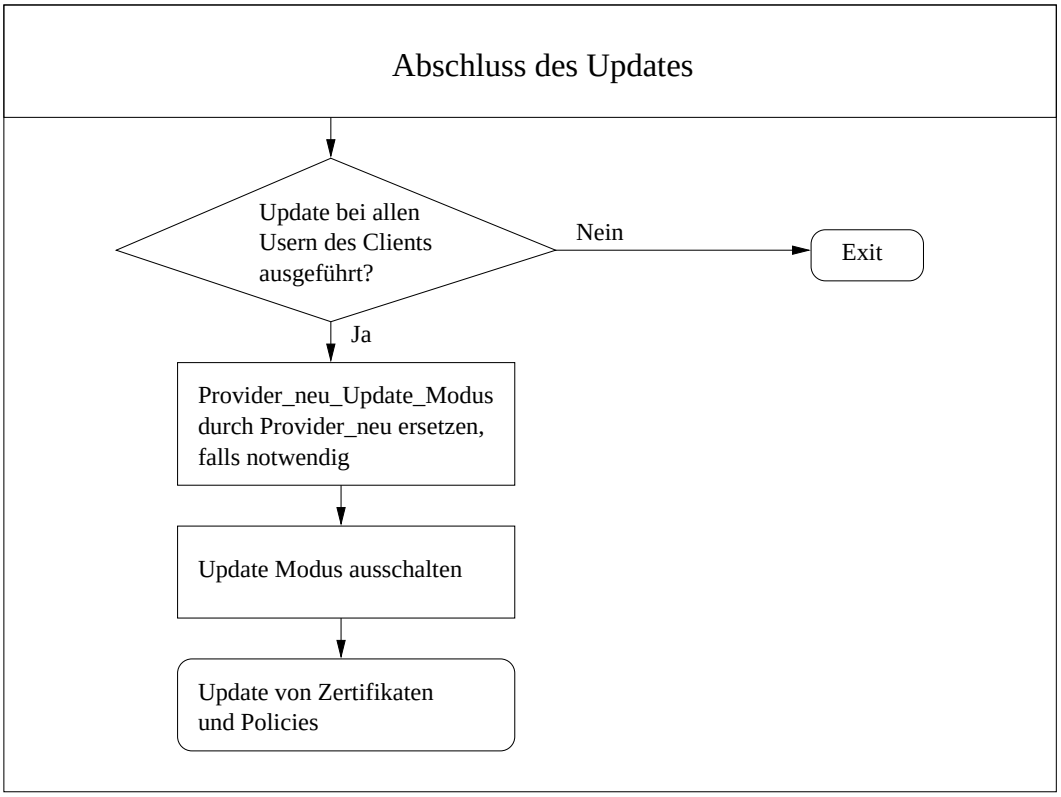

Abbildung 4.17: Phase 7: Abschluss des Updates

Graphisch lässt sich der Abschluss des Update Management Protocols wie in Abbildung 4.17

dargestellt zusammenfassen.

Bei einer Störung im Ablauf in Phase 7 kann eine Wiederholung durchgeführt werden.

Zertifikatsinhaber benötigen neue Zertifikate, falls sie Schlüssel neu generiert und über den Update Service zur Zertifizierungsinstanz transportiert haben, und ggf. neue Policies. Wie sie diese erhalten, ist Thema in Phase 8.

4.5.11 Phase 8: Austausch von Zertifikaten und Policies

Der Abschluss ist vollzogen. Neue Verfahren und Sicherheitsanker wurden ausgetauscht und eine Bestätigung mit möglichen neuen Schlüssel an den Update Service geschickt. Zertifikate und Policies müssen u. U. ausgetauscht werden. Woher bekommen die Certificate Holder ihre neuen Zertifikate? Und veränderte Policies? Dieser Abschnitt beschäftigt sich mit diesen Themen.

Zertifikate und Policies sind über Verzeichnisse verfügbar und mit den Verfahren und Sicherheitsankern bei den PSEs der Certificate Holder verifizierbar, so dass ein Download über das Verzeichnis DIR und eine Verifikation problemlos möglich ist. Darüber hinaus stehen etablierte Protokolle bereit, um Zertifikate zu übermitteln; beispielsweise das Certificate Management Protocol [CMP99] [CMP01] oder als PKCS#7-Attachment. Ein beim Client angekommenes Zertifikat kann mit den standardisierten Chipkarten-Kommandos (siehe [ISO95] oder [ISO98]) zur ICC transportiert werden.

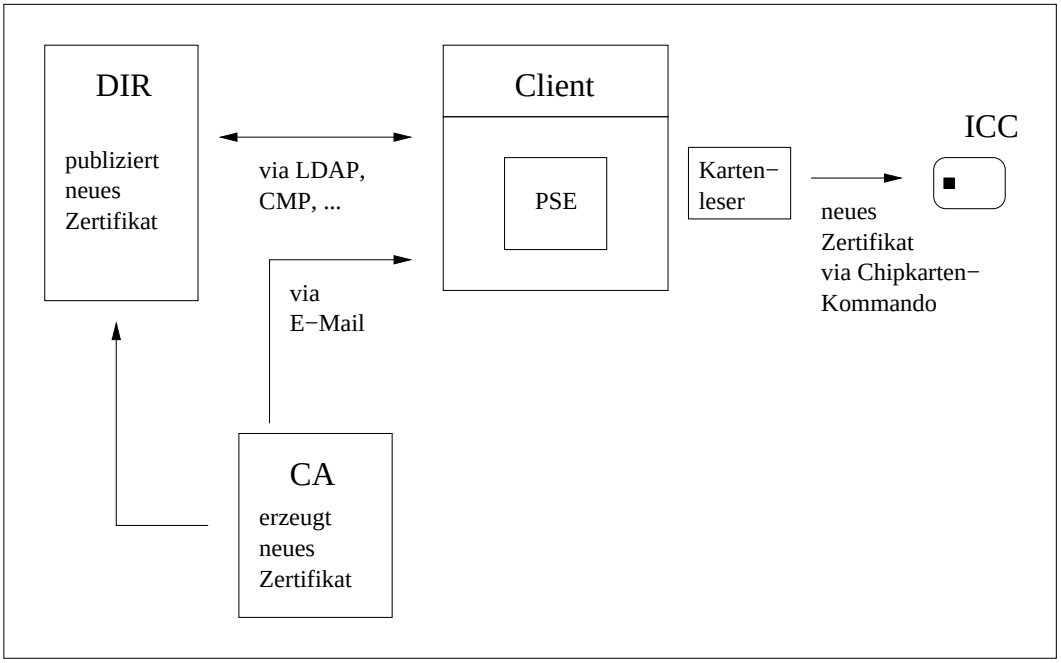

Abbildung 4.18: Phase 8: Zertifikate und Policies austauschen

Bei SigG-konformen Anwendungen ist nach §5 Abs. 2 der Signaturverordnung [SigV01, §5] eine Bestätigung der erhaltenen Zertifikate notwendig, wobei diese Bestätigung auch mit einer digitalen Signatur erfolgen kann. Für diese digitale Bestätigung stehen in der Fail-Safe-PKI Mechanismen bereit. So kann z. B. ein Zertifikatsinhaber in einer SigG-konformen Signatur-Anwendung dem Trust Center die Bestätigung via PKCS#7 als Mail schicken, wobei die Form der Bestätigung vorgegeben sein kann, so dass das Trust Center dies automatisiert bearbeiten kann. Erst

danach darf das Zertifikat in das Verzeichnis eingestellt werden; ausdrücklich „darf“, denn nach [SigG01, §5] kann ein Zertifikatsinhaber verfügen, dass seine Zertifikate nicht im Verzeichnis veröffentlicht werden dürfen.

Abbildung 4.18 illustriert den Ablauf von Phase 8.

Falls ein Zertifikat beim Zertifikatsinhaber nicht ankommt, kann sich dieser bei seinem Trust Center melden und den Ablauf wiederholen.

Nachdem neue multiple Komponenten in der Fail-Safe-PKI etabliert wurden, können sie genutzt werden – beispielsweise, um digitale Signaturen nachzusignieren.

4.5.12 Phase 9: Applikationsdatenpflege

Digitale Signaturen sind nur begrenzt „haltbar“, weil sich kryptographische Verfahren und ihre Parameter im Laufe der Zeit ändern können. Die Problematik wurde in Abschnitt 2 auf Seite 39 ausführlich dargelegt. Aus diesem Grund werden geeignete Kryptoverfahren jährlich veröffentlicht und in den Zertifikaten Gültigkeitszeiträume vermerkt. Ein Nachsignieren oder Re-Signieren ist immer dann sinnvoll, wenn die Eignung der eingesetzten Verfahren oder die Gültigkeit eines Zertifikats zu einem Dokument abzulaufen droht und – speziell in einer Fail-Safe-PKI – wenn ein Teil einer multiplen digitalen Signatur durch einen Schadensfall ohne Beweiskraft ist. Wie in Abschnitt 2.1.3 auf Seite 45 ausgeführt, ist derjenige für ein Re-Signieren zuständig, der eine digitale Signatur auch über den Ablauf der Eignung eines Verfahrens oder den Gültigkeitszeitraum zugehöriger Zertifikate als Beweismittel nutzen will. In diesem Abschnitt wird das Re-Signieren erläutert.

Die grundsätzliche Idee ist, dass – wie in Abschnitt 3.3 auf Seite 75 ausgeführt – wichtige Dokumente multipel signiert werden, damit sie ihre Beweiskraft behalten, falls eine der Signaturen kompromittiert sein sollte. Wenn nun neue kryptographische und einsatzspezifische Komponenten durch das Update Management Protocol verteilt wurden, können multipel signierte Dokumente, deren eine Signatur kompromittiert wurde, mit einer erneuten multiplen digitalen Signatur nachsigniert werden.

Da es – wenn es um die Beweiskraft von digitalen Signaturen geht – unumgänglich ist, einen Zeitstempel zu nutzen, wurde in Abschnitt 3.3 eine Idee zur Effizienzsteigerung beschrieben: Das zeitzustempelnde Dokument ist nur über eine einfache Signatur, der Zeitstempel jedoch über eine multiple digitale Signatur abgesichert.

Deshalb wird ein Re-Signieren nur für einen Zeitstempel beschrieben. Das Szenario ist, dass zum Zeitpunkt t_0 ein mit einer einfachen Signatur signiertes Dokument D aus beweistechnischen Gründen mit einem multiplen digitalen Zeitstempel versehen wird: $\mathrm{TSP}(D, t_0)$. Dieser Zeitstempel wird von einer autorisierten und vertrauenswürdigen Zeitstempelinstanz (Time Stamping Authority – TSA) erzeugt. Zum Nachsignieren wird zum Zeitpunkt t_1, $t_1 > t_0$, ein erneuter Zeitstempel erzeugt:

$$\mathrm{TSP}(D|\mathrm{TSP}(D, t_0), t_1)$$

Diese Vorgehensweise entspricht den rechtlichen Regelungen, siehe Abschnitt 2.1.3 auf Seite 45.

Im Sinne des Signatur/Verifikations-Modells für multiple digitale Signaturen erfolgt ein Re-Signieren wie folgt:

Erstes Signieren zum Zeitpunkt t_0:

Signaturverfahren zum Zeitpunkt t_0	$\{\text{sign}^{i_1}, \text{sign}^{i_2} \mid \text{Fail-Safe-Konzept-geeignet}\}$
Input von Benutzer an TSA	Tupel von Hashwerten $(h_1, h_2) = (\kappa^1(D^*), \kappa^2(D^*))$ für ein Dokument D^* und zwei voneinander unabhängige Hashfunktionen κ^1 und κ^2
Input D	$(h_1, h_2) \mid$ Zeit t_0
Signierer	Zeitstempelinstanz (TSA)
S^*	multipel signierter Zeitstempel (D, S^{i_1,i_2}), optional sämtliche Zertifikate und Sperrinformationen, die bis zum Zeitpunkt t_1 gültig sind, wobei die aktuelle Zeit t_0 vor t_1 liegt: $t_0 < t_1$
Aufzubewahren	Dokument D^* und multiplen Zeitstempel S^*
Verifizieren von S^* bis t_1	Eine der zwei Signaturen zu verifizieren und zu validieren, reicht normalerweise, wenn kein Zertifikat revoziert ist, aus: O. B. d. A. wird i_1 ausgewertet: S^* ist gültig, wenn $\text{verify}^{i_1}((h_1, h_2) \mid t_0, \text{sign}^{i_1}((h_1, h_2) \mid t_1, \text{prK}^{i_1}_{\text{TSA}}), \text{puK}^{i_1}_{\text{TSA}}) =$ TRUE.
Validieren von S^* bis t_1	Ist das Zertifikat des TSA mathematisch korrekt und nicht revoziert, so ist der Zeitstempel S^* korrekt.
Gehört Zeitstempel S^* zu D^*?	Prüfen, ob $\kappa^1(D^*) = h_1$ und/oder $\kappa^2(D^*) = h_2$ gelten. Eine Signatur reicht aus, wenn das TSA-Zertifikat gültig ist.
Validieren nach t_1 oder wenn TSA-Zertifikat revoziert	Nicht mehr möglich, weil das Zertifikat zum Zeitpunkt der Verifikation gültig sein muss. Es ist nicht möglich, ein Zertifikat zu akzeptieren, dass heute (Zeitpunkt t) revoziert ist, aber gestern (Zeitpunkt t', $t' < t$) erzeugt wurde, wenn nicht sicher gestellt ist, dass es wirklich gestern erzeugt wurde. Da dieser Zeitpunkt erst durch den Zeitstempel belegt wird, muss der Zeitstempel jetzt (t) gültig sein, d. h. nicht abgelaufen und nicht revoziert.

Re-Signieren zum Zeitpunkt t_2, $t_0 < t_2 < t_1$:

Signaturverfahren zum Zeitpunkt t_2	$\{\text{sign}^{i_1}, \text{sign}^{i_3} \mid \text{Fail-Safe-Konzept-geeignet}\}$. Nach Voraussetzung ist von einem Schaden höchstens ein Signaturverfahren betroffen. O. B. d. A. sei dies sign^{i_2}, so dass sign^{i_3} entweder ein neues, zu sign^{i_1} unabhängiges Verfahren oder zu sign^{i_2} identisch ist, falls ein Re-Signieren wegen Ablaufs eines Zertifikats nötig wird.
Input von Benutzer an TSA	Tupel von Hashwerten $(h'_1, h'_2) = (\kappa^1(D^{**}), \kappa^3(D^{**}))$, wobei D^{**} das ursprüngliche Dokument D^* und den nachzusignierenden Zeitstempel S^* enthält, und κ^1 und κ^3 zwei voneinander unabhängige Hashfunktionen sind. κ^3 kann κ^2 ersetzt haben oder κ^2 sein.
Input D'	$(h'_1, h'_2) \mid$ Zeit t_2
Signierer	Zeitstempelinstanz (TSA)
S^{**}	multipel signierter Zeitstempel (D', S^{i_1,i_3}), optional sämtliche Zertifikate und Sperrinformationen, die bis zum Zeitpunkt t_3 gültig sind, wobei die aktuelle Zeit t_2 vor t_3 liegt

Verifizieren von S^{**} bis t_3	O. B. d. A. wird i_1 ausgewertet: S^{**} ist gültig, wenn $\text{verify}^{i_1}((h'_1, h'_2) \mid t_0, \text{sign}^{i_1}((h'_1, h'_2) \mid t_1, \text{prK}^{i_1}_{\text{TSA}}), \text{puK}^{i_1}_{\text{TSA}}) = \text{TRUE}$.
Validieren von S^{**} bis t_3	Ist das Zertifikat des TSA mathematisch korrekt und nicht revoziert, so ist der Zeitstempel S^{**} korrekt.
Gehört Zeitstempel S^{**} zu D^{**}?	Prüfen, ob $\kappa^1(D^{**}) = h'_1$ und/oder $\kappa^3(D^{**}) = h'_2$ gelten. Ist ein Hashwert korrekt, dessen Hashfunktion als kryptographisch geeignet gilt, dann gehört Dokument D^{**} zum Zeitstempel S^{**}. Es gilt $D^{**} = D^*\|S^*$, so dass im Folgenden der Zeitstempel S^* verifiziert wird.
Verifizieren und Validieren von S^*	Analoges Vorgehen wie oben bis t_1. Es wird geprüft, ob der Zeitstempel S^* und das damalige Zertifikat des TSA (bis t_1) mathematisch korrekt sind.
Gehört Zeitstempel S^* zu D^*?	Prüfen, ob $\kappa^1(D^*) = h_1$ und/oder $\kappa^2(D^*) = h_2$ gelten.
Zusammenfassung	Sind die Zeitstempel S^* und S^{**} korrekt und sind die zeitgestempelten Dokumente D^* aus S^* und S^{**} identisch, so existierte D^* bereits zum Zeitpunkt t_0.

Das Nachsignieren eines Dokuments ohne Zeitstempel funktioniert analog, indem statt Zeitstempel D von der TSA Dokumente D^* von einem CH nachsigniert werden.

Die Applikationsdatenpflege betrifft auch Verschlüsselungen. Verschlüsselte Daten, die vor dem Austausch eines Verschlüsselungsverfahrens entschlüsselt wurden und nun im Klartext vorliegen, können jetzt mit dem neuen Verfahren erneut verschlüsselt werden. Diese Art des "Re-Encrypting" wird hier nicht weiter thematisiert.

Damit ist der Ablauf im Schadensfall vollständig durchgeführt.

4.6 Verallgemeinerung des Modells

Der Beschreibung im Schadensfall liegt das Modell einer einstufigen PKI-Hierachie zugrunde. Besteht die PKI in einer Verallgemeinerung aus einer RootCA und mehreren von ihr zertifizierten CAs, die ihrerseits CAs und CHs zertifizieren – wie in Abbildung 1.4 auf Seite 21 abgebildet –, findet der Austausch sukzessive statt: In einem ersten Schritt initiiert die RootCA den Austausch bei ihren Zertifikatsinhabern – also den von ihr zertifizierten CAs. In den folgenden Schritten wenden alle CAs, die selber einen Austausch erfahren haben, dieselbe Prozedur auf ihre Zertifikatsinhaber an. Auf diese Weise verbreitet sich das UpdateComponent in einer Fail-Safe-PKI, die aus mehr als einer Hierchie-Stufe besteht.

4.7 Performance und Kosten einer Fail-Safe-PKI

Welcher Mehraufwand muss bei einer Fail-Safe-PKI gegenüber einer herkömmlichen PKI einkalkuliert werden? Und welcher Minderaufwand ist zu erwarten? Eine erste Einschätzung zu diesen Fragen ergab in Abschnitt 3.7 auf Seite 78,

- dass die zusätzlichen Kosten für Räume, Personal und Technik im Trust Center relativ gering sein werden,
- dass die Performance-Verluste im täglichen Umgang mit multiplen Signaturen und iterativen Verschlüsselungen von den konkreten Anwendungen, dem eingesetzten Equipment und dem geforderten Sicherheitsniveau abhängen,
- dass die Performance des Austauschs kompromittierter Komponenten aufgrund der gesicherten Verfügbarkeit unkritisch ist und
- dass der dynamische Austausch im Fail-Safe-Konzept deutlich schneller als die Verteilung neuer Verfahren, Schlüssel und Zertifikate auf herkömmlichem Weg ist.

In diesem Abschnitt werden diese ersten Einschätzungen bzgl. organisatorischer Kosten für ein Trust Center, Kosten für zusätzliches technisches Equipment und Performance in der Nutzungsphase sowie im Schadensfall präzisiert. Für konkrete Angaben zu Performance und Kosten einer speziellen Anwendung, die als Fail-Safe-PKI ausgelegt ist, sind diese allgemeinen Angaben zu konkretisieren; siehe dazu die Beispiele in Kapitel 5.

Organisatorische Kosten

Jede PKI braucht ein Trust Center, das die Funktionalitäten der Registrierungs- und Zertifizierungsinstanz übernimmt sowie den Verzeichnis- und Zeitstempeldienst anbietet. Dazu sind Räume und geschultes Personal nötig. Wenn das Trust Center oder Teilbereiche ausgegliedert werden, müssen entsprechende Kosten für die Dienstleistung aufgewendet werden. Für die Fail-Safe-PKI ist nur ein Mehraufwand für die Administration des Update Services nötig.

Technisches Equipment

Server im Trust Center sowie Clients und Chipkarten bei den Zertifikatsinhabern müssen den Erfordernissen der Fail-Safe-PKI angepasst werden:

- Update Service im Trust Center:
  Der Update Service muss Informationen zu den Zertifikatsinhaber – z. B. E-Mail-Adresse oder MessageID – vorhalten, sowie UpdateComponents und zugehörige -Responses verwalten und verarbeiten. Dazu sind zusätzlicher Speicherplatz und erweiterte Funktionalitäten notwendig.
- Verzeichnisdienst im Trust Center:
  Das Verzeichnis muss doppelt soviele Zertifikate vorhalten, was Speicherplatz kostet. Darüber hinaus muss zusätzliche Software vorhanden sein, um Anfragen nach multipel signierten CRLs und multipel signierten OCSP-Antworten (OCSPv11) interpretieren und entsprechende Antworten erzeugen und ggf. UpdateComponents darin integrieren zu können.
- Zeitstempelserver:
  Der Zeitstempelserver muss das erweiterte Zeitstempelprotokoll (TSPv11) interpretieren und bearbeiten können, wozu zusätzliche Software nötig ist.
- Client:
  Der Client benötigt zusätzlichen Speicherplatz für multiple Verfahren und zusätzliche Software für:

  - erweitertes Online Certificate Status Protocol (OCSPv11)
  - erweitertes Time Stamp Protocol (TSPv11)
  - multipel signierte Sperrlisten
  - in PKCS#7 codierte multiple digitale Signaturen
  - in PKCS#7 codierte iterative Verschlüsselungen
  - Update Management Protocol (UMP)

- Chipkarte:
  Chipkarten müssen zusätzliche multiple Verfahren vorhalten und das Update Management Protocol ausführen können. Dazu sind zusätzliche Software und zusätzliche Ressourcen bzgl. Speicherplatz und Prozessorleistung nötig.

Dabei ist zu beachten, dass der Preis für zusätzliche Software in Anbetracht der sonstigen Software als eher gering einzuschätzen ist. Die Prozessoren und Speicher müssen den zusätzlichen Aufwand bewerkstelligen können, was nur bei Chipkarten aufgrund ihrer eingeschränkten Ressourcen Probleme bereiten kann. Hierbei sind Fortschritte in der Zukunft zu berücksichtigen: Infineon plant beispielsweise 256 KB – 1 MB große nicht-flüchtige Speicher (EEPROM) und Prozessoren mit 32-Bit-Technologie.

Nutzungsphase

Welche Mehrbelastung hat das Fail-Safe-Konzept in der Nutzung? Dazu ist ein anwendungsunabhängiger und ein anwendungsspezifischer Mehraufwand zu berücksichtigen.

Zunächst der anwendungsunabhängige Mehraufwand: Das Registrieren der Zertifikatsinhaber in der Registration Authority verändert sich in der Fail-Safe-PKI nicht gegenüber einer herkömmlichen PKI; genausowenig wie das Verteilen von Schlüsseln und Zertifikaten. Ein Mehraufwand besteht in der Schlüssel- und Zertifikatserzeugung, weil in der CA pro CH doppelt so viele Schlüssel und Zertifikate erzeugt werden müssen, sowie in der Zertifikatsverwaltung beim Verzeichnisdienst. Dieser Mehraufwand wird bei den heute verfügbaren Rechnern in CA und DIR keine ernsten Probleme bereiten. Ein Problem könnte aber die Schlüsselerzeugung auf der Chipkarte sein, wenn diese mit ihren knappen Ressourcen jeweils zwei Schlüssel erzeugen muss.

Anwendungsspezifische Überlegungen zum Mehraufwand werden für Signatur- und Authentisierungs-Anwendungen jeweils bzgl. Rechenleistung und Netzlast vorgenommen.

1. Mehraufwand bei Signatur-Anwendungen bzgl. Rechenleistung:
   Mail Client:

   - Für die Erzeugung einer multiplen digitalen Signatur ist eine zusätzliche Signatur zu erzeugen.
   - Für die Verifikation einer multiplen digitalen Signatur ist bei hohem Sicherheitsniveau, wenn beide Signaturen verifiziert werden, eine zusätzliche Signatur zu verifizieren, sonst nicht.
   - Für die Anforderung eines erweiterten Zeitstempelprotokolls (TSPv11) ist ein zusätzlicher Hashwert zu erzeugen.
   - Für die Verifikation eines multipel signierten TSPs ist bei hohem Sicherheitsniveau, wenn beide Signaturen verifiziert werden, eine zusätzliche Signatur zu verifizieren, sonst nicht.

- Für die Anforderung einer multipel signierten Sperrliste ($CRL^{i,j}$) ist kein Mehraufwand nötig.
- Für die Verifikation einer multipel signierten Sperrliste ist eine zusätzliche Signatur zu verifizieren.
- Für die Anforderung einer erweiterten Zertifikats-Status-Antwort (OCSPv11) ist kein Mehraufwand nötig, lediglich eine andere Versionsnummer.
- Für die Verifikation eines multipel signierten OCSP-Responses ist eine zusätzliche Signatur zu verifizieren.
- Iteratives Verschlüsseln erfordert eine zusätzliche Verschlüsselung.
- Entschlüsseln einer iterativen Verschlüsselung erfordert eine zusätzliche Entschlüsselung.

Server für TSPv11, OCSPv11 und $CRL^{i,j}$:

- Für multiple digitale Signaturen ist je eine zusätzliche Signatur zu erzeugen.

Chipkarte:

- Für die Erzeugung einer multiplen digitalen Signatur ist eine zusätzliche Signatur zu erzeugen.
- Die Verifikation einer multiplen digitalen Signatur erfordert bei hohem Sicherheitsniveau eine zusätzliche Verifikation der Chipkarte.
- Iteratives Verschlüsseln erfordert eine zusätzliche Verschlüsselung.
- Entschlüsseln einer iterativen Verschlüsselung erfordert eine zusätzliche Entschlüsselung.

2. Mehraufwand bei Signatur-Anwendungen bzgl. Netzlast:

   - Bei multiplen digitalen Signaturen ist in entsprechendem PKCS#7, TSPv11, OCSPv11 und $CRL^{i,j}$ je eine zusätzliche Signatur zu übertragen; bei TSPv11 darüber hinaus ein zusätzlicher Hashwert.
   - Bei iterativen Verschlüsselungen müssen die Verschlüsselungsinformationen zur zweiten Verschlüsselung zusätzlich übertragen werden.

   Die Performance-Verluste hängen stark vom signierten oder verschlüsselten Dokument ab: Soll ein Word-Dokument von 1 MB Größe mit RSA (1024 Bit Signatur) und zusätzlich mit ECDSA (160 Bit Signatur) signiert werden, so ist auch unter Berücksichtigung weiterer Bits für die Codierung einer Signatur der Mehraufwand vergleichsweise gering. Analoges gilt für die Verschlüsselung eines Dokuments mit RSA und ECIES.
   Demgegenüber wirkt sich eine zweite Signatur in Zeitstempeln und Zertifikats-Status-Antworten deutlicher aus, weil die zu signierenden Informationen relativ klein sind. In den konkreten Beispielen zu den geleisteten OCSPv11- und TSPv11-Implementierungen ergab sich folgendes:

   - OCSP-v1-Request mit einer Signatur: 1196 Bytes
   - OCSP-v1-Response mit einer Signatur: 1680 Bytes
   - OCSP-v11-Request mit einer Signatur: 1201 Bytes
   - OCSP-v11-Response mit multipler Signatur: 2595 Bytes

- TSP-v1-Request mit einem Hashwert: 43 Bytes
- TSP-v1-Response mit einer Signatur: 1342 Bytes
- TSP-v11-Request mit zwei Hashwerten: 79 Bytes
- TSP-v11-Response mit multipler Signatur: 2462 Bytes

Siehe dazu Abschnitte A.4 und A.5 auf den Seiten 198 und 203.

3. Mehraufwand bei Authentisierungs-Anwendungen bzgl. Rechenleistung:
   Zugangskontroll-Client sowie Rechner S und C:

   - Für die Anforderung einer multipel signierten Sperrliste ($\mathrm{CRL}^{i,j}$) ist kein Mehraufwand nötig.
   - Für die Verifikation einer multipel signierten Sperrliste ist eine zusätzliche Signatur zu verifizieren.
   - Für die Anforderung einer erweiterten Zertifikats-Status-Antwort (OCSPv11) ist kein Mehraufwand nötig, lediglich eine andere Versionsnummer.
   - Für die Verifikation eines multipel signierten OCSP-Responses ist eine zusätzliche Signatur zu verifizieren.
   - Erfordert ein hohes Sicherheitsniveau eine iterative Verschlüsselung, so ist eine zusätzliche Ver- und Entschlüsselung durchzuführen.
   - In OCSPv11 und $\mathrm{CRL}^{i,j}$ müssen UpdateComponents integriert werden.

   Server des Trust Centers für OCSPv11 und $\mathrm{CRL}^{i,j}$:

   - Für multiple digitale Signaturen ist je eine zusätzliche Signatur zu erzeugen.

   Chipkarte:

   - Es ist kein Mehraufwand zu erwarten, weil weder multiple Signaturen noch iterative Verschlüsselungen genutzt werden.

4. Mehraufwand bei Authentisierungs-Anwendungen bzgl. Netzlast:

   - Bei multiplen digitalen Signaturen ist in entsprechendem OCSPv11 und $\mathrm{CRL}^{i,j}$ je eine zusätzliche Signatur zu übertragen.
   - Bei Übertragung eines UpdateComponents über OCSPv11, $\mathrm{CRL}^{i,j}$ oder TLS/SSL erhöht sich die Netzlast um dieses UpdateComponent.
   - Bei iterativen Verschlüsselungen müssen die Verschlüsselungsinformationen zur zweiten Verschlüsselung zusätzlich übertragen werden.

Performance und Kosten im Schadensfall

Welche Performance und welche Kosten sind im Schadensfall zu erwarten? In diesem Fall bietet die Fail-Safe-PKI vor allem Vorteile, weil die Verfügbarkeit der PKI gewährleistet ist, multipel signierte Dokumente ihre Beweiskraft erhalten, Revokationsinformationen weiterhin sicher transportiert werden können und iterative Verschlüsselungen vertraulich bleiben.

In einer herkömmlichen PKI müsste neue Software etwa auf CD-ROMs oder Disketten verfügbar gemacht werden, da für einen Download u. U. keine PKI-Mechanismen zur Absicherung bereit

Tabelle 4.3: Mehraufwand einer Fail-Safe-PKI

„Kostenstelle“	Präzisierte „Kostenstelle“	Mehraufwand Fail-Safe-PKI gegenüber PKI ohne Fail-Safe-Konzept
Organisatorische Kosten im Trust Center	Räume, Personal, Schulung, Betriebskosten	Administrator für Update Service
Technisches Equipment	Update Service, DIR, TSP, Client, Chipkarte	Zusätzliche Software und Speicher
in Nutzungsphase	RA, CA	Schlüssel- und Zertifikatserzeugung verdoppelt sich
	DIR	Zertifikatsverwaltung verdoppelt sich
Rechenleistung in Signatur-Anwendung	Client	je nach Sicherheitsniveau: eine zusätzliche Signaturerzeugung und -verifikation, eine zusätzliche Ver- und Entschlüsselung, eine zusätzliche Hashwertberechnung
	Server	eine zusätzliche Signaturerzeugung
	Chipkarte	je nach Sicherheitsniveau: eine zusätzliche Signaturerzeugung und -verifikation, eine zusätzliche Ver- und Entschlüsselung
Netzlast in Signatur-Anwendung	Multiple digitale Signaturen	eine zusätzliche Signatur, Aufschlag der Netzlast hängt von signiertem Dokument ab
	Iterative Verschlüsselung	eine zusätzliche Verschlüsselung, Aufschlag der Netzlast hängt von verschlüsseltem Dokument ab
Rechenleistung in Authentisierungs-Anwendung	Client	je nach Sicherheitsniveau: eine zusätzliche Signaturerzeugung und -verifikation, eine zusätzliche Ver- und Entschlüsselung
	Server	eine zusätzliche Signaturerzeugung
	Chipkarte	kein Mehraufwand
Netzlast in Authentisierungs-Anwendung	Multiple digitale Signaturen	eine zusätzliche Signatur, Aufschlag der Netzlast hängt von signiertem Dokument ab
	Iterative Verschlüsselung	eine zusätzliche Verschlüsselung, Aufschlag der Netzlast hängt von verschlüsseltem Dokument ab

stehen. Sind Chipkarten involviert, müssten sie ggf. erst produziert, personalisiert und über den Postweg o. ä. verteilt werden. Während dieser Zeit wäre die Verfügbarkeit der PKI nicht gewährleistet.

Demgegenüber kann in einer Fail-Safe-PKI neue Software mit dem Update Management Protocol (UMP) relativ schnell verteilt werden, während die Verfügbarkeit gesichert bleibt. Das Update Management Protocol ist aufgrund des Fehlens von Code relativ kompakt. Ein UpdateComponent ist je nach Informationsgehalt 3 – 8 KB groß. Wenn Code von einem Server geladen wird,

dauert dies erwartungsgemäß seine Zeit. Da aber die Nutzer den Code wahrscheinlich zeitlich versetzt downloaden, kann diese Aktion ohne größere Probleme durchgeführt werden. Darüber hinaus funktioniert ein Austausch ohne das UpdateComponent nicht, so dass der Update Service die Menge der Zugriffe auf den Server steuern kann. Der Austausch beim Zertifikatsinhaber selbst kann eine Weile dauern, insbesondere wenn neue Schlüssel generiert werden.

Für die Gesamtzeit der Austauschmaßnahme ist zu berücksichtigen:

- Zeit für Prüfung des Schadens
- Zeit für Entwicklung und Prüfung der Gegenmaßnahme
- Zeit für Administrieren und Prüfen des UMPs
- Zeit für den Austausch bei den Zertifikatsinhabern

Fazit: Der Mehraufwand und die Nachteile im täglichen Betrieb der Fail-Safe-PKI gegenüber einer „normalen“ PKI ohne Fail-Safe-Konzept sind nicht zu hoch – gerade im Hinblick auf die höhere Sicherheit im Schadensfall. Tabellen 4.3 und 4.4 fassen die Ergebnisse dieses Abschnitts zusammen.

Tabelle 4.4: Vergleich „normale“ PKI gegenüber Fail-Safe-PKI

Funktionalität im Schadensfall	„Normale“ PKI	Fail-Safe-PKI
Verfügbarkeit der PKI	nein	ja
Zur Wiederherstellung der vollen Verfügbarkeit nötig:		
- Verteilung neuer Software	“out-of-band“, z. B. über CD-ROM oder Diskette, etwa per Post	im laufenden Betrieb, online
- benötigte Zeit	Tage – Wochen, währenddessen die PKI nicht verfügbar ist	Stunden – Tage, währenddessen die Fail-Safe-PKI verfügbar ist
Beweisbarkeit von Signaturen	nein	ja
Wirkungsvolle Revokationen	nein	ja
Gewährleistung von Verschlüsselungen	nein	ja

4.8 Zusammenfassung

In diesem Kapitel wurde das in Kapitel 3 vorgestellte Fail-Safe-Konzept für Public-Key-Infrastrukturen anhand der Fail-Safe-PKI in allen Details dargestellt. Diesem Kapitel ist zu entnehmen,

- wann zwei kryptographische Algorithmen voneinander unabhängig sind und für welche Algorithmen dies derzeit zutrifft,

- wie multiple digitale Signaturen und iterative Verschlüsselungen definiert sind,
- wie die Fail-Safe-PKI mit Registration und Certification Authority, Verzeichnis- und Revokationsdienst, Zeitstempelservice und Zertifikatsinhabern aufgebaut ist und welche Unterschiede sie mit dem Update Service zu herkömmlichen PKIs aufweist,
- welche kryptographischen Verfahren in der Fail-Safe-PKI verwendet werden und insbesondere welche Anforderungen an ihre Unabhängigkeit gestellt werden,
- welche herkömmlichen Funktionalitäten auch in der Fail-Safe-PKI Verwendung finden,
- wie multiple digitale Signaturen in Dokumenten, E-Mails, Policies, Sperrlisten, Zertifikats-Status-Antworten und Zeitstempeln eingesetzt werden,
- wie iterative Verschlüsselungen bei der Verschlüsselung von Dokumenten oder E-Mails und in Authentisierungen zur Anwendung kommen,
- welche Funktionalitäten durch das neue Update Management Protocol geschaffen wurden, um kryptographische Komponenten, Sicherheitsanker und Schlüssel auf sichere Weise, im laufenden Betrieb und auch im Schadensfall in allen Komponenten einer PKI zu aktualisieren,
- wie der exakte Ablauf der Aktualisierung für verschiedene Schadensfälle und verschiedene Anwendungszenarien erfolgt, wobei die Szenarien drei Anwendungsklassen – Signatur-Anwendung, reale Authentisierungs-Anwendung und virtuelle Authentisierungs-Anwendung –, das dort mögliche Equipment – Mail Client, Multi User Client, Chipkarten, Zugangskontroll-Client sowie Rechner in Client-Server-Architekturen – und die daraus resultierenden verschiedenen Transportwege des Update Management Protocols mit dem UpdateComponent und UpdateComponentResponse abdecken, und
- welche Performance und Kosten eine Fail-Safe-PKI für verschiedene Anwendungen und Sicherheitsniveaus hat, wobei zusätzliche Kosten für die Organisation und das technische Equipment sowie Performance-Verluste in der Nutzungsphase bei Mail Clients, Chipkarten sowie Zeitstempel-, Revokations- und Verzeichnisservern den Vorteilen im Schadensfall entgegengesetzt werden müssen, wenn die Verfügbarkeit von PKI-Anwendungen gewährleistet, die Beweisbarkeit digitale Signaturen erhalten, die Revokationsmechanismen praktikabel und verschlüsselt übertragene Daten vertraulich bleiben.

Zwei Beispiele zeigen im nächsten Kapitel 5 die konkrete Gestaltung einer Fail-Safe-PKI für eine Signatur- und eine Authentisierungs-Anwendung. Diese Beispiele verdeutlichen den konkreten Mehraufwand für das Fail-Safe-Konzept und schildern die Funktionalitäten in der Nutzungsphase und insbesondere beim Auftritt von Schäden.

Kapitel 5

Beispiele

Wie in Kapitel 1 erläutert, lassen sich PKI-Anwendungen in Signatur- und Authentisierungs-Anwendungen einteilen. In diesem Abschnitt wird für diese beiden Anwendungen das konkrete Fail-Safe-Konzept und insbesondere der Austausch von Komponenten in den drei Schadensfällen „RSA kompromittiert“, „CA-Schlüssel kompromittiert“ und „CH-Schlüssel kompromittiert“ diskutiert.

5.1 Beispiel einer Signatur-Anwendung

In diesem Beispiel wird eine Signatur-Anwendung beschrieben, in der die Benutzerin Alice zuhause an ihrem Rechner einen Mail Client mit einer Software-PSE installiert hat, so dass sie eigene E-Mails signieren und fremde E-Mails verifizieren kann – illustriert in Graphik 5.1.

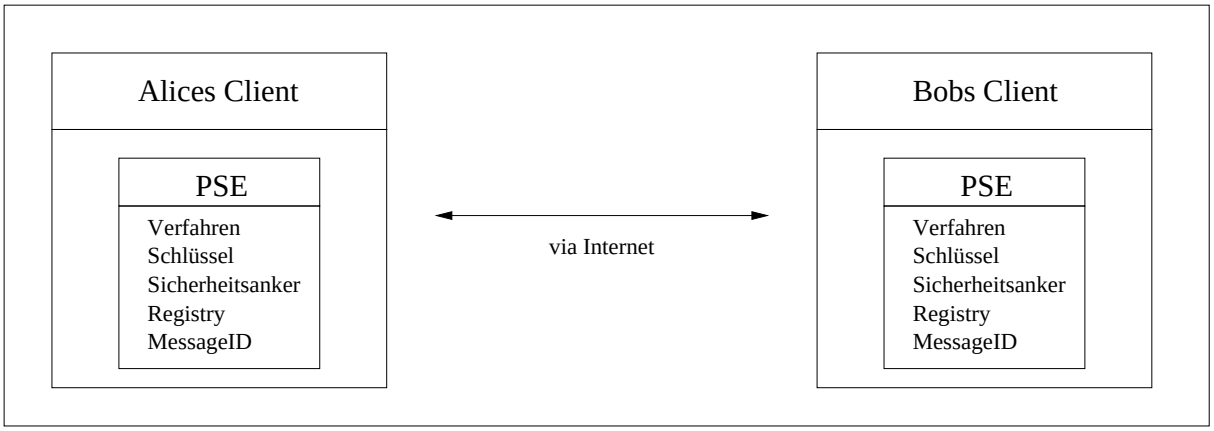

Abbildung 5.1: Beispiel einer Signatur-Anwendung

Alice ist also die Zertifikatsinhaberin; sie nutzt Zertifikate einer Zertifizierungsinstanz, die in dieser einstufigen PKI auch die RootCA ist. In der Fail-Safe-PKI sind zwei Teil-PKIs PKI^1 und PKI^2 installiert; PKI^1 nutzt RSA und PKI^2 ECDSA als Signaturalgorithmus. Ihr Kommunikationspartner Bob ist ebenfalls ein Zertifikatsinhaber in dieser PKI. Das Szenario, welches in diesem Beispiel diskutiert wird, ist: Was passiert, wenn RSA kompromittiert wird?

Das Beispiel ist so gestaltet, dass zunächst die konkrete Konfiguration von CA und CHs mit Verfahren, Schlüsseln und Zertifikaten vorgestellt und anschließend die normale und erweiterte Funktionsweise mit Nutzung multipler digitaler Signaturen beschrieben wird. Hauptaugenmerk dieses Abschnitts liegt auf der Beschreibung des Ablaufs im Schadensfall, um Kryptoverfahren, Schlüssel und Sicherheitsanker sicher und im laufenden Betrieb auszutauschen. Eine Aufwandsabschätzung rundet dieses Beispiel ab.

5.1.1 Konfiguration

Welche Kryptoverfahren, Schlüssel und Zertifikate sind verfügbar?

Zertifizierungsinstanz und Zertifikatsinhaber verfügen über folgende kryptographische Komponenten:

- Signaturalgorithmen: $\sigma^1 = \text{RSA}$; $\sigma^2 = \text{ECDSA}$
- Voneinander unabhängige Hashfunktionen: $\kappa^1 = \text{RIPEMD-160}$; $\kappa^2 = \text{SHA-1}$
- Voneinander unabhängige Formatierungen: $\phi^1 = \text{PKCS\#1}$ bei RSA; $\phi^2 = 1$ bei ECDSA, da dort keine Formatierung nötig ist
- Fail-Safe-Konzept-geeignete Signaturverfahren: $\text{sign}^1 \mathrel{\widehat{=}}$ rsasignatureWithripemd160 und $\text{sign}^3 \mathrel{\widehat{=}}$ sha1WithRSAENC in PKI^1; $\text{sign}^2 \mathrel{\widehat{=}}$ ecdsa-with-SHA1 und $\text{sign}^4 \mathrel{\widehat{=}}$ ecSignWithripemd160 in PKI^2

Die CA verfügt über folgende Schlüssel, Zertifikate und Sperrlisten:

- Schlüsselpaare der CA: $(\text{prK}^1_{\text{CA}}, \text{puK}^1_{\text{CA}})$ zu $\sigma^1 = \text{RSA}$; $(\text{prK}^2_{\text{CA}}, \text{puK}^2_{\text{CA}})$ zu $\sigma^2 = \text{ECDSA}$
- Zertifikate der CA: $\text{cert}^1_{\text{CA}}$ zu σ^1; $\text{cert}^2_{\text{CA}}$ zu σ^2
- Weitere Zertifikate: $\{\text{cert}^1_{\text{CH}} \mid \text{alle CHs}\}$; $\{\text{cert}^2_{\text{CH}} \mid \text{alle CHs}\}$
- Sperrlisten: CRL^1, einfach signiert mit sign^1; CRL^2, einfach signiert mit sign^2; $\text{CRL}^{1,2}$, multipel signiert mit sign^1 und sign^2

Eine Dienstleistung der CA ist der Zeitstempeldienst (TSS), der von der Zeitstempelinstanz (TSA) angeboten wird. Die TSA verfügt ebenfalls über die beschriebenen kryptographischen Komponenten sowie

- Schlüsselpaare, $(\text{prK}^1_{\text{TSA}}, \text{puK}^1_{\text{TSA}})$ zu σ^1 und $(\text{prK}^2_{\text{TSA}}, \text{puK}^2_{\text{TSA}})$ zu σ^2, und
- Zertifikate, $\text{cert}^1_{\text{TSA}}$ zu σ^1 und $\text{cert}^2_{\text{TSA}}$ zu σ^2.

Jeder CH verfügt über folgende Schlüssel, Zertifikate und Sicherheitsanker:

- Schlüsselpaare jedes CHs: $(\text{prK}^1_{\text{CH}}, \text{puK}^1_{\text{CH}})$ zu σ^1; $(\text{prK}^2_{\text{CH}}, \text{puK}^2_{\text{CH}})$ zu σ^2
- Zertifikate jedes CHs: $\text{cert}^1_{\text{CH}}$ zu σ^1; $\text{cert}^2_{\text{CH}}$ zu σ^2
- Sicherheitsanker: $\text{cert}^1_{\text{CA}}$ mit puK^1_{CA} zu σ^1; $\text{cert}^2_{\text{CA}}$ mit puK^2_{CA} zu σ^2

5.1.2 Normale Funktionsweise

Im Rahmen der normalen Funktionsweise können Alice und Bob E-Mails mit einfachen digitalen Signaturen versehen und einfach signierte E-Mails verifizieren. Dazu können Zertifikate, Sperrlisten und Zertifikats-Status-Anfragen angefordert, verifiziert und ausgewertet werden. Des Weiteren nutzen Alice und Bob den Zeitstempeldienst. In diesem Abschnitt wird die normale Funktionsweise kurz erläutert.

Alice und Bob stellen jeweils bei ihrem Mail Client – z. B. Microsoft Outlook – ein, ob sie E-Mails signiert versenden wollen. Dazu wählen sie aus einem Menü aus, welches der vorhandenen Signaturverfahren eine Signatur erzeugen soll: $\text{sign}^1 \mathrel{\widehat{=}}$ rsasignatureWithripemd160, $\text{sign}^2 \mathrel{\widehat{=}}$ ecdsa-with-SHA1, $\text{sign}^3 \mathrel{\widehat{=}}$ sha1WithRSAENC oder $\text{sign}^4 \mathrel{\widehat{=}}$ ecSignWithripemd160. Die signierte E-Mail wird in PKCS#7 mit S/MIME an den Empfänger verschickt. Der Mail Client des Empfängers interpretiert das PKCS#7-Datenpaket und extrahiert aus SignedData das verwendete Signaturverfahren, um die Signatur zu verifizieren. Der Mail Client zeigt das Ergebnis der Prüfung dem Benutzer an. Der Benutzer kann seinen Client so konfiguriert haben, dass dieser eine Validierung durchführt. In dem Fall besorgt sich der Client die benötigten Zertifikate via LDAP und Revokationsinformationen in Form von Sperrlisten oder einer Zertifikats-Status-Anfrage, verifiziert die Signaturen der Revokationsinformationen und validiert damit die Zertifikate zu prüfender Signaturen. Um den Zeitstempeldienst (TSS) zu nutzen, verfügen Alices und Bobs Clients über eine Applikation, die das zeitzustempelnde Dokument hasht – wobei eingestellt werden kann, ob RIPEMD-160 oder SHA-1 genutzt wird – und via Zeitstempelprotokoll TSP einen Zeitstempel anfordert und erhält.

5.1.3 Erweiterte Funktionsweise

Alice und Bob möchten die erweiterten Funktionalitäten der Fail-Safe-PKI nutzen, um die Probleme im Schadensfall zu umgehen. Dieser Abschnitt beschreibt, was sie dazu tun.

Alice und Bob haben ihren Mail Client jeweils so konfiguriert, dass multipel signierte Sperrlisten über eine gesonderte LDAP-Adresse und multipel signierte Zertifikats-Status-Antworten über OCSPv11 angefordert und ausgewertet werden. Das Beispiel eines OCSPv11-Protokolls ist in Abschnitt A.4 auf Seite 198 abgebildet.

Ihre normale Kommunikation via E-Mail signieren sie weiterhin einfach. Sie könnten aber auch multiple Signaturen nutzen, indem sie in den Einstellungen des Mail Clients einstellen, dass zwei Signaturen erzeugt werden sollen. Der Mail Client erkennt bei einer empfangenen multipel signierten E-Mail zwei in PKCS#7 integrierte Signaturen automatisch und wertet – je nach Systemkonfiguration – beide oder nur eine aus und zeigt dem Benutzer das Ergebnis an.

Wichtige Dokumente, die Beweischarakter haben, lassen Alice und Bob vom Zeitstempeldienst mit einer multiplen digitalen Signatur zeitstempeln. Dazu fordern sie in der Zeitstempel-Applikation das Zeitstempelprotokoll TSPv11 an, so dass ihr zeitzustempelndes Dokument zweimal mit RIPEMD-160 und SHA-1 gehasht und diese beiden Hashwerte zusammen mit der Zeit vom Zeitstempeldienst multipel signiert werden. Die TSA nutzt zwei voneinander unabhängige Signaturverfahren, d. h. in diesem Fall entweder rsasignatureWithripemd160 und ecdsa-with-SHA1 oder sha1WithRSAENC und ecSignWithripemd160. Das Beispiel eines TSPv11-Protokolls ist in Abschnitt A.5 auf Seite 203 beschrieben.

5.1.4 Ablauf im Schadensfall – Austausch kompromittierter Komponenten

Angenommen, RSA sei kompromittiert! Angenommen, es sei möglich, aus einem öffentlichen Schlüssel mit einem neuen Faktorisierungsalgorithmus auf effektive Weise den privaten Schlüssel zu berechnen, so dass bei keiner Signatur mehr davon auszugehen ist, sie stamme tatsächlich vom behaupteten Signierer. Nach heutigem Kenntnisstand ist von einem solchen Schaden ECDSA unberührt. Was im Folgenden passiert, um die Sicherheit wiederherzustellen und um in der Fail-Safe-PKI dynamisch RSA-Komponenten durch noch sichere Komponenten zu ersetzen, ist Inhalt dieses Abschnitts.

Zunächst würde das BSI etwa in einer Eilmitteilung eine neue Liste geeigneter Kryptoalgorithmen herausgeben, woraufhin die Trust Center Zertifikate, die RSA-Schlüssel zertifizieren, und mit RSA signierte Zertifikate revozieren würden.

Da Sperrlisten und Zertifikats-Status-Antworten multipel signiert werden und Alice und Bob stets beide Signaturen auswerten, können sie diesen Sperrinformationen weiterhin Glauben schenken. Da Alice und Bob wichtige Dokumente mit multipel signierten Zeitstempeln versehen haben, von denen die ECDSA-Signatur weiterhin sicher ist, bleiben diese Dokumente trotz Schadensfall beweiskräftig.

Darüber hinaus bleibt die Verfügbarkeit der PKI gewährleistet: Alice und Bob stellen in ihrem Mail Client ein, dass sie zukünftig ihre E-Mails mit ecdsa-with-SHA1 oder ecSignWithripemd160 signieren möchten, so dass sie auch weiterhin sicher miteinander kommunizieren können.

Bisher sind also Phase 0 „Eintritt Schaden“ und Phase 1 „Revokation“ des Ablaufs im Schadensfall entsprechend Abbildung 4.3 auf Seite 109 durchgeführt worden.

Angenommen, es stehe ein neuer Signaturalgorithmus praktisch bereit, den der neue Faktorisierungsalgorithmus, welcher RSA kompromittierte, nicht tangiert und der zur elliptischen Kurven-Kryptographie unabhängig ist – sei dies z. B. IQDSA. Aus diesem Signaturalgorithmus ergeben sich zusammen mit RIPEMD-160 und SHA-1 zwei neue Signaturverfahren RipeMD160/IQDSASignatureAlgorithm und SHA1/IQDSASignatureAlgorithm (entsprechend den offiziellen OID-Distinguished Names). Praktisch ausgetauscht werden Provider. Der bisherige `FlexiCoreProvider`, der u. a. alle RSA nutzenden Signaturverfahren enthält, soll durch den `FlexiNFProvider` ersetzt werden, der dieselben Verfahren wie der `FlexiCoreProvider` enthält, aber mit dem Unterschied, dass statt RSA stets IQDSA verwendet wird. `FlexiNFProvider` enthält somit neben RipeMD160/IQDSASignatureAlgorithm und SHA1/IQDSASignatureAlgorithm auch z. B. DES, IDEA, RIJNDAEL, MD5, SHA-1 oder RIPEMD-160. Darüber hinaus wird ein Übergangs-Provider `Provider_neu_Update_Modus` zusammengestellt, der die Vereinigung aus `FlexiCoreProvider` und `FlexiNFProvider` bildet und in dem insbesondere die kompromittierten als auch die neuen Komponenten enthalten sind. Die Konstruktion des Übergangs-Providers ist nötig, um bei Störungen des Ablaufs sicherzustellen, dass Kryptoverfahren für ein erneutes UpdateComponent inklusive UMP-Verifikation zur Verfügung stehen.

Der Ablauf, um RSA-Komponenten durch IQDSA-Komponenten zu ersetzen, wird im Folgenden beschrieben und durch Abbildung 5.2 illustriert; dabei entsprechen die Nummern in der Abbildung der folgenden Nummerierung.

1. Der Update Service generiert in Phase 2 ein UpdateComponent, welches
   - Informationen über die anstehenden Aktionen,
   - Informationen über zu löschende Verfahren, RSA-Sicherheitsanker und eigene RSA-Schlüssel,

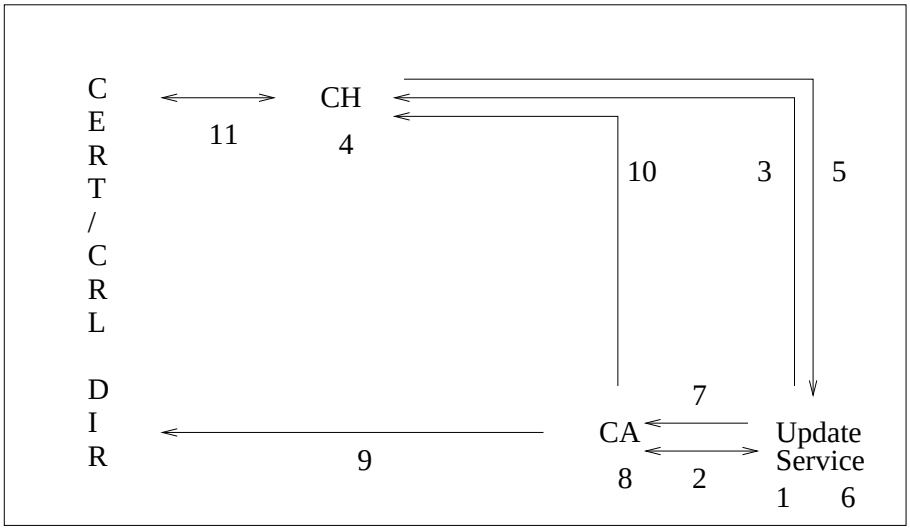

Abbildung 5.2: Ablauf in Signatur-Anwendung (Beispiel)

- einen Link für die zu installierenden kryptographischen Komponenten bzw. Provider,
- einen neuen Sicherheitsanker zu IQDSA,
- das Kommando, einen neuen Schlüssel für IQDSA zu generieren, und
- eine neue Registry

enthält.

2. Der Update Service lässt das UpdateComponent von der Certification Authority entweder mit rsasignatureWithripemd160 und ecdsa-with-SHA1 oder mit sha1WithRSAENC und ecSignWithripemd160 signieren.

3. Der Update Service schickt dieses UpdateComponent in PKCS#7 codiert als E-Mail direkt an alle involvierten Zertifikatsinhaber.

4. In Phase 3 empfängt Alice das UpdateComponent in dem Moment, in dem sie die Mail vom Update Service erhält und öffnet. Der Mail Client erkennt das UpdateComponent, verifiziert eine(!) Signatur zur frühzeitigen Erkennung von Denial-of-Service-Attacken und springt in den Update Modus, sofern die Signatur gültig, d. h. mathematisch korrekt, war. Der Client interpretiert das UpdateComponent: Zunächst wird die Versionsnummer ausgewertet und in eine entsprechende Routine verzweigt. Alice wird der Text angezeigt, der sie über die anstehenden Aktionen unterrichtet. Sie wird gefragt, ob sie den Austausch jetzt oder später durchführen will. Falls sie dies später tun will, wird der Austausch beendet. Der Austausch kann erneut angestoßen werden, wenn die E-Mail ein weiteres Mal geöffnet wird. Will Alice den Austausch durchführen, so bestätigt sie den entsprechenden Button auf der ihr angezeigten Meldung.
In Phase 4 werden Client-relevante Aktionen durchgeführt: Zunächst wird in einer UMP-Verifikation geprüft, ob das Format korrekt ist, ob zwei Signaturen vorhanden und mathematisch korrekt sind und es wird anhand der Registry überprüft, ob die beiden Signaturen für diesen Update geeignet sind – d. h. konkret, ob das UpdateComponent mit

rsasignatureWithripemd160 und ecdsa-with-SHA1 oder mit sha1WithRSAENC und ecSignWithripemd160 signiert wurde. Ist die UMP-Verifikation korrekt, wird fortgefahren. Da ein Algorithmus auszutauschen ist, werden die neuen Provider `FlexiNFProvider` und `Provider_neu_Update_Modus` via http geladen. Die beiden Signaturen, welche die Provider absichern, werden verifiziert und es wird geprüft, ob es dieselben Signaturverfahren wie für das UpdateComponent sind. Ist diese Prüfung korrekt, wird `Provider_neu_Update_Modus` installiert. Anschließend wird der neue Sicherheitsanker installiert, ein neues Schlüsselpaar erzeugt und die neue Registry installiert.
Ist die Installation erfolgreich durchgeführt worden, werden die Provider gelöscht, die vom Schaden „RSA kompromittiert“ betroffen sind. Anhand der Registry kann der Client feststellen, welche Provider dies sind. In diesem Fall ist es der `FlexiCoreProvider`. Anschließend werden RSA-Schlüssel und die alte Registry gelöscht.
Phase 5 entfällt in diesem Beispiel, weil keine Chipkarten beteiligt sind.
In Phase 6 wird die Bestätigung des erfolgreichen Updates mit dem neu generierten Schlüssel zum Update Service geschickt: Der Client erzeugt das UpdateComponentResponse mit dem Namen des Zertifikatsinhabers, der UMP-ID und dem neuen Schlüssel und signiert es zweimal – mit RipeMD160/IQDSASignatureAlgorithm und ecdsa-with-SHA1 oder mit SHA1/IQDSASignatureAlgorithm und ecSignWithripemd160.

5. Der Client sendet das UpdateComponentResponse mit dem neu generierten Schlüssel in PKCS#7 codiert via E-Mail an den Update Service zurück, dessen Adresse im UpdateComponent übertragen wurde.
   In Phase 7 werden zum Abschluss des Updates die alten Sicherheitsanker deaktiviert und der Übergangs-Provider `Provider_neu_Update_Modus` durch `FlexiNFProvider` ersetzt. Durch diese Aktionen verlässt der Client den Update Modus.

6. Der Update Service verifiziert das UpdateComponentResponse dahingehend, dass er prüft, ob dieser Zertifikatsinhaber ein UpdateComponent mit dieser UMP-ID erhalten hat.

7. Ist dem so, sendet der Update Service dieses UpdateComponentResponse an die Certification Authority weiter.

8. Die Certification Authority zertifiziert den neuen IQDSA-Schlüssel in einem Zertifikat. Grundlage für diese Zertifizierung ist,

   (a) dass die Anforderung vom Update Service kam,

   (b) dass die IQDSA-Signatur mathematisch korrekt ist,

   (c) dass die ECDSA-Signatur mathematisch korrekt und nicht revoziert ist und

   (d) dass der IQDSA-Schlüssel kryptographisch geeignet und dublettenfrei ist.

   Andernfalls schickt der Update Service Alice ein neues UpdateComponent, dass mit ECDSA und IQDSA abgesichert ist.

9. In Phase 8 stellt die Certification Authority das neue Zertifikat in das Verzeichnis ein, sofern sie dazu von Alice berechtigt ist.

10. Die Certification Authority kann das neue Zertifikat aber auch direkt an Alice schicken – z. B. PKCS#7-codiert in einer E-Mail oder via Certificate Management Protocol (CMP). Analog wird eine neue Policy verteilt.

11. Alternativ kann Alice ihr neues Zertifikat auch via LDAP vom Verzeichnis der gültigen Zertifikate ziehen.
    In Phase 9 wird Applikationsdatenpflege betrieben: Die mit TSPv11 zeitgestempelten Dokumente, die vor Eintritt eines Schadens multiple Signaturen aufwiesen und von denen jetzt eine ungültig ist, werden durch ein Re-Signieren wieder mit gültigen multiplen Signaturen versehen. Dazu werden das ursprüngliche Dokument D zusammen mit dem früheren Zeitstempel $\mathrm{TSP}(D, t_0)$ erneut zeitgestempelt: $\mathrm{TSP}(D|\mathrm{TSP}(D, t_0), t_1)$. Der frühere Zeitstempel ist damit verlängert worden. Denn: Der neue Zeitstempel der Zeitstempelinstanz bestätigt mit einer IQDSA- und einer ECDSA-Signatur, dass ihr zum Zeitpunkt t_1 der Hashwert aus Dokument D und altem Zeitstempel $\mathrm{TSP}(D, t_0)$ vorgelegen hat. Da zum Zeitpunkt t_1 ECDSA sicher ist und der alte Zeitstempel eine ECDSA-Signatur aufweist, kann – sofern die Signatur korrekt ist – davon ausgegangen werden, dass zum früheren Zeitpunkt t_0 das Dokument D bereits existierte.

Damit ist in der Fail-Safe-PKI RSA durch IQDSA im laufenden Betrieb ersetzt worden. Im Folgenden können die Nutzer wieder eine normale oder erweiterte Funktionalität nutzen.

Für diese Anwendung sind einige Komponenten implementiert worden. Details finden sich in Abschnitt 6 auf Seite 177.

5.1.5 Aufwandsabschätzung

Der Mehraufwand dieser Beispiel-Fail-Safe-PKI gegenüber einer PKI ohne Fail-Safe-PKI bezieht sich zunächst auf Komponenten der multiplen Kryptographie, die bei CA und Clients installiert sein müssen. Bei den heute üblichen Speichergrößen sollte dies kein Problem darstellen. Im täglichen Umgang sind es die multipel signierten Sperrlisten und Zertifikats-Status-Antworten, die etwas mehr Zeit zum Erzeugen und Verifizieren der zweiten digitalen Signatur und etwas mehr Netzlast beanspruchen. In Anbetracht der heute üblichen Prozessoren und der Datenmengen, die über das Internet transferiert werden, stellt dieser Mehraufwand – gerade im Hinblick auf ein höheres Sicherheitsniveau – keine unüberwindbare Hürde dar. Der Austausch selbst ist im laufenden Betrieb unter Nutzung einer PKI wesentlich schneller und komfortabler durchzuführen als ohne Nutzung einer PKI, wenn Teilnehmer neue Software, Sicherheitsanker und Zertifikate zugeschickt bekommen, sich von ihrer Authentizität überzeugen und diese installieren müssen.

5.2 Beispiel einer Authentisierungs-Anwendung

Das Beispiel einer Authentisierungs-Anwendung ist Thema dieses Abschnitts. In dieser Anwendung authentisieren sich Mitarbeiter eines Unternehmens mit ihrer Chipkarte gegenüber einem Zugangskontroll-Client mit Kartenleser, dessen Aufgabe es ist, den Zugang zu einem Raum zu kontrollieren. Graphik 5.3 illustriert dieses Szenario.

Der Raum ist als sicherheitskritisch eingestuft. Deshalb validiert der Zugangskontroll-Client bei jeder Chipkarte das Zertifikat via OCSP auf seine aktuelle Gültigkeit.

Die Fail-Safe-PKI ist einstufig. An der Spitze steht die CA, welche vom Unternehmen organisiert wird und welche die RootCA dieser PKI ist. Zertifikatsinhaber sind die Chipkarten der Mitarbeiter und der Zugangskontroll-Client. In der Fail-Safe-PKI sind zwei Teil-PKIs PKI^1 und PKI^2 installiert; PKI^1 nutzt RSA und PKI^2 ECDSA als Signaturalgorithmus. In diesem Beispiel

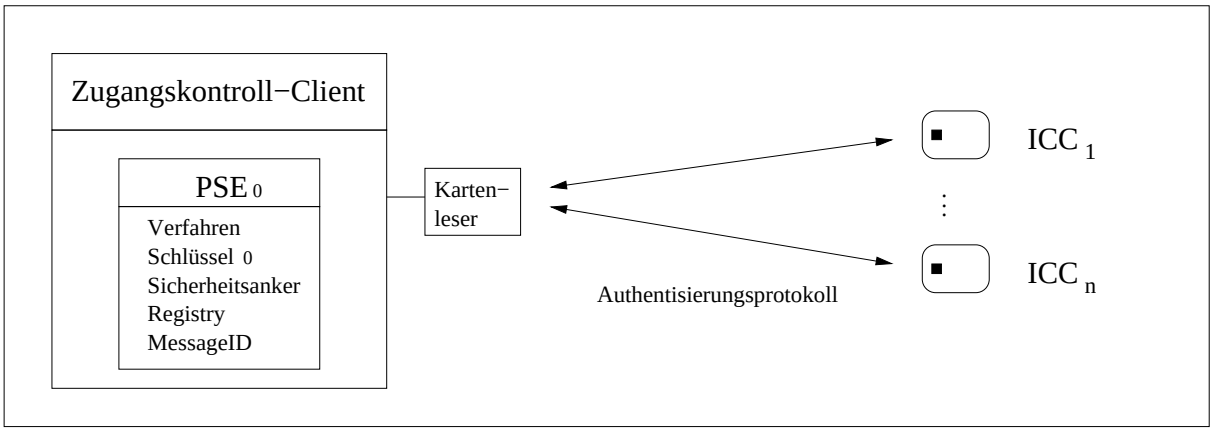

Abbildung 5.3: Beispiel einer Authentisierungs-Anwendung

werden zwei Schäden diskutiert: Kompromittieren des CA-Schlüssels und eines Schlüssels einer Chipkarte.

In diesem Abschnitt wird zunächst die konkrete Konfiguration von CA und CHs mit Verfahren, Schlüsseln und Zertifikaten vorgestellt und anschließend die Funktionsweise beschrieben. Hauptaugenmerk liegt wieder auf der Beschreibung des Ablaufs im Schadensfall und des Transports des UpdateComponents zu den Chipkarten und zurück zum Update Service, um den Sicherheitsanker und den Schlüssel der Chipkarte sicher im laufenden Betrieb auszutauschen. Eine Aufwandsabschätzung rundet dieses Beispiel ab.

5.2.1 Konfiguration

Welche Kryptoverfahren, Schlüssel und Zertifikate sind verfügbar?

Zertifizierungsinstanz und Zertifikatsinhaber verfügen über folgende kryptographische Komponenten:

- Signaturalgorithmen: σ^1 = RSA; σ^2 = ECDSA
- Voneinander unabhängige Hashfunktionen: κ^1 = RIPEMD-160; κ^2 = SHA-1
- Voneinander unabhängige Formatierungen: ϕ^1 = PKCS#1 bei RSA; $\phi^2 = 1$ bei ECDSA, da dort keine Formatierung nötig ist
- Fail-Safe-Konzept-geeignete Signaturverfahren: $\text{sign}^1 \mathrel{\widehat{=}}$ rsasignatureWithripemd160 und $\text{sign}^3 \mathrel{\widehat{=}}$ sha1WithRSAENC in PKI^1; $\text{sign}^2 \mathrel{\widehat{=}}$ ecdsa-with-SHA1 und $\text{sign}^4 \mathrel{\widehat{=}}$ ecSignWithripemd160 in PKI^2

Die CA verfügt über folgende Schlüssel und Zertifikate:

- Schlüsselpaare der CA: $(\text{prK}^1_{\text{CA}}, \text{puK}^1_{\text{CA}})$ zu σ^1 = RSA; $(\text{prK}^2_{\text{CA}}, \text{puK}^2_{\text{CA}})$ zu σ^2 = ECDSA
- Zertifikate der CA: $\text{cert}^1_{\text{CA}}$ zu σ^1; $\text{cert}^2_{\text{CA}}$ zu σ^2
- Weitere Zertifikate:
  $\{\text{cert}^1_{\text{CH}} \mid \text{alle CHs}\}$; $\{\text{cert}^2_{\text{CH}} \mid \text{alle CHs}\}$

Jeder CH – also der Zugangskontroll-Client und jede Chipkarte – verfügt über folgende Schlüssel, Zertifikate und Sicherheitsanker:

- Schlüsselpaare jedes CHs: $(\mathrm{prK}^1_{\mathrm{CH}}, \mathrm{puK}^1_{\mathrm{CH}})$ zu σ^1; $(\mathrm{prK}^2_{\mathrm{CH}}, \mathrm{puK}^2_{\mathrm{CH}})$ zu σ^2
- Zertifikate jedes CHs: $\mathrm{cert}^1_{\mathrm{CH}}$ zu σ^1; $\mathrm{cert}^2_{\mathrm{CH}}$ zu σ^2
- Sicherheitsanker: $\mathrm{cert}^1_{\mathrm{CA}}$ mit $\mathrm{puK}^1_{\mathrm{CA}}$ zu σ^1; $\mathrm{cert}^2_{\mathrm{CA}}$ mit $\mathrm{puK}^2_{\mathrm{CA}}$ zu σ^2

5.2.2 Funktionsweise

Die normale Authentisierung verläuft wie folgt:

1. Eine Chipkarte wird in den Kartenleser eingeführt.
2. Der Zugangskontroll-Client spricht die Chipkarte an und stellt Schlüssel-ID für $\mathrm{prK}^1_{\mathrm{CH}}$ und Algorithmus-ID zu sign^1 ein. D. h. es wird defaultmäßig ausschließlich das Signaturverfahren $\mathrm{sign}^1 \mathrel{\widehat{=}}$ rsasignatureWithripemd160 genutzt.
3. Der Client lässt sich von der Chipkarte eine zufällige Challenge RND signieren, welche daraufhin $S = \mathrm{sign}^1(\mathrm{RND}, \mathrm{prK}^1_{\mathrm{CH}})$ im Response zurückliefert. Um den Signaturvorgang in der Karte zu autorisieren, muss der Benutzer der Karte eine PIN in das PIN-Pad des Kartenlesers eingeben.
4. Der Client erhält außerdem das Zertifikat $\mathrm{cert}^1_{\mathrm{CH}}$ der Chipkarte.
5. Der Client verifiziert das Zertifikat der Chipkarte und anschließend mit dem ihm nun bekannten authentischen öffentlichen Schlüssel der Chipkarte die Signatur S der Challenge.
6. Der Client validiert das Zertifikat, indem er via OCSPv11 den Status des Zertifikats abfragt. Die OCSP-Antwort ist multipel signiert.
7. Sind die Signaturen der OCSP-Antwort korrekt und ist das Zertifikat gültig, kann der Zugangskontroll-Client davon ausgehen, dass tatsächlich die im Zertifikat angegebene Chipkarte mit ihm kommuniziert.
8. Um dem Benutzer der spezifischen Chipkarte nun den Zugang zum Raum gewähren zu dürfen, prüft der Client, ob im Zertifikat entsprechende Attribute bzgl. Zugang festgelegt sind.

Das Authentisierungsprotokoll zwischen Client und Chipkarte nutzt ausschließlich PKI^1. PKI^2 wird nur zwischen CA und Zugangskontroll-Client genutzt, um bei diesem sicherheitskritischen Raum die Zertifikats-Status-Antworten abzusichern. Das Beispiel eines OCSPv11-Protokolls ist in Abschnitt A.4 auf Seite 198 abgebildet.

5.2.3 Ablauf im Schadensfall – Austausch des Sicherheitsankers

Angenommen, der RSA-Schlüssel der CA sei kompromittiert und die CA bemerke dies! Angenommen, der private Schlüssel der CA ($\mathrm{prK}^1_{\mathrm{CA}}$) sei aus der CA gestohlen worden. Da mit diesem Schlüssel beliebige gültige Zertifikate erzeugt werden können, ist bei keiner Authentisierung mehr davon auszugehen, dass tatsächlich die behauptete Chipkarte authentisiert wird.

Nach Voraussetzung werden die CA-Schlüssel unabhängig voneinander aufbewahrt, so dass davon ausgegangen werden kann, dass die CA-Schlüssel zu ECDSA weiterhin sicher sind. Was im Folgenden passiert, um die Sicherheit wiederherzustellen und um in der Fail-Safe-PKI dynamisch den Sicherheitsanker zu ersetzen, ist Inhalt dieses Abschnitts.

Zunächst wird in einem solchen Fall das Root-Zertifikat revoziert.

Da Zertifikats-Status-Antworten multipel signiert werden und der Zugangskontroll-Client stets beide Signaturen auswertet, kann der Client diesen Sperrinformationen weiterhin Glauben schenken.

Darüber hinaus bleibt die Verfügbarkeit der PKI gewährleistet: Der zuvor gezeigte Ablauf der normalen Authentisierung ändert sich ausschließlich in den Punkten 2. 3. und 4.:

- zu 2.:
  Der Zugangskontroll-Client spricht die Chipkarte an und stellt Schlüssel-ID für $\mathrm{prK}^2_{\mathrm{CH}}$ und Algorithmus-ID zu $\mathrm{sign}^2 \mathrel{\widehat{=}}$ ecdsa-with-SHA1 ein.

- zu 3.:
  Der Client lässt sich von der Chipkarte eine zufällige Challenge RND signieren, welche daraufhin $S = \mathrm{sign}^2(\mathrm{RND}, \mathrm{prK}^2_{\mathrm{CH}})$ im Response zurückliefert. Um den Signaturvorgang in der Karte zu autorisieren, muss der Benutzer der Karte eine PIN in das PIN-Pad des Kartenlesers eingeben.

- zu 4.:
  Der Client erhält außerdem das Zertifikat $\mathrm{cert}^2_{\mathrm{CH}}$ der Chipkarte.

Bisher sind also Phase 0 „Eintritt Schaden“ und Phase 1 „Revokation“ des Ablaufs im Schadensfall entsprechend Abbildung 4.3 auf Seite 109 durchgeführt worden.

Angenommen, das „Sicherheitsloch“ in der CA – das es ermöglichte, den RSA-Schlüssel zu stehlen – sei beseitigt worden und es sei ein neues RSA-Schlüsselpaar der CA erzeugt worden – $(\mathrm{prK}^{1\_neu}_{\mathrm{CA}}, \mathrm{puK}^{1\_neu}_{\mathrm{CA}})$.

Der Ablauf, um den Sicherheitsanker bei den Clients zu ersetzen, wird im Folgenden beschrieben und durch Abbildung 5.4 illustriert; dabei entsprechen die Nummern in der Abbildung der folgenden Nummerierung. Bei dieser Beschreibung ist besonders der Transport des Update-Components zu den Chipkarten von Interesse. Das linke Feld “CERT/CRL DIR“ (in Abb. 5.4) schließt die OCSP-Dienstleistung mit ein.

1. Der Update Service generiert in Phase 2 ein UpdateComponent, welches
   - die Information, den RSA-Sicherheitsanker zu löschen, und
   - einen neuen Sicherheitsanker zu RSA, den es zu installieren gilt, enthält.
   - Desweiteren ist eine MessageID vergeben, die größer als alle bisher vergebenen MessageIDs ist. Im Authentisierungsprotokoll kann der Client auf diese Weise schnell herausfinden, ob die betreffende Chipkarte bereits mit einem UpdateComponent versehen wurde.

2. Der Update Service lässt das UpdateComponent von der Certification Authority entweder mit rsasignatureWithripemd160 und ecdsa-with-SHA1 oder mit sha1WithRSAENC und ecSignWithripemd160 signieren.

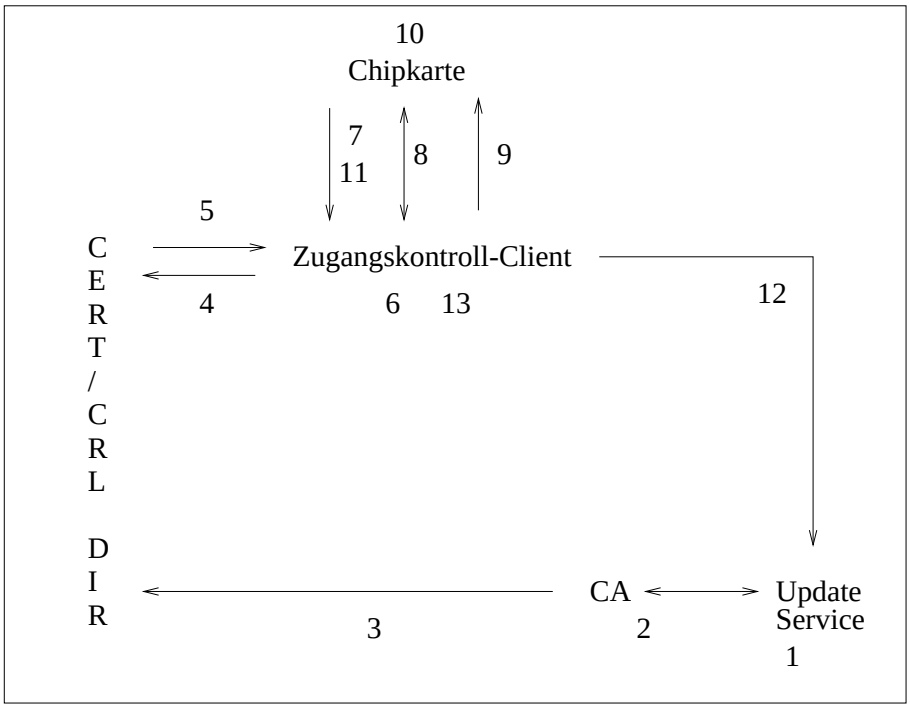

Abbildung 5.4: Ablauf in Authentisierungs-Anwendung (Austausch Sicherheitsanker)

3. Die Certification Authority stellt dieses UpdateComponent in das Verzeichnis DIR ein, so dass

4. bei jeder OCSP-Anfrage über OCSPv11 vom Zugangskontroll-Client dieses UpdateComponent in die

5. OCSP-Antwort integriert wird.

6. In Phase 3 empfängt der Zugangskontroll-Client das UpdateComponent in dem Moment, in dem er die OCSP-Antwort auswertet. Der Zugangskontroll-Client erkennt das UpdateComponent, verifiziert eine(!) Signatur zur frühzeitigen Erkennung von Denial-of-Service-Attacken und springt in den Update Modus, sofern die Signatur gültig war.
   Der Client interpretiert das UpdateComponent: Zunächst wird die Versionsnummer ausgewertet und in eine entsprechende Routine verzweigt.
   In Phase 4 werden Client-relevante Aktionen durchgeführt: Zunächst wird in einer UMP-Verifikation geprüft, ob das Format korrekt ist, ob zwei Signaturen vorhanden und mathematisch korrekt sind und es wird anhand der Registry überprüft, ob die beiden Signaturen für diesen Austausch geeignet sind. Ist die UMP-Verifikation korrekt, wird fortgefahren und der neue Sicherheitsanker im Zugangskontroll-Client installiert.
   Der Client befindet sich im Update Modus, der besagt, dass der Client bei jede Authentisierung versucht, das UpdateComponent an Chipkarten abzusetzen.

7. Eine Chipkarte beginnt mit der Authentisierung. Dieser Schritt entfällt beim ersten Mal, weil während dieser Authentisierung die OCSP-Antwort mit dem UpdateComponent erst beim Client ankommt.

8. Der Zugangskontroll-Client testet, ob die Chipkarte das UpdateComponent bereits erhielt, indem er prüft, ob die bei der Chipkarte gespeicherte sicherheitsrelevante MessageID kleiner als die MessageID des UpdateComponents ist. Es gibt zwei Möglichkeiten: Entweder liest er die MessageID aus, wobei die Chipkarte dafür Sorge zu tragen hat, dass diese ID nicht von außen verändert werden kann. Oder der Client schickt ein neu zu definierendes Kommando an die Karte mit der MessageID des UpdateComponents, woraufhin die Chipkarte den Vergleich durchführt und im Response entsprechend antwortet.

9. Der Zugangskontroll-Client schickt in Phase 5 das UpdateComponent entsprechende Chipkarten-Kommando UPDATE COMPONENT COMMAND an die Chipkarte, sofern der Test ergeben hat, dass bei dieser Chipkarte UMP noch nicht ausgeführt wurde. Zuvor muss der Client über den Link in UpdateComponent das für die Chipkarte bestimmte UPDATE COMPONENT COMMAND laden.

10. In der Chipkarte wird der Austausch durchgeführt: Zunächst wird geprüft, ob die übertragene MessageID größer der gespeicherten ist. Ist dem so, wird in einer UMP-Verifikation geprüft, ob das Format korrekt ist, ob zwei Signaturen vorhanden und mathematisch korrekt sind und es wird anhand der Registry überprüft, ob die beiden Signaturen für diesen Austausch geeignet sind. Ist die UMP-Verifikation korrekt, wird fortgefahren und der neue Sicherheitsanker in der Karte installiert.

11. In Phase 6 generiert die Chipkarte das UPDATE COMPONENT RESPONSE, in das sie ihre eigene Identität und die UMP-ID des UPDATE COMPONENT COMMANDs integriert und zweimal – mit rsasignatureWithripemd160 und ecdsa-with-SHA1 oder mit sha1WithRSAENC und ecSignWithripemd160 – signiert. Dieses RESPONSE sendet die Karte an den Zugangskontroll-Client, der

12. dieses UPDATE COMPONENT REPONSE in ein ASN.1-codiertes UpdateComponent-Response integriert, das er in PKCS#7 codiert via E-Mail an den Update Service sendet, dessen Adresse im UpdateComponent übertragen wurde. Analog verfährt der Client mit dem UpdateComponentResponse des für ihn bestimmten UpdateComponents.

13. Falls die OCSP-Antworten nicht mehr das UpdateComponent enthalten, weil die Trust-Center-Administratoren festgestellt haben, dass der Austausch bei hinreichend vielen Chipkarten durchgeführt wurde, wird das UpdateComponent aus dem Verzeichnis entfernt. Alle nun folgenden OCSP-Antworten enthalten dieses UpdateComponent nicht mehr.
    Wenn der Zugangskontroll-Client eine OCSP-Antwort ohne dieses UpdateComponent empfängt, leitet der Client in Phase 7 den Abschluss des Updates ein: Der alte Sicherheitsanker im Client wird deaktiviert und der Update Modus verlassen. Im Folgenden wird der Client nicht mehr versuchen, UpdateComponents an Chipkarten abzusetzen oder zu testen, ob der Update bei einer Chipkarte bereits durchgeführt wurde.

Damit ist in der Fail-Safe-PKI ein neuer RSA-Sicherheitsanker im laufenden Betrieb ersetzt worden.
Für diese Anwendung ist OCSPv11 implementiert worden. Details finden sich in Abschnitt 6.2 auf Seite 178.

5.2.4 Ablauf im Schadensfall - Austausch des Schlüssels einer Chipkarte

Angenommen, der RSA-Schlüssel einer Chipkarte A (prK_A^1) sei kompromittiert und der CA sei dies bekannt! Was im Folgenden passiert, um die Sicherheit wiederherzustellen und um in

der Fail-Safe-PKI dynamisch die Chipkarte mit einem neuen RSA-Schlüssel und Zertifikat zu versorgen, ist Bestandteil dieses Abschnitts.

Zunächst wird in einem solchen Fall das Zertifikat cert_A^1 revoziert. Da der Zugangskontroll-Client stets Zertifikats-Status-Antworten auswertet, ist die Sicherheit des Raumes ab Revokationszeitpunkt gewährleistet.

Wie erhält diese Chipkarte nun einen neuen zertifizierten RSA-Schlüssel? Abbildung 5.5 illustriert den Ablauf, der im Folgenden beschrieben wird. Dabei ist besonders der Transport des UpdateComponentResponses mit dem neuen Schlüssel von der Chipkarte zur CA von Interesse. Wieder schließt das linke Feld "CERT/CRL DIR" (in Abb. 5.5) einen OCSP-Dienst mit ein.

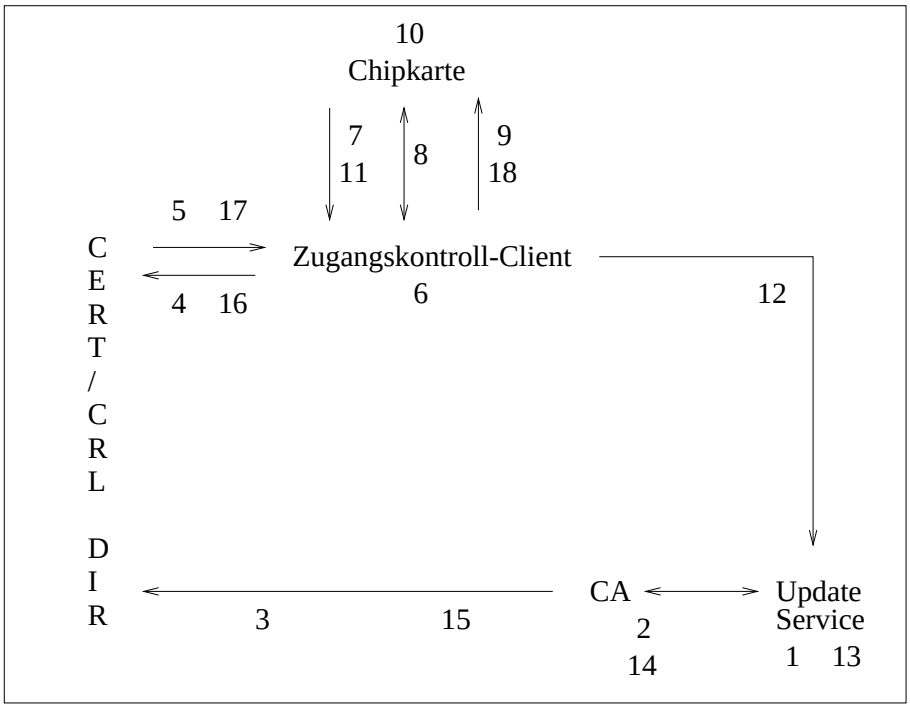

Abbildung 5.5: Ablauf in Authentisierungs-Anwendung (Austausch CH-Schlüssel)

1. Der Update Service generiert in Phase 2 ein UpdateComponent, welches
   - die Information, den RSA-Schlüssel zu löschen, und
   - das Kommando, einen neuen RSA-Schlüssel zu erzeugen, enthält.
   - Desweiteren ist die Chipkartenseriennummer als Identität des CHs angegeben und
   - eine MessageID vergeben, die größer als die bei Chipkarte A gespeicherte MessageID ist.
2. Der Update Service lässt das UpdateComponent von der Certification Authority zweimal signieren.
3. Die Certification Authority stellt dieses UpdateComponent in das Verzeichnis DIR ein, so dass

4. bei jeder OCSP-Anfrage über OCSPv11 vom Zugangskontroll-Client dieses UpdateComponent in die

5. OCSP-Antwort integriert wird.

6. In Phase 3 empfängt der Zugangskontroll-Client das UpdateComponent in dem Moment, in dem er die OCSP-Antwort auswertet. Der Zugangskontroll-Client erkennt das UpdateComponent, verifiziert eine(!) Signatur zur frühzeitigen Erkennung von Denial-of-Service-Attacken und springt in den Update Modus, sofern die Signatur gültig, d. h. mathematisch korrekt, war.
   Der Client interpretiert das UpdateComponent: Zunächst wird die Versionsnummer ausgewertet und in eine entsprechende Routine verzweigt.
   Phase 4 wird nicht durchlaufen, weil keine Client-relevanten Aktionen auszuführen sind. Der Client befindet sich im Update Modus, der besagt, dass der Client bei jeder Authentisierung versucht, das UpdateComponent an die richtige Chipkarte A abzusetzen.

7. Eine Chipkarte beginnt mit der Authentisierung. Dieser Schritt entfällt beim ersten Mal, weil während dieser Authentisierung die OCSP-Antwort mit dem UpdateComponent erst beim Client ankommt.

8. Der Zugangskontroll-Client testet, ob die Chipkarte der aktuellen Authentisierung eine für das UpdateComponent bestimmte Karte ist, indem er die Chipkartenseriennummer ausliest und mit der Nummer des UpdateComponents vergleicht.
   Sind die Nummern verschieden, führt er eine normale Authentisierung durch.
   Sind die Nummern identisch, so lädt der Client das für die Chipkarte bestimmte UPDATE COMPONENT COMMAND über den Link und

9. überträgt das UPDATE COMPONENT COMMAND in Phase 5 an die Chipkarte.

10. In der Chipkarte wird der Austausch durchgeführt: Zunächst werden Adresse und MessageID geprüft. Ist die Chipkarte korrekt adressiert und ist der Austausch noch nicht ausgeführt worden (über den Vergleich übertragener und gespeicherter MessageIDs), wird fortgefahren. In einer UMP-Verifikation wird geprüft, ob das Format korrekt ist, ob zwei Signaturen vorhanden und mathematisch korrekt sind und es wird anhand der Registry überprüft, ob die beiden Signaturen für diesen Austausch geeignet sind. Ist die UMP-Verifikation korrekt, wird ein neues Schlüsselpaar erzeugt und das alte gelöscht.

11. In Phase 6 generiert die Chipkarte das UPDATE COMPONENT RESPONSE, in welchem ihre Identität, die UMP-ID, die aktuelle MessageID und der neue öffentliche Schlüssel zweimal – mittels RSA und ECDSA – signiert enthalten sind, und sendet es an den Zugangskontroll-Client.
    Die Authentisierung der Chipkarte gegenüber dem Raum kann mit PKI^2 durchgeführt und anschließend beendet werden; es braucht nicht auf das neue Zertifikat gewartet zu werden.

12. Der Zugangskontroll-Client integriert dieses UPDATE COMPONENT RESPONSE in ein ASN.1-codierte UpdateComponentResponse und schickt es via PKCS#7 an den Update Service.

13. Der Update Service verifiziert das UpdateComponentResponse dahingehend, dass er prüft, ob diese Chipkarte ein UpdateComponent erhalten hat.

14. Ist dem so, sendet der Update Service dieses UpdateComponentResponse an die Certification Authority weiter, die daraufhin den neuen RSA-Schlüssel in einem Zertifikat zertifiziert. Grundlage für diese Zertifizierung ist, dass die Anforderung vom Update Service kam, dass die RSA-Signatur korrekt ist, dass die ECDSA-Signatur mathematisch korrekt und das zugehörige Zertifikat nicht revoziert ist und dass der RSA-Schlüssel kryptographisch geeignet und dublettenfrei ist. Andernfalls schickt der Update Service der Chipkarte A ein erneutes UpdateComponent.
    Die CA entfernt das UpdateComponent für A aus dem Verzeichnis, so dass bei einer nun folgenden OCSP-Antwort dieses UpdateComponent nicht mehr enthalten ist. Der Zugangskontroll-Client registriert dies und verlässt – nun in Phase 7 – den Update Modus für dieses UpdateComponent und versucht daraufhin nicht mehr, das UpdateComponent an Chipkarte A abzusetzen.

15. In Phase 8 stellt die Certification Authority das neue Zertifikat in das Verzeichnis ein.

16. Wenn in einem Authentisierungsprotokoll der Chipkarte ihr Zertifikat fehlt, fordert der Zugangskontroll-Client dieses Zertifikat via LDAP vom Verzeichnis an.

17. Nach Erhalt des Zertifikats nutzt der Client es für die Authentisierung und

18. schreibt es auf die Chipkarte.

Damit ist in der Fail-Safe-PKI für eine Chipkarte A ein neuer RSA-Schlüssel im laufenden Betrieb ausgetauscht worden.

Für diese Anwendung ist OCSPv11 implementiert worden. Details finden sich in Abschnitt 6.2 auf Seite 178.

5.2.5 Aufwandsabschätzung

Der Mehraufwand dieser Beispiel-Fail-Safe-PKI gegenüber einer PKI ohne Fail-Safe-PKI bezieht sich zunächst auf Komponenten der multiplen Kryptographie, die bei CA, Zugangskontroll-Client und Chipkarten installiert sein müssen. Bei CA und Client sollte dies bei den heute üblichen Speichergrößen kein Problem darstellen; bei der Chipkarte ist dies aufgrund ihrer beschränkten Ressourcen kritischer. Im täglichen Umgang sind es die multipel signierten Zertifikats-Status-Antworten, die etwas mehr Zeit zum Erzeugen und Verifizieren der zweiten digitalen Signatur und etwas mehr Netzlast beanspruchen. In Anbetracht der heute üblichen Prozessoren bei Rechnern und der vergleichsweise geringen zusätzlichen Datenmengen stellt dieser Mehraufwand keine unüberwindbare Hürde dar, zumal die Chipkarte in der täglichen Nutzung keine Mehraufgabe zu bewältigen hat. Der Austausch selbst ist im laufenden Betrieb unter Nutzung einer PKI wesentlich schneller und komfortabler durchzuführen als ohne Nutzung einer PKI, wenn neue Chipkarten bestellt, geliefert, personalisiert und an die Mitarbeiter verteilt werden müssen. Es ist auch zu berücksichtigen, dass in diesem Beispiel im Schadensfall kein Ausfall der Verfügbarkeit der PKI zu erwarten ist, was gerade ohne ein Redundanzsystem hohe Kosten verursachen würde, wenn der zu schützende Raum anderweitig abgesichert werden müsste.

5.3 Zusammenfassung

In diesem Kapitel wurde an den beiden Beispielen Signatur- und Authentisierungs-Anwendung demonstriert, über welche Komponenten die entsprechenden Fail-Safe-PKIs verfügen, welche erweiterte Funktionalität sie nutzen können und vor allem wie sich die Schäden „RSA kompromittiert“, „CA-Schlüssel kompromittiert“ und „CH-Schlüssel kompromittiert“ auf die Fail-Safe-PKI auswirken und welcher konkrete Ablauf im Schadensfall vorgesehen ist, um kompromittierte Komponenten im laufenden Betrieb auszutauschen. An diesen Beispielen lässt sich ablesen, dass sich der Mehraufwand der Fail-Safe-PKI im Vergleich zu einer herkömmlichen PKI in Grenzen hält und eine deutlich höhere Sicherheit im Schadensfall bietet.

Kapitel 6

Implementierungen

Zur Realisierung des Fail-Safe-Konzepts für Public-Key-Infrastrukturen sind folgende Protokolle zu implementieren: PKCS#7 mit multiplen digitalen Signaturen und iterativen Verschlüsselungen, multipel signierte Sperrlisten, multipel signierte Zertifikats-Status-Antworten, multipel signierte Zeitstempel und das Update Management Protocol.

Um die Korrektheit und Praktikabilität der neuen Protokolle zu untermauern, wurden Teile des Fail-Safe-Konzepts als Proof-of-Concept implementiert:

- Update Management Protocol (UMP) auf Trust-Center- und Client-Seite
- Online Certificate Status Protocol in Version v11 (OCSPv11) mit multiplen digitalen Signaturen
- Time Stamp Protocol in Version v11 (TSPv11) mit multiplen digitalen Signaturen

PKCS#7 und multipel signierte Sperrlisten wurden im Rahmen dieser Disseration nicht implementiert. PKCS#7 unterstützt die für das Fail-Safe-Konzept geforderten Funktionalitäten zur multiplen Kryptographie bereits. Mail Clients müssen deshalb für das Fail-Safe-Konzept so konzipiert sein, dass sie multiple digitale Signaturen und iterative Verschlüsselungen bearbeiten können – wie etwa das in der FlexiPKI entstandene Plug-In für Microsoft Outlook. Multipel signierte Sperrlisten unterscheiden sich von standardkonformen Sperrlisten nur durch die zweite Signatur, die von entsprechenden Servern erzeugt, dem LDAP-Verzeichnis zur Verfügung gestellt und von Clients ausgewertet werden. Die bei der Implementierung zu OCSPv11 gemachten Erfahren sind in dieses Protokoll eingeflossen.

In diesem Kapitel wird über die Implementierungen im Rahmen der FlexiPKI [FlexiPKI] an der TU Darmstadt berichtet.

6.1 Update Management Protocol

Das Erzeugen eines UpdateComponents im Trust Center sowie das Interpretieren und Ausführen des UpdateComponents und Erzeugen eines -Responses beim Client wurde im Rahmen der Diplomarbeit „Implementierung des Austausches kryptographischer Komponenten in FlexiPKI mittels Update Management Protocol“ von Igor Kalenderian [Kale01] implementiert.

Es ist nunmehr möglich, mit der Trust-Center-Software ein UpdateComponent am Bildschirm zusammenzustellen und an die angegebenen Adressaten zu verschicken. Die Erweiterung des

Microsoft Outlook Plug-Ins erkennt ein in PKCS#7 codiertes UpdateComponent, führt nach Bestätigung des Benutzers den Update durch, lädt dazu nötige Provider über http oder ftp herunter und schickt ein UpdateComponentResponse mit dem neu generierten Schlüssel an das Trust Center zurück. Für Details sei auf [Kale01] verwiesen.

In der Diplomarbeit sind Zeiten genannt, die sich auf einen PC mit Pentium 2 Prozessor mit 500 MHz Taktfrequenz und 128 MB RAM unter Windows NT 4.0 beziehen und in dem zu Testzwecken ECDSA durch RSA ersetzt wurde: Die Gesamtausführungszeit beträgt zwischen 80 und 100 Sekunden, ohne interaktive Benutzereingaben. Davon dauert das Laden des Providers über das Internet zwischen 25 und 30 Sekunden und das Generieren des neuen Schlüsselpaares etwa 50 Sekunden. Die UMP-Nachrichten sind zwischen 3 und 8 KB groß.

6.2 Erweitertes Online Certificate Status Protocol

Eine Zertifikats-Status-Anfrage mit dem Online Certificate Status Protocol (OCSP) verläuft wie folgt: Ein Client fordert bei einem Server ein “OCSPRequest“ an, in dem der OCSP-Client die Zertifikate angibt, von denen er wissen möchte, ob sie zur Zeit gültig und nicht revoziert sind. Der OCSP-Server sendet ein digital signiertes “OCSPResponse“ mit den gewünschten Informationen.

Unter Fail-Safe-Gesichtspunkten kann ein solches Response unzureichend sein, denn in einem Schadensfall, welcher die Signatur des OCSP-Servers berührt, ist das Response wertlos. Aus diesem Grund kann in der Fail-Safe-PKI das Response multipel signiert werden. Dazu kennzeichnet der OCSP-Client im Request durch Angabe einer Versionsnummer, ob er im Response – wie bisher – eine einfache Signatur oder eine multiple Signatur erwartet:

- Einfache Signatur im Response: Version v1(0)
  Im de facto Standard OCSPv1 [OCSP99] ist die Versionsnummer default auf Version v1 eingestellt und das Response wird einfach signiert. v1 wird intern als Integer 0 dargestellt – daher die im folgenden benutzte Darstellung „Version v1(0)“.

- Multiple Signatur im Response: Version v11(10)
  OCSP entwickelt sich weiter: Zum Beispiel kursiert im Internet derzeit ein Draft von OCSP, der als Version v2(1) gehandelt wird [OCSP01]. Um bei diesen und weiteren Entwicklungen nicht in Konflikt mit den Versionsnummern zu geraten, wird die Versionsnummer des OCSP-Requests, der eine multiple Signatur erwartet, mit Version v11(10) definiert.

Clients zu Version v1 und zu Version v11 sind zueinander nicht kompatibel. Entweder setzt der Client ein v1-Request ab und erwartet ein v1-Response, oder er setzt ein v11-Request ab und erwartet ein v11-Response. Beide Requests werden einfach signiert. Der OCSP-Server hingegen ist zu beiden Versionen kompatibel. Der Server antwortet auf ein v1-Request mit einem v1-Response und auf ein v11-Request mit einem v11-Response.

Im Rahmen der Dissertation wurden OCSP-Client und -Server für Version v11 entwickelt. In diesem Abschnitt wird ein Überblick über die Implementierung wiedergegeben, der nicht die Dokumentation darstellt. Die ausführliche Dokumentation findet sich beim Source-Code.

Realisierung

In der FlexiPKI gibt es bereits einen in Java implementierten OCSP-Client (realisiert als `OCSP-Client`) und -Server (`OCSPServlet`), die den oben beschriebenen Request (`OCSPRequest`) bzw. Response (`OCSPResponse`) einfach signiert absetzen und die Signatur verifizieren. Im Sinne der FlexiPKI-Philosophie sind die Verfahren austauschbar gestaltet; es gibt eine Konfigurationsdatei (`ocspserver.config`), in der die unterstützten Verfahren eingestellt werden.

Diese bestehenden Programme wurden im Rahmen der Fail-Safe-PKI nun um multiple Komponenten erweitert. Dabei unterscheiden sich grundsätzlich die auf Version v11 geänderte Java-Klassen von den ursprünglichen Klassen durch den Zusatz „v11“, z. B. `OCSPv11Response`, `OCSP-v11Servlet` oder `Signaturev11`.

Konfigurationsdateien

`ocspv11server.config` stellt für den Server (`OCSPv11Servlet`) die Verfahren zur Verfügung. Für diese erste Implementierung sind in der Konfigurationsdatei `ocspv11server.config` zwei Provider installiert, so dass das Response durch die zwei voneinander unabhängigen Verfahren MD5withRSA und SHA1withECDSA signiert wird. `ocspv11server.config` gibt beispielsweise an, wo sich die Schlüssel und Zertifikate für den OCSP-Server befinden und welche Signaturverfahren und Provider eingesetzt werden:
Für die Nutzung von MD5withRSA:

```
KEYSTORE_TYPE=JKS
KEYSTORE_PROVIDER=
KEY_FILE=./keystore/ocsp-server.ks
KEY_PWD=12345678
ALIAS=default
SIGNATURE_ALGO=MD5withRSA
SIGNATURE_PROVIDER=CDCStandard
```

Für die Nutzung von SHA1withECDSA:

```
KEYSTORE_TYPE_2=JKS
KEYSTORE_PROVIDER_2=
KEY_FILE_2=./keystore/ocsp-serverEC.ks
KEY_PWD_2=12345678
ALIAS_2=default
SIGNATURE_ALGO_2=SHA1withECDSA
SIGNATURE_PROVIDER_2=CDCEC
```

Für die Nutzung von cdc-standard- und cdc-ec-Provider:

```
INSTALL_PROVIDERS=cdc.standard.CDCStandardProvider,cdc.ec.CDCECProvider
```

Der OCSPv11-Server greift für die Zertifikats-Status-Antwort auf folgende Zertifikate und Sperrlisten zu:

```
ISSUER_1_CA=./CAs/LiDIACA.crt
ISSUER_1_CRL=./CAs/LiDIACArevoc.crl
ISSUER_2_CA=./CAs/TestCA.crt
ISSUER_2_CRL=./CAs/TestCArevoc.crl
```

ASN.1-Strukturen von OCSPv11 in Java

Die Datenstrukturen des Online Certificate Status Protcols werden üblicherweise in Abstract Syntax Notation One beschrieben. Eine exakte ASN.1-Notation von OCSP für Version v1 und v11 findet sich in Abschnitt A.4 auf Seite 197. Die ASN.1-Strukturen werden über entsprechende Java-Klassen realisiert: `Accuracy`, `BasicOCSPv11Response`, `CertID`, `CertStatus`, `CRLReason`, `KeyHash`, `OCSPv11Response`, `OCSPResponseStatus`, `OCSPRequest`, `ResponderID`, `ResponseBytesv11`, `ResponseData`, `Request`, `RevokedInfo`, `Signaturev11`, `SingleResponse`, `TBSRequest`, `Version`.

Für das Fail-Safe-Konzept sind die mit „v11“ gekennzeichneten Klassen erstellt worden, die auf `BasicOCSPResponse`, `OCSPResponse`, `ResponseBytes` und `Signature` basieren und so weit wie möglich das Vererbungskonzept nutzen. Zum Beispiel ist die ursprüngliche Syntax von `BasicOCSPResponse`,

```
BasicOCSPResponse ::= SEQUENCE {
   tbsResponseData      ResponseData,
   signatureAlgorithm   AlgorithmIdentifier,
   signature            BIT STRING,
   certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL
   }
```

in OCSPv11 in `BasicOCSPv11Response` und `Signaturev11` aufgeteilt worden:

```
BasicOCSPv11Response ::= SEQUENCE {
   tbsResponseData   ResponseData,
   signatures        SEQUENCE OF Signaturev11
   }

Signaturev11 ::= SEQUENCE {
   signatureAlgorithm   AlgorithmIdentifier,
   signature            BITSTRING,
   certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL
   }
```

`BasicOCSPv11Response` lässt somit im Gegensatz zu `BasicOCSPResponse` multiple Signaturen zu, weil `signatures` auf `SEQUENCE OF Signature` verweist, die in `Signaturev11` realisiert ist. In `Signaturv11` wurde zudem eine neue Methode `getSignatures(int number)` implementiert, mit der von außen auf die gewünschte Signatur in der Unterstruktur von `Signaturev11` zugegriffen werden kann.

Desweiteren sind die vier Java-Klassen `BuildOCSPv11Response`, `CheckResponsev11`, `OCSPv11-Client` und `OCSPv11Servlet` entstanden.

Weitere Methoden

Das Zusammensetzen der Antwort mit multiplen digitalen Signaturen wird in `BuildOCSPv11-Response` durchgeführt, das entsprechend um weitere Variablen und Aktionen erweitert wurde; z. B. wird zum Signieren des BuildOCSPv11Responses der Befehl `bocspResp.setSignature` zweimal aufgerufen. `setSignature` beinhaltet auch `bocspResp.addCert(srvCert_)`, das für das Anhängen der Zertifikate zuständig ist.

Für OCSPv11 wurde stets der neu vergebene OID {1.3.6.1.4.1.8301.3.3.2.5} und die neue Versionsnummer v11(10) gesetzt; für weitere Informationen sei auf Appendix B.5 auf Seite 227 verwiesen.

Um geänderte von bestehenden Java-Klassen besser unterscheiden zu können, wurden sämtliche geänderte Klassennamen mit dem Zusatz „v11“ versehen, was allerdings dazu führt, dass die Änderungen auch in den Klassen nachgezogen werden mussten, die ausschließlich eine geänderte Klasse aufrufen, ansonsten aber keine substanzielle Änderung erfahren haben.

main-Methoden

Die beiden Java-Klassen `OCSPv11Servlet` und `OCSPv11Client` enthalten die main-Methoden für OCSPv11-Server und -Client.

Der Server wird über ein Servlet simuliert. Über die Properties `v11Servlet.properties` wird angegeben, welches Servlet das simulierte sein soll – in diesem Fall `OCSPv11Servlet`. Über die Konfigurationsdatei `ocspv11server.config` werden die Verfahren eingestellt.

`OCSPv11Servlet` ist kompatibel zu v1(0) (default) und v11(10). Nach Empfang eines OCSP-Requests interpretiert und prüft `OCSPv11Servlet` die Versionsnummer: Ist sie 0, dann wird über eine Verzweigung wie bisher fortgefahren. Bei Version v11(10) wird entsprechend verzweigt und mit `BuildOCSPv11Response` ein Response vom neuen Typ `OCSPv11Response` generiert. Variablen und Aktionen sind entsprechend doppelt vorhanden.

`OCSPv11Client` – im Gegensatz zum Server inkompatibel zu Version v1 – fordert ein OCSP-Request zu Version v11 an. Dazu wird im `buildRequest` die Version v11(10) explizit angegeben. Um neben RSA ECDSA verarbeiten zu können, wurde in der main-Methode `java.security.Security.addProvider(new cdc.ec.CDCECProvider())` hinzugefügt. Nach Absetzen des Requests erwartet der OCSPv11-Client eine Antwort vom Typ `OCSPv11Response` statt vormals `OCSPResponse`. In die Prüfung des Responses werden beide Signaturen einbezogen, weshalb `CheckResponse` auf `CheckResponsev11` geändert wurde.

6.3 Erweitertes Time Stamp Protocol

Bei der Anforderung nach einem Zeitstempel mit dem Time Stamp Protocol (TSP) fordert ein Client bei einem Server ein “TimeStampRequest“ an, in dem der Client den Hashwert des zeitzustempelnden Dokuments überträgt. Der TSP-Server sendet ein “TimeStampResponse“, das den übermittelten Hashwert und eine Zeitangabe enthält und vom TSP-Server digital signiert ist.

Unter Fail-Safe-Gesichtspunkten kann ein solches Response unzureichend sein, denn in einem Schadensfall, welcher die Signatur des TSP-Servers berührt, ist der Zeitstempel ohne Beweiswert. Aus diesem Grund kann in der Fail-Safe-PKI der Zeitstempel zwei verschiedene Hashwerte enthalten und multipel signiert sein. Dazu überträgt der Client im Request zwei Hashwerte und kennzeichnet das Request mit einer Versionsnummer, so dass der Server aus diesen beiden Hashwerten einen multipel signierten Zeitstempel im Response generiert.

Da sich dieser Request durch die Übergabe zweier Hashwerte vom bisherigen TSP-Standard [TSP01] unterscheidet – denn dort ist ausschließlich ein(!) Hashwert zulässig –, ist dieser neue “TimeStampRequest“ nicht zum alten kompatibel. Zur besseren Unterscheidung erhält der neue

Request den neuen Namen “TimeStampRequestv11“. Wie bei OCSPv11 wird für diese Weiterentwicklung im Fail-Safe-Konzept die Versionsnummer v11(10) definiert, um nicht mit Versionsnummern künftiger Entwicklungen zu kollidieren.

Ein TSP-Client zu Version v1(0) – entsprechend Standard [TSP01] – ist zu einem TSP-Client zu Version v11(10) inkompatibel, ebenso die TSP-Server zu den beiden Versionen. Die Kommunikation zwischen TSP-Client und -Server verläuft wie folgt:

1. Version v1(0):
   Der TSPClient setzt ein v1-Request mit Übergabe eines Hashwertes an eine ihm bekannte Adresse ab, unter der der entsprechende TSP-Server die Anfrage verarbeitet und zeitstempelt. Der Client erhält eine v1-konforme Antwort.

2. Version v11(10):
   Der TSP-Client setzt ein v11-Request mit Übergabe einer Sequence von zwei Hashwerten an eine ihm bekannte Adresse ab, die v11-spezifisch ist und unter der der entsprechende TSP-Server die Anfrage verarbeitet und multipel zeitstempelt. Der Client erhält eine v11-konforme Antwort.

Im Rahmen der Dissertation wurden TSP-Client und -Server für Version v11 entwickelt. In diesem Abschnitt wird ein Überblick über die Implementierung wiedergegeben, der nicht die Dokumentation darstellt. Die ausführliche Dokumentation findet sich beim Source-Code.

Realisierung

In der FlexiPKI gibt es bereits einen in Java implementierten TSP-Client (realisiert als `TSP-Client`) und -Server (`TSPServlet`), die den oben beschriebenen Request mit einem Hashwert (`TimeStampReq`) bzw. Response mit einer Signatur (`TimeStampResp`) jeweils absetzen und die Signatur verifizieren. Im Sinne der FlexiPKI-Philosophie sind die Verfahren austauschbar gestaltet; es gibt eine Konfigurationsdatei (`tspserver.config`), in der die unterstützten Verfahren eingestellt werden.

Diese bestehenden Programme werden im Rahmen der Fail-Safe-PKI um multiple Komponenten erweitert. Dabei unterscheiden sich grundsätzlich auf Version v11 geänderte Java-Klassen von den ursprünglichen Klassen durch den Zusatz „v11“, beispielsweise `TimeStampRespv11`. Das führt jedoch dazu, dass die Änderungen auch in den Klassen nachgezogen werden mussten, die ausschließlich eine geänderte Klasse aufrufen, ansonsten aber keine substanzielle Änderung erfahren haben.

Konfigurationsdateien

`tspv11server.config` stellt für den Server (`TSPv11Servlet`) die Verfahren zur Verfügung. Für diese erste Implementierung sind in der Konfigurationsdatei `tspv11server.config` zwei Provider installiert, so dass das Response durch die zwei voneinander unabhängigen Verfahren MD5withRSA und SHA1withECDSA signiert wird. Die Konfigurationsdatei `tspv11server.config` gibt an, wo sich die Schlüssel und Zertifikate für den TSP-Server befinden und welche Signaturverfahren und Provider eingesetzt werden:

Für die Nutzung von MD5withRSA:

```
KEYSTORE_TYPE=JKS
KEYSTORE_PROVIDER=
KEY_FILE=./keystore/tsp-server.ks
KEY_PWD=12345678
ALIAS=default
SIGNATURE_ALGO=MD5withRSA
SIGNATURE_PROVIDER=CDCStandard
```

Für die Nutzung von SHA1withECDSA:

```
KEYSTORE_TYPE_2=JKS
KEYSTORE_PROVIDER_2=
KEY_FILE_2=./keystore/ocsp-serverEC.ks
KEY_PWD_2=12345678
ALIAS_2=default
SIGNATURE_ALGO_2=SHA1withECDSA
SIGNATURE_PROVIDER_2=CDCEC
```

Für die Nutzung von cdc-standard- und cdc-ec-Provider:

```
INSTALL_PROVIDERS=cdc.standard.CDCStandardProvider,cdc.ec.CDCECProvider
```

ASN.1-Strukturen von TSPv11 in Java

Die Datenstrukturen des Time Stamp Protocols werden üblicherweise in Abstract Syntax Notation One beschrieben. Eine exakte ASN.1-Notation von TSP für Version v1 und v11 findet sich in Abschnitt A.5 auf Seite 202. Die ASN.1-Strukturen werden über entsprechende Java-Klassen realisiert: `Accuracy`, `PKIStatusInfo`, `GeneralName`, `MessageImprint`, `TimeStampRespv11`, `TimeStampReqv11`, `TSTInfov11`.

Wesentliche Änderungen für das Fail-Safe-Konzept sind in den mit v11 gekennzeichneten Java-Klassen vorgenommen worden. Zum Beispiel sind in `TimeStampReqv11`,

```
TimeStampReqv11 ::= SEQUENCE {
   version           INTEGER { v11(10) },
   messageImprints   SEQUENCE OF MessageImprint,
   reqPolicy         TSAPolicyId                OPTIONAL,
   nonce             INTEGER                    OPTIONAL,
   certReq           BOOLEAN DEFAULT FALSE,
   extensions        [0] IMPLICIT Extensions    OPTIONAL,
   }
```

und im `TSTInfov11`, das den eigentlichen Zeitstempel in `TimeStampRespv11` enthält,

```
TSTInfov11 ::= SEQUENCE {
   version           INTEGER { v11(10) },
   policy            OBJECT IDENTIFIER,
   messageImprints   SEQUENCE OF MessageImprint,
   serialNumber      INTEGER,
   genTime           GENERALIZED TIME,
   accuracy          ACCURACY                   OPTIONAL,
   ordering          BOOLEAN DEFAULT FALSE,
   nonce             INTEGER                    OPTIONAL,
   tsa               [0] GeneralName            OPTIONAL,
   extensions        [1] IMPLICIT Extensions    OPTIONAL
   }
```

eine neue Versionsnummer und eine Sequence of MessageImprints hinzugekommen.

Über `TimeStampingOIDRegistyv11` wurde stets der neu definierte OID {1.3.6.1.4.1.8301.3.3.2.5} gesetzt – siehe auch Appendix B.5 auf Seite 227.

main-Methoden

Die beiden Java-Klassen `TSPClientv11` und `TSPServletv11` enthalten die main-Methoden für TSPv11-Client und -Server.

Der Server wird über ein Servlet simuliert. Über die Properties `servletv11.properties` wird angegeben, welches Servlet das simulierte sein soll – in diesem Fall `TSPServletv11`. `TSPServletv11` unterstützt ausschließlich Version v11(10) und ist inkompatibel zu v1(0) (default).

`TSPClientv11` – wie der Server inkompatibel zu Version v1 – fordert einen Zeitstempel zu Version v11 an. Dazu wird das zeitzustempelnde Dokument zweimal mit zwei verschiedenen Hashfunktionen gehasht – in dieser Implementierung mit MD5 und SHA1. Zum Signieren mit ECDSA wurde in der main-Methode `java.security.Security.addProvider(new cdc.ec.CDCECProvider())` hinzugefügt. Nach Absetzen des TSP-Requests erwartet der TSPv11-Client eine Antwort vom Typ `TimeStampRespv11` statt `TimeStampResp`. Die Antwort ist über PKCS#7 vom TSPServlet in Version v11 multipel signiert. Die Intention ist, dass der TSPClientv11 nicht zwingend beide Signaturen verifiziert, sondern nur dann, wenn eine ungültig werden sollte. In dieser Implementierung ist aus Testgründen jedoch eine Verifikation eingebaut, die beide Signaturen verifiziert. In einer späteren Implementierung sollte dies vom Benutzer einstellbar sein.

6.4 Zusammenfassung

In diesem Kapitel wurden die im Kontext der Dissertation entstandenen Implementierungen vorgestellt: Das Update Management Protocol, das erweiterte Online Certificate Status Protocol und das erweiterte Time Stamp Protocol. Intention war der Proof-of-Concept, so dass durch diese Arbeiten gezeigt wurde, dass die neuen und erweiterten Datenformate und Protokolle nicht nur theoretisch, sondern auch praktisch funktionieren. Dieser Prozess führte zu einigen Änderungen an den neuen Protokollen – insbesondere am Update Management Protocol durch die Diplomarbeit von Igor Kalenderian [Kale01].

Kapitel 7

Zusammenfassung und Ausblick

Public-Key-Infrastrukturen sind ein wichtiger Baustein, um Sicherheit in der elektronischen Kommunikation zu realisieren. Die moderne Informationsgesellschaft nutzt schon heute viele Anwendungen mit Public-Key-Infrastrukturen. Einige Stichworte sind E-Mail, Datenverschlüsselung, Homebanking, digitale Wahlen, E-Commerce oder E-Government. Es ist anzunehmen, dass PKI-Anwendungen in Zukunft noch zunehmen werden.

Obwohl Public-Key-Infrastrukturen bereits vielfältig eingesetzt werden, gibt es noch häufig unbeachtete Probleme, weil Verfahren der Public-Key-Kryptographie nicht beweisbar sicher sind, so dass das Kompromittieren einer kryptographischen oder einsatzspezifischen Komponente nicht auszuschließen ist, und weil Implementierungsfehler immer wieder vorkommen. Diese Probleme sind nicht nur theoretischer Natur, sondern – wie die Vergangenheit zeigt – durchaus ernst zu nehmen. Aufgrund der weiter zunehmenden Anwendung von Public-Key-Infrastrukturen können die aus Schadensfällen resultierenden Folgen gravierend sein:

- Verlust der Beweisbarkeit digitaler Signaturen, was gerade im Hinblick auf die rechtliche Gleichstellung zur händischen Unterschrift und auf Langzeitsignaturen zu großen Problemen führen kann
- Verlust der Vertraulichkeit verschlüsselt übertragener sensibler Daten
- Verlust von praktikablen und verbindlichen Revokationsmechanismen
- und vor allem: Verlust der Verfügbarkeit von PKI-Anwendungen, wenn Zertifikate aufgrund eines Schadens revoziert wurden und die sichere Kommunikation zu davon betroffenen Zertifikatsinhabern unterbrochen ist

Die vorliegende Dissertation zeigt mit dem Fail-Safe-Konzept für Public-Key-Infrastrukturen eine Lösung dieser Probleme. Das Konzept besteht aus den vier Bausteinen: Integration multipler Kryptographie in eine PKI mit dynamischer Aktualisierbarkeit, Einsatz multipler digitaler Signaturen, Verwendung erweiterter Revokationsmechanismen und Nutzung iterativer Verschlüsselungen. Das Fail-Safe-Konzept unterstützt Interoperabilität zu bestehenden Standards, zum deutschen Signaturgesetz und zu bestehenden PKIs ohne Fail-Safe-Konzept. Wichtig für die Akzeptanz ist, dass der für das Konzept benötigte Mehraufwand nicht zu hoch ist – gerade unter Berücksichtigung des höheren Sicherheitsniveaus. Die Machbarkeit des theoretischen Konzepts wurde mit einem Proof-of-Concept innerhalb der FlexiPKI [FlexiPKI] durch die Implementierungen des Update Management Protocols in [Kale01], des erweiterten Zeitstempelprotokolls und des erweiterten Zertifikats-Status-Protokolls nachgeweisen.

Welche Zukunft hat das Fail-Safe-Konzept? Anwender, die auf eine ständige Verfügbarkeit ihrer PKI und eine zuverlässige sichere Kommunikation Wert legen, werden auf dem Fail-Safe-Konzept bestehen, so dass es in immer mehr PKIs Einzug halten wird. Um das Bewusstsein für die Probleme zu schärfen, wurde das Fail-Safe-Konzept auf mehreren Konferenzen vorgestellt – beispielsweise auf der „KSI 2001 – Kommunikationssicherheit im Zeichen des Internet“ [HaMa01a], der „Gemplus Developers Conference“ in Paris [HaMa01d], der „VIS 2001 – Verlässliche IT-Systeme“ [HaMa01b], der „IECON'01 - The 27th Annual Conference of the IEEE Industrial Electronics Society“ in Denver/Colorado [HaMa01c] oder dem „12. SIT-SmartCard Workshop“ in Darmstadt [Mase02]. Die Fail-Safe-PKI dieser Dissertation orientiert sich eng an der FlexiPKI, die bereits heute käuflich zu erwerben ist. Für die Erweiterung der FlexiPKI um das Fail-Safe-Konzept muss noch einige Implementierungsarbeit geleistet werden und insbesondere sind Chipkarten um austauschbare kryptographische Komponenten und das Fail-Safe-Konzept zu erweitern. Dadurch wird sich die FlexiPKI einen Wettbewerbsvorteil sichern, so dass andere Anbieter von Public-Key-Infrastrukturen werden nachziehen müssen. Das Fail-Safe-Konzept bietet diese Möglichkeit, weil es auch für PKIs offen ist, die nicht auf Java oder dem Provider-Konzept basieren. Voraussetzung ist allerdings, dass diese PKIs austauschbare Kryptographie unterstützen. Es ist also zu erwarten, dass das Fail-Safe-Konzept für Public-Key-Infrastrukturen in Zukunft einen festen Platz innerhalb der PKI-Technologie einnehmen wird.

Neben dem Fail-Safe-Konzept für Public-Key-Infrastrukturen kann das in dieser Dissertation eingeführte allgemeine Fail-Safe-Konzept mit seinen drei Charakteristika Redundanz, Unabhängigkeit und Nachhaltigkeit auch in anderen Bereichen eingesetzt werden, um die Verfügbarkeit von sicherheitskritischen Systemen im Schadensfall gewährleisten zu können.

Appendix A

Datenformate und Protokolle

Kapitel 1 und 4 beschreiben Public-Key-Infrastrukturen im Allgemeinen und die Fail-Safe-PKI im Besonderen. Die existierenden und neuen Datenformate und Protokolle werden dort abstrakt notiert und in diesem Appendix in der für Standards üblichen Syntax spezifiziert – in Abstract Syntax Notation One (ASN.1) [ASN1]. Darüber hinaus wird in diesem Abschnitt der Vorschlag einer Policy für eine Fail-Safe-PKI angegeben.

A.1 Zertifikate

Zertifikate sind in Definition 8 auf Seite 15 eingeführt worden. Ein de facto Standard für Zertifikate ist X.509 [ISO97c] [CCRL00]. Dort finden sich auch detaillierte Erklärungen. X.509-Zertifikate werden auch in der Fail-Safe-PKI unterstützt.

Zertifikate nach X.509

```
Certificate ::= SEQUENCE {
   tbsCertificate       TBSCertificate,
   signatureAlgorithm   AlgorithmIdentifier,
   signatureValue       BIT STRING
   }

TBSCertificate ::= SEQUENCE {
   version                [0] EXPLICIT Version DEFAULT v1,
   serialNumber           CertificateSerialNumber,
   signature              AlgorithmIdentifier,
   issuer                 Name,
   validity               Validity,
   subject                Name,
   subjectPublicKeyInfo   SubjectPublicKeyInfo,
   issuerUniqueID         [1] IMPLICIT UniqueIdentifier  OPTIONAL,
   subjectUniqueID        [2] IMPLICIT UniqueIdentifier  OPTIONAL,
   extensions             [3] EXPLICIT Extensions        OPTIONAL
   }

Version ::= INTEGER { v1(0), v2(1), v3(2) }
```

```
CertificateSerialNumber ::= INTEGER

Validity ::= SEQUENCE {
   notBefore   Time,
   notAfter    Time
   }

Time ::= CHOICE {
   utcTime       UTCTime,
   generalTime   GeneralizedTime
   }

UniqueIdentifier ::= BIT STRING

SubjectPublicKeyInfo ::= SEQUENCE {
   algorithm          AlgorithmIdentifier,
   subjectPublicKey   BIT STRING
   }

Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension

Extension ::= SEQUENCE {
   extnID      OBJECT IDENTIFIER,
   critical    BOOLEAN DEFAULT FALSE,
   extnValue   OCTET STRING
   }

AlgorithmIdentifier ::= SEQUENCE {
   algorithm    OBJECT IDENTIFIER,
   parameters   ANY DEFINED BY algorithm OPTIONAL
   }

Name ::= CHOICE { RDNSequence }

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::= SET OF AttributeTypeAndValue

AttributeTypeAndValue ::= SEQUENCE {
   type    AttributeType,
   value   AttributeValue
   }

AttributeType ::= OBJECT IDENTIFIER

AttributeValue ::= ANY DEFINED BY AttributeType

DirectoryString ::= CHOICE {
   teletexString     TeletexString (SIZE (1..MAX)),
   printableString   PrintableString (SIZE (1..MAX)),
   universalString   UniversalString (SIZE (1..MAX)),
   utf8String        UTF8String (SIZE (1.. MAX)),
   bmpString         BMPString (SIZE (1..MAX))
   }
```

Kurzerklärung: `certificate` enthält drei Einträge: `tbsCertificate`, `signatureAlgorithm` und `signatureValue`. "tbs" steht für To Be Signed. `tbsCertificate` enthält Versions- und Seriennummer, Algorithmus Identifier des Signaturalgorithmus, mit dem die CA das Zertifikat signiert, Namen des Herausgebers (`issuer`), Güligkeitszeitraum, Namen des Inhabers (`subject`), öffentlichen Schlüssel des CH und zugehörigen Signaturalgorithmus. Die Zertifikatsnummern (`CertificateSerialNumber`) sind eindeutig innerhalb des Issuers, also der CA. Optional sind zusätzliche eindeutige Kennzeichnungen von Herausgeber und Inhaber sowie Erweiterungen. `signatureAlgorithm` kennzeichnet den Algorithmus, mit dem die CA das Zertifikat signiert. `signatureValue` ist die Signatur der CA.
Eine wichtige Extension ist zum Beispiel "Key Usage":

```
id-ce-keyUsage OBJECT IDENTIFIER ::= { id-ce 15 }

KeyUsage ::= BIT STRING {
   digitalSignature   (0),
   nonRepudiation     (1),
   keyEncipherment    (2),
   dataEncipherment   (3),
   keyAgreement       (4),
   keyCertSign        (5),
   cRLSign            (6),
   encipherOnly       (7),
   decipherOnly       (8)
   }
```

Kurzerklärung: Die Key Usage `digitalSignature` wird in Authentisierungs-Anwendungen genutzt, `nonRepudiation` in Signatur-Anwendungen, wenn Nicht-Abstreitbarkeit gefordert wird, `keyEncipherment` zum Schlüsseltransport unter Verschlüsselung eines asymmetrischen Verfahrens, `keyAgreement` zum Schlüsselaustausch unter Verwendung von z. B. Diffie-Hellman, `dataEncipherment` zur Verschlüsselung mit einem asymmetrischen Verfahren, `keyCertSign` zum Verfizieren von Signaturen von Zertifikaten und `cRLSign` zum Verifizieren von Signaturen von Sperrlisten. `encipherOnly` und `decipherOnly` sind nur in Zusammenhang mit `keyAgreement` definiert und regeln, ob ein Schlüssel nur zum Ver- oder auch zum Entschlüsseln verwendet werden darf [CCRL00].

Ein Beispiel für ein X.509-Zertifikat ist auf Seite 16 abgebildet.

A.2 Attributzertifikate

Attributzertifikate binden ein bestimmtes Attribut an den öffentlichen Schlüssel eines Public-Key-Zertifikats. Der Grundaufbau eines Attributzertifikats ist in [ISO97c] spezifiziert. Es gibt einige standardisierte Attribute, es können aber auch proprietäre Attribute genutzt werden. Für die Fail-Safe-PKI gibt es ein neu definiertes Attribut.

Attributzertifikat nach X.509

```
AttributeCertificate ::= SEQUENCE {
   tbsAttributeCertificate   TBSAttributeCertificate,
   signatureAlgorithm        SignatureAlgorithm,
   signature                 BIT STRING
   }
```

```
TBSAttributeCertificate ::= SEQUENCE {
   version                 Version DEFAULT v1,
   subject                 CHOICE {
      baseCertificateID   [0] IssuerSerial, -- associated with a Public Key Certificate
      subjectName         [1] GeneralNames, -- associated with a name
      }
   issuer                  GeneralNames,            -- CA issuing the attribute cert.
   signature               AlgorithmIdentifier,
   serialNumber            CertificateSerialNumber,
   attCertValidityPeriod   AttCertValidityPeriod,
   attributes              SEQUENCE OF Attribute,
   issuerUniqueID          UniqueIdentifier         OPTIONAL,
   extensions              Extensions               OPTIONAL
   }

IssuerSerial ::= SEQUENCE {
   issuer      GeneralNames,
   serial      CertificateSerialNumber,
   issuerUID   UniqueIdentifier         OPTIONAL
   }

AttCertValidityPeriod ::= SEQUENCE {
   notBeforeTime   GeneralizedTime,
   notAfterTime    GeneralizedTime
   }

Attribute ::= SEQUENCE {
   type     AttributeType,
   values   SET OF AttributeValue
   }

AttributeType ::= ATTRIBUTE.&id

AttributeValue ::= ATTRIBUTE.&Type

ATTRIBUTE ::= CLASS {
   &id OBJECT IDENTIFIER UNIQUE,
   &Type
   }
WITH SYNTAX {
   SYNTAX &Type
   IDENTIFIED BY &id
   }
```

Kurzerklärung: Der Aufbau von `AttributeCertificate` ist analog zu `Certificate` gestaltet. `tbsAttributeCertificate` enthält die zu signierenden Informationen, die im Wesentlichen aus `subject` und `attributes` bestehen, weil hier der Bezug zum Public-Key-Zertifikat über `baseCertificateID` und die vergebenden Attribute vermerkt sind.

Attribute

In `Attribute` können beliebig viele Attribute angegeben werden. Es können standardisierte Attribute, z. B. aus [ISO97c], oder proprietäre Attribute verwendet werden. Aus [BSI-SigI99a]

werden zwei Beispiele aufgeführt; sie betreffen die Vertretungsmacht (`procuration`) und monetäre Beschränkungen (`monetaryLimit`):

```
id-sigi-at-procuration OBJECT IDENTIFIER ::= { 1 3 36 8 3 2 }

atProcuration ATTRIBUTE ::= {
WITH SYNTAX ProcurationSyntax
   SINGLE VALUE
   ID id-sigi-at-procuration
   }

ProcurationSyntax ::= SEQUENCE {
   country              PrintableString(SIZE(2)) OPTIONAL,
   typeOfSubstitution   DirectoryString          OPTIONAL,
   signingFor           SigningFor
   }

DirectoryString ::= CHOICE {
   printableString   PrintableString (SIZE (1..maxSize)),
   teletexString     TeletexString (SIZE (1..maxSize)),
   bmpString         BMPString (SIZE (1..maxSize)),
   universalString   UniversalString (SIZE (1..maxSize))
   }

SigningFor ::= CHOICE {
   thirdPerson   GeneralName,
   certRef       IssuerAndSerial
   }


id-sigi-at-monetaryLimit OBJECT IDENTIFIER ::= {1 3 36 8 3 4}

atMonetaryLimit ATTRIBUTE ::= {
WITH SYNTAX MonetaryLimitSyntax
   SINGLE VALUE
   ID id-sigi-at-monetaryLimit
   }

MonetaryLimitSyntax ::= SEQUENCE {
   currency   PrintableString (SIZE(3)),
   amount     INTEGER,
   exponent   INTEGER
   }
```

Attributzertifikat in Fail-Safe-PKI

Das neue Update Management Protocol wird mittels UpdateComponent initiiert, welches der Update Service an die Certificate Holder schickt. Es ist entweder von der CA signiert oder vom Update Service, welches in diesem Fall mit einem besonderen Attributzertifikat von der CA autorisiert ist. Ein Vorschlag für dieses Attributzertifikat ist das Folgende:

```
id-ump-umpAuthorization OBJECT IDENTIFIER ::= { cdc-ump 3 }
```

```
umpAuthorization ATTRIBUTE ::= {
WITH SYNTAX UMPAuthorizationSyntax
   SINGLE VALUE
   ID id-ump-umpAuthorization
   }

UMPAuthorizationSyntax ::= SEQUENCE {
   authorization BOOLEAN
   }
```

Kurzerklärung: Dieses Attributzertifikat ist an das Public-Key-Zertifikat gebunden, das die CA einem Update Service ausgestellt hat, und enthält das Attribut `umpAuthorization`, was besagt, dass der Inhaber dieses Zertifikats das Recht hat, UpdateComponents zu signieren. In `UMPAuthorizationSyntax` muss dazu das Flag `authorization` auf TRUE gesetzt sein. Der OID für dieses Attribut leitet sich aus dem OID ab, der vom Fachgebiet „Kryptographie, Computeralgebra (cdc)“ der TU Darmstadt für das Update Management Protocol (UMP) vergeben wurde:

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

Damit ist UMP und dieses Attribut weltweit eindeutig zu identifizieren.

Da Chipkarten derzeit keine X.509-Zertifikate interpretieren können, muss dieses Attribut zusätzlich in ein für Chipkarten lesbares Card Verifiable (CV) Certificate umgesetzt werden.

A.3 Sperrlisten

Ein de facto Standard für Sperrlisten ist die von PKIX propagierte Certificate Revocation List (CRL) [CCRL00]. Für die Fail-Safe-PKI sind Erweiterungen nötig.

Certificate Revocation List nach PKIX

```
CertificateList ::= SEQUENCE {
   tbsCertList          TBSCertList,
   signatureAlgorithm   AlgorithmIdentifier,
   signatureValue       BIT STRING
   }

TBSCertList ::= SEQUENCE {
   version               Version                  OPTIONAL,
   signature             AlgorithmIdentifier,
   issuer                Name,
   thisUpdate            Time,
   nextUpdate            Time                     OPTIONAL,
   revokedCertificates   SEQUENCE OF SEQUENCE {
      userCertificate      CertificateSerialNumber,
      revocationDate       Time,
      crlEntryExtensions   Extensions               OPTIONAL
      } OPTIONAL,
   crlExtensions         [0] EXPLICIT Extensions  OPTIONAL
   }
```

```
Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension

Extension ::= SEQUENCE {
   extnID      OBJECT IDENTIFIER,
   critical    BOOLEAN DEFAULT FALSE,
   extnValue   OCTET STRING
   }

id-ce-cRLReason OBJECT IDENTIFIER ::= { id-ce 21 }

reasonCode ::= { CRLReason }

CRLReason ::= ENUMERATED {
   unspecified            (0),
   keyCompromise          (1),
   cACompromise           (2),
   affiliationChanged     (3),
   superseded             (4),
   cessationOfOperation   (5),
   certificateHold        (6),
   removeFromCRL          (8)
   }
```

Kurzerläuterung: Der Aufbau von `CertificateList` ähnelt der von Zertifikaten, weil in `tbsCertList` die zu signierende Sperrliste enthalten ist. `TBSCertList` enthält neben den schon von X.509-Zertifikaten bekannten Feldern `version`, `signature` und `issuer` den Zeitpunkt, zu dem diese Sperrliste erzeugt wurde (`thisUpdate`) und wann die nächste Sperrliste veröffentlicht wird (`nextUpdate`). Unter `revokedCertificates` findet sich eine Liste von Zertifikatsseriennummern (`userCertificate`) zusammen mit dem Sperrzeitpunkt (`revocationDate`). Darüber hinaus kann in den Extensions z. B. der Sperrgrund (`CRLReason`) vermerkt sein.

Certificate Revocation List für Fail-Safe-PKI

Sperrlisten in der Fail-Safe-PKI können drei Erweiterungen aufweisen:

- Signieren von CRL mit einer multiplen digitalen Signatur
- Sperrung ganzer Bereiche von Zertifikatsnummern, entsprechend Kapitel 3.4 auf Seite 76
- UMP-Transport für Authentisierungs-Anwendungen, d. h. Integration mehrerer Update-Components in einer CRL

Diese neue CRL unterscheidet sich von der Standard-CRL durch einen neu vergebenen Object Identifier (OID).

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

CertificateListMultipleSigned ::= SEQUENCE {
  tbsCertList   TBSCertListMultipleSigned,
  signatures    SEQUENCE OF Signatures     -- multiple digitale Signaturen
  }
```

```
TBSCertListMultipleSigned ::= SEQUENCE {
   version               Version {v1}                      OPTIONAL,
   signature             SEQUENCE OF AlgorithmIdentifier, -- mehrere Signaturalgorithm.
   issuer                Name,
   thisUpdate            Time,
   nextUpdate            Time                              OPTIONAL,
   revokedCertificates   SEQUENCE OF SEQUENCE {
      userCertificate      EnhancedCertSerialNumber, -- Seriennummer oder Sperrbereich
      revocationDate       Time,
      crlEntryExtensions   Extensions               OPTIONAL
      } OPTIONAL,
   crlExtensions         [0] Extensions                    OPTIONAL
   }

EnhancedCertSerialNumber ::= INTEGER -- Vergabe der Nummern entsprechend Abschnitt 3.4

Signatures ::= SEQUENCE {
   signatureAlgorithm   AlgorithmIdentifier,
   signature            BIT STRING
   }

id-ump-umpTransport OBJECT IDENTIFIER ::= { cdc-ump 4 } -- neue Extension

UMPTransport ::= SEQUENCE OF UpdateComponent -- enthält UpdateComponents
```

Kurzerläuterung: Der Grundaufbau (`CertificateListMultipleSigned`) unterscheidet sich insofern vom Standard, als dass mehrere Signaturen in `signatures` möglich sind. `Signatures` selbst weist wie beim Standard die Felder `signatureAlgorithm` und `signature` auf. In der zu signierenden Sperrliste `TBSCertListMultipleSigned` sind mehrere Signaturalgorithmen statt nur einem anzugeben. In der Liste der gesperrten Zertifikate (`revokedCertificates`) ist der Typ von `userCertificate` nun statt `CertificateSerialNumber` der neue Typ `EnhancedCertSerialNumber`, der darauf hinweist, dass auch solche Nummern zulässig sind, die einen ganzen Bereich von Nummern sperren – entsprechend der in Abschnitt 3.4 auf Seite 76 beschriebenen Konventionen. Der Transport mehrerer UpdateComponents erfolgt über eine neue Extension mit OID `cdc-ump 4`, welche in `UMPTransport` mehrere `UpdateComponent` enthält.

A.4 Zertifikats-Status-Anfragen

Ein de facto Standard für Zertifikats-Status-Anfragen ist das von der PKIX veröffentlichte Online Certificate Status Protocol (OCSP) [OCSP99]. OCSP besteht aus einem Request und einem Response. Für die Fail-Safe-PKI sind Erweiterungen nötig.

Online Certificate Status Protocol nach PKIX

Request Syntax:

```
OCSPRequest ::= SEQUENCE {
   tbsRequest          TBSRequest,
   optionalSignature   [0] EXPLICIT Signature OPTIONAL
   }
```

```
TBSRequest ::= SEQUENCE {
   version             [0] EXPLICIT Version DEFAULT v1,
   requestorName       [1] EXPLICIT GeneralName         OPTIONAL,
   requestList         SEQUENCE OF Request,
   requestExtensions   [2] EXPLICIT Extensions          OPTIONAL
   }

Signature ::= SEQUENCE {
   signatureAlgorithm   AlgorithmIdentifier,
   signature            BIT STRING,
   certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL
   }

Version ::= INTEGER { v1(0) }

Request ::= SEQUENCE {
   reqCert                    CertID,
   singleRequestExtensions    [0] EXPLICIT Extensions OPTIONAL
   }

CertID ::= SEQUENCE {
   hashAlgorithm    AlgorithmIdentifier,
   issuerNameHash   OCTET STRING,           -- Hash of Issuer's DN
   issuerKeyHash    OCTET STRING,           -- Hash of Issuers public key
   serialNumber     CertificateSerialNumber
   }
```

Kurzerläuterung: Der `OCSPRequest` besteht aus der Anfrage (`tbsRequest`), die optional signiert (`optionalSignature`) sein kann. In `TBSRequest` ist in `requestList` die Liste von Zertifikatsseriennummern (`serialNumber` in `CertID`) enthalten, deren Status angefragt wird.

Response Syntax:

```
OCSPResponse ::= SEQUENCE {
   responseStatus   OCSPResponseStatus,
   responseBytes    [0] EXPLICIT ResponseBytes OPTIONAL
   }

OCSPResponseStatus ::= ENUMERATED {
   successful         (0), -- Response has valid confirmations
   malformedRequest   (1), -- Illegal confirmation request
   internalError      (2), -- Internal error in issuer
   tryLater           (3), -- Try again later
                           -- (4) is not used
   sigRequired        (5), -- Must sign the request
   unauthorized       (6)  -- Request unauthorized
   }

ResponseBytes ::= SEQUENCE {
   responseType   OBJECT IDENTIFIER,
   response       OCTET STRING
   }

id-pkix-ocsp         OBJECT IDENTIFIER ::= { id-ad-ocsp }
```

```
id-pkix-ocsp-basic   OBJECT IDENTIFIER ::= { id-pkix-ocsp 1 }

BasicOCSPResponse ::= SEQUENCE {
   tbsResponseData      ResponseData,
   signatureAlgorithm   AlgorithmIdentifier,
   signature            BIT STRING,
   certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL
   }

ResponseData ::= SEQUENCE {
   version              [0] EXPLICIT Version DEFAULT v1,
   responderID          ResponderID,
   producedAt           GeneralizedTime,
   responses            SEQUENCE OF SingleResponse,
   responseExtensions   [1] EXPLICIT Extensions          OPTIONAL
   }

ResponderID ::= CHOICE {
   byName   [1] Name,
   byKey    [2] KeyHash
   }

KeyHash ::= OCTET STRING

SingleResponse ::= SEQUENCE {
   certID             CertID,
   certStatus         CertStatus,
   thisUpdate         GeneralizedTime,
   nextUpdate         [0] EXPLICIT GeneralizedTime OPTIONAL,
   singleExtensions   [1] EXPLICIT Extensions      OPTIONAL
   }

CertStatus ::= CHOICE {
   good      [0] IMPLICIT NULL,
   revoked   [1] IMPLICIT RevokedInfo,
   unknown   [2] IMPLICIT UnknownInfo
   }

RevokedInfo ::= SEQUENCE {
   revocationTime     GeneralizedTime,
   revocationReason   [0] EXPLICIT CRLReason OPTIONAL
   }

UnknownInfo ::= NULL -- this can be replaced with an enumeration
```

Kurzerläuterung: Die Antwort `OCSPResponse` besteht aus Status der Antwort und der Antwort selbst, sofern der Status “successful“ gesetzt ist. Die Antwort (`BasicOCSPResponse`) besteht aus der signierten Antwort `tbsResponseData`, in der neben `version`, `responderID` und Zeit (`producedAt`) eine Liste von Einzelantworten (`SingleResponse`) aufgeführt ist. Jedes `SingleResponse` enthält zu einer Zertifikatsseriennummer (`certID`) den Status `CertStatus` zum angegebenen Zeitpunkt `thisUpdate`. Der Status kann auf „nicht gesperrt“ und „bekannt“ (`good`), „revoziert“ und „bekannt“ (`revoked`) oder „unbekannt“ (`unknown`) lauten.

Online Certificate Status Protocol für Fail-Safe-PKI

Für die Fail-Safe-PKI sind folgende Erweiterungen nötig:

- Die Antwort ist multipel digital signiert
- In die Antwort sind UpdateComponents integriert (für Authentisierungs-Anwendungen)

Request:
Die Anforderung nach einer multipel signierten OCSP-Antwort unterscheidet sich von der Anforderung nach einer einfach signierten OCSP-Antwort darin, dass in das Request eine andere Versionsnummer codiert ist: Version v1(0) ist die Versionsnummer, die im OCSP-Standard [OCSP99] vorgegeben ist. Da bereits ein Vorschlag für eine Änderung dieses Standards vorliegt [OCSP01], in der Versionsnummer v2(1) vorgesehen ist, wird für die Erweiterung in der Fail-Safe-PKI die Versionsnummer v11(10) gewählt. Dieser Sprung in der Nummerierung soll ein Vermischen dieser Versionsnummer mit etwaigen Erweiterungen des Standards verhindern. Nach wie vor kann der Request optional signiert werden; es reicht allerdings eine einfache digitale Signatur, weil der OCSP-Server diese Signatur gegen eine Sperrliste validieren kann.

Request Syntax:

```
TBSRequest ::= SEQUENCE {
   version             [0] EXPLICIT Version DEFAULT v11, -- v11 ist neu
   requestorName       [1] EXPLICIT GeneralName          OPTIONAL,
   requestList         SEQUENCE OF Request,
   requestExtensions   [2] EXPLICIT Extensions           OPTIONAL
   }

Version ::= INTEGER { v1(0), v2(1), v11(10) }
```

Kurzerläuterung: Es ist neben den in [OCSP99] bzw. [OCSP01] vorgesehenen Versionsnummern v1(0) und v2(1) in der Fail-Safe-PKI die Nummer v11(10) für OCSPv11 zulässig.

Response:
Die Zertifikats-Status-Antwort ist multipel signiert und enthält die Versionsnummer v11(10) für OCSPv11. In den Extensions kann – wie bei den Sperrlisten – eine Liste von UpdateComponents integriert sein.

Request Syntax:

```
cdc-ocspv11 OBJECT IDENTIFIER ::= { cdc-ump 5 }

BasicOCSPv11Response ::= SEQUENCE {
   tbsResponseData   ResponseData,
   signatures        SEQUENCE OF Signaturev11 -- mehrere Signaturen
   }

Signaturev11 ::= SEQUENCE {
   signatureAlgorithm   AlgorithmIdentifier,
   signature            BIT STRING,
   certs                [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL
   }
```

```
ResponseData ::= SEQUENCE {
   version              [0] EXPLICIT Version v11,   -- neue Versionsnummer
   responderID          ResponderID,
   producedAt           GeneralizedTime,
   responses            SEQUENCE OF SingleResponse,
   responseExtensions   [1] EXPLICIT Extensions     OPTIONAL
   }

id-ump-umpTransport OBJECT IDENTIFIER ::= { cdc-ump 4 } -- neue Extension

UMPTransport ::= SEQUENCE OF UpdateComponent -- enthält UpdateComponents
```

Kurzerläuterung: Im `BasicOCSPv11Response` sind mehrere Signaturen einzubinden; die Einzelsignaturen sind in `Signaturev11` ausgelagert. `ResponseData` unterscheidet sich lediglich in der Versionsnummer.

Ein OCSP-Server in der Fail-Safe-PKI ist interoperabel, so dass er sowohl die standardkonformen Versionen v1 und v2 als auch die neue v11-Version unterstützt. Je nach Request formt der Server ein entsprechendes Response. Der OCSP-Client, der den Request absendet, erwartet ein entsprechendes Response.

Beispiel zu OCSPv11

Im OCSP Request wird der Request zu zwei Zertifikaten angefordert. Die Versionsnummer ist v11(10), so dass der OCSP-Server mit einer multipel signierten Antwort im OCSP Response antwortet.

```
OCSP Request {
TBS Request {
   Version: Integer 10
   RequestorName:[CONTEXT SPECIFIC 1] EXPLICIT GeneralName {
      CN=ocsp-client, O=TU Darmstadt, OU=CDC, C=DE
   }
   SequenceOf cdc.ocsp.asn1.Request {
      Request {
         CertID {
            X.509 AlgorithmIdentifier 1.3.14.3.2.26 (SHA)
            Octet String.c6.ae.37.d3.72.5.9c.46.48.5d.4a.bb.d8.27.a.28.4c.22.75.a3
            Octet String.e1.51.46.3f.b1.1f.11.a3.55.d2.84.b0.69.3d.b8.86.cb.6.a4.8d
            Integer 958
         }
         <no singleReqExt>
      }
      Request {
         CertID {
            X.509 AlgorithmIdentifier 1.3.14.3.2.26 (SHA)
            Octet String.c6.ae.37.d3.72.5.9c.46.48.5d.4a.bb.d8.27.a.28.4c.22.75.a3
            Octet String.e1.51.46.3f.b1.1f.11.a3.55.d2.84.b0.69.3d.b8.86.cb.6.a4.8d
            Integer 957
         }
         <no singleReqExt>
      }
   }
```

```
   <no requestextensions>
}
OCSP Signature {
   Algorithm: MD5withRSA
   Signature: BitString {0111001101000000010101011001011100011011110110110101110011
      1100100101111000110100110010101111001001001000001011110111010001000111000011111
      0011011111101001111000110011101001111111000110100101011011110011111110010111010
      1011001100001111011101011110010000001101100100111110000011001100100111100100001
      0
      1100011011000011000000100000001011101111101000010111100100100101011110000111000
      0
      1010000101011101101101010000010111100001011000010011100001100010000000000011100
      1
      1100111100011100000000101011011100101100110011011111111010000100101010101011101
      1
      1110101010010010000111100010100001001011010110011010111000100011001001011101101
      0
      0101011100000111100100010001100010000010011111111011000011100001011010100011000
      1
      0001100010011110101011001011011100100010100001011011011110101101110000011010101
      0
      0111110010101100000000110010001101111010111010001010100011010010111111001101000
      1
      0100000111000001111011001100010000110011011010100010001100010010101000111110111
      1
      0111001000111101110111000011001011101010001011111111101010111011000101010101101
      1
      1010100}
      Certificate for CN=ocsp-client, O=TU Darmstadt, OU=CDC, C=DE
   }
}

Response {
Status: Response-Status:successful (0)
[CONTEXT SPECIFIC 0] EXPLICIT ResponseBytesv11 {
   Type: 1.3.6.1.4.1.8301.3.3.2.5.
   BasicOCSPv11Response {
   ResponseData {
   [CONTEXT SPECIFIC 0] EXPLICIT Integer 10
   (CHOICE) [CONTEXT SPECIFIC 1] EXPLICIT CN=OCSP Responder, O=TU Darmstadt, OU=CDC,
      C=DE, EMAILADDRESS=ca-admin@cdc.informatik.tu-darmstadt.de
   GeneralizedTime "20020318132750Z"
   SequenceOf cdc.ocsp.asn1.SingleResponse {
      SingleResponse {
      CertID {
         X.509 AlgorithmIdentifier 1.3.14.3.2.26 (SHA)
         Octet String.c6.ae.37.d3.72.5.9c.46.48.5d.4a.bb.d8.27.a.28.4c.22.75.a3
         Octet String.e1.51.46.3f.b1.1f.11.a3.55.d2.84.b0.69.3d.b8.86.cb.6.a4.8d
         Integer 958
      }
      Status: GOOD
      This update: Wed Jul 19 18:01:17 GMT+02:00 2000
      Next update: Sat Aug 19 18:01:17 GMT+02:00 2000
      }
      SingleResponse {
      CertID {
         X.509 AlgorithmIdentifier 1.3.14.3.2.26 (SHA)
         Octet String.c6.ae.37.d3.72.5.9c.46.48.5d.4a.bb.d8.27.a.28.4c.22.75.a3
         Octet String.e1.51.46.3f.b1.1f.11.a3.55.d2.84.b0.69.3d.b8.86.cb.6.a4.8d
         Integer 957
      }
      Status: GOOD
      This update: Wed Jul 19 18:01:17 GMT+02:00 2000
      Next update: Sat Aug 19 18:01:17 GMT+02:00 2000
```

```
      }
      }
      [CONTEXT SPECIFIC 1] EXPLICIT SequenceOf codec.x509.X509Extension {
      }
   }
   SequenceOf cdc.ocsp.asn1.Signaturev11 {
   OCSP Signature {
      Algorithm: MD5withRSA
      Signature: BitString {00111110010111011111001110101111001110001001000010010110
      0
      1101101001000101010111010001011110101110111100011000001000000110001101100111011
      0
      1010000011011010110110100111011100010011011001100010011001101100110111111011
      0
      1111111000011110100010000010111100101100100011110100110010000000001110000011001
      0
      1010101111011010101111011111100110111010110111010001011011010110000111010000110
      0
      1111000111110000000111110011101111010010010100110000110010101001111100101111111
      0
      0110100100000100110101001100011001101011110011011101111001011100111110100111010
      0
      1101111001101000001010100110001110001001011000011010110110111001010110101100011
      0
      1010111011100000001000000011010110110110001010000101011011001110101001110000110
      0
      0011001101000110100010010010111000011000011100000000101101000100101011000111100
      00
      1001001110100000100111000011011100010101101110100000111000010001001010001011010
      0
      0110100100001101000000110010100001101111101010010010011001010101100010000010011
      0
      0111100110100011101110101010111110100101101110101010110111110100010100100110101
      10
      0101101}
      Certificate for CN=OCSP Responder, O=TU Darmstadt, OU=CDC, C=DE, EMAILADDRESS=
      ca-admin@cdc.informatik.tu-darmstadt.de
      }
   OCSP Signature {
      Algorithm: SHA1withECDSA
      Signature: BitString {00110000001101000000001000011000010111100010101101001111
      1
      0110111101000111111000010000011110101100110110110100000110001101111001100110110
      1
      1010100111100101101110101000111101001000000001110100011110110000111111010111000
      0
      0111001000000100001100000011111101111011011010111001111000111010010110001111
      10
      1101001011000001110100111011111100100010101110001111010011000111100000100101101
      0
      1100101010010100101110011110001110101100001100100001111}
      Certificate for CN=OCSP Responder, O=TU Darmstadt, OU=CDC, C=DE, EMAILADDRESS=
      ca-admin@cdc.informatik.tu-darmstadt.de
      }
   }
   }
   }
}
```

An diesem Beispiel lässt sich auch der Mehraufwand einer ECDSA-Signatur gegenüber einer RSA-Signatur leicht ablesen.

A.5 Zeitstempel

Ein de facto Standard für Zeitstempel ist das von der PKIX herausgegebene Time Stamp Protocol (TSP) [TSP01]. Das Protokoll besteht aus Request und Response. Für die Fail-Safe-PKI sind Änderungen notwendig.

Time Stamp Protocol nach PKIX

Request Syntax:

```
id-kp-timeStamping OBJECT IDENTIFIER ::= {iso(1) identified-organization(3) dod(6)
internet(1) security(5) mechanisms(5) pkix(7) kp(3) timestamping(8)}

TimeStampReq ::= SEQUENCE {
   version          INTEGER { v1(1) },
   messageImprint   MessageImprint,         -- a hash algorithm OID and the hash
                                            -- value of the data to be time stamped
   reqPolicy        TSAPolicyId             OPTIONAL,
   nonce            INTEGER                 OPTIONAL,
   certReq          BOOLEAN DEFAULT FALSE,
   extensions       [0] IMPLICIT Extensions OPTIONAL
   }

MessageImprint ::= SEQUENCE {
   hashAlgorithm   AlgorithmIdentifier,
   hashedMessage   OCTET STRING
   }

TSAPolicyId ::= OBJECT IDENTIFIER
```

Kurzerläuterung: In der Anfrage nach einem Zeitstempel (`TimeStampReq`) wird der Hashwert (in `messageImprint`) übertragen, der zeitgestempelt werden soll.

Response Syntax:

```
TimeStampResp ::= SEQUENCE {
   status           PKIStatusInfo,
   timeStampToken   TimeStampToken OPTIONAL
   }

PKIStatusInfo ::= SEQUENCE {
   status         PKIStatus,
   statusString   PKIFreeText    OPTIONAL,
   failInfo       PKIFailureInfo OPTIONAL
   }

PKIStatus ::= INTEGER {
   granted                 (0), -- Zeitstempel gewährt
   grantedWithMods         (1), -- Zeitstempel gewährt mit Modifikation
   rejection               (2),
   waiting                 (3),
   revocationWarning       (4), -- Warnung: Revokation steht bevor
   revocationNotification  (5)  -- Revokation ist aufgetreten
   }

PKIFailureInfo ::= BIT STRING {
   badAlg                (0), -- unrecognized or unsupported Algo
   badRequest            (2), -- transaction not permitted or supported
   badDataFormat         (5), -- the data submitted has the wrong format
   timeNotAvailable      (14),-- the TSA's time source is not available
   unacceptedPolicy      (15),-- the requested TSA policy is not supported
   unacceptedExtension   (16),-- the requested extension is not supported
   addInfoNotAvailable   (17),-- the additional info. requested could not
                              -- be understood or is not available
```

```
   systemFailure          (25) -- request cannot be handled (system failure)
   }

TimeStampToken ::= ContentInfo
   -- contentType is id-signedData as defined in [CMS99]
   -- content is SignedData as defined in([CMS99])
   -- eContentType within SignedData is id-ct-TSTInfo
   -- eContent within SignedData is TSTInfo

id-ct-TSTInfo OBJECT IDENTIFIER ::= { iso(1) member-body(2)
us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) ct(1) 4}

TSTInfo ::= SEQUENCE {
   version          INTEGER { v1(1) },
   policy           TSAPolicyId,
   messageImprint   MessageImprint,         -- MUST have the same value as the
                                            -- similar field in TimeStampReq
   serialNumber     INTEGER,                -- Time Stamps users MUST be ready to
                                            -- accommodate integers up to 160 bits.
   genTime          GeneralizedTime,
   accuracy         Accuracy                OPTIONAL,
   ordering         BOOLEAN DEFAULT FALSE,
   nonce            INTEGER                 OPTIONAL, -- see TimeStampReq
   tsa              [0] GeneralName         OPTIONAL,
   extensions       [1] IMPLICIT Extensions OPTIONAL
   }

Accuracy ::= SEQUENCE {
   seconds   INTEGER              OPTIONAL,
   millis    [0] INTEGER (1..999) OPTIONAL,
   micros    [1] INTEGER (1..999) OPTIONAL
   }
```

Kurzerläuterung: In der Antwort des Zeitstempel-Servers ist ein Status (`status`) und der Zeitstempel (`timeStampToken`) enthalten. Der Status gibt an, ob der Zeitstempel erzeugt wurde oder, falls nicht, weshalb nicht. `TimeStampToken` besteht aus einem – nach [CMS99] oder [PKCS7] – signierten Datenfeld, welches `TSTInfo` heißt. `TSTInfo` enthält u. a. die Versionsnummer, den in der Anfrage übertragenden Hashwert (`messageImprint`), eine Seriennummer und die Zeit, zu der der Zeitstempel erzeugt wurde.

Time Stamp Protocol für Fail-Safe-PKI

In der Fail-Safe-PKI können Zeitstempel multiple digitale Signaturen aufweisen, welche zwei verschiedenen Hashwerte beinhalten.

Request:
Intention ist, dass aus einem Dokument durch zwei verschiedene Hashfunktionen zwei verschiedene Hashwerte extrahiert werden, die in der Zeitstempel-Anfrage zum Zeitstempel-Server übertragen werden. Der Request enthält darüber hinaus eine neue Versionsnummer – analog zur Argumentation in Abschnitt A.4 auf Seite 197 wird die Versionsnummer v11 gewählt. Der TSP-Server erkennt den Wunsch nach einem multipel signierten Zeitstempel an zwei Merkmalen: An den zwei Hashwerten und an der Versionsnummer v11.
Request Syntax:

```
cdc-tspv11 OBJECT IDENTIFIER ::= { cdc-ump 6 }

TimeStampReqv11 ::= SEQUENCE {
   version          INTEGER { v11(10) },        -- neue Versionsnummer
   messageImprint   SEQUENCE OF MessageImprint, -- mehrere Hashwerte
   reqPolicy        TSAPolicyId                 OPTIONAL,
   nonce            INTEGER                     OPTIONAL,
   certReq          BOOLEAN DEFAULT FALSE,
   extensions       [0] IMPLICIT Extensions     OPTIONAL
   }
```

Kurzerläuterung: In `messageImprint` kann eine Liste von `MessageImprint` enthalten sein.

Response:
Die multiplen digitalen Signaturen werden über PKCS#7 (siehe Abschnitt A.6 auf Seite 206) codiert. Die Informationen des Zeitstempels werden in TSTInfov11 codiert, in welchen eine neue Versionsnummer und eine Liste von Hashwerten integriert ist.

Response Syntax:

```
cdc-TSTInfov11 OBJECT IDENTIFIER ::= { cdc-tspv11 1 }

TSTInfov11 ::= SEQUENCE {
   version           INTEGER { v11(10) },        -- neue Versionsnummer
   policy            OBJECT IDENTIFIER,
   messageImprints   SEQUENCE OF MessageImprint, -- mehrere Hashwerte
   serialNumber      INTEGER,
   genTime           GENERALIZED TIME,
   accuracy          ACCURACY                    OPTIONAL,
   ordering          BOOLEAN DEFAULT FALSE,
   nonce             INTEGER                     OPTIONAL,
   tsa               [0] GeneralName             OPTIONAL,
   extensions        [1] IMPLICIT Extensions     OPTIONAL
   }
```

Kurzerläuterung: `TSTInfov11` unterscheidet sich von `TSTInfo` durch eine neue Versionsnummer und die Möglichkeit, eine Liste von `MessageImprint` zuzulassen.

Intention ist, dass der Zeitstempel-Server prüft, ob die beiden Hashfunktionen, die die beiden übertragenen Hashwerte erzeugten, derzeit geeignet und voneinander unabhängig sind. Andernfalls soll der Zeitstempel-Server einen Zeitstempel ablehnen.

Der TSP-Server ist nicht interoperabel! In der Fail-Safe-PKI muss es zwei verschiedene Server geben, die mit verschiedenen Adressen angesprochen werden: Einen für TSPv1 nach [TSP01] und einen für TSPv11 mit multipel signierten Zeitstempeln. Der Grund ist, dass der TSP-Server zunächst überprüft, ob er das übertragene Request interpretieren kann; d. h. er erwartet genau einen(!) Hashwert entsprechend Standard [TSP01] oder er lässt auch mehrere Hashwerte entsprechend dieser Erweiterungen zu.

Beispiel zu TSPv11

Ein Vertrag, in dem Alice bestätigt, dass sie drei Waschmaschinen kauft, soll mit einem Zeitstempel versehen werden. Der Anforderer des Zeitstempels erzeugt aus der Datei Vertrag.doc mittels

der beiden Hashfunktionen MD5 und SHA-1 die beiden hexadezimal dargestellten Hashwerte “96.cc.28.e9.b6.f2.d2.3.89.18.b5.ea.65.bf.d4.a9“ und “d1.cc.75.f6.42.14.cf.8d.4.bb.5b.b3.83.ef.ff.bf.a2.82.87.65“. Diese beiden Hashwerte werden im TimeStampRequest (`TimeStampReqv11`) an den TSP-Server übermittelt, der einen multipel signierten Zeitstempel (`TimeStampRespv11`) erzeugt.

```
TimeStampReqv11 {
   Integer 10
   SequenceOf cdc.tsp.ietf.MessageImprint {
      < MessageImprint
      Algorithm:MD5
      Message:Octet String.96.cc.28.e9.b6.f2.d2.3.89.18.b5.ea.65.bf.d4.a9
      >
      < MessageImprint
      Algorithm:SHA
      Message:Octet String.d1.cc.75.f6.42.14.cf.8d.4.bb.5b.b3.83.ef.ff.bf.a2.82.87.65
      >
      }
   0.0
   Integer 0
   BOOLEAN true
   [CONTEXT SPECIFIC 0] IMPLICIT SequenceOf codec.x509.X509Extension {
   }
   }

TimeStampRespv11 {
   PKIStatusInfo {
      PKIStatus: GRANTED
   }
   ContentInfo {
      1.2.840.113549.1.7.2
   [CONTEXT SPECIFIC 0] EXPLICIT -- PKCS#7 SignedData --
   SignedData {
      Integer 1
      SetOf codec.x509.AlgorithmIdentifier {
         X.509 AlgorithmIdentifier 1.2.840.113549.2.5 (MD5)
         X.509 AlgorithmIdentifier 1.3.14.3.2.26 (SHA)
         }
      ContentInfo {
         1.3.6.1.4.1.8301.3.3.2.6.1
      [CONTEXT SPECIFIC 0] EXPLICIT encoded TSTInfov11 {PKCS#7 Data { <TimeStampToken
         Version: Integer 10
         Policy OID:0.0
         Message imprints:
            Message imprint hash algorithm:
               X.509 AlgorithmIdentifier 1.2.840.113549.2.5 (MD5)
            Message imprint hash value:
               Octet String.96.cc.28.e9.b6.f2.d2.3.89.18.b5.ea.65.bf.d4.a9
            Message imprint hash algorithm:
               X.509 AlgorithmIdentifier 1.3.14.3.2.26 (SHA)
            Message imprint hash value:
               Octet String.d1.cc.75.f6.42.14.cf.8d.4.bb.5b.b3.83.ef.ff.bf.a2.82.87.65
         Serial Number:1
         Signing Time:GeneralizedTime "20020318200230Z"
```

```
      <no accuracy>
      Ordering: FALSE
      <no nonce>
      }
   }
   [CONTEXT SPECIFIC 0] IMPLICIT Certificates codec.asn1.ASN1Opaque {
      Certificate for CN=Test Timestamping Authority (Single Key), OU=CDC,
      O=TU Darmstadt, C=de
      Certificate for CN=Test Timestamping Authority (Single Key), OU=CDC,
      O=TU Darmstadt, C=de
      }
   [CONTEXT SPECIFIC 1] IMPLICIT SetOf codec.asn1.ASN1Opaque {
   }
   SetOf codec.pkcs7.SignerInfo {
   PKCS#7 SignerInfo {
      Version : Integer 1
      Issuer : CN=TestCA, L=Darmstadt, O=TU Darmstadt, OU=CDC, C=DE
      Serial : Integer 1236
      Algorithm : MD5withRSA
      Auth A : 2 elements
      Unauth A : 0 elements
      Signature : Octet String.1.34.b4.b2.1f.d4.13.15.42.74.54.93.2f.5c.90.77.59.b.
      5f.be.50.cc.6e.7f.ce.47.96.4.5e.70.4c.d9.6c.b8.46.8a.9b.1f.ca.d0.88.d.ec.1a.
      d8.94.f5.a2.6.10.60.85.82.ef.39.2a.5c.70.35.43.ae.a6.d7.45.7a.1e.d6.52.45.fa.
      e2.14.14.5e.bc.e4.a6.c1.e0.53.42.bb.ed.ef.ee.3f.a4.fe.6f.14.cb.5e.4c.9d.d8.
      2d.4c.23.e6.68.10.e1.42.10.9.3c.1.fe.91.b2.9f.3.24.c2.b9.1f.f9.c7.8e.dd.40.
      55.52.98.1f.19.5a.61
      Attributes codec.x501.Attribute {
         Attribute {
         1.2.840.113549.1.9.3
         Set {
         1.3.6.1.4.1.8301.3.3.2.6.1
         }
         }
         Attribute {
         1.2.840.113549.1.9.4
         Set {
         Octet String.bb.71.a5.a6.6b.e8.de.ce.62.62.bf.2f.fd.81.66.2a
         }
         }
   }
   }
   PKCS#7 SignerInfo {
      Version : Integer 1
      Issuer : CN=TestCA, L=Darmstadt, O=TU Darmstadt, OU=CDC, C=DE
      Serial : Integer 1247
      Algorithm : SHA1withECDSA
      Auth A : 2 elements
      Unauth A : 0 elements
      Signature : Octet String.30.34.2.18.15.39.96.2d.90.6b.af.7b.14.d1.8e.3c.f6.af.
      e7.5a.58.58.b7.4.9b.2c.e4.ad.2.18.6e.76.22.e3.7e.dd.7e.6c.1e.12.b1.24.45.eb.ce.
      a3.2e.ce.b1.75.bc.6d.68.ad
      Attributes codec.x501.Attribute {
         Attribute {
```

```
            1.2.840.113549.1.9.3
            Set {
            1.3.6.1.4.1.8301.3.3.2.6.1
            }
            }
            Attribute {
            1.2.840.113549.1.9.4
            Set {
            Octet String.48.c3.1c.d3.a8.fb.58.f5.6.e9.ec.f7.fa.34.9d.71.c7.a0.b8.f5
            }
            }
      }
      }
}
}
}
}
```

A.6 PKCS#7

Ein de facto Standard für Datentransfer ist der von RSA Security Inc. propagierte Public Key Cryptography Standard #7 (PKCS#7) [PKCS7]. Es können u. a. Klartexte (data), signierte Texte (signedData), verschlüsselte Texte (envelopedData) und signierte Kryptogramme (signedAndEnvolopedData) über ASN.1 codiert werden.

Das in PKIX entstandene Cryptographic Message Syntax (CMS) [CMS99] ist leicht abgewandelt in PKCS#7 wiederzufinden. Wegen der weiten Verbreitung von PKCS#7 wird nur PKCS#7 betrachtet.

PKCS#7 gestattet Dokumente multipel zu signieren und iterativ zu verschlüsseln, weshalb für die Fail-Safe-PKI keine Änderungen notwendig sind. Allerdings wird UMP in PKCS#7 integriert.

PKCS#7-Grundaufbau

Der Grundaufbau von PKCS#7 enthält Typ (`contentType`) und Inhalt (`content`), auf den der Typ hinweist.

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


ContentType ::= OBJECT IDENTIFIER
```

Der Object Identifier für PKCS#7 ist {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) }.

PKCS#7-Typen

Klartexte werden über `data` codiert, d. h. { pkcs-7 1 } ist der `contentType` und `Data` der `content`:

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

Data ::= OCTET STRING
```

Signierte Texte werden über `signedData` codiert:

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

SignedData ::= SEQUENCE {
   version            Version,
   digestAlgorithms   DigestAlgorithmIdentifiers,
   contentInfo        ContentInfo,
   certificates       [0] IMPLICIT ExtendedCertificatesAndCertificates OPTIONAL,
   crls               [1] IMPLICIT CertificateRevocationLists          OPTIONAL,
   signerInfos        SignerInfos                 -- SignierInfo is a collection
                                                  -- of per-signer information
   }

SignerInfos ::= SET OF SignerInfo

SignerInfo ::= SEQUENCE {
   version                   Version,
   issuerAndSerialNumber     IssuerAndSerialNumber,
   digestAlgorithm           DigestAlgorithmIdentifier,
   authenticatedAttributes   [0] IMPLICIT Attributes             OPTIONAL,
   digestEncryptionAlgorithm DigestEncryptionAlgorithmIdentifier,
   encryptedDigest           EncryptedDigest,
   unauthenticatedAttributes [1] IMPLICIT Attributes             OPTIONAL
   }

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

EncryptedDigest ::= OCTET STRING
```

Kurzerläuterung: `SignedData` enthält im Wesentlichen den zu signierenden Inhalt (`contentInfo`) und die Signatur (`signerInfos`). Unter `SignedData` können optional Zertifikate oder Sperrlisten mitgeliefert werden. `signerInfos` enthält mehrere Signaturen in einer Folge von `SignerInfo`. Jedes `SignerInfo` enthält im Wesentlichen Angaben zum Signierer (`issuerAndSerialNumber`), zur Hashfunktion (`digestAlgorithm`) und zum Signaturverfahren (`digestEncryptionAlgorithm`) sowie die Signatur selbst (`encryptedDigest`). Die Angabe der Hashfunktion findet sich zweimal wieder – unter `SignedData` und unter `SignerInfo` –, falls aus Effizienzgründen frühzeitig mit Hashen begonnen werden soll.

Das Beispiel einer multiplen digitalen Signatur mit PKCS#7 ist im Beispiel des Zeitstempels nachzulesen; siehe Abschnitt A.5 auf Seite 203.

Verschlüsselte Texte werden über `envelopedData` codiert:

```
envelopedData OBJECT IDENTIFIER ::= { pkcs-7 3 }

EnvelopedData ::= SEQUENCE {
   version                Version,
   recipientInfos         RecipientInfos,
   encryptedContentInfo   EncryptedContentInfo
   }

RecipientInfos ::= SET OF RecipientInfo

EncryptedContentInfo ::= SEQUENCE {
   contentType                  ContentType,
   contentEncryptionAlgorithm   ContentEncryptionAlgorithmIdentifier,
   encryptedContent             [0] IMPLICIT EncryptedContent         OPTIONAL
   }

EncryptedContent ::= OCTET STRING

RecipientInfo ::= SEQUENCE {
   version                   Version,
   issuerAndSerialNumber     IssuerAndSerialNumber,
   keyEncryptionAlgorithm    KeyEncryptionAlgorithmIdentifier,
   encryptedKey              EncryptedKey
   }

EncryptedKey ::= OCTET STRING
```

Kurzerläuterung: `EnvelopedData` enthält Informationen für den Empfänger (`recipientInfos`) und zum Kryptogramm (`EncryptedContentInfo`). Die Verschlüsselung wird mit einem hybriden Verschlüsselungsverfahren durchgeführt: Der Klartext vom Typ (`contentType`) ist mit einem symmetrischen Verfahren (`contentEncryptionAlgorithm`) zum Kryptogramm (`encryptedContent`) verschlüsselt worden. Den zum Entschlüsseln nötigen geheimen Schlüssel (`encryptedKey`) kann der Empfänger (entsprechend `issuerAndSerialNumber`) mit seinem privaten Schlüssel und dem asymmetrischen Verschlüsselungsverfahren (`keyEncryptionAlgorithm`) wieder entschlüsseln.

Signierte und verschlüsselte Texte werden über `signedAndEnvelopedData` codiert:

```
signedAndEnvelopedData OBJECT IDENTIFIER ::= { pkcs-7 4 }

SignedAndEnvelopedData ::= SEQUENCE {
   version                 Version,
   recipientInfos          RecipientInfos,
   digestAlgorithms        DigestAlgorithmIdentifiers,
   encryptedContentInfo    EncryptedContentInfo,
   certificates            [0] IMPLICIT ExtendedCertificatesAndCertificates OPTIONAL,
   crls                    [1] IMPLICIT CertificateRevocationLists          OPTIONAL,
   signerInfos             SignerInfos
   }
```

Kurzerläuterung: `SignedAndEnvelopedData` kombiniert `SignedData` und `EnvelopedData`.

Update Management Protocol in PKCS#7

Das neue Update Management Protocol in der Fail-Safe-PKI wird in PKCS#7 codiert. Dazu gibt es neue Object Identifier, die vom Fachgebiet „Kryptographie, Computeralgebra (cdc)“ der TU Darmstadt für das Update Management Protocol (UMP) vergeben wurden:

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

UpdateComponent OBJECT IDENTIFIER ::= { cdc-ump 1 }

UpdateComponentResponse OBJECT IDENTIFIER ::= { cdc-ump 2 }
```

A.7 S/MIME

Secure/Multipurpose Internet Mail Extensions (S/MIME) unterstützt einen sicheren Daten-Transfer - basierend auf dem Internet MIME-Standard. Folgende kryptographischen Sicherheitsdienste werden bereitgestellt: Authentizität, Integrität und Nicht-Abstreitbarkeit von Nachrichten über digitale Signaturen sowie Vertraulichkeit und Daten-Sicherheit über Verschlüsselungen. Im Body einer S/MIME-Nachricht können mit CMS [CMS99] bzw. PKCS#7 [PKCS7] codierte Daten integriert werden [SMIME]. Es gibt verschiedene Möglichkeiten, S/MIME zu nutzen:

1. S/MIME enthält ausschließlich ein PKCS#7-Format (SignedData)
2. Signierter Inhalt und Signatur werden über S/MIME getrennt codiert. Die Signatur wiederum wird über ein PKCS#7-Format übertragen
3. Verschlüsselungen werden in einem PKCS#7-Format (EnvelopedData) codiert und über S/MIME übertragen
4. Die für die Fail-Safe-PKI neu definierten Datentypen UpdateComponent und UpdateComponentResponse zum Update von kryptographischen und einsatzspezifischen Komponenten werden in PKCS#7 integriert

S/MIME enthält ein PKCS#7-Format (SignedData)

```
From: maseberg@cdc.informatik.tu-darmstadt.de
To: maseberg@darmstadt.gmd.de
Date: Mon, 02 Apr 2001 09:41:09 -0800 (PST)
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: application/pkcs7-mime; smime-type=signed-data;
name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

<<the ASN.1 DER-encoded PKCS#7 message, base64-encoded>>

567GhIGfHfYT6ghyHhHUujpfyF4f8HHGTrfvhJhjH776tbB9HG4VQbnj777n8HHGT9HG4VQpfyF467GhIGfHf
YT6rfvbnj756tbBghyHhHUujhJhjHHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H7n8HHGghyH
h6YT64V0GhIGfHfQbnj75
```

Text über S/MIME und Signatur separat über PKCS#7-Format

```
From: maseberg@cdc.informatik.tu-darmstadt.de
To: maseberg@darmstadt.gmd.de
Date: Mon, 02 Apr 2001 09:41:09 -0800 (PST)
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: multipart/signed;
protocol="application/pkcs7-signature";
micalg=sha1; boundary=boundary42

--boundary42
Content-Type: text/plain

This is a clear-signed message.

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

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT64VQpfyF467GhIGfHfYT6jH77n8HH
GghyHhHUujhJh756tbB9HGTrfvbnjn8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF
47GhIGfHfYT64VQbnj756

--boundary42--
```

Verschlüsselungen in PKCS#7-Format (EnvelopedData) codiert, Übertragen via S/MIME

```
From: maseberg@cdc.informatik.tu-darmstadt.de
To: maseberg@darmstadt.gmd.de
Date: Mon, 02 Apr 2001 09:41:09 -0800 (PST)
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m

rfvbnj756tbBghyHhHUujhJhjH77n8HHGT9HG4VQpfyF467GhIGfHfYT67n8HHGghyHhHUujhJh4VQpfyF467
GhIGfHfYGTrfvbnjT6jH7756tbB9Hf8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF
40GhIGfHfQbnj756YT64V
```

Update Management Protocol in S/MIME

```
From: flexipki-ca@cdc.informatik.tu-darmstadt.de
To: maseberg@cdc.informatik.tu-darmstadt.de
Date: Mon, 04 Apr 2001 09:41:09 -0800 (PST)
Subject: Formatted text mail
MIME-Version: 1.0
Content-Type: application/cdc-ump
Content-Transfer-Encoding: base64
```

```
<<the ASN.1 DER-encoded cdc-ump message, base64-encoded>>

567GhIGfHfYT6ghyHhHUujpfyF4f8HHGTrfvhJhjH776tbB9HG4VQbnj777n8HHGT9HG4VQpfyF467GhIGfHf
YT6rfvbnj756tbBghyHhHUujhJhjHHUujhJh4VQpfyF467GhIGfHfYGTrfvbnjT6jH7756tbB9H7n8HHGghyH
h6YT64VOGhIGfHfQbnj75
```

A.8 TLS/SSL

Transport Layer Security (TLS) ist ein de facto Standard für Authentisierungen zwischen Client und Server. Secure Socket Layer (SSL) ist ein früherer Standard, der in TLS integriert ist.

Im Rahmen der Fail-Safe-PKI kann TLS vom Standard abweichen, wenn das Update Management Protocol über TLS übertragen wird. Eine Anwendung ist die virtuelle Zugangskontrolle (siehe Abschnitt 1.3.2 auf Seite 28) zwischen Rechner S – dem Server – und Rechner C – dem Client – in Phase 3 des Updates (siehe Abschnitt 4.5.4 auf Seite 119). Die folgenden Überlegungen basieren auf dem TLS-Standard [TLS99] und beleuchten, wie ein UpdateComponent mit TLS vom Server zum Client und wie ein UpdateComponentResponse vom Client zum Server transportiert werden kann.

Wie in Abschnitt 1.3.2 auf Seite 27 ausgeführt, ist die Grundidee zur gegenseitigen Authentisierung die Folgende: Der Client schickt dem Server ein “ClientHello“, in das eine Zufallszahl integriert ist. Der Server signiert diese Zufallszahl zusammen mit Parametern zum Schlüsselaustausch und sendet die Signatur zusammen mit einer eigenen Zufallszahl und seinem Zertifikat zum Client zurück, welcher daraufhin prüfen kann, ob der Server authentisch ist, d. h. der behauptete ist. Anschließend sendet der Client sein Zertifikat mit der Zufallszahl des Servers und Informationen zum Schlüsselaustausch signiert an den Server zurück. Die nachfolgende Kommunikation kann mit dem ausgehandelten geheimen Schlüssel und einem symmetrischen Verfahren verschlüsselt werden.

Dieses Prinzip ist in TLS derart realisiert, dass in einem „vollständigen Handshake“ folgendes erfolgen kann:

- Versionsabstimmung (2.0, 3.0, 3.1)
- Abstimmung der kryptographischen Verfahren
- Austausch von Zertifikaten (X.509)
- Schlüsselaustausch zum Festlegen von symmetrischen Schlüsseln
- Authentisierung
- Zuweisung einer Session-ID

In einem „kurzen Handshake“ wird eine frühere Session fortgesetzt, so dass einige Schritte entfallen können.

Die folgende Graphik stellt einen vollständigen Handshake dar, wobei der Stern „*“ optionale Schritte kennzeichnet:

Server		Client
	⟵	`ClientHello`
`ServerHello`		
`Certificate*`		
`ServerKeyExchange*`		
`CertificateRequest*`		
`ServerHelloDone`	⟶	
		`Certificate*`
		`ClientKeyExchange`
		`CertificateVerify*`
		`[ChangeCipherSpec]`
	⟵	`Finished`
`[ChangeCipherSpec]`		
`Finished`	⟶	
Application Data	⟷	Application Data

Im Folgenden wird die Syntax näher erläutert. Für Details sei auf [TLS99] verwiesen.

ClientHello:

```
struct {
   uint32 gmt_unix_time;
   opaque random_bytes[28];
} Random;

opaque SessionID<0..32>;

uint8 CipherSuite[2];

enum { null(0), (255) } CompressionMethod;

struct {
   ProtocolVersion   client_version;
   Random            random;
   SessionID         session_id;
   CipherSuite       cipher_suites<2..2^16-1>;
   CompressionMethod compression_methods<1..2^8-1>;
} ClientHello;
```

Kurzerklärung: `ClientHello` enthält die höchste unterstützte Protokollversion (`ProtocolVersion`), eine Client-Nonce (`Random`) – welche sich aus aktueller Zeit `uint32` und einer 28 Byte langen Zufallszahl `opaque` zusammensetzt –, ggf. einer `SessionID` – um an eine Session anzuschließen – und eine Liste von `CipherSuite`. Eine CipherSuite enthält die unterstützten kryptographischen Verfahren.

ServerHello:

```
struct {
   ProtocolVersion   server_version;
   Random            random;
   SessionID         session_id;
   CipherSuite       cipher_suite;
   CompressionMethod compression_method;
} ServerHello;
```

Kurzerklärung: `ServerHello` enthält die Protokollversion, einen Server-Nonce, ggf. eine Session-ID bei kurzem Handshake und vom Server gewählte CipherSuites.

Weitere Protokollschritte:

```
struct {
   ASN.1Cert certificate_list<1..2^24-1>;
} Certificate;

enum { rsa, diffie_hellman } KeyExchangeAlgorithm;

struct {
   opaque RSA_modulus<1..2^16-1>;
   opaque RSA_exponent<1..2^16-1>;
} ServerRSAParams;

struct {
   opaque DH_p<1..2^16-1>;
   opaque DH_g<1..2^16-1>;
   opaque DH_Ys<1..2^16-1>;
} ServerDHParams;

struct {
   select (KeyExchangeAlgorithm) {
      case diffie_hellman:
         ServerDHParams params;
         Signature      signed_params;
      case rsa:
         ServerRSAParams params;
         Signature       signed_params;
   };
} ServerKeyExchange;

enum { anonymous, rsa, dsa } SignatureAlgorithm;

select (SignatureAlgorithm) {
   case anonymous: struct { };
      case rsa:
         digitally-signed struct {
         opaque md5_hash[16];
         opaque sha_hash[20];
      };
      case dsa:
         digitally-signed struct {
         opaque sha_hash[20];
      };
} Signature;
```

```
enum {
   rsa_sign(1), dss_sign(2), rsa_fixed_dh(3), dss_fixed_dh(4),
   (255)
} ClientCertificateType;

opaque DistinguishedName<1..2^16-1>;

struct {
   ClientCertificateType certificate_types<1..2^8-1>;
   DistinguishedName     certificate_authorities<3..2^16-1>;
} CertificateRequest;

struct { } ServerHelloDone;

struct {
   select (KeyExchangeAlgorithm) {
      case rsa: EncryptedPreMasterSecret;
      case diffie_hellman: DiffieHellmanClientPublicValue;
   } exchange_keys;
} ClientKeyExchange;

struct {
   ProtocolVersion client_version;
   opaque          random[46];
} PreMasterSecret;

struct {
   public-key-encrypted PreMasterSecret pre_master_secret;
} EncryptedPreMasterSecret;

enum { implicit, explicit } PublicValueEncoding;

struct {
   select (PublicValueEncoding) {
      case implicit: struct {};
      case explicit: opaque DH_Yc<1..2^16-1>;
   } dh_public;
} ClientDiffieHellmanPublic;

struct {
   Signature signature;
} CertificateVerify;

struct {
   opaque verify_data[12];
} Finished;
```

TLS in der Fail-Safe-PKI

In der Fail-Safe-PKI soll ein UpdateComponent mit einem ServerHello vom Server zum Client und ein UpdateComponentResponse mit einem ClientHello vom Client zum Server transportiert werden. Ein Vorschlag, wie TLS zu erweitern wäre:

```
struct {
   ProtocolVersion    server_version;
   Random             random;
   SessionID          session_id;
   CipherSuite        cipher_suite;
   CompressionMethod  compression_method;
   TLSUpdateComponent tls_update_component;
} ServerHello;

struct {
   ProtocolVersion            client_version;
   Random                     random;
   SessionID                  session_id;
   CipherSuite                cipher_suites<2..2^16-1>;
   CompressionMethod          compression_methods<1..2^8-1>;
   TLSUpdateComponentResponse tls_update_component_response;
} ClientHello;
```

Kurzerklärung: Zur Realisierung dieser Idee muss eine neue Protokollversion (`ProtocolVersion`) etabliert werden und das Protokoll dahingehend erweitert werden, dass jeweils ein weiterer Eintrag erlaubt ist: `TLSUpdateComponent` beim `ServerHello` und `TLSUpdateComponentResponse` beim `ClientHello`, in die ein UpdateComponent bzw. -Response integriert ist.

A.9 Policies

Policies sind nicht standardisiert. Es gibt in [CP99] einen Vorschlag, was eine Certificate Policy enthalten soll. Die Überschriften der – recht umfangreichen – Liste sind im Folgenden abgedruckt. Für detaillierte Informationen sei auf [CP99] verwiesen. Für die Fail-Safe-PKI sind Erweiterungen der Policy nötig.

Vorschlag zu Policies nach [CP99]

1. INTRODUCTION
1.1 Overview
1.2 Identification
1.3 Community and Applicability
1.3.1 Certification authorities, 1.3.2 Registration authorities, 1.3.3 End entities, 1.3.4 Applicability
1.4 Contact Details
1.4.1 Specification administration organization, 1.4.2 Contact person, 1.4.3 Person determining CPS suitability for the policy
2. GENERAL PROVISIONS
2.1 Obligations
2.1.1 CA obligations, 2.1.2 RA obligations, 2.1.3 Subscriber obligations, 2.1.4 Relying party obligations, 2.1.5 Repository obligations
2.2 Liability
2.2.1 CA liability, 2.2.2 RA liability
2.3 Financial responsibility

2.3.1 Indemnification by relying parties, 2.3.2 Fiduciary relationships, 2.3.3 Administrative processes
2.4 Interpretation and Enforcement
2.4.1 Governing law, 2.4.2 Severability, survival, merger, notice, 2.4.3 Dispute resolution procedures
2.5 Fees
2.5.1 Certificate issuance or renewal fees, 2.5.2 Certificate access fees, 2.5.3 Revocation or status information access fees, 2.5.4 Fees for other services such as policy information, 2.5.5 Refund policy
2.6 Publication and Repository
2.6.1 Publication of CA information, 2.6.2 Frequency of publication, 2.6.3 Access controls, 2.6.4 Repositories
2.7 Compliance audit
2.7.1 Frequency of entity compliance audit, 2.7.2 Identity/qualifications of auditor, 2.7.3 Auditor's relationship to audited party, 2.7.4 Topics covered by audit, 2.7.5 Actions taken as a result of deficiency, 2.7.6 Communication of results
2.8 Confidentiality
2.8.1 Types of information to be kept confidential, 2.8.2 Types of information not considered confidential, 2.8.3 Disclosure of certificate revocation/suspension information, 2.8.4 Release to law enforcement officials, 2.8.5 Release as part of civil discovery, 2.8.6 Disclosure upon owner's request, 2.8.7 Other information release circumstances
2.9 Intellectual Property Rights
3. IDENTIFICATION AND AUTHENTICATION (34)
3.1 Initial Registration
3.1.1 Types of names, 3.1.2 Need for names to be meaningful, 3.1.3 Rules for interpreting various name forms, 3.1.4 Uniqueness of names, 3.1.5 Name claim dispute resolution procedure, 3.1.6 Recognition, authentication and role of trademarks, 3.1.7 Method to prove possession of private key, 3.1.8 Authentication of organization identity, 3.1.9 Authentication of individual identity
3.2 Routine Rekey
3.3 Rekey after Revocation
3.4 Revocation Request
4. OPERATIONAL REQUIREMENTS (34)
4.1 Certificate Application
4.2 Certificate Issuance
4.3 Certificate Acceptance
4.4 Certificate Suspension and Revocation
4.4.1 Circumstances for revocation, 4.4.2 Who can request revocation, 4.4.3 Procedure for revocation request, 4.4.4 Revocation request grace period, 4.4.5 Circumstances for suspension, 4.4.6 Who can request suspension, 4.4.7 Procedure for suspension request, 4.4.8 Limits on suspension period, 4.4.9 CRL issuance frequency (if applicable), 4.4.10 CRL checking requirements, 4.4.11 On-line revocation/status checking availability, 4.4.12 On-line revocation checking requirements, 4.4.13 Other forms of revocation advertisements available, 4.4.14 Checking requirements for other forms of revocation advertisements, 4.4.15 Special requirements re key compromise
4.5 Security Audit Procedures
4.5.1 Types of event recorded, 4.5.2 Frequency of processing log, 4.5.3 Retention period for audit log, 4.5.4 Protection of audit log, 4.5.5 Audit log backup procedures, 4.5.6 Audit collection system (internal vs external), 4.5.7 Notification to event-causing subject, 4.5.8 Vulnerability assessments

4.6 Records Archival
4.6.1 Types of event recorded, 4.6.2 Retention period for archive, 4.6.3 Protection of archive, 4.6.4 Archive backup procedures, 4.6.5 Requirements for time-stamping of records, 4.6.6 Archive collection system (internal or external), 4.6.7 Procedures to obtain and verify archive information
4.7 Key changeover
4.8 Compromise and Disaster Recovery
4.8.1 Computing resources, software, and/or data are corrupted, 4.8.2 Entity public key is revoked, 4.8.3 Entity key is compromised, 4.8.4 Secure facility after a natural or other type of disaster
4.9 CA Termination
5. PHYSICAL, PROCEDURAL, AND PERSONNEL SECURITY CONTROLS (34)
5.1 Physical Controls
5.1.1 Site location and construction, 5.1.2 Physical access, 5.1.3 Power and air conditioning, 5.1.4 Water exposures, 5.1.5 Fire prevention and protection, 5.1.6 Media storage, 5.1.7 Waste disposal, 5.1.8 Off-site backup
5.2 Procedural Controls
5.2.1 Trusted roles, 5.2.2 Number of persons required per task, 5.2.3 Identification and authentication for each role
5.3 Personnel Controls
5.3.1 Background, qualifications, experience, and clearance requirements, 5.3.2 Background check procedures, 5.3.3 Training requirements, 5.3.4 Retraining frequency and requirements, 5.3.5 Job rotation frequency and sequence, 5.3.6 Sanctions for unauthorized actions, 5.3.7 Contracting personnel requirements, 5.3.8 Documentation supplied to personnel
6. TECHNICAL SECURITY CONTROLS (34)
6.1 Key Pair Generation and Installation
6.1.1 Key pair generation, 6.1.2 Private key delivery to entity, 6.1.3 Public key delivery to certificate issuer, 6.1.4 CA public key delivery to users, 6.1.5 Key sizes, 6.1.6 Public key parameters generation, 6.1.7 Parameter quality checking, 6.1.8 Hardware/software key generation, 6.1.9 Key usage purposes (as per X.509 v3 key usage field)
6.2 Private Key Protection
6.2.1 Standards for cryptographic module, 6.2.2 Private key (n out of m) multi-person control, 6.2.3 Private key escrow, 6.2.4 Private key backup, 6.2.5 Private key archival, 6.2.6 Private key entry into cryptographic module, 6.2.7 Method of activating private key, 6.2.8 Method of deactivating private key, 6.2.9 Method of destroying private key
6.3 Other Aspects of Key Pair Management
6.3.1 Public key archival, 6.3.2 Usage periods for the public and private keys
6.4 Activation Data
6.4.1 Activation data generation and installation, 6.4.2 Activation data protection, 6.4.3 Other aspects of activation data
6.5 Computer Security Controls
6.5.1 Specific computer security technical requirements, 6.5.2 Computer security rating
6.6 Life Cycle Technical Controls
6.6.1 System development controls, 6.6.2 Security management controls, 6.6.3 Life cycle security ratings
6.7 Network Security Controls
6.8 Cryptographic Module Engineering Controls
7. CERTIFICATE AND CRL PROFILES

7.1 Certificate Profile
7.1.1 Version number(s), 7.1.2 Certificate extensions, 7.1.3 Algorithm object identifiers, 7.1.4 Name forms, 7.1.5 Name constraints, 7.1.6 Certificate policy Object Identifier, 7.1.7 Usage of Policy Constraints extension, 7.1.8 Policy qualifiers syntax and semantics, 7.1.9 Processing semantics for the critical certificate policy extension
7.2 CRL Profile
7.2.1 Version number(s), 7.2.2 CRL and CRL entry extensions
8. SPECIFICATION ADMINISTRATION
8.1 Specification change procedures
8.2 Publication and notification policies
8.3 CPS approval procedures

Erweiterte Policy

Die Besonderheiten einer Fail-Safe-PKI sollten sich in der Policy wiederfinden. Unter einem eigenen Object Identifier

```
cdc-cp OBJECT IDENTIFIER ::={iso(1) identified-organization(3) dod(6)
internet(1) private(4) enterprises(1) TUD(8301) cdc(3) policy(4) FSPKI(1)}
```

wird eine Fail-Safe-PKI-Policy mit den besonderen kryptographischen Verfahren und Protokollen definiert. Von [CP99] weichen insbesondere folgende Punkte ab:

- *1.3 Community and Applicability*
  wird um den Update Service im Trust Center erweitert
- *2.1.4 Relying party obligations*
  regeln die Verifikation multipler digitaler Signaturen
- *3.2 Routine Rekey* und *3.3 Rekey after Revocation*
  erklären, wie über den Update Service Schlüssel erneuert werden
- *4.4 Certificate Suspension and Revocation*
  definiert auch multipel signierte Sperrlisten und multipel signierte OCSP-Antworten sowie ihre Verifikation
- *4.7 Key changeover*
  beschreibt die Methoden, neue Schlüssel zu verteilen
- *4.8 Compromise and Disaster Recovery*
  erklärt die Prozeduren des Fail-Safe-Konzepts im Schadensfall entsprechend Abschnitt 4.5
- *6. TECHNICAL SECURITY CONTROLS*
  wird um Erfordernisse der multiplen Kryptographie erweitert
- *7.2 CRL Profile*
  definiert multipel signierte Sperrlisten

Wie eine Policy abgesichert ist, ist nicht standardisiert. Ein Vorschlag ist, diese Policy mit multiplen digitalen Signaturen in PKCS#7 zu codieren.

Appendix B

Update Management Protocol

Ein Highlight dieser Dissertation ist das in den Kapiteln 3 und 4 vorgestellte Update Management Protocol (UMP). Im Folgenden werden die Datenformate des Update Management Protocols spezifiziert – allgemein in Abstract Syntax Notation One (ASN.1) [ASN1] und für Chipkarten in einem speziellen Chipkarten-Kommando entsprechend ISO 7816 [ISO95] [ISO98]. Darüber hinaus werden die Transportwege des UMPs zusammengefasst, die Registry in ASN.1 codiert und die neuen Object Identifier des Fail-Safe-Konzepts zusammenfassend notiert.

B.1 Update Management Protocol in ASN.1

Das Update Management Protocol (UMP) besteht aus den beiden Teilen UpdateComponent, das der Update Service an die Clients der Certificate Holder schickt, und UpdateComponent-Response, welches der Client als Bestätigung an den Update Service sendet.

UpdateComponent

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

UpdateComponent OBJECT IDENTIFIER ::= { cdc-ump 1 }

UpdateComponent ::= SEQUENCE {
   version          INTEGER DEFAULT 1,
   updateService    UTF8String,
   umpId            INTEGER,
   freeText         [0] UTF8String         OPTIONAL,
   identity         [1] Identity           OPTIONAL,
   clientRelevant   [2] ClientRelevant     OPTIONAL,
   iccRelevant      [3] Link               OPTIONAL,
   protections      SEQUENCE OF Protection -- zwei Signaturen gefordert
   }

Identity ::= SEQUENCE {
   subjectName   [0] Name    OPTIONAL, -- aus Zertifikat, analog zu X.509
   iccsn         [1] INTEGER OPTIONAL  -- ICCSN von Chipkarte
   }
```

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

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

AlgorithmIdentifier ::= SEQUENCE { -- entsprechend [CCRL00]
   algorithm    OBJECT IDENTIFIER,
   parameters   ANY DEFINED BY algorithm OPTIONAL
   }

KeyOrCertToDelete ::= SEQUENCE {
   algorithm  AlgorithmIdentifier,           -- Lösche Schlüssel/Zertifikate zu
   usage      [0] KeyUsage         OPTIONAL, -- diesem Verfahren und diesem Usage
   keyLength  [1] INTEGER          OPTIONAL  -- maximale kritische Schlüssellänge
                                             -- z.B. 1023, wenn 1024 gefordert
   }

KeyUsage ::= BIT STRING { -- entsprechend [CCRL00]
   digitalSignature   (0), -- security mechanismen other than nonRepudiation
   nonRepudiation     (1), -- digital signature of a dokument
   keyEncipherment    (2), -- public key is used for key transport
   dataEncipherment   (3), -- for enciphering user data
   keyAgreement       (4), -- Diffie-Hellman key management
   keyCertSign        (5), -- for certificate signing
   cRLSign            (6), -- for revocation information signing
   encipherOnly       (7), -- for enciphering data while performing key agreement
   decipherOnly       (8)  -- for deciphering data while performing key agreement
   }

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

Link ::= UTF8String -- der eine http- oder ftp-Adresse spezifiziert

Registry -- siehe Abschnitt B.4 auf Seite 225

TrustAnchorToInstall ::= SEQUENCE {
   algorithm     AlgorithmIdentifier,           -- Installiere TrustAnchor zu diesem
   usage         [0] KeyUsage         OPTIONAL, -- Verfahren und diesem Usage
   trustAnchor   TrustAnchor
   }
```

```
TrustAnchor ::= CHOICE {
   certificate   [0] Certificate,
   key           [1] BIT STRING
   }

CHKeyToInstall ::= SEQUENCE {
   algorithm   AlgorithmIdentifier,           -- Generiere Key zu diesem Verfahren
   usage       [0] KeyUsage         OPTIONAL, -- und diesem Usage in PSE
   keylength   INTEGER                        -- geforderte Schlüssel-Länge
   }

Protection ::= SEQUENCE {
   signatureAlg   AlgorithmIdentifier,
   signature      BIT STRING,
   certificates   [0] CCertificate     OPTIONAL
   }

CCertificate ::= SEQUENCE {
   certificate            [0] Certificate          OPTIONAL, -- siehe A.1 auf Seite 187
   attributeCertificate   [1] AttributeCertificate OPTIONAL  -- siehe A.2 auf Seite 189
   }

UpdateComponentCode ::= SEQUENCE {
   uccNumber     INTEGER,
   provider      SEQUENCE OF Provider,
   protections   SEQUENCE OF Protection -- zwei gefordert: mit kompromittiertem
                                        -- und sicherem Signaturverfahren erzeugt
   }

Provider ::= SEQUENCE {
   providerName   UTF8String, -- z.B. FlexiCoreProvider, FlexiECProvider, ...
   code           BIT STRING  -- im Klartext oder verschlüsselt
   }
```

Kurzerklärung: Der Object Identifier { 1 3 6 1 4 1 8301 3 3 2 } weist weltweit eindeutig auf das Update Management Protocol hin. In UpdateComponent muss eine `version` enthalten sein; z. Zt. ist dies default 1. `updateService` enthält die (E-Mail-)Adresse, zu der das UpdateComponentResponse geschickt werden soll. `umpId` ist eine Kennung dieses UpdateComponents, die auch im zugehörigen -Response verwendet wird, um eine Zuordnung zu erhalten. Optional ist ein freier Text (`freeText`), der dem Benutzer angezeigt werden kann. Die Identität des Certificate Holders (`identity`) ist entweder der Name aus dem Zertifikat oder die Seriennummer der Chipkarte (`iccsn`) in einer Authentisierungs-Anwendung. Ein `UpdateComponent` enthält Code für einen Client (`clientRelevant`) und/oder für eine Chipkarte (`iccRelevant`), mindestens jedoch eines. In `protections` sind die zwei Signaturen gefordert, die mit für dieses Update geeigneten Signaturverfahren erzeugt werden.

`ClientRelevant` enthält optional eine MessageID (`messageID`) und Informationen über zu löschende Komponenten (`deletions`) und/oder Informationen über zu installierende Komponenten (`installations`), mindestens jedoch eine Information muss gesetzt sein. `deletions` geben zu löschende Komponenten an: Eine kryptographische Primitive oder ein mathematisches Basisproblem in `algorithm` – über die Registry findet der Client alle auf dieser Primitiven oder diesem Basisproblem beruhenden Verfahren –, einen Sicherheitsanker (`trustAnchor`) – der als Schlüssel oder Zertifikat in der PSE des Clients installiert sein kann – und/oder einen eigenen

Schlüssel (`chKey`). Schlüssel können über den Algorithmus und die Key Usage angesprochen werden. Falls ein Schlüssel aufgrund seiner Schlüssellänge kompromittiert wurde, kann eine maximale kritische Schlüssellänge (`keyLength`) angegeben werden, z. B. 1023, wenn 1024 Bit lange Schlüssel gefordert sind. `Key Usage` entspricht der Bedeutung aus [CCRL00].

Unter `Installations` wird angegeben, unter welchem Link (`provider`) neue Provider zu finden sind. `Installations` kann darüber hinaus eine neue Registry (`registry`), einen neuen Sicherheitsanker (`trustAnchor`) als Schlüssel oder Zertifikat und/oder den Befehl enthalten, einen eigenen neuen Schlüssel zu einer gegebenen Schlüssellänge zu generieren (`chKey`). `Link` enthält eine Adresse, über die via http oder ftp neue Provider gezogen werden können. Wie in Abschnitt 4.5 auf Seite 107 ausgeführt, besteht eine feste Konvention, wie aus der angegebenen URL und Informationen zur eigenen Systemkonfiguration die URL zusammenzusetzen ist, unter welcher genau die Provider zu finden sind, die zum eigenen System passen. Die Provider sind in `UpdateComponentCode` integriert – jeweils mit `providerName` und `code`. `UpdateComponentCode` ist für Verwaltungszwecke mit einer Nummer (`uccNumber`) gekennzeichnet und zweimal über `protections` signiert.

`iccRelevant` ist ein Link. Der Client fügt auf die beschriebene Konvention an die übermittelte URL Informationen zur Chipkarte an und erhält so den vollständigen Link, unter dem er das Chipkarten-Kommando UPDATE COMPONENT COMMAND findet, welches dem UpdateComponent für Clients entspricht. UPDATE COMPONENT COMMAND ist in Abschnitt B.2 auf Seite 223 beschrieben.

`protections` enthält zwei Signaturen vom Typ `Protection`, die jeweils Signaturverfahren (`signatureAlg`) und Signatur (`signature`) sowie optional Zertifikat (`certificate`) und Attributzertifikat (`attributeCertificate`) in `CCertificate` enthalten. Das Attributzertifikat muss den Signierer – sofern es nicht die CA zum Sicherheitsanker ist – für das UpdateComponent autorisieren. Ein geeignetes Attributzertifikat ist in Abschnitt A.2 auf Seite 191 angegeben.

Für weitere Erklärungen zum UpdateComponent sei auf Abschnitt 4.5 auf Seite 107 verwiesen.

Unter `deletions` im Eintrag `algorithm` kann über einen Object Identifier eine kryptographische Primitive oder ein mathematisches Basisproblem eindeutig angesprochen werden. Während die OIDs der Primitiven international bekannt sind, werden für die Basisprobleme die folgenden OIDs definiert:

```
cdc-problem OBJECT IDENTIFIER ::= { cdc-ump 7 }
cdc-factor OBJECT IDENTIFIER ::= { cdc-problem 1 }
cdc-dl-ff OBJECT IDENTIFIER ::= { cdc-problem 2 }
cdc-dl-ec OBJECT IDENTIFIER ::= { cdc-problem 3 }
cdc-dl-iq OBJECT IDENTIFIER ::= { cdc-problem 4 }
```

“cdc-factor“ steht für das Faktorisierungsproblem, “cdc-dl-ff“ für das DL-Problem in endlichen Körpern, “cdc-dl-ec“ für das DL-Problem auf elliptischen Kurven und “cdc-dl-iq“ für das DL-Problem in imaginär-quadratischen Zahlkörpern.

UpdateComponentResponse

```
UpdateComponentResponse OBJECT IDENTIFIER ::= { cdc-ump 2 }

UpdateComponentResponse ::= SEQUENCE {
   version          INTEGER DEFAULT 1,
```

```
   clientRelevant   [0] UCRClientRelevant OPTIONAL,
   iccRelevant      [1] UCRICCRelevant    OPTIONAL
   }

UCRClientRelevant ::= SEQUENCE {
   umpId             INTEGER,
   subjectName       Name,
   messageClientID   [0] INTEGER          OPTIONAL,
   subjectPKInfo     [1] SEQUENCE OF SubjectPublicKeyInfo{{ PKInfoAlgorithms }}
                                        OPTIONAL, -- neuer Schlüssel,
                                                  -- entsprechend PKCS#10 codiert
   protection        SEQUENCE OF Protection
   }

UCRICCRelevant ::= BIT STRING

SubjectPublicKeyInfo {ALGORITHM: IOSet} ::= SEQUENCE { -- entsprechend PKCS#10
   algorithm          AlgorithmIdentifier {{IOSet}},
   subjectPublicKey   BIT STRING
   }
```

Kurzerläuterung: Das UpdateComponentResponse als Bestätigung für ein erfolgreiches Ausführen des UpdateComponents bezieht sich auf den Update beim Client (`clientRelevant`) und/oder bei der Chipkarte `iccRelevant` und enthält eine Versionsnummer. Der Inhalt des `iccRelevant` besteht aus dem von der Chipkarte erzeugten und an den Client gesendeten UPDATE COMPONENT RESPONSE. Falls ein CH sowohl über Client als auch über Chipkarte verfügt, beinhaltet dieses UpdateComponentResponse beide Informationen.

`UCRClientRelevant` enthält UMP-ID des UpdateComponents, den Namen des Zertifikatsinhabers, die messageID des Clients (`messageClientID`), optional Informationen zu neu generierten Schlüsseln entsprechend PKCS#10 und ist in `protection` zweimal signiert: Optional ist eine Signatur mit dem neuen Algorithmus aus `subjectPKInfo` erzeugt, damit die CA prüfen kann, ob Private und Public Key zusammenpassen. Mindestens eine Signatur ist mit einem noch sicheren Verfahren erzeugt worden, auf dessen Grundlage der Update Service der Bestätigung glauben und die CA dem Certificate Holder ein neues Zertifikat mit dem neuen öffentlichen Schlüssel ausstellen kann.

`UCRICCRelevant` enthält im Wesentlichen dieselben Informationen wie `UCRClientRelevant`, ist allerdings für Chipkarten anders codiert, so dass `UCRICCRelevant` innerhalb des UpdateComponentResponses als ein Bit String übertragen wird.

Für weitere Erklärungen zum UpdateComponentResponse sei auf Abschnitt 4.5 auf Seite 107 verwiesen.

B.2 Update Management Protocol in ISO-7816-Notation

Das zuvor beschriebene Update Management Protocol in ASN.1 zirkuliert zwischen einem Server und einem Client. Ist eine Chipkarte involviert, die ASN.1 nicht interpretieren kann, muss das Update Management Protocol mit den für Chipkarten üblichen Kommandos beschrieben werden. Ein Standard für Chipkarten-Kommandos ist die ISO-7816-Reihe. Zwei wesentliche Teile dieser Reihe sind [ISO95] und [ISO98].

Eine Analyse bestehender ISO-Kommandos hat gezeigt, dass für das Update Management Protocol keine Ausdrucksmittel bereitstehen. Aus diesem Grund ist ein neues Chipkarten-Kommando namens UPDATE COMPONENT entstanden, das im Folgenden beschrieben wird.

Personal Security Environments und insbesondere Chipkarten in der Fail-Safe-PKI sind das Thema der Dissertation „Gesicherter Austausch von kryptographischen Basiskomponenten in Personal Security Environments“ von Michael Hartmann. Für nähere Informationen sei deshalb auf [Hart02] verwiesen.

Das Kommando UPDATE COMPONENT besteht aus dem COMMAND (Tab. B.1), welches vom Client zur Chipkarte gereicht wird, und der Antwort RESPONSE (Tab. B.2) der Chipkarte. COMMAND und RESPONSE bestehen aus TLV-Datenobjekten, die in Tabelle B.3 aufgeführt sind.

Tabelle B.1: UPDATE COMPONENT COMMAND

CLA	’80’	proprietäres Kommando
INS	’XY’	UPDATE COMPONENT
P1	’00’	
P2	xx	Anzahl der folgenden Data-Blöcke
Lc	L	
Data	xx	Konkatenation von TLV-Datenobjekten
Le	-	

Tabelle B.2: UPDATE COMPONENT RESPONSE

Data	xx	Konkatenation von TLV-Datenobjekten oder leer
SW1 - SW2	xx	Status

Kurzerklärung: UPDATE COMPONENT COMMAND ist ein proprietäres Kommando, das im Wesentlichen eine Konkatenation von TLV-codierten Datenobjekten enthält, in denen das UpdateComponent chipkartenspezifisch beschrieben ist. TLV steht für Tag Length Value. Im UPDATE COMPONENT RESPONSE finden sich TLV-Datenobjekte der Antwort, in welcher das UpdateComponentResponse der Chipkarte enthalten ist. Die TLV-Datenobjekte sind analog zum UpdateComponent resp. -Response aufgebaut. Falls aufgrund eines Fehlers keine erfolgreiche Bestätigung erfolgt, bleibt das Data-Feld leer und der Status kennzeichnet eine Fehlermeldung.

B.3 Transportwege des Update Management Protocols

Wie werden UpdateComponent und UpdateComponentResponse transportiert? Das Update Management Protocol (UMP) kann integriert werden in:

- PKCS#7 – siehe Abschnitt A.6 auf Seite 209
- CRL-Sperrliste – siehe Abschnitt A.3 auf Seite 193

Tabelle B.3: TLV-Datenobjekte für UPDATE COMPONENT COMMAND und RESPONSE

Tag	Daten-Element
'01'	Versionsnummer
'02'	UMP-ID
'03'	ICCID der ICC-Applikation, für die dieses UpdateComponent gilt
'04'	Identität des Certificate Holders
'05'	MessageICCID – MessageID der Chipkarte
'06'	zu löschen (constructed tag)
'07'	zu installieren (constructed tag)
'08'	Algorithmus-Identifier
'09'	CA-Key – Ansprechen eines CA-Keys / Sicherheitsanker
'0A'	CH-Key – Ansprechen eines CH-Keys
'0B'	Algorithmus zum Key
'0C'	KeyUsage des Keys – als BIT STRING entsprechend [CCRL00]
'0D'	KeyLength zum Key
'0E'	Code als – u. U. codierter – BIT STRING für Algorithmus
'0F'	Registry als BIT STRING, so wie sie die Karte ablegt und verarbeiten kann
'11'	Key als BIT STRING
'12'	CH-Key erzeugen – Kommando zum Erzeugen eines neuen Schlüsselpaares
'13'	Signatur-Informationen – entsprechend der Protection
'14'	Algorithmus-Identifier des Signaturverfahrens
'15'	Signatur als BIT STRING
'16'	CV-Zertifikat im Sinne eines Attributzertifikats, falls Sicherheitsanker nicht zu dieser Signatur gehört

- OCSP-Antwort – siehe Abschnitt A.4 auf Seite 197
- TLS – siehe Abschnitt A.8 auf Seite 214

Die dazu neu definierten Object Identifier sind in Abschnitt B.5 auf Seite 227 zusammenfassend nachzulesen.

B.4 Registry des Fail-Safe-Konzepts

Die Registry definiert für jede kryptographische Primitive, zu denen in diesem Zusammenhang als übergeordnete Primitive auch mathematische Basisprobleme gezählt werden, und jedes Verfahren die zum Löschen nötigen Signaturverfahren, die in der UMP-Verifikation präsentiert werden müssen, damit der entsprechende Security Status gesetzt und ein Deaktivieren ermöglicht wird. Die Festlegung, welche Security Conditon pro Primitive oder zusammengesetztem Verfahren gesetzt werden muss, ist von Menschen im Trust Center festzulegen.

Da die Registry sicherheitskritisch ist, wird die Registry selbst multipel signiert, falls der Client nicht für ihre Unversehrtheit garantieren kann. Die Registry hat selbst die Security Conditions, dass Lesen immer zugelassen ist, ein Update von außen nie und intern nur innerhalb eines UPDATE MANAGEMENT PROTOCOLs.

In der Registry sind alle Primitiven und Verfahren enthalten, die vom Trust Center benutzt werden. Es müssen nicht alle Verfahren bei jedem Client vorhanden sein. Dadurch hält das Trust Center nur eine Registry für alle Clients vor, so dass verhindert werden kann, dass jeder Client eine andere Registry bekommt und das Trust Center all die dafür nötigen Informationen speichern müsste.

Die in Abschnitt 4.5.5 dargestellte Tabelle (auf Seite 128) wird im Client intern in einer ASN.1-Struktur realisiert, die im Folgenden vorgestellt wird:

```
Registry ::= SEQUENCE {
   entries      SEQUENCE OF Entry,
   protection   [0] SEQUENCE OF Protection OPTIONAL
   }

Entry ::= SEQUENCE {
   identifier               INTEGER,
   primitiveOrConstructed   PrimitiveOrConstructed
   }

PrimitiveOrConstructed ::= CHOICE {
   primitive     [0] Primitive,  -- kryptographische Primitive oder math. Basisproblem
   constructed   [1] Constructed
   }

Primitive ::= SEQUENCE {
   oid                  OBJECT IDENTIFIER,
   contained_in         SEQUENCE OF INTEGER,    -- Querverweis zu anderen Identifier
   securityConditions   SEQUENCE OF ORCondition
   }

Constructed ::= SEQUENCE {
   oid                  OBJECT IDENTIFIER,
   securityConditions   SEQUENCE OF ORCondition
   }

ORCondition ::= SEQUENCE { -- logische AND-Verknüpfung
   entry_one   INTEGER,
   entry_two   INTEGER
   }
```

Kurzerklärung: Die Registry setzt sich aus `entries` zusammen und kann mit der `protection` vor Veränderung geschützt werden. Jeder `Entry` ordnet einer kryptographischen Primitive (`Primitive`), zu der in diesem Zusammenhang als übergeordnete Primitive auch ein mathematisches Basisproblem gezählt wird, oder einem zusammengesetzten Verfahren (`Constructed`) einen `identifier` zu. Jede `Primitive` enthält einen Object Identifier des mathematischen Problems oder der kryptographischen Primitive, eine Liste von `identifier` – in denen diese `Primitive` enthalten ist – und die `securityConditions`. Jedes `Constructed` enthält einen Object Identifier des entsprechenden Verfahrens und die `securityConditions`. Die Security Conditions setzen

sich aus beliebig vielen Paaren (`entry_one`, `entry_two`) zusammen – mindestens jedoch einem. Die Einträge eines Paares sind mit logischem AND und die Paare der Liste mit logischem OR verknüpft.
Die Registry in einer Chipkarte kann z. B. mit einer der ASN.1-Notation entsprechenden TLV-Codierung realisiert werden.

B.5 Object Identifier des Update Management Protocols

Neue Protokolle und Datenformate innerhalb der Fail-Safe-PKI erhalten neue Object Identifier (OID), mit denen sie sich weltweit eindeutig identifizieren lassen.
Die neuen Object Identifier sind unterhalb des OID des Fachgebiets „Kryptographie, Computeralgebra (cdc)“ der TU Darmstadt unter der Kennzahl 2 für “ump“ angesiedelt:

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

Unter dem OID für cdc-ump – {`1 3 6 1 4 1 8301 3 3 2`} – wurden weitere OIDs vergeben:

1. für das UpdateComponent und -Response:

   ```
   UpdateComponent OBJECT IDENTIFIER ::= { cdc-ump 1 }
   UpdateComponentResponse OBJECT IDENTIFIER ::= { cdc-ump 2 }
   ```

2. für ein neues Attribut für Attributzertifikate:

   ```
   id-ump-umpAuthorization OBJECT IDENTIFIER ::= { cdc-ump 3 }
   ```

3. für eine neue Erweiterung zum Transport des UpdateComponents:

   ```
   id-ump-umpTransport OBJECT IDENTIFIER ::= { cdc-ump 4 }
   ```

4. für die erweiterten OCSP- und TSP-Protokolle:

   ```
   cdc-ocspv11 OBJECT IDENTIFIER ::= { cdc-ump 5 }
   cdc-tspv11 OBJECT IDENTIFIER ::= { cdc-ump 6 }
   cdc-TSTInfov11 OBJECT IDENTIFIER ::= { cdc-tspv11 1 }
   ```

5. für die mathematischen Basisprobleme:

   ```
   cdc-problem OBJECT IDENTIFIER ::= { cdc-ump 7 }
   cdc-factor OBJECT IDENTIFIER ::= { cdc-problem 1 }
   cdc-dl-ff OBJECT IDENTIFIER ::= { cdc-problem 2 }
   cdc-dl-ec OBJECT IDENTIFIER ::= { cdc-problem 3 }
   cdc-dl-iq OBJECT IDENTIFIER ::= { cdc-problem 4 }
   ```

Darüber hinaus ist ein OID für multipel signierte Sperrlisten,

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

und ein OID für eine dieses Fail-Safe-Konzept respektierende Policy,

```
cdc-cp OBJECT IDENTIFIER ::={iso(1) identified-organization(3) dod(6)
internet(1) private(4) enterprises(1) TUD(8301) cdc(3) policy(4) FSPKI(1)},
```

vergeben worden.

Literaturverzeichnis

[AiLe98] Alexander Aiken und Raph Levien. *Attack-resistant trust metrics for public key certification*. In: *Proceedings of the 7th on USENIX Security Symposium*, Seiten 229–241. USENIX Assoc., 1998.

[ANX] Advanced Network eXchange (ANX). http://www.anx.com.

[Arch] Beweiskräftige und sichere Langzeitarchivierung digital signierter Dokumente (ArchiSig). http://www.archisig.de.

[ASN1] Griffin Consulting. *Abstract Syntax Notation One (ASN.1)*. http://asn-1.com.

[Austria] Österreich. *Digitale Signatur für Österreich*. http://www.austriacard.at.

[Beck99] Hubert Becker. *Logistik – Ein Überblick*. http://home.t-online.de/home/becker2/log3_1_1.htm, Juni 1999.

[Bern01] Daniel J. Bernstein. *Circuits for Integer Factorization: A Proposal*. http://cr.yp.to/papers.html/nfscircuit, November 2001.

[BiSh97] Eli Biham und Adi Shamir. *Differential Fault Analysis of Secret Key Cryptosystems*. In: *CRYPTO'97*, Lecture Notes in Computer Science 1294, Seiten 513–525. Springer, 1997.

[Blei98] Daniel Bleichenbacher. *Chosen Ciphertext Attacks against Protocols Based on the RSA Encryption Standard PKCS#1*. In: *CRYPTO'98*, Lecture Notes in Computer Science 1462, Seiten 1–12. Springer, 1998.

[Blei00] Holger Bleich. *Wenn der Rechner zweimal klingelt....* *c't*, 15: Seiten 124–126, 2000.

[BlKr01] Gerrit Bleumer und Heinrich Krüger-Gebhard. *Sicherheit moderner Frankiersysteme*. In: Dirk Fox, Marit Köhntopp und Andreas Pfitzmann, Hrsg., *Verlässliche IT-Systeme 2001*, Seiten 135–146. Vieweg, 2001.

[BMIa] Bundesinnenministerium. *Dienstausweis*. http://www.bsi.de/aktuell/exauschr/digital.htm.

[BMIb] Bundesinnenministerium. *Sphinx-Projekt*. http://www.bsi.de/aufgaben/projekte/sphinx/sphinx.htm.

[BNotK] Bundesnotarkammer. http://www.bnotk.de.

[BPRS02] Ralf Brandner, Ulrich Pordesch, Alexander Roßnagel und Joachim Schachermayer. *Langzeitsicherung qualifizierter elektronischer Signaturen*. *DUD*, 2/2002: Seiten 97–103, 2002.

[Bürger] Zentrum für sichere Informationstechnologie – AUSTRIA (A-SIT). *Die Bürgerkarte*. http://www.buergerkarte.at.

[BSI] Bundesamt für Sicherheit in der Informationstechnik (BSI). http://www.bsi.de.

[BSI-SigI99a] Bundesamt für Sicherheit in der Informationstechnik (BSI). *Spezifikation zur Entwicklung interoperabler Verfahren und Komponenten nach SigG/SigV – Signatur-Interoperabilitätsspezifikation (SigI) – Abschnitt A1 Zertifikate*, 30. April 1999.

[BSI-SigI99b] Bundesamt für Sicherheit in der Informationstechnik (BSI). *Spezifikation zur Entwicklung interoperabler Verfahren und Komponenten nach SigG/SigV – Signatur-Interoperabilitätsspezifikation (SigI) – Abschnitt B5 Mehrfachsignaturen/Erneute Digitale Signatur*, 30. August 1999.

[BSI01] Bundesamt für Sicherheit in der Informationstechnik (BSI). *Geeignete Kryptoalgorithmen*, 5. Juli 2001.

[BSI02] Bundesamt für Sicherheit in der Informationstechnik (BSI). *Geeignete Kryptoalgorithmen*, 15. April 2002.

[Buch99] Johannes Buchmann. *Einführung in die Kryptographie*. Springer, 1999.

[Buch01] Johannes Buchmann. *Wie sicher kann Sicherheit sein?*. Technical Report No. TI-5/01, 28.03.2001, Technische Universität Darmstadt, Institut für Theoretische Informatik, 2001. http://www.informatik.tu-darmstadt.de/TI/Veroeffentlichung/TR.

[BuMa00] Johannes Buchmann und Markus Maurer. *Wie sicher ist die Public-Key-Kryptographie?*. In: Patrick Horster, Hrsg., *Systemsicherheit*, Seiten 105–116. Vieweg, 2000.

[Bund] BundOnline 2005. http://www.bund0nline2005.de.

[BuRT00] Johannes Buchmann, Markus Ruppert und Markus Tak. *FlexiPKI – Realisierung einer flexiblen Public-Key-Infrastruktur*. In: Patrick Horster, Hrsg., *Systemsicherheit*, Seiten 309–314. Vieweg, 2000.

[CCRL99] Network Working Group. *Internet X.509 Public Key Infrastructure – Certificate and CRL Profile*. Request for Comments 2459, Januar 1999.

[CCRL00] Network Working Group. *Internet X.509 Public Key Infrastructure – Certificate and CRL Profile*. Internet Draft, März 2000.

[CMC01] PKIX Working Group. *Internet X.509 Public Key Infrastructure – Certificate Mangement Messages over CMS*. Internet Draft, Juli 2001.

[CMP99] Network Working Group. *Internet X.509 Public Key Infrastructure – Certificate Management Protocols*. Request for Comments 2510, März 1999.

[CMP01] Network Working Group. *Internet X.509 Public Key Infrastructure – Certificate Management Protocols*. Internet Draft, Mai 2001.

[CMS99] Network Working Group. *Cryptographic Message Syntax*. Request for Comments 2630, Juni 1999.

[CP99] Network Working Group. *Internet X.509 Public Key Infrastructure – Certificate Policy and Certification Practices Framework*. Request for Comments 2527, März 1999.

[DINSig99] DIN NI-17.4. *Spezifikation der Schnittstelle zu Chipkarten mit Digitaler Signatur-Anwendung/Funktion nach SigG und SigV, Version 1.0*, 20. April 1999.

[DiSc01] Manja Diering und Klaus Schmeh. *Zertifizierter Paragrafendschungel*. *c't*, 13: Seiten 182–184, 2001.

[Dobb96] Hans Dobbertin. *Cryptanalysis of MD4*. In: Dieter Gollmann, Hrsg., *Fast Software Encryption: Third International Workshop*, Lecture Notes in Computer Science 1039, Seiten 53–69. Springer, 1996.

[Duden96] Duden. *Die deutsche Rechtschreibung*. Duden-Verlag, 21. Ausgabe, 1996.

[Duden97] Duden. *Das Herkunftswörterbuch*. Duden-Verlag, 2. Ausgabe, 1997.

[EcEG00] Claudia Eckert, Florian Erhard und Johannes Geiger. *GSFS – Ein gruppenfähiges, verschlüsselndes Dateisystem*. In: Patrick Horster, Hrsg., *Systemsicherheit*, Seiten 29–40. Vieweg, 2000.

[eLex] d vision. *Lexikon*. http://www.d-vision.de/ecommerce/lexikon, 15.5.2001.

[Elster] Elster. *Elektronische Steuererklärungen*. http://www.elster.de.

[EU A9C01] EU A9C (Article 9 Committee). *Algorithms and Parameters for Secure Electronic Signatures, V.144 Draft vom 4. Mai 2001*, 2001.

[EU99] Das euroäische Parlament und der Rat der europäischen Union. *Richtlinie 1999/93/EG des europäischen Parlaments und des Rates vom 13. Dezember 1999 über gemeinschaftliche Rahmenbedingungen für elektronische Signaturen*. Amtsblatt der Europäischen Gemeinschaften 19.1.2000 L 13/12-20, 13. Dezember 1999.

[FairPay] FairPay – Verläßlichkeit im elektronischen Zahlungsverkehr. http://fairpay.dfki.de.

[Fell01] Hans-Willi Fell. *Interoperabilität in PKI-Anwendungen – Notwendigkeit oder Marketingfloskel*. *DUD*, 9/2001: Seiten 536–538, 2001.

[Finnland] Finnland. *Finn-ID*. http://www.vaestorekisterikeskus.fi/prc.htm.

[FJPP95] Hannes Federrath, Anja Jerichow, Andreas Pfitzmann und Birgit Pfitzmann. *Mehrseitig sichere Schlüsselerzeugung*. In: Patrick Horster, Hrsg., *Trust Center*. Vieweg, 1995.

[FlexiPKI] Technische Universität Darmstadt. *Institut für Theoretische Informatik. Lehrstuhl Prof. Dr. Buchmann. Forschungsprojekt „FlexiPKI“*. http://www.informatik.tu-darmstadt.de/TI/Forschung/Flexi-PKI/Welcome.html.

[GBO] Der Gesetzgeber. *Grundbuchordnung (GBO)*.

[GiSc00] Ernst-Günter Giessmann und Roland Schmitz. *Zum Gültigkeitsmodell für elektronische Signaturen nach SigG und X.509*. *DUD*, 7/2000: Seiten 401–404, 2000.

[HaMa00] Michael Hartmann und Sönke Maseberg. *Fail-Safe-Konzept für FlexiPKI*. Technical Report No. TI-11/00, 14.12.2000, Technische Universität Darmstadt, Institut für Theoretische Informatik, 2000.

[HaMa01a] Michael Hartmann und Sönke Maseberg. *Fail-Safe-Konzept für FlexiPKI*. In: Patrick Horster, Hrsg., *Kommunikationssicherheit im Zeichen des Internet*, Seiten 128–138. Vieweg, 2001.

[HaMa01b] Michael Hartmann und Sönke Maseberg. *Fail-Safe-Konzept für Public-Key-Infrastrukturen*. In: Dirk Fox, Marit Köhntopp und Andreas Pfitzmann, Hrsg., *Verlässliche IT-Systeme – VIS 2001*, Seiten 179–191. Vieweg, 2001.

[HaMa01c] Michael Hartmann und Sönke Maseberg. *Replacement of Components in Public Key Infrastructures*. In: *Proceedings of IECON'01 - The 27th Annual Conference of the IEEE Industrial Electronics Society*, Seiten 2012–2016. IEEE, 2001.

[HaMa01d] Michael Hartmann und Sönke Maseberg. *SmartCards for the FlexiPKI Environment*. In: *Gemplus Developers Conference, Paris*, 2001. Verfügbar als Technical Report No. TI-9/01, 29.05.2001, unter http://www.informatik.tu-darmstadt.de/TI/Veroeffentlichung/TR.

[Hart02] Michael Hartmann. *Gesicherter Austausch von kryptographischen Basiskomponenten in Personal Security Environments*. Dissertation, Technische Universität Darmstadt, 2002. In Vorbereitung.

[HBCI] Zentraler Kredit Ausschuß (ZKA). *Home Banking Computer Interface*. http://www.hbci-zka.de.

[Iden] Identrus. http://www.identrus.com.

[IETF] Internet Engineering Task Force (IETF). http://www.ietf.org.

[Irel] The Irish Times vom 5. März 2000. *French banks panic over electronic cards*. http://www.ireland.com/newspaper/finance/2000/0315/fin18.htm.

[ISO89] ISO/IEC. *7498-2, Information processing system – Open systems interconnection – Basic reference model – Part 2: Security architecture*, 1989.

[ISO95] ISO/IEC. *7816-4 – Information Technology – Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 4: Interindustry Commands for Interchange*, 1995.

[ISO97a] ISO/IEC. *9594-8: — ITU-T Recommendation X.509 (1997E) – Annex A – Authentication Framework in ASN.1*, 1997.

[ISO97b] ISO/IEC. *9796, Information Technology – Security techniques – Digitale Signature schemes giving message recovery – Part 2: Mechanisms using a hash-function*, 1997.

[ISO97c] International Telecommunication Union. *ITU-T Recommendation X.509 (06/97) – Information Technology – Open Systems Interconnection – the Directory: Authentication Framework*, 1997.

[ISO98] ISO/IEC. *FDIS 7816-8 – Identification Cards – Integrated Circuit(s) Cards with Contacts – Part 8: Security Related Interindustry Commands*, 1998.

[iVote] i-Vote Wahlen im Internet. http://www.ivote.de.

[IZN] Informatikzentrum Niedersachsen (IZN). http://www.izn.de.

[JaJK99] Joshua Jaffe, Benjamin Jun und Paul C. Kocher. *Differential Power Analysis*. In: *CRYPTO'99*, Lecture Notes in Computer Science 1666, Seiten 388–397. Springer, 1999.

[Kale01] Igor Kalenderian. *Implementierung des Austausches kryptographischer Komponenten in FlexiPKI mittels Update Management Protocol*. Diplomarbeit, Technische Universität Darmstadt, 2001.

[KeSc01] Thomas Keinz und Markus Schneider. *Proof of Authorship for Copyright Protection in OPELIX*. In: *Präsentiert auf der Electronic Imaging in the Visual Arts (EVA 2001) in Florence, Italien*, 26.-30. März 2001.

[Koch96] Paul C. Kocher. *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*. In: *CRYPTO'96*, Lecture Notes in Computer Science 1109, Seiten 104–113. Springer, 1996.

[Koch98] Paul C. Kocher. *On Certificate Revocation and Validation*. In: Rafael Hirschfeld, Hrsg., *Financial Cryptography*, Vol. 1465, Seiten 172–177. Springer, 1998.

[LDAP99] Network Working Group. *Internet X.509 Public Key Infrastructure – Operational Protocols – LDAPv2*. Request for Comments 2559, April 1999.

[LeLe93] Arjen K. Lenstra und Henrik W. Lenstra Jr., Hrsg. *The development of the number field sieve*. Lecture Notes in Mathematics 1554. Springer, 1993.

[LeVe00] Arjen K. Lenstra und Eric R. Verheul. *Selecting Cryptographic Key Sizes*. In: *Public Key Cryptography 2000*, Seiten 446–465, 2000.

[LLMP90] Arjen K. Lenstra, Henrik W. Lenstra Jr., Mark S. Manasse und John M. Pollard. *The number field sieve*. In: *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, Seiten 564–572, 1990.

[LLMP93] Arjen K. Lenstra, Henrik W. Lenstra Jr., Mark S. Manasse und John M. Pollard. *The factorization of the ninth Fermat number*. *Mathematics of Computation*, 61: Seiten 319–349, 1993.

[Mase02] Sönke Maseberg. *Konzeption von Fail-Safe-Systemen*. In: Bruno Struif, Hrsg., *Tagungsband 12. SIT-SmartCard Workshop*, 2002.

[Maur96] Ueli Maurer. *Modelling a Public-Key Infrastructure*. In: *European Symposium on Research in Computer Security (ESORICS)*, Lecture Notes in Computer Science. Springer, 1996.

[McRu00] Patrick McDaniel und Aviel Rubin. *A Response to "Can We Eliminate Certificate Revocation Lists?"*. In: *Proceedings of Financial Cryptography 2000*, 2000.

[MeOV97] Alfred Menezes, Paul van Oorschot und Scott Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[Mica96] Silvio Micali. *Efficient Certificate Revocation*. Technical Report Technical Memo MIT/LCS/TM-542b, Massachusetts Institute of Technology, 1996.

[Moor65] Gordon E. Moore. *Cramming more components onto integrated circuits*. *Electronics Magazine*, 38: Seiten 114–117, April 1965.

[MTT] TeleTrusT. *MailTrusT (AG 8) – Pilotprojekt Digitale Signatur für den Dokumentenaustausch*. http://www.darmstadt.gmd.de/mailtrust.

[Myer98] Michael Myers. *Revocation: Options and Challenges*. In: Rafael Hirschfeld, Hrsg., *Financial Cryptography*, Vol. 1465, Seiten 165–172. Springer, 1998.

[NaPT01] David Naccache, David Pointcheval und Christophe Tymen. *Monotone Signatures*. In: P. Syverson, Hrsg., *Proceedings of Financial Cryptography 2001*. Springer, 2001.

[NIST] National Institute of Standards and Technology (NIST) – Computer Securi- ty Resource Center (CSRC). *AES – Advanced Encryption Standard*. http://csrc.nist.gov/encryption/aes.

[OCSP99] Network Working Group. *Internet X.509 Public Key Infrastructure – Online Certificate Status Protocol – OCSP*. Request for Comments 2560, Juni 1999.

[OCSP01] Network Working Group. *Online Certificate Status Protocol, version 2*. Internet Draft, März 2001.

[Odly95] Andrew M. Odlyzko. *The future of integer factorization*. *CryptoBytes*, 1, 2 (1995): Seiten 5–12, 1995.

[Odly99] Andrew M. Odlyzko. *Discrete logarithms: The past and the future*. *AT&T Labs – Research*, 19. Juli 1999.

[OIC00] TeleTrusT. *German Office Identity Card (Dienstausweis), Version 0.9*, 5. März 2000.

[Opelix] Open Personalized Electronic Information Commerce System (Opelix). http://www.opelix.org.

[P1363a01] Institute of Electrical und Electronics Engineers (IEEE). *P1363a/D9 – Standard Specifications for Public Key Cryptography: Additional Techniques*. Draft Version 9, Juni 2001.

[PePf97] Torben Pryds Pedersen und Birgit Pfitzmann. *Fail-Stop Signatures*. *SIAM Journal on Computing*, 26(2): Seiten 291–330, 1997.

[Pfit01] Andreas Pfitzmann. *Zur Technik der digitalen Signaturen*. In: Otto Ulrich Christian J. Langenbach, Hrsg., *Elektronische Signaturen. Kulturelle Rahmenbedingungen einer technischen Entwicklung. Wissenschaftsethik und Technikfolgenbeurteilung, Bd. 12*. Springer, 2001.

[PGP] Phil Zimmermann. *Pretty Good Privacy*. http://www.pgpi.com.

[PKCS1a] RSA Laboratories. *Public-Key Cryptography Standards #1: RSA Encryption Standard, Version 1.5 vom 1. November 1993*. http://www.rsalabs.com/pkcs/pkcs-1.

[PKCS1b] RSA Laboratories. *Public-Key Cryptography Standards #1: RSA Cryptography Standard, Version 2.0 vom 1. Oktober 1998*. http://www.rsalabs.com/pkcs/pkcs-1.

[PKCS10] RSA Laboratories. *Public-Key Cryptography Standards #10: Certification Request Syntax Standard, Version 1.7 vom Mai 2000*. http://www.rsalabs.com/pkcs/pkcs-10 und Request for Comments 2314.

[PKCS7] RSA Laboratories. *Public-Key Cryptography Standards #7: Cryptographic Message Syntax Standard, Version 1.5 vom 1. November 1993*. http://www.rsalabs.com/pkcs/pkcs-7 und Request for Comments 2315.

[PKIX] IETF Working Group: Public Key Infrastructure (X.509) (PKIX). http://www.ietf.org/html.charters/pkix-charter.html.

[Post01] Deutsche Post. *Allgemeine Geschäftsbedingungen für die eTrust-Dienstleistungen der Deutschen Post Signtrust*. http://www.signtrust.de, Januar 2001.

[Quasi] Qualitätssicherung in der Nierenersatztherapie (Quasi-Niere). http://www.quasi-niere.de.

[Rive98] Ronald L. Rivest. *Can We Eliminate Certificate Revocation Lists?*. In: Rafael Hirschfeld, Hrsg., *Financial Cryptography*, Vol. 1465, Seiten 178–183. Springer, 1998.

[Road00] PKIX Working Group. *Internet X.509 Public Key Infrastructure – PKIX Roadmap*. Internet Draft, Mai 2000.

[RoSe01] Tanja Römer und Jean-Pierre Seifert. *Information Leakage Attacks against Smart Card Implementations of the Elliptic Curve Digital Signature Algorithm*. In: Isabelle Attali und Thomas Jensen, Hrsg., *Smart Card Programming and Security (e-smart 2001)*, Lecture Notes in Computer Science 2140, Seiten 211–219. Springer, 2001.

[RSA99] RSA. *The Faktorization of RSA-140*. *RSA Laboratories – Bulletin – Number 10*, 8. März 1999.

[RZ] Rhein-Zeitung. *RZ-Online Archiv vom 13. und 21. März 2000 und 4. April 2000*. http://rhein-zeitung.de.

[SaSc75] Jerome H. Saltzer und Michael D. Schroeder. *The protection of information in computer systems*. In: *Proceedings of the IEEE*, Vol. 63(9), Seiten 1278–1308, 1975.

[Schn96] Bruce Schneier. *Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C*. Addison-Wesley, 1996.

[Schr01] Bundeskanzler Gerhard Schröder. *Rede anlässlich der Tagung der Behördenleiter des Bundes zur Initiative BundOnline 2005 "Moderne Verwaltung in der Informationsgesellschaft" am 14. Mai 2001 in Berlin (Museum für Kommunikation).* http://www.bundonline2005.de/de/index.html, 2001.

[Schu01] Christiane Schulzki-Haddouti. *Karten neu gemischt*. *c't*, 25: Seiten 274–281, 2001.

[Secude] Secude. http://www.secude.de.

[Seif02] Jean-Pierre Seifert. *Seitenkanal-Attacken auf SmartCard-Implementierungen des ECDSA*. In: Bruno Struif, Hrsg., *Tagungsband 12. SIT-SmartCard Workshop*, 2002.

[Shou98] Victor Shoup. *Why Chosen Ciphertext Security Matters*. Report RZ 3076, IBM Research, November 1998.

[Siet01a] Richard Sietmann. *Digitale Signaturen mit Hindernissen*. c't-Ticker unter http://www.heise.de/newsticker/data/jk-17.01.01-010, 17. Januar 2001.

[Siet01b] Richard Sietmann. *Verführerischer Charme.... c't*, 11: Seiten 22–23, 2001.

[SigG97] Der Gesetzgeber. *Gesetz zur digitalen Signatur (Signaturgesetz – SigG)*, 13. Juni 1997.

[SigG01] Der Gesetzgeber. *Gesetz zur digitalen Signatur (Signaturgesetz – SigG)*, 1. Mai 2001.

[SigV-B] Der Gesetzgeber. *Begründung zur Verordnung zur digitalen Signatur (in der Fassung des Beschlusses der Bundesregierung vom 8.10.1997).*

[SigV97] Der Gesetzgeber. *Verordnung zur digitalen Signatur (Signaturverordnung – SigV)*, 8. Oktober 1997.

[SigV01] Der Gesetzgeber. *Verordnung zur elektronischen Signatur (Signaturverordnung – SigV)*, 16. November 2001.

[SMIME] IETF Working Group: S/MIME Mail Security (smime). http://www.ietf.org/html.charters/smime-charter.html.

[SSL] Netscape. *SSL 3.0 Specification*. http://home.netscape.com/eng/ssl3/ssl-toc.html.

[Stie01] Wolfgang Stieler. *Drei mal fünf ist fünfzehn*. c't-Ticker unter http://www.heise.de/newsticker/data/wst-20.12.01-003, 20. Dezember 2001.

[Tage] ARD. *Tagesschau vom 14. März 2000*. http://www.tagesschau.de.

[TC] TC TrustCenter. http://www.trustcenter.de.

[Tele01] Deutsche Telekom. *Public Key Service – Allgemeine Geschäftsbedingungen*, 2001.

[TLS99] Network Working Group. *The TLS Protocol Version 1.0*. Request for Comments 2246, Januar 1999.

[TPS] Trusted Pocket Signer (TruPoSign). http://www.vernet-info.de/projekt07.html.

[TSP01] PKIX Working Group. *Internet X.509 Public Key Infrastructure – Time Stamp Protocol (TSP)*. Internet Draft, Mai 2001.

[TZI] Universität Bremen. *Arbeitsgruppe Telekomunikation, Projekt "Media@Komm"*. http://www.mediakomm.net.

[Vernet] Sichere und verlässliche Transaktion in offenen Kommunikationsnetzen (Vernet). http://www.vernet-info.de.

[VOB] Der Gesetzgeber. *Verdingungsordnung für Bauleistungen (VOB)*.

[VOL] Der Gesetzgeber. *Verdingungsordnung für Leistungen (VOL)*.

[Wohl00] Petra Wohlmacher. *Konzepte zur multiplen Kryptographie*. In: Patrick Horster, Hrsg., *Systemsicherheit*, Seiten 357–369. Vieweg, 2000.

[Wohl01] Petra Wohlmacher. *Digitale Signaturen und Sicherheitsinfrastrukturen*. IT Verlag für Informationstechnik, 2001.

Abbildungsverzeichnis

Tabellenverzeichnis

Index

Curriculum Vitae (Wissenschaftlicher Werdegang)

31.12.1970	Geboren in Bremen, Deutschland
9/77 – 6/91	Schulausbildung
6/91	*Abitur*, Gymnasium in Bremen
10/92 – 12/98	Studium der Mathematik und Informatik an der Universität Bremen
12/98	Diplom in Mathematik (*Dipl.-Math.*)
seit 5/99	Doktorand am Fraunhofer-Institut für Sichere Telekooperation (SIT) (vormals GMD-Forschungszentrum Informationstechnik) in Darmstadt
seit 10/99	Promotionsstudent bei Prof. Dr. J. Buchmann an der Technischen Universität Darmstadt
seit 3/01	Wissenschaftlicher Mitarbeiter an der Technischen Universität Darmstadt im Rahmen des „FairPay“-Projekts für „Verläßlichkeit im elektronischen Zahlungsverkehr“