# Hardware-Crypto-Token gestütztes Single Sign-On für zertifikatsbasierte Authentifizierung

Dissertation

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

vorgelegt der

Naturwissenschaftlichen Fakultät III
Agrar-, Geowissenschaften, Mathematik und Informatik
(Institut für Informatik)
der Martin-Luther-Universität Halle-Wittenberg

von Herrn Sandro Wefel
geb. am 28. April 1972 in Wittenberg

Halle (Saale), Oktober 2009

# Inhaltsverzeichnis

# Kapitel 1

# Einleitung

Das Leben im beginnenden 21. Jahrhundert wird zunehmend vom Einsatz vernetzter digitaler Technik und den damit verbundenen Technologien geprägt. Neue Dienste, wie interaktives Fernsehen, *just-in-time* Zugriff auf weltweite Informationsspeicher, Online-Bibliotheken und die Möglichkeit eigene kreative Werke ohne kostspieligen Zusatzaufwand einer breiten Öffentlichkeit zur Verfügung zu stellen, sind nur einige Beispiele dieser neuen vernetzten Zukunft. Insbesondere junge Menschen nutzen zunehmend diese Möglichkeiten. Aktuelle Trends, wie *Web2.0* mit seiner kollaborativen Ausprägung für die eigene kreative Nutzung des Internets und die unter diesem Stichpunkt zu nennenden Portale und Netzwerke *YouTube*, *Wikipedia* oder *studiVZ* zeigen das enorme Potential, welches vernetzte Datenangebote besitzen. Darüber hinaus liefern die zunehmenden Nutzerzahlen dieser Portale den Beweis, dass Interesse nicht nur auf einen kleinen fachkundigen Kreis beschränkt ist, sondern innerhalb breiter Schichten der Bevölkerung besteht.

Zwangsläufig wird eine bedarfsgerechte Bereitstellung dieser Angebote die aktuelle und zukünftige Herausforderung an die IT-Strukturen in größeren, aber auch bereits in kleineren Firmen, Ämtern und Bildungseinrichtungen sein. Aufgrund des nachhaltigen Wunsches auf zeitlich und räumlich unabhängigen Zugriff auf Informationen - „Zugriff zu jeder Zeit an jedem Ort“ - erfolgt ein stetiger Ausbau der Angebote mit steigender Tendenz zur Interaktivität. Dabei handelt es sich um Angebote, die über den Abruf der vom Anbieter bereitgestellten statischen Information hinaus gehen und den Anwender einbeziehen. Es ergeben sich personalisierte Informationen oder Dienstleistungen. In vielen Arbeitsbereichen ist der Zugriff auf die „im Netz gespeicherten“ Informationen mittlerweile eine Voraussetzung für eine optimale Arbeitsumgebung. Zunehmend wird auch selbständiges Lernen auf Informationen „aus dem Netz“ aufbauen.

Der Trend zum weiteren Ausbau netzgestützter Technologien und der Einsatz neuer Dienste ist folglich ungebrochen. Einige mit den zugrunde liegenden Techniken verbundene Probleme können aber zu einer zunehmenden Distanzierung von angebotenen Diensten führen. Man denke dabei an den unzureichenden Datenschutz für die Menge an Informationen, die bei den Anbietern gesammelt und hinterlegt werden [Gar01]. Hier gibt es offensichtlich eine Diskrepanz zwischen den Interessen von Unternehmen und der Privatsphäre der Nutzer, welche zu einer missbräuchlichen Anwendung von anvertrauten persönlichen Daten führen kann.

Zu großen Teilen folgt der Missbrauch persönlicher Daten aber auch aufgrund des Unverständnisses gegenüber den möglichen und notwendigen Schutzmaßnahmen beim Umgang mit sensiblen Daten und deren Übertragung in Netzwerken. Möchte man Dienste im Netz nutzen, so muss man insbesondere mit persönlichen Daten - und dazu gehört bereits die scheinbar unverfängliche Email-Adresse - sorgsam umgehen. Bei der Übermittlung sollten, insbesondere wenn es sich um vertrauliche Da-

ten handelt, technische Maßnahmen ergriffen werden, um diese bei der Übertragung vor unbefugtem Zugriff zu schützen. Verschlüsselte Übertragung sensibler Daten wäre eine Möglichkeit zur Lösung dieses Problems. Verschlüsselung allein ist jedoch nicht ausreichend, da für eine wirklich sichere Übertragung bestimmte Bedingungen zu beachten sind, was sowohl das Verfahren, Schlüsselerzeugung, Schlüsseltausch und Schlüsselaufbewahrung angeht. Eine Beachtung dieser Bedingungen ist wichtig, um Angriffe auf scheinbar sicher verschlüsselte Daten zu vermeiden. So gilt es insbesondere „*real world*“ Schwachstellen, die bei der praktischen Umsetzung und dem realen Einsatz erst entstehen, zu erkennen und zu beseitigen [Sch07a]. Als Beispiel sei die WLAN-Absicherung genannt. Viele Hersteller von WLAN-Hardware integrieren prinzipiell als sicher geltende Verfahren, liefern aber ihre Produkte mit Standardpassworten aus. Ein Nachweis der Sicherheit des Verfahrens unter Laborbedingungen hilft hier nicht weiter, wenn die Kunden das Passwort zum realen Betrieb nicht ändern müssen.
Die Praxis zeigt, dass die oben genannten Bedingungen für den Einsatz kryptographischer Verfahren viele Anwender überfordern. Verfügbare Techniken zur sicheren Kommunikation sind oftmals zu unhandlich und damit anfällig für *real world* Schwachstellen. So werden weiterhin Techniken angeboten, die zwar funktional sind, dafür aber die Anforderungen an Daten- und Informationssicherheit nur unzureichend erfüllen. Aus diesem Grund verzichten Nutzer möglicherweise komplett auf adäquate Schutzmaßnahmen oder nutzen diese nur unzulänglich. Werden Anwender häufig mit Informationen über mögliche, für sie aber unverständliche Sicherheitsgefährdungen konfrontiert, sind sie desensibilisiert und können leicht Opfer eines realen Angriffes werden. Als Beispiel seien die häufigen Browserwarnungen über ungültige Server-Zertifikate genannt, die letztendlich dazu führen, dass viele Nutzer bedenkenlos jedes Zertifikat ungeprüft akzeptieren [CCR09]. Ein Angriff, der über ein ungültiges Zertifikat erkennbar wäre, bietet dem Angreifer somit eine große Menge möglicher Opfer. Folglich besteht Bedarf, bei der Kommunikation im Netz den Einsatz kryptographischer Verfahren zu erleichtern. Die Unterstützung der Kryptographie in Standardapplikationen und die sich daraus ergebenden Möglichkeiten müssen für den Nutzer transparenter und letztendlich einfacher anwendbar sein. Für die zum Einsatz von Kryptographie erforderlichen Grundvoraussetzungen, zu denen Schlüsselerzeugung, Zertifikatsausstellung, Einbindung in Applikationen u.v.m. gehören, existieren vielfältige, zum Teil sehr unterschiedliche technische Lösungen. Ein Endanwender ohne dahingehendes Interesse wird diese jedoch nicht nutzen, insofern sie nicht für ihn ohne Zusatzaufwand sofort einsetzbar sind.
Kryptographische Maßnahmen helfen nicht zwangsläufig bei Problemen des Datenschutzes, wie z.B. dem Missbrauch personenbezogener Daten. Zwar würden fremde Personen, die lediglich Zugriff auf den Datenverkehr erlangen, beim Ausspähen geschützter Daten behindert. Allerdings besteht weiterhin die Gefahr des Missbrauchs durch Anbieter oder Betreiber der Dienste im Netz. So ist auch ein Schutz der Daten vor dem Missbrauch durch Mitarbeiter mit administrativem Zugriff auf Dienste bzw. Rechner, welche die Dienste anbieten, erforderlich. Ohne Schutzmaßnahmen ist es einem Techniker mit hinreichenden administrativen Befugnissen durchaus möglich, die Identität einer Person im betreuten System anzunehmen und in deren Namen zu agieren. Schutz vor diesbezüglichen Missbrauch bietet die elektronische Signatur sämtlicher Inhalte. Die Signatur muss aber vor Verwendung der Inhalte geprüft werden und die Prüfung muss auf Anwenderseite erfolgen, da eine anbieterseitige Prüfung natürlich ebenfalls manipuliert werden kann. Allerdings ergibt sich damit wieder eine technische Hürde für den Anwender. Würde man Anreize zur umfassenden Einführung von Nutzer-Zertifikaten schaffen und deren einfache Einbindung in die verwendete Kommunikationssoftware (z.B. Webbrowser) ermöglichen, so wäre bereits eine gute Grundlage vorhanden, um in Zukunft auch für dieses Problem eine technische Lösung anzubieten.
Neben der Schaffung gesicherter elektronischer Übertragungswege und der Verhinderung des Miss-

brauchs durch administrative Stellen gehört die **Nutzerauthentifizierung** zu den wichtigsten Punkten, bei denen kryptographische Verfahren zur Absicherung netzgestützter Dienste hilfreich sind. Eine erfolgreiche Authentifizierung führt im anschließenden Schritt, der Autorisierungsphase, zur Prüfung und Zuweisung der Nutzerrechte, welche die weitere Rolle des Nutzers im System bestimmen. Somit ist die erfolgreiche Authentifizierung die Basis für die Nutzung der Dienste. Gerade deshalb ist aus Sicht der Daten- und Informationssicherheit auf eine Authentifizierung, welche eine zuverlässige Prüfung der behauptete Identität erlaubt, besonderes Augenmerk zu legen. Leider ist dieser Punkt oftmals unzureichend gelöst.

## 1.1 Authentifizierung

Zur Authentifizierung können verschiedene Verfahren genutzt werden. Allen Verfahren gemein ist, dass die verwendeten Authentisierungsmerkmale gewisse Qualitätskriterien erfüllen müssen, z.B. wenn es um die Wahl der Passwörter geht. Außerdem sind die Merkmale vor unbefugtem Zugriff zu schützen. Gerade im Hinblick auf einen breiten Anwenderkreis mit sehr divergentem technischem Kenntnisstand kann nicht vorausgesetzt werden, dass diese Tatsache allen Anwendern hinreichend bewusst ist. Insbesondere die sekundären Auswirkungen, die eine Weitergabe der Authentisierungsmerkmale haben können, werden gern unterschätzt. So ist seitens der Anwender nicht zwangsläufig der Schutzbedarf von Zugangsdaten einsichtig und es werden häufig Zugangskennungen für Rechnersysteme oder das WLAN an dritte Personen weitergegeben. Oder es wird mit den Daten nicht sorgsam umgegangen bzw. sie werden zu leichtfertig herausgegeben. Untersuchungen zeigen die Anfälligkeit vieler Anwender für Social-Engineering-Angriffe [Lip09]. Bei erfolgreichen Angriffen erhaltene Passwörter offenbaren gleichzeitig einen weiteren Angriffspunkt. Über zwei Drittel der Anwender nutzen weniger als drei Passwörter für verschiedene Applikationen, bis zu 30% verwenden sogar nur ein Passwort für alle netzgestützten Dienste. Das deckt sich mit weiteren Umfragen[1].
Daraus ergibt sich die Anforderung, dem Anwender eine komfortable Möglichkeit zur Authentifizierung an Rechnersystemen und Netzdiensten zu bieten und dabei gleichzeitig zu verhindern, dass die dazu genutzten Authentisierungsmerkmale kopiert und missbräuchlich verwendet werden können.
Authentifizierungsverfahren arbeiten mit persönlichen Merkmalen, die durch geheimes Wissen oder einen Besitz oder körperliche, also biometrische Merkmale oder der Kombination verschiedener Merkmale gegeben sind. Biometrische Verfahren haben den Vorteil, dass bei Personen die Authentisierungsmerkmale immer verfügbar und schwer auf eine andere Person übertragbar sind. Dagegen spricht, dass die Verfahren oftmals schwer handhabbar sind und aufwendige Hardware erfordern. Außerdem wird Biometrie von einigen Nutzern aufgrund eines Misstrauens gegenüber der Technologie und dem erforderlichen Schutz der Daten prinzipiell abgelehnt [Way08]. Obwohl sich dies in den letzten Jahren zu Gunsten der Biometrie verschiebt, so existieren immer noch Vorbehalte [Uni06].
Besser geeignet scheinen besitzbasierten Authentifizierungsverfahren zu sein. Das Merkmal liegt dabei in Form eines Objekts vor, das der Nutzer während der Authentifizierung „besitzen“ muss. Ist das Anlegen einer Kopie des Objekts sehr aufwendig, kann die versehentliche oder beabsichtigte Weitergabe der Authentisierungsmerkmale verhindert werden. Vertreter dieser Variante sind kryptographische Verfahren in Kombination mit Hardware-Token. Aufgabe der Token ist es, das Auslesen der für die Authentifizierung benötigten Informationen (der privaten Schlüssel) zu verhindern.

[1] McAfee-Studie zum Passwort-Gebrauch:
`http://www.mcafee.com/de/about/press/corporate/2007/20071009_162200_h.html`
Sophos-Internet-Umfrage:
`http://www.sophos.de/pressoffice/news/articles/2006/04/pr_de_passpoll06.html`

## 1.2 Zum Inhalt der Arbeit

Die Arbeit beschäftigt sich schwerpunktmäßig mit der Schaffung einer nutzerfreundlichen und - gemessen mit den passwortbasierten Verfahren - aus Sicht der Dienstanbieter sichereren Lösung zur Nutzerauthentifizierung. Dazu wurden typische Einsatzfälle zur Authentifizierung an einer Hochschule analysiert. Hochschulen bieten ein sehr heterogenes Testfeld, in dem unterschiedliche Anforderungen abgedeckt werden müssen. Gerade in diesem Umfeld sind häufig Defizite bei der Nutzerauthentifizierung an zentralen Diensten oder am Netzwerk zu finden.
Die verbesserte Nutzerfreundlichkeit ergibt sich durch Reduzierung der erforderlichen Authentifizierungen. Durch Single Sign-On (SSO) wird nur eine Authentifizierung bei der Anmeldung am jeweils genutzten Rechner benötigt. Weitere Authentifizierungen erfolgen unbemerkt im Hintergrund. Dies darf aber nicht auf Kosten der Sicherheit gehen. Als Basis dient deshalb ein etabliertes, als sicher geltendes Authentifizierungsverfahren auf Basis asymmetrischer Verschlüsselung. Gleichzeitig müssen dabei die Anforderungen, welche der Einsatz dieses kryptographischen Verfahrens an den Nutzer stellt, möglichst gering gehalten werden. Um dies zu erreichen, werden Hardware-Token zum Schlüsseltransport in Verbindung mit einer SSO-Software eingesetzt.
Die wesentliche Neuerung gegenüber etablierten Verfahren zum Single Sign-On ist, dass dieses System nur eine minimale clientseitige Erweiterung erfordert. Minimal bedeutet an dieser Stelle, dass lediglich eine Schnittstelle erweitert und an einen clientseitigen Dienst angebunden wird. Applikationen müssen nicht im Quelltext angepasst, lediglich rekonfiguriert werden. Im Vergleich zu anderen clientseitig arbeitenden Lösungen ist der entwickelte Ansatz unabhängig von einer speziellen herstellerspezifischen Hardware oder einem Betriebssystem oder einem hierfür vorzusehenden zentralen Authentifizierungsservice. Im Hinblick auf eine mögliche Kostenersparnis sollte dies im Hochschulbereich von Vorteil sein.
Das System hat allerdings auch Nachteile. Für den Einsatz der Token muss zusätzliche Hard- und Middleware vorhanden sein, was an einer Hochschule kein größeres Problem sein sollte. Für den Nutzer stellt es allerdings einen gewissen Aufwand dar, das Token mitzuführen und mit dem Rechner zu verbinden. Um die Nutzer trotzdem zur Annahme des für sie neuen Authentifizierungsverfahrens zu bewegen und eine Blockadehaltung zwecks Beibehaltung alter, passwortbasierter Verfahren zu vermeiden, muss ein Mehrwert erkennbar sein. Die Vereinfachung der Authentifizierung durch SSO ist dabei bereits ein wesentlicher Punkt. Hinzu kommen weitere Anwendungsfelder, welche die im Token enthaltenen Schlüssel und Zertifikate und deren sicherer Transport bieten, z.B. der Einsatz für elektronische Signatur und Möglichkeiten zur Verschlüsselung von Daten bei Übertragungen im Netz oder der Ablage auf dem lokalen Rechner. Letztendlich wird eine Grundlage für den vereinfachten Einsatz kryptographischer Verfahren auf Seiten der Nutzer gelegt. Dies liefert einen Baustein zur „Steigerung der Sicherheit durch Anwendung einfach bedienbarer kryptographischer Werkzeuge in den Händen der Nutzer“.

Die Arbeit gliedert sich wie folgt. Im zweiten Kapitel werden die kryptographischen Verfahren kurz vorgestellt, die als Grundlage für das hier vorgestellte System dienen. Das dritte Kapitel erläutert anhand der im Hochschulbereich verwendeten Netzstrukturen die Authentifizierungsschnittstellen, welche das vorgestellte System vollständig unterstützen soll. Die sich daraus ergebenden Anforderungen für das Authentifizierungsverfahren mit SSO werden im Anschluss analysiert. Im vierten Kapitel werden mögliche und aktuell verfügbare Verfahren zur Authentifizierung und für Single Sign-On vorgestellt und eine Begründung für die Wahl der zertifikatsbasierten Authentifizierung gegeben. Da zur sicheren Schlüsselaufbewahrung Hardware-Token hinzugezogen werden müssen, gibt Kapitel 5 einen Überblick über Token und deren System-Einbindung, insbesondere im Hinblick auf eine betriebssystemunabhängige Einbettung in die angestrebte Authentifizierungslösung auf Basis asymmetrischer

kryptographischer Verfahren. Im sechsten Kapitel wird die sich daraus ergebende Idee einer clientseitigen zertifikatsbasierten SSO-Umgebung erläutert, deren Realisierung Kapitel 7 beschreibt. In diesem Kapitel wird auch gezeigt, wie Server- und Clientanwendungen in das SSO-System integriert werden können. Aktuelle Software für die häufig genutzten Dienste kann zum Großteil unverändert eingebunden werden. Außerdem wird gezeigt, dass mit den Token und der gewählten Token-Schnittstelle neben der Authentifizierung weitere Anwendungen, wie die Verschlüsselung und Signatur von Daten möglich sind und dafür ebenfalls auf bestehende Software, ohne diese zu modifizieren, zurück gegriffen werden kann. Die am aufgebauten Testsystem ermittelten Ergebnisse untermauern die Aussage, dass der Ansatz im größeren Umfang praktisch eingesetzt werden kann und im Vergleich zu anderen SSO-Verfahren keine merklichen Nachteile in Bezug auf Ressourcenverbrauch besitzt. Dazu wurde die Systembelastung (Speicher, CPU-Zeit) und die Verzögerung während der Authentifizierung im praktischen Einsatz ermittelt. Die experimentell erhaltenen Ergebnisse werden im Kapitel 8 vorgestellt, zusammen mit einer Bewertung der Sicherheit der SSO-Lösung.

# Kapitel 2

# Grundlagen

## 2.1 Begriffe und ihre Verwendung

Einige der in dieser Arbeit verwendeten Begriffe werden in der Literatur zum Teil unterschiedlich verwendet. Wir wollen aus diesem Grund diese Begriffe kurz vorstellen, insbesondere wie sie in dieser Arbeit zu verstehen sind.

**Authentizität/Integrität** Die Authentizität beschreibt die Echtheit eines Objekts oder Subjekts. Ein Dokument ist authentisch, wenn eine unabstreitbare, feste Zuordnung zur Identität des Verfassers möglich ist. Die Zuordnung muss geprüft werden können. Unter Integrität versteht man die Gewährleistung, dass ein Objekt nicht unerlaubt und unbemerkt veränderbar ist. Um die Integrität eines Dokumentes zu gewährleisten, muss der Inhalt dem Verfasser zugeordnet werden können. Eine Veränderung, die nicht durch den Verfasser erfolgte, muss erkennbar sein.

Authentizität und Integrität, verbunden mit Wahrung der Vertraulichkeit des Inhaltes von Daten sind wesentliche Schutzziele zur Schaffung von Informationssicherheit (eng. *security*) [Eck06].

**Authentisierung** Das Vorlegen von **Authentisierungsmerkmalen** gegenüber einer anderen Person oder einem Dienst wird als Authentisierung bezeichnet. Die Merkmale dienen dazu, die eigene Identität zu belegen. Eine behauptete Identität kann sowohl von einer Person, als auch von einem sächlichen Gegenstand stammen. So kann sich eine Person gegenüber einer anderen Person oder einem Dienst authentisieren, z.B. beim Abrufen der Email. Ein Objekt kann sich gegenüber einem Dienst authentisieren, z.B. ein Notebook, welches sich an einem Netzwerkzugangsknoten anmeldet.

**Authentifizierung** Der Vorgang der Überprüfung vorgelegter Authentisierungsmerkmale wird als Authentifizierung bezeichnet. Während des Vorgangs wird die Integrität und Authentizität der Merkmale geprüft. Die mittels der vorgelegten Merkmale behauptete Identität einer Person oder eines Objekts wird während der Authentifizierung verifiziert [MA08].

**Autorisierung** Als Autorisierung wird der Schritt bezeichnet, bei dem einer Person oder einem Objekt Rechte eingeräumt werden. Eine erfolgreiche Authentifizierung ist die Voraussetzung, um anhand der bestimmten Identität spezifische Rechte zu vergeben. Eine nicht erfolgreiche Authentifizierung ist meist gleichzusetzen mit der Einräumung eingeschränkter Rechte, auch als Gast-Status bezeichnet [Rie07].

**Dienst** Der Begriff Dienst ist eine Bezeichnung für eine bestimmte Funktionalität, die ein Computer zur Verfügung stellt (engl. *Service*). In dieser Arbeit ist, wenn nicht gesondert angemerkt, darunter ein Netzwerkdienst zu verstehen. Ein Netzwerkdienst ist eine Funktion, die über das Computernetz von einem entfernten Computersystem genutzt werden kann. Ein Beispiel dafür wären Email-Systeme, die das Abholen und das Versenden von Emails erlauben. In diesem Falle bietet der Mailserver zwei Dienste an, die als *Fetch Mail* und *Submit Mail* bezeichnet werden können.

Netzwerkdienste werden von einem Anbieter (engl. *Service Provider*) zur Verfügung gestellt.

Mit dem Begriff Dienst werden auch (System-)Programme bezeichnet, die im Hintergrund auf einem Rechner laufen und bestimmte Funktionalitäten bereitstellen. Sie entsprechen einem Dämon bzw. *Daemon* (engl. ***D**isk **a**nd **E**xecution **Mon**itor*). Die Bezeichnungen Dämon und Netzwerkdienst können ein laufendes Programm bezeichnen, da ein Netzwerkdienst von einem Computer fast immer durch einen Dämon bereitgestellt wird.

**Server** Mit dem Begriff Server wird in der Literatur sowohl die Hardware, also ein Computer bezeichnet, der Dienste im Rechnernetzwerk zur Verfügung stellt, als auch ein Dämon, welcher einen Dienst anbietet. Ein Serverprogramm zeichnet aus, dass es zugehörige Clientprogramme gibt.

**Client** Analog zum Server wird der Begriff Client für Computer oder Programme verwendet. Ein Clientprogramm nutzt die Serverdienste, indem es über festgelegte Protokolle mit dem Serverprogramm kommuniziert.

**Protokoll** Ein Protokoll für Netzwerkdienste (Netzwerkprotokoll) ist eine Beschreibung für die zur Kommunikation zwischen Client und Server notwendigen Formate und Abläufe vom Aufbau der Verbindung, über den Ablauf der Datenübertragung bis zum Verbindungsabbau. Es besteht aus einer syntaktischen Beschreibung der verwendeten Formate, den Regeln zur Kommunikation und einer semantischen Beschreibung des Inhalts. Hierbei können die Inhalte im Datenteil eines Netzwerkprotokolls wiederum durch ein anderes Protokoll beschrieben werden, Protokolle können somit ineinander verschachtelt werden. Dies wird am ISO/OSI-Referenzmodell deutlich, dem Protokollstack, der für die Übertragung in Netzwerken angewendet wird. Eine Übertragung erfolgt über mehreren Ebenen mit den jeweils zur Ebene gehörenden Protokollen. Diese Aufsplittung dient der Vereinfachung eines komplexen Ablaufes, da man sich bei dem Protokoll einer Ebene auf die vorgesehene Funktionalität konzentrieren kann [KR07, Tan96].

## 2.2 Kryptographie

Kryptographie, die Wissenschaft von der Ver- und Entschlüsselung von Informationen zur Geheimhaltung gegenüber Dritten liefert im Umfeld der Computer- und Netztechnik eine Grundlage zur Schaffung der Daten- bzw. Informationssicherheit. Ihre Werkzeuge ermöglichen die Wahrung der Vertraulichkeit und Integrität und dienen zum Nachweis der Authentizität gespeicherter und übertragener Informationen.
Im Rahmen dieser Arbeit werden Authentifizierungsverfahren betrachtet, die auf Anwendung kryptographischer Funktionen beruhen. Die kryptographischen Verfahren werden im Folgenden insofern vorgestellt, wie es für die Erklärung der Authentifizierung und der sich daraus ergebenden Möglichkeiten erforderlich ist.

### 2.2.1 Symmetrische Verfahren

Symmetrische kryptographische Verfahren basieren auf der Nutzung eines gemeinsamen Schlüssels, der allen an der Kommunikation beteiligten Parteien bekannt sein muss.

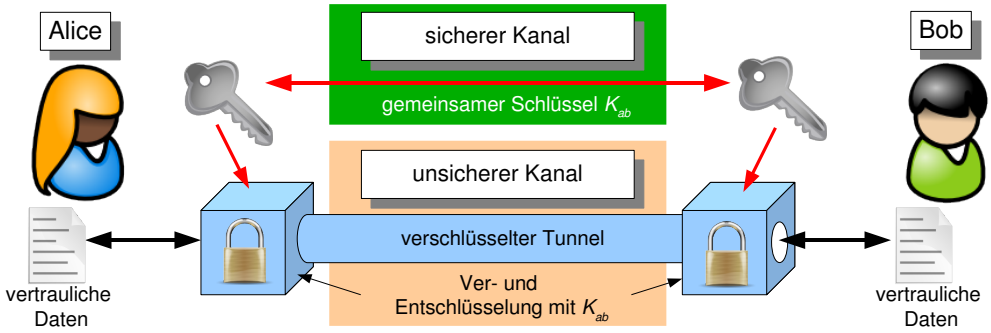



**Abb. 2.1 —** Prinzip der Datenübertragung mit symmetrischer Verschlüsselung

Abbildung 2.1 zeigt den Ablauf am Beispiel der gesicherten Übermittlung von Informationen zwischen Alice und Bob über einen unsicheren Kanal. Beide benötigen hierzu einen gemeinsamen, nur ihnen bekannten geheimen Wert, den Schlüssel $K_{ab}$, der im Vorfeld über einen gesicherten Kanal zwischen den Parteien übermittelt werden muss. Mittels dieses Schlüssels können die zu übertragenden Information vom Sender verschlüsselt und vom Empfänger entschlüsselt werden.

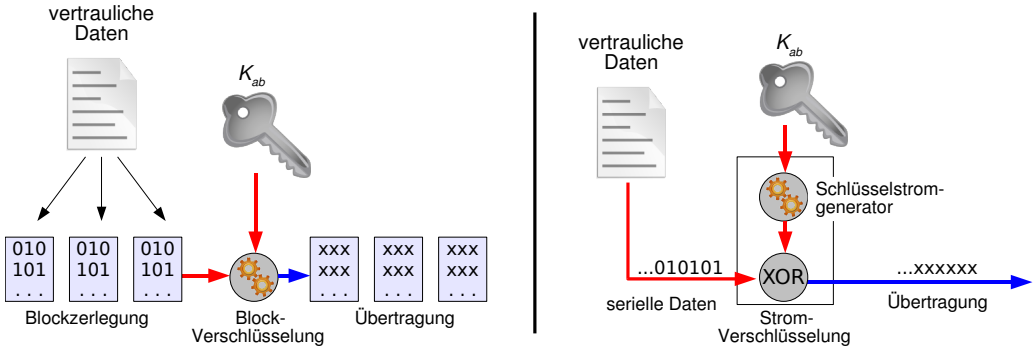



**Abb. 2.2 —** **Blockchiffre**: die Daten werden in einzelne Blöcke zerlegt und jeder Block als eine Einheit verschlüsselt, wobei eine Verkettung der Blöcke möglich ist. **Stromchiffre**: die Daten werden als serieller Strom ohne Blockbegrenzung durch eine XOR-Verknüpfung mit einem pseudozufälligen Schlüsselstrom verschlüsselt.

Man unterscheidet zwischen Block- und Stromchiffren. Bei Blockchiffren werden Datenblöcke mit bestimmter Länge unter Anwendung des Schlüssels verarbeitet. Sind mehr Informationen zu verarbeiten, so ist diese Operation mehrmals auszuführen. Um zu vermeiden, dass gleiche Eingaben zu gleichen Ergebnisblöcken führen, werden nachfolgende Blöcke oder der verwendete Schlüssel durch eine Verknüpfung mit dem Ergebnis der vorhergehenden Blockverschlüsselung verarbeitet [BSN05]. Ver- und Entschlüsselung basieren auf der Nutzung von Operationen, bei denen die ursprüngliche Information durch die Abfolge dieser Operationen dermaßen verändert wird, dass eine Rückgewinnung nur durch Umkehrung der einzelnen Schritte möglich sein sollte. Die Schritte sind für das jeweilige Verfahren bekannt, die Sicherheit hängt somit nur vom Schlüssel ab, der als Parameter einfließt.
Viele eingesetzte Verfahren auf Blockebene sind Varianten der Feistelchiffre, die unter Nutzung einer

Transformationsfunktion mit Substitution und Permutation auf Bitebene und Schlüsseladdition (XOR) in mehreren Runden arbeiten. Eine Umkehrung, also die Entschlüsselung ist dabei mittels der gleichen Transformationsfunktion bei Kenntnis des Schlüssels einfach möglich [Mil03]. Bekannte und aktuell eingesetzte Vertreter sind 3DES (*Triple-DES*, eine Erweiterung des ***D**ata **E**ncryption **S**tandard*-Verfahrens) [BSN05], Blowfish [Sch93], Twofish [SKW$^+$00] sowie RC5 [Riv97] und RC6 [RRSY98, Han05]. Das heute am häufigsten eingesetzte symmetrische Verfahren AES (***A**dvanced **E**ncryption **S**tandard*) arbeitet zwar nicht auf Basis einer Feistelchiffre, nutzt aber ebenfalls mehrere Runden mit Substitutions- und Permutations-Operationen und die XOR-Schlüsseladdition [DR02].

Stromchiffre hingegen verschlüsseln die Daten, indem sie den Klartext-Datenstrom mit einem Schlüsselstrom derart verknüpfen, dass aus dem entstehenden chiffrierten Strom bei Kenntnis des Schlüsselstroms wieder der ursprüngliche Klartext erzeugt werden kann. Eine XOR-Verknüpfung zwischen Daten- und Schlüsselstrom ermöglicht dies. Der gleiche Schlüsselstrom muss dann sowohl bei der Verschlüsselung, als auch bei der Entschlüsselung vorliegen. Dazu dient ein Generator, der im Allgemeinen bekannt ist. Die Sicherheit des Verfahrens beruht also nicht auf der Geheimhaltung des Algorithmus. Deshalb muss der Generator in der Lage sein, mit bestimmten Startparametern, meist sind dies ein offen übertragener Initialisierungsvektor und ein aus dem geheimen Schlüssel gewonnener Wert, eine pseudozufällige Bitfolge zu erzeugen. Diese darf ohne Kenntnis des Schlüssels nur derart schwer rekonstruierbar sein, dass sie praktisch als unvorhersagbar gilt. Das bedeutet auch, dass einem Angreifer, dem sowohl ein Klartextbereich als auch der zugehörige chiffrierte Bereich bekannt ist, aus dem hieraus gewonnenen Teil des Schlüsselstroms keine Voraussagen über den weiteren Schlüsselstrom gewinnen darf oder gar die Startparameter bzw. den Schlüssel rekonstruieren kann. So führten Schwächen im Entwurf der in den ersten WLAN-Netzen eingesetzten WEP-Verschlüsselung (***W**ired **E**quivalent **P**rivacy*) dazu, dass der Schlüssel für den eingesetzten RC4-Stromchiffre-Algorithmus mit geringem Aufwand aus „abgehörten“ Übertragungen gewonnen werden konnte [TWP07].

Bekannte Stromchiffren bzw. Generatoren von Schlüsselströmen sind die in GSM-Mobilfunknetzen eingesetzten A5/1 und A5/2-Algorithmen [BD00], die aber mit der Einführung vom UMTS durch Blockverschlüsselung ersetzt wurden, sowie der bereits genannte RC4-Generator [Fon05]. RC4 wird auch im WEP-Nachfolger WPA (**W**i-Fi **P**rotected **A**ccess) eingesetzt. Dessen Nachfolger WPA2 nutzt dagegen das Blockchiffre-Verfahren AES [IEE04a].

Prinzipiell lässt sich aus einer Blockchiffre eine Stromchiffre generieren, indem die Verschlüsselung eines Initialisierungsvektors (IV) mittels des Schlüssels als Basis für den Schlüsselstrom genutzt wird. Der entstehende Ausgabeblock ist der erste Teil des Schlüsselstroms. Im nächsten Schritt wird der IV durch eine Rückkopplung der Ausgabe (Output Feedback Mode) oder des erzeugten Ergebnisses nach der XOR-Verknüpfung (Cipher Feedback Mode) erneut mittels des symmetrischen Verfahrens verschlüsselt [BSN05]. Diese Rückkopplung ermöglicht einen kontinuierlichen Schlüsselstrom.

Symmetrischen Strom- und Blockchiffren gemein ist, dass zur Ver- und Entschlüsselung der gleiche Schlüssel bzw. leicht daraus abzuleitende Schlüssel genutzt werden. Der gemeinsame Schlüssel $K_{ab}$ muss deshalb, wie in Abbildung 2.1 dargestellt, über einen sicheren Kanal übertragen werden. Dies kann natürlich im Vorfeld erfolgen, z.B. durch Austausch über einen Datenträger bei einem früheren Treffen zwischen Alice und Bob.

Problematisch erweist sich der Schlüsseltausch, wenn der Schlüssel spontan den Kommunikationspartnern bereitgestellt werden soll. Man bräuchte dann einen gesicherten Online-Kanal zur Schlüsselübertragung. Hier muss man sich aber die Frage stellen, wenn ein derartiger Kanal existiert, warum man diesen nicht gleich zur Übertragung der Daten nutzt. Im Normalfall gibt es aber keinen gesicherten Online-Kanal, so dass sich die Frage erübrigt. Der Austausch bzw. die Verteilung neuer Schlüssel stellt somit eine Herausforderung bei der Verwendung symmetrischer Verfahren dar.

### 2.2.2 Asymmetrische Verfahren

Die praktische Sicherheit kryptographischer Systeme hängt von der Frage ab, mit welchem Aufwand eine erfolgreiche Kryptoanalyse zu leisten ist [Eck06]. Außerdem sind mögliche *Real World* Schwachstellen zu bedenken, die ohne „Brechen“ der Verschlüsselung den unberechtigten Zugriff auf die vertraulichen Daten erlauben. Die Sicherheit symmetrischer Verfahren basiert deshalb, abgesehen von der praktischen Sicherheit des Verfahrens an sich, auf der Lösung des Schlüsselverteilungsproblems.
Demgegenüber arbeiten asymmetrische Verfahren mit verschiedenen Schlüsseln zur Ver- und Entschlüsselung, wie in Abbildung 2.3 gezeigt.

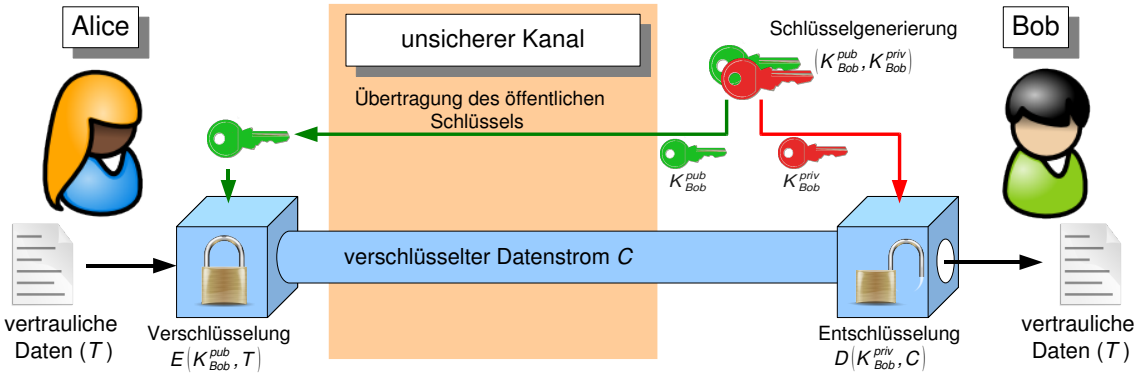



**Abb. 2.3 —** Prinzip der asymmetrischen Datenverschlüsselung

In diesem Beispiel erstellt Bob ein Schlüsselpaar $(K^{pub}, K^{priv})$ und nutzt ein Verfahren mit der Eigenschaft, dass Daten, die mit $K^{pub}$ verschlüsselt werden, nur unter Zuhilfenahme von $K^{priv}$, aber nicht mit $K^{pub}$ allein wieder entschlüsselt werden können. $K^{priv}$ darf sich natürlich nicht einfach aus $K^{pub}$ berechnen lassen. Es muss also gelten, dass ein Klartext $T$ durch die Anwendung des Verschlüsselungsverfahrens $E$ (engl. für *Encryption*) in den chiffrierten Text $C$ überführt werden kann: $C = E(K^{pub}, T)$. Mittels einer weiteren Funktion $D$ (engl. für *Decryption*) kann unter Zuhilfenahme des privaten Schlüssels $K^{priv}$ wieder der Klartext $T$ aus $C$ zurückgewonnen werden: $T = D(K^{priv}, C)$.
Gibt es ein Verfahren mit hinreichend vielen Schlüsselpaaren, welche diese Eigenschaft erfüllen, so kann Bob den generierten öffentlichen Schlüssel $K^{pub}$ (engl. *public key*) beliebig verteilen, insbesondere auch über ungeschützte Kanäle. Wenn Alice eine vertrauliche Nachricht versenden möchte, die nur Bob lesen darf, so kann sie sich einfach den öffentlichen Schlüssel von Bob „besorgen“, die Nachricht verschlüsseln und das Ergebnis zu Bob übertragen. Da nur Bob über den geheimen Schlüssel $K^{priv}$ (engl. *private key*) verfügt, kann nur er diese Nachricht entschlüsseln. Ein mitlesender Angreifer, der nicht im Besitz von $K^{priv}$ ist, bleibt außen vor [BSN05].
Ein Schlüssel $K^{pub}$ kann beliebig verteilt werden. Darauf aufbauende Kryptosysteme bezeichnet man als Public-Key-Verfahren. Diese erleichtern das Schlüsselverteilungsproblem. So können zum Einen die Schlüssel über öffentliche Kanäle übertragen werden, anderseits werden bei mehr als drei Kommunikationsteilnehmern insgesamt weniger Schlüssel benötigt. Das beschriebene Verfahren ist nur einseitig, also von Alice zu Bob skizziert. Für die umgekehrte Kommunikationsrichtung muss Alice ebenfalls ein Schlüsselpaar generieren und ihren öffentlichen Schlüssel an Bob übertragen. Bei $n$ Teilnehmern wären jedoch insgesamt nur $n$ Schlüsselpaare für beliebige bidirektionale Kommunikation erforderlich. Würde man dies von einem symmetrischen Verfahren verlangen, so müssten $n \cdot (n-1)/2$ Schlüssel erstellt und sicher übermittelt werden.

Die Schlüsselverteilung kann bei asymmetrischen Verfahren über unsichere Kanäle erfolgen, ist jedoch nicht gegen das Einschleusen eines Schlüssels abgesichert. Zertifikate können für die Gültigkeitsprüfung der Schlüssel genutzt werden (Kapitel 2.3). Dazu werden elektronische Signaturen benötigt.

### Elektronische Signaturen

Gilt für ein asymmetrisches Verfahren $D, E$ und für die Schlüsselpaare $(K^{pub}, K^{priv})$:

$$\forall T : D(K^{priv},\ E(K^{pub}, T)) = T \ \text{ und } \ E(K^{pub},\ D(K^{priv}, T)) = T, \tag{2.1}$$

dann eignet sich das Verfahren neben der Verschlüsselung auch zur Erstellung elektronischer Signaturen [Eck06]. Die Eigenschaft besagt, dass die Anwendung der Kodierungsfunktionen $D$ und $E$ getauscht werden kann. Was mit dem einen Schlüssel kodiert wird, kann mit dem jeweils anderen Schlüssel dekodiert werden. Für die Signatur braucht man diese Eigenschaft im umgekehrten Sinne zur Verschlüsselung. Etwas, das mit einem privaten Schlüssel kodiert wurde, kann mit dem zugehörigen öffentlichen Schlüssel dekodiert und damit geprüft werden. Da im Gegensatz zum öffentlichen Schlüssel nur der Inhaber Zugriff auf den privaten Schlüssel haben sollte, kann man durch Dekodierung mit dem öffentlichen Schlüssel prüfen, ob der Inhaber das Dokument „kodiert“ hat. Dies entspricht einer Unterschrift.
In Abbildung 2.3 besitzt Alice den öffentlichen Schlüssel von Bob. Insofern sich Alice sicher ist, dass es sich bei $K^{pub}_{Bob}$ um den öffentlichen Schlüssel von Bob handelt, kann Alice prüfen, ob ein übertragener Text von Bob verfasst wurde. Hierzu müsste Bob, wie in Abbildung 2.4 skizziert, den Text $T$ elektronisch signieren, indem er $S = D(K^{priv}_{Bob}, T)$ berechnet und diese Signatur zusammen mit dem Text überträgt $(T, S)$[1].

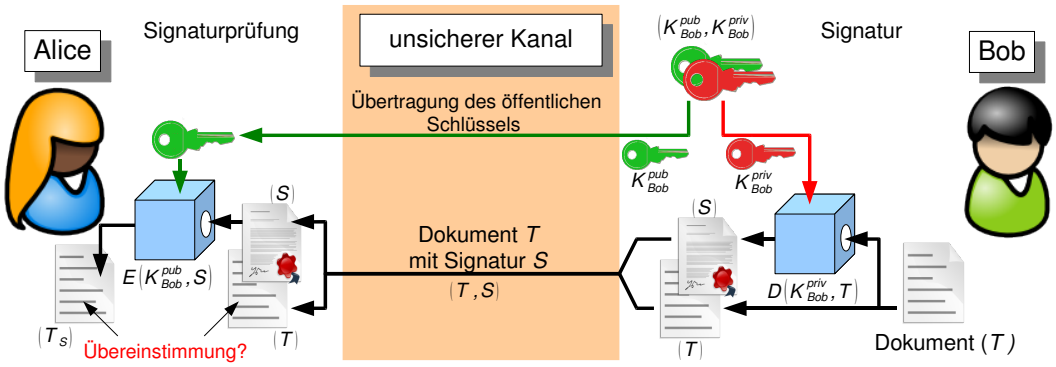



**Abb. 2.4 —** Signatur mit asymmetrischen Verfahren

Alice berechnet mit der Signatur $S$ und dem öffentlichen Schlüssel von Bob $T_S = E(K^{pub}_{Bob}, S)$ und vergleicht anschließend $T$ mit $T_S$. Insofern beide gleich sind, kann Alice davon ausgehen, dass der ursprüngliche Text mit dem geheimen privaten Schlüssel von Bob, den nur Bob besitzen sollte, signiert wurde und somit authentisch ist. Die Integrität wird gewährt, da eine nicht durch Bob erfolgte Änderung am Dokument über die Abweichung der Signatur erkannt wird.
Handelt es sich bei dem Text um ein relativ langes Dokument, erhält man eine „große“ Signatur, deren Berechnung sehr aufwendig sein kann. Außerdem ermöglicht man, falls das ganze Dokument nicht

[1] Die Signatur $S$ kann auch getrennt vom Text übertragen oder verwahrt werden, wenn eine spätere Zuordnung geklärt ist.

verschlüsselt wurde, einem Angreifer die Zuordnung zwischen Klartext und einem verschlüsselten Text, da die Signatur ja unter Verwendung des privaten Schlüssels erzeugt wurde. Dies liefert dem Angreifer die Mittel einer *Known-Plaintext* Analyse (Kapitel 8.2.1), welche möglicherweise Rückschlüsse auf den geheimen Schlüssel erlaubt. Ein einfacher Ausweg besteht darin, nicht das Dokument $T$ zu signieren, sondern einen Hashwert $H(T)$ zu bilden und diesen zu signieren: $D(K_{Bob}^{priv}, H(T))$. Hashwerte sind in ihrer Länge begrenzt und meist kürzer als das Dokument. Verwendet man eine Hashfunktion, für die es praktisch unmöglich ist, den gleichen Hashwert bei einer „sinnvoll“ veränderten Eingabe zu erzeugen, so bleibt Integrität und Authentizität gewahrt (siehe Kapitel 4.2.1). Der Angreifer kann den Dokumentinhalt nicht in seinem Sinne verändern, da er somit einen neuen Hashwert erhält und ohne Besitz des privaten Schlüssels keine gültige Signatur erzeugen kann.

### Das RSA-Verfahren

In dieser Arbeit werden Hardware-Token verwendet. Hierbei handelt es sich um Mikrorechner in kompakter Bauform, z.B. als Chipkarten oder USB-Sticks, deren Prozessoren für die Ausführung kryptographischer Operationen ausgelegt sind. Für das asymmetrische Prinzip existieren mehrere Verfahren (DSS - Digital Signature Standard [DSS09], ElGamal [Elg85] oder Elliptic Curve Algorithmen [Mil85] wie ECDSA [JMV01, X9.62]), welche von den Token unterschiedlich unterstützt werden. Ein Verfahren, das die meisten Token beherrschen, ist das RSA-Verfahren, welches hier in Kürze erläutert werden soll.
Das RSA-Verfahren (***R**ivest-**S**hamir-**A**dleman*), benannt nach den Entwicklern, ist ein asymmetrisches Verfahren, für das die gewünschte Signatureigenschaft (2.1) gilt [PKCS1]. Die Operationen verwenden sogar die gleiche Funktion, da für einen Schlüssel $K^x$ aus einem Schlüsselpaar $(K^{pub}, K^{priv})$ gilt: $E(K^x, T) = D(K^x, T)$. Nur der Einsatz des jeweiligen Schlüssels $K^x$ entscheidet, ob es sich um eine Verschlüsselung ($x = pub$) oder eine Entschlüsselung bzw. Signatur ($x = priv$) auf der Eingabe $T$ handelt. Wir bezeichnen die RSA-Funktion im Folgenden als $\mathrm{RSA}(K, T)$.
RSA basiert auf modularer Arithmetik [BSW06]. Eine Eigenschaft dieser Arithmetik besagt, dass für das Produkt $N = p \cdot q$ zweier Primzahlen $p$ und $q$ für alle natürlichen Zahlen $k$ und $m$ mit $m \leq N$ gilt:

$$m^{k\cdot(p-1)\cdot(q-1)+1} \bmod N = m \tag{2.2}$$

Wenn man den Exponenten über das Produkt zweier natürlicher Zahlen $e$ und $d$ darstellt

$$e \cdot d = k \cdot (p-1) \cdot (q-1) + 1 \quad \text{mit } k \in \mathbb{N} \tag{2.3}$$

dann gilt

$$m^{e\cdot d} \bmod N = (m^e)^d \bmod N = m \quad \text{bzw.} \quad (m^d)^e \bmod N = m \tag{2.4}$$

Die Werte $e$ und $d$ bilden jeweils einen Teil des RSA-Schlüssels. Kennt man nur den Modulus $N$ und einen Teil des Schlüssels, so kann der andere schwer berechnet werden, da man dafür $p-1$ und $q-1$ nach (2.3) bräuchte. Um diese Werte zu erhalten, müsste man $N$ in seine Faktoren $p$ und $q$ zerlegen, was für große Zahlen relativ schwer ist.
Zur Generierung eines RSA-Schlüsselpaares benötigt man somit zwei „große“ Primzahlen, wobei darauf zu achten ist, dass diese gewissen Bedingungen genügen [Eck06]. So sollten sie keinesfalls benachbart sein und sich in ihrer Länge um mehrere Stellen unterscheiden. Um vor aktuellen Angriffen sicher zu sein, sollten die Werte derart groß gewählt werden, dass eine Primfaktorisierung mit aktuellen bzw. absehbar verfügbaren Hardware-Ressourcen und den bekannten Algorithmen nicht in akzeptabler Zeit möglich ist. Schwierigkeiten kann es deshalb bereits bei der Bestimmung derart großer Primzahlen geben [Lis05, AKS04].

Wurden die beiden Zahlen $p$ und $q$ generiert, berechnet man hieraus das Produkt $N = p \cdot q$ sowie die eulersche Funktion $\varphi(N)$, welche die Anzahl der Werte kleiner $N$ liefert, die teilerfremd zu $N$ sind. Da es sich um Primzahlen handelt, gilt

$$\varphi(p) = p - 1, \quad \phi(q) = q - 1 \text{ und auch } \varphi(N) = (p-1) \cdot (q-1)$$

Über $\varphi(N)$ lassen sich die beiden Werte $e$ und $d$ berechnen. Hierzu wählt man eine natürliche Zahl $e$ mit $1 < e < \varphi(N)$, die teilerfremd zu $\varphi(N)$ ist - eine Primzahl bietet sich an - und ermittelt den zugehörigen Wert $d$, so dass Gleichung (2.3) erfüllt ist. Dazu muss gelten

$$e \cdot d \bmod \varphi(N) = 1.$$

Der Wert von $d$ lässt sich durch Lösung der diophantischen Gleichung $e \cdot d + k \cdot \varphi(N) = 1$ für $k \in \mathbb{N}$ mittels des erweiterten euklidischen Algorithmus ermitteln. Die Kombination $(e, N)$ bildet einen Teil des Schlüssels, $(d, N)$ den anderen. Da beide Schlüssel „gleichwertig“ sind, soll $(e, N)$ den öffentlichen Schlüssel darstellen ($e$ für *E*ncryption), der an beliebige Teilnehmer verteilt werden kann.

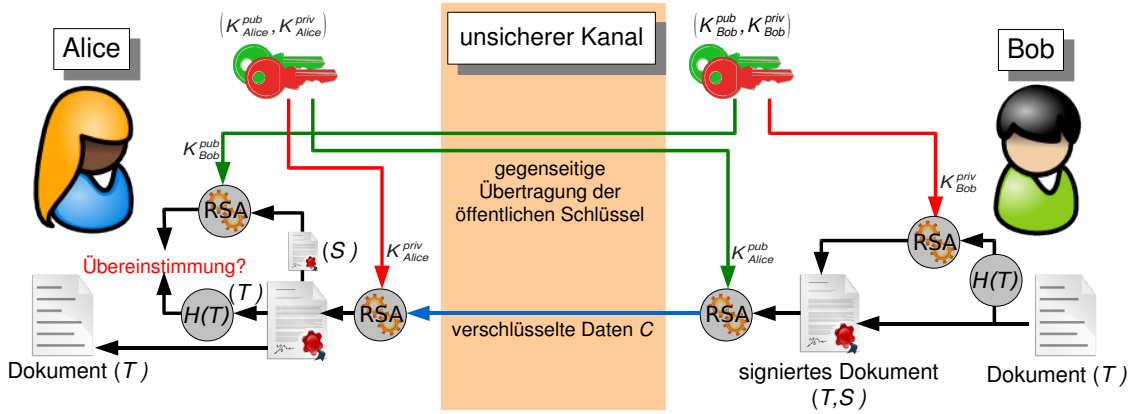



**Abb. 2.5 —** Verwendung des RSA-Verfahrens für Signatur und Verschlüsselung

In Abbildung 2.5 übersenden sich Alice und Bob gegenseitig ihre öffentlichen Schlüssel. Bob kann daraufhin eine geheime Nachricht $T$ an Alice übertragen. Um die Integrität der Nachricht zu gewährleisten und die Authentizität des Absenders zu garantieren, erzeugt er zuerst eine elektronische Signatur $S$, die er mit seinem geheimen Schlüssel aus dem Hashwert $H(T)$ der Nachricht durch Berechnung von

$$S = \mathrm{RSA}(K^{priv}_{Bob}, H(T)) = H(T)^{d_{Bob}} \bmod N_{Bob}$$

ermittelt. Anschließend verschlüsselt er die konkatenierten Werte $(T, S)$ mit dem öffentlichen Schlüssel von Alice

$$C = \mathrm{RSA}(K^{pub}_{Alice}, (T, S)) = (T, S)^{e_{Alice}} \bmod N_{Alice}$$

und überträgt das Ergebnis an Alice. Dabei wird $(T, S)$ als ein numerischer Wert $x$ aufgefasst. Wegen (2.2) muss gelten $x \leq N_{Alice}$, um die spätere Entschlüsselung zu ermöglichen. Andernfalls muss $(T, S)$ zerlegt und in mehreren Teilen verschlüsselt werden.
Alice kann den Inhalt entschlüsseln. Dazu berechnet Alice nach (2.4) mit ihrem geheimen Schlüssel

$$\mathrm{RSA}(K^{priv}_{Alice}, C) = C^{d_{Alice}} \bmod N_{Alice} = ((T, S)^{e_{Alice}})^{d_{Alice}} \bmod N_{Alice} = (T, S)$$

und erhält die entschlüsselte Nachricht $T$ und die Signatur $S$. Anschließend prüft Alice die Authentizität. Analog zur Entschlüsselung nach Gleichung (2.4) berechnet sie auf der Signatur $S$ mit Bobs öffentlichem Schlüssel

$$\mathrm{RSA}(K_{Bob}^{pub}, S) = S^{e_{Bob}} \bmod N_{Bob} = (H(T)^{d_{Bob}})^{e_{Bob}} \bmod N_{Bob} = H(T)$$

und erhält damit den von Bob als Signatur hinterlegten Hashwert $H(T)$. Anschließend bildet Alice aus dem entschlüsseltem Text $T$ mit dem gleichen Hashverfahren ebenfalls einen Hashwert $H_1(T)$ und vergleicht $H(T)$ und $H_1(T)$. Stimmen die Hashwerte überein, so kann Alice davon ausgehen, dass ihr die unveränderte Nachricht des Urhebers Bob vorliegt. Dies gilt aber nur, insofern Alice auch sicher sein kann, dass es sich bei $K_{Bob}^{pub}$ tatsächlich um den öffentlichen Schlüssel von Bob handelt.
Empfehlungen zur Implementierung und Verwendung des RSA-Verfahrens inklusive erforderlicher Sicherheitsmaßnahmen liefert die PKCS#1-Spezifikation [PKCS1]. Diese von der Firma RSA Inc.[2] vorgestellten „Public Key Cryptograhic Standards“ (PKCS) umfassen eine Reihe von Normen im kryptographischen Umfeld, von denen weitere in den Kapiteln 5.1.4 und 5.2.2 verwendet werden.

### Diffie-Hellman-Merkle-Schlüsselvereinbarung

Einen Spezialfall, der zu den asymmetrischen Verfahren gerechnet wird, stellt die Diffie-Hellman-Merkle-Schlüsselvereinbarung dar. Es handelt sich hierbei nicht um die Verschlüsselung einer zu übertragenden Nettoinformation, sondern um die Vereinbarung eines gemeinsamen geheimen Wertes. Dieses Verfahren wird auch Diffie-Hellman-Schlüsseltausch genannt, obwohl kein Schlüssel ausgetauscht wird. Vielmehr kann dieses Verfahren dazu genutzt werden, um sich über einen unsicheren Übertragungskanal auf ein gemeinsames Geheimnis zu einigen, welches vor einem Angreifer, der die Kommunikation im Kanal mitlesen, aber nicht verändern kann, verborgen bleibt.
In Verbindung mit Token nutzen Applikationen diese Schlüsselvereinbarung gern für die schnelle Generierung eines gemeinsamen Sitzungsschlüssels. Im Token wird lediglich die RSA-Signatur zur Echtheitsbestätigung berechnet.

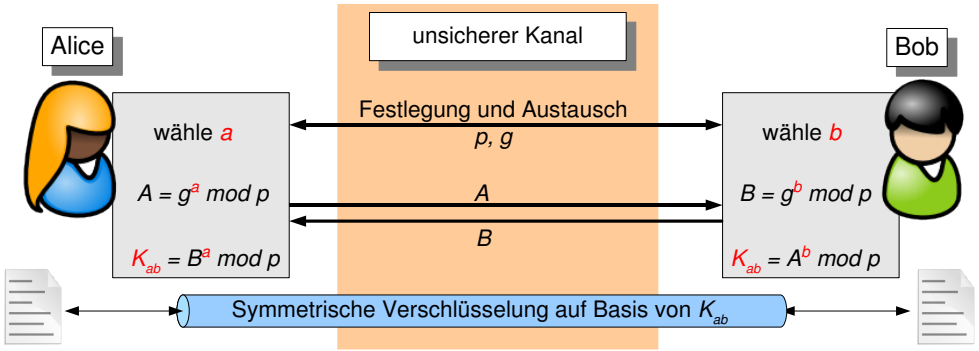



**Abb. 2.6 —** Diffie-Hellman-Merkle Schlüsselvereinbarung

Beide Parteien (Alice und Bob) benötigen hierzu gemeinsam bekannte, nicht geheime Werte $p$ und $g$, wobei $p$ eine „hinreichend“ große Primzahl und $g$ eine Primitivwurzel dieser Primzahl sein muss. Außerdem benötigen Alice und Bob jeweils einen geheim zu haltenden, zufällig gewählten Wert $a$ bzw. $b$ kleiner als $p - 2$. Nach Berechnung von

$$A = g^a \bmod p \quad \text{und} \quad B = g^b \bmod p$$

[2]Tochtergesellschaft der EMC Corporation, entstanden nach diversen Übernahmen aus der Firma RSA Data Security, die im Jahre 1982 von den Entwicklern des Verfahrens gegründet wurde.

auf der jeweiligen Seite und Austausch von $A$ und $B$ über den unsicheren Kanal, können beide Seiten den gemeinsamen Wert

$$\begin{aligned} K_{ab} &:= B^a \bmod p \\ &= (g^b \bmod p)^a \bmod p = g^{ba} \bmod p = (g^a \bmod p)^b \bmod p \\ &= A^b \bmod p \end{aligned}$$

berechnen. Der Wert $K_{ab}$ kann anschließend als Schlüssel bzw. als Basis zur Generierung eines symmetrischen Schlüssels dienen, so wie dies für das *PreMasterSecret* im SSL/TLS-Standard (siehe Kapitel 4.3.2) vorgesehen ist.
In diesem Falle kann man von der Verbindung eines asymmetrischen Verfahrens (beide Teilnehmer hatten unterschiedliche geheime Werte, die anderen gemeinsamen Werte waren öffentlich) mit einem symmetrischen Verfahren sprechen und erhält somit ein hybrides Verfahren.

### 2.2.3 Hybride Verfahren

Neben der Diffie-Hellman-Merkle-Schlüsselvereinbarung in Verbindung mit einem symmetrischen Verfahren lassen sich weitere hybride Varianten finden. So besteht die Möglichkeit der direkten Verwendung eines asymmetrischen Verfahrens zum Austausch eines symmetrischen Schlüssels, kombiniert mit anschließender symmetrischer Verschlüsselung.

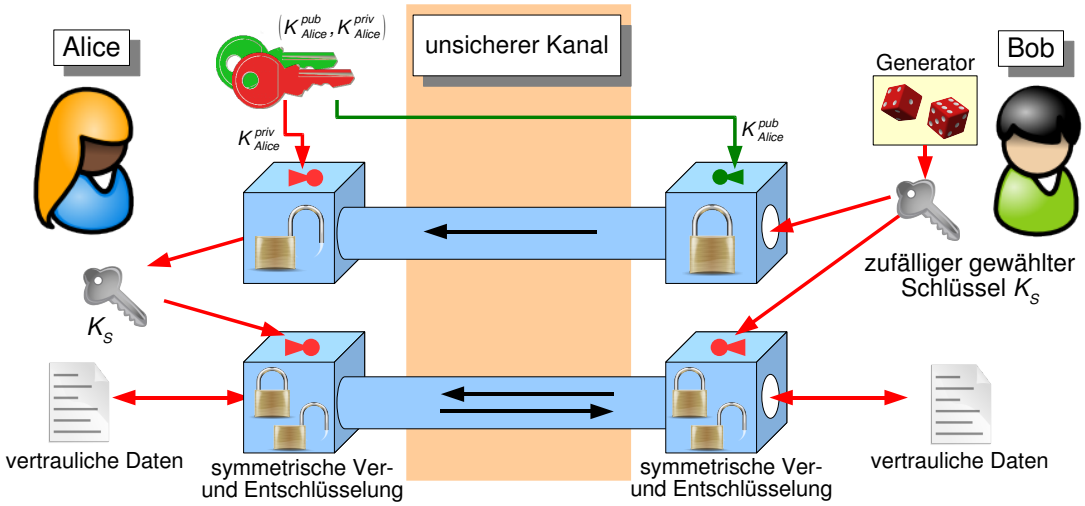



**Abb. 2.7 —** Hybride Verschlüsselung

Abbildung 2.7 verdeutlicht die Funktion. Die asymmetrischer Verschlüsselung ermöglicht es, mit dem öffentlichen Schlüssel des Kommunikationspartners einen erzeugten zufälligen Schlüssel $K_S$ für die Verwendung in einem symmetrischen Verfahren zu verschlüsseln und diesen sicher zu übertragen. Diese Funktion lässt sich noch erweitern, indem der Sender in Verbindung mit dem eigenen privaten Schlüssel durch eine Signatur des symmetrischen Schlüssels analog Abbildung 2.4 die Urheberschaft des erzeugten Schlüssels $K_S$ bescheinigt.
Ein wesentlicher Vorteil hybrider Verfahren ist deren hohe Geschwindigkeit durch Verwendung symmetrischer Verfahren zur Verschlüsselung der Nettoinformation unter gleichzeitiger Vermeidung des diesen Verfahren anhaftenden Schlüsseltauschproblems.

## 2.3 Vertrauensnetze, Zertifikate und PKI

Während der Übertragung im unsicheren Transportkanal können Daten durch einen Angreifer, der Kontrolle über den Kanal erlangt, manipuliert werden kann. Insbesondere kann dieser Angreifer den Schlüsseltausch oder die Schlüsselvereinbarung derart beeinflussen, dass er selber zum Kommunikationspartner beider Parteien wird. Die Daten würden somit von den Parteien verschlüsselt zum Angreifer übertragen, welcher sie mitlesen, für die jeweils andere Partei erneut verschlüsseln und an diese weiterleiten kann, ohne dass eine Manipulation auffallen würde. Dies wäre eine Form eines *Man-In-The-Middle-Angriffs* (siehe Kapitel 8.2.1). Deshalb sind Vorkehrungen zu treffen, um die Authentizität des Kommunikationspartners prüfen zu können.
Es existieren verschiedene Lösungen. Diese beruhen aber zumeist auf der Prüfung einer digitalen Signatur der Schlüssel. So kann in einem hybriden Verfahren (Abbildung 2.7) der für die symmetrische Verschlüsselung genutzte Schlüssel $K_S$ mit dem privaten Schlüssel von Bob vor der Übertragung signiert werden. Allerdings hilft dies nicht weiter, da Alice dem öffentlichen Schlüssel $K^{pub}_{Bob}$ von Bob nicht trauen kann. Umgekehrt muss Bob ja auch dem öffentlichen Schlüssel $K^{pub}_{Alice}$ von Alice vertrauen, bevor er diesen zur Verschlüsselung nutzt. Demzufolge wird eine Bestätigung der Authentizität öffentlicher Schlüssel benötigt. Dazu gibt es zwei Ansätze.

### 2.3.1 Vertrauensmodelle

Beide Ansätze basieren darauf, dass ein Schlüssel fest zu einer Person (oder Maschine) zugeordnet werden kann. Diese Person muss eindeutig spezifiziert werden, z.B. anhand einer eindeutigen ID. Im einfachsten Falle nutzt man hierzu eine Emailadresse, die nur der betreffenden Person zugeordnet ist. Diese Verknüpfung der ID mit dem Schlüssel wird elektronisch signiert. Die Frage, wer die Signatur ausstellt, liefert zwei Vertrauensmodelle [Sch07a], das dezentrale Web/Net-Of-Trust- und das hierarchische Modell.

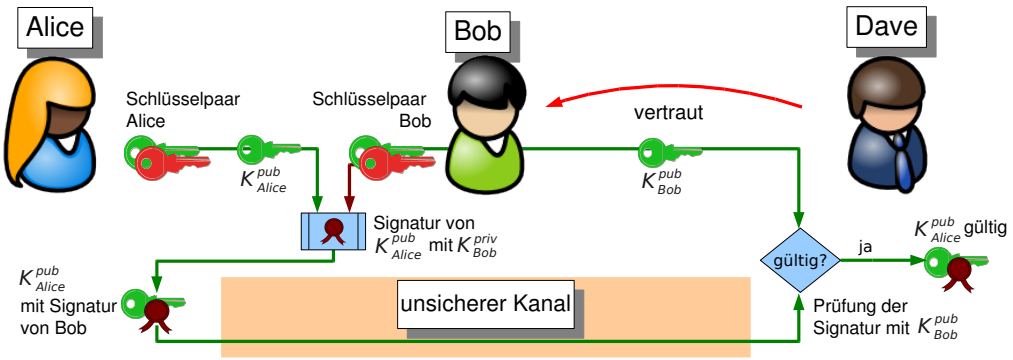



**Abb. 2.8 —** Net-of-Trust: Strukturen auf Basis gegenseitigen Vertrauens

**Web/Net-Of-Trust:** Die Idee besteht darin, dass Nutzer mit ihrem eigenen privaten Schlüssel die Schlüssel-ID-Kombinationen anderer Nutzer signieren, insofern sie sicher sind, dass es sich hierbei tatsächlich um die Schlüssel der betreffenden Personen handelt (Abbildung 2.8). Dieses Verfahren lässt sich transitiv fortsetzen. Bei bestehendem Vertrauensverhältnis zu anderen Personen können Schlüssel, die von diesen Personen signiert wurden, mit einer vom Vertrauensverhältnis abhängigen Wichtung als authentisch angesehen und genutzt werden. Somit erhält eine Schlüssel/ID-Verknüpfung

eine Menge von Signaturen, die eine Aussage über die Authentizität ermöglichen. Die Entscheidung darüber obliegt allerdings jeweils dem Nutzer.
PGP[3] ist eine Applikation, welches dieses Verfahren nutzt und z.B. zur Verschlüsselung von Emails oder Dokumenten gern verwendet wird. Der Vorteil besteht in der einfachen Erstellung von Schlüsseln in Verbindung mit Signaturen durch viele nahestehende Personen, insofern diese bereits über einen Schlüssel verfügen. Somit kann das System ohne weitere Dienste schnell aufgebaut werden.
Bei diesem Verfahren gibt es keine zentrale Stelle, welche die Schlüssel signiert, sondern es bildet sich ein Netz auf Vertrauensbasis zwischen den Nutzern. Ein Nachteil des Verfahrens ist der Fall, in dem ein privater Schlüssel kompromittiert wurde. Aufgrund der fehlenden zentralen Stelle gibt es keine einfache Lösung, diesen Umstand sämtlichen Teilnehmern im System kurzfristig mitzuteilen.

**Hierarchische Strukturen mit Zertifikaten:** Die Kombination aus öffentlichen Schlüssel und einer eindeutigen ID (als *Subject* bezeichnet) in Verbindung mit einer elektronischen Signatur von einer „vertrauenswürdigen“ Stelle wird als elektronisches Zertifikat bezeichnet. Ein Aussteller bescheinigt die Authentizität der Schlüssel-ID-Kombination eines Antragstellers mit seiner Signatur, nachdem er die Identität des Antragstellers geprüft hat. Insofern ein Nutzer dem Aussteller vertraut, kann er Zertifikate anderer Nutzer prüfen, indem er deren Zertifikats-Signatur mit dem öffentlichen Schlüssel des Ausstellers prüft. Dies erfolgt analog zur Prüfung der Signatur eines Dokuments nach Abbildung 2.4.
Unter dem Begriff **PKI (Public Key Infrastructure)** versteht man strukturierte Systeme auf Basis von Zertifikaten [AL02]. Neben den Zertifikaten gehören die Stellen zur Entgegennahme der Anträge (RA: Registration Authority), die vertrauenswürdigen Stellen, welche im Endeffekt Zertifikate ausstellen (CA: Certificate Authority), die Verwaltung möglicher Zertifikatswiderrufslisten (CRL: Certificate Revocation List) und Stellen zur Prüfung von Zertifikaten zu einer PKI. Eine Policy, an die sich alle Teilnehmer halten müssen, beschreibt Aufbau der PKI und Verwendbarkeit der Zertifikate.

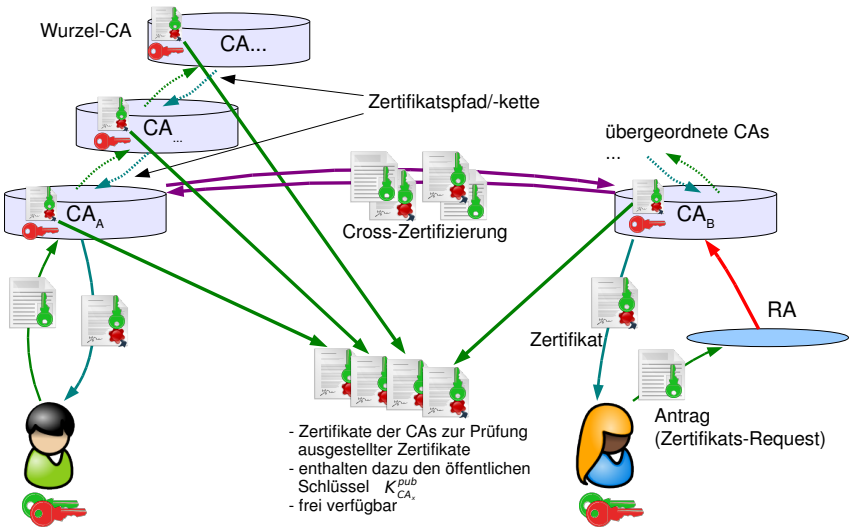



**Abb. 2.9 —** Hierarchischer Aufbau einer PKI

Das gesamte System ergibt somit eine Struktur von Zertifikatsinhabern und Zertifizierungsstellen, die wiederum selber über ein Zertifikat, ausgestellt von einer weiteren Zertifizierungsstelle verfügen

[3] PGP: *Pretty Good Privacy*, vom Entwickler Phil Zimmermann hinsichtlich Nutzerfreundlichkeit ausgelegte Software für Ver- und Entschlüsselung von Daten und zum Erzeugen und Prüfen von Signaturen [Gar96, RFC2440, RFC4880].

können. Diese meist streng hierarchische Verknüpfung ermöglicht Strukturen, wie sie in der Abbildung 2.9 gezeigt werden. Eine CA ist ein Knoten, welcher Zertifikate für die darunter liegende Ebene ausstellt. Die CA muss also die ID/Schlüssel-Authentizität und somit die Identität des Inhabers prüfen, bevor sie das Zertifikat ausstellt. Kann der Nutzer seinen Antrag nicht der CA selber überreichen, so kann er eine RA im Umfeld hinzuziehen. Die RA nimmt Anträge entgegen, prüft deren Authentizität und leitet sie an die CA weiter. Die CA, vorausgesetzt sie vertraut der RA, kann daraufhin die Zertifikate ausstellen.
Der Pfad vom Nutzer bis zur Wurzel dieses Baumes (Root-CA) wird als Validierungspfad oder Zertifikatskette bezeichnet. Insofern ein Nutzer einem der Aussteller im Pfad vertraut, so gelten für ihn alle im Unterbaum enthaltenen Zertifikate als gültig. Er muss hierzu die Kette von Zertifikaten bis zum vertrauenswürdigen Knoten zurück verfolgen können.
Zur Aufbrechung der Baum-Hierarchie besteht die Möglichkeit der Cross-Zertifizierung, bei dem sich Zertifizierungsstellen in gleichen Hierarchieebenen gegenseitig Zertifikate ausstellen, oder die Einbindung von Bridge-CAs, die mehrere Root-CAs verknüpfen. Allerdings wird ohne die strenge Baum-Hierarchie ein Abruf der CA-Zertifikate und damit die Zertifikats-Prüfung aufwendiger. In der in Kapitel 7.2 vorgestellten Software ist eine derartige Prüfung meist nicht implementiert. Als Abhilfe könnten aber Online-Dienste zur Zertifikats-Prüfung herangezogen werden.

### 2.3.2 X.509-Zertifikate

Allgemein gebräuchlich sind Zertifikate nach dem ITU-T[4] Standard X.509, welcher mittlerweile in Version 3 vorliegt [X509]. X.509-Zertifikate enthalten

- eine eindeutige ID des Ausstellers (*Issuer*),
- die ID des Inhabers (*Subject*),
- den öffentlichen Schlüssel des Inhabers,
- einen Gültigkeitszeitraum
- und die Signatur des Ausstellers.

Version 3 ermöglicht darüber hinaus Erweiterungen, z.B. zusätzliche Attribute, welche die Verwendung des mit dem Zertifikat verbundenen Schlüsselpaares festlegen. Diese Festlegung betrifft die Verwendbarkeit zur Verschlüsselung, zur Signatur von Daten oder zur Ausstellung weiterer Zertifikate (z.B. als Zertifikat für eine Sub-CA), zur Nutzung als Personen- oder Maschinenzertifikat uvm. Darüber hinaus besteht die Möglichkeit, weitere Informationen abzulegen. Als Beispiel sei die URI genannt, unter der Zertifikatssperrlisten im Netz abgerufen werden können. Ein Beispielzertifikat mit Erklärung der Inhalte befindet sich im Anhang A.
Für Systeme auf Basis einer dezentralen Vertrauensstruktur, wie z.B. PGP, lassen sich auch Zertifikate ausstellen. Diese Zertifikate können die Signaturen mehrerer Aussteller enthalten [RFC2440, RFC4880]. Sie werden aber meist nicht als Zertifikate, sondern lediglich als PGP-PublicKey-Paket bezeichnet. Im folgenden Text bezieht sich deshalb der Begriff Zertifikat auf die X.509-Variante.

[4] ITU: International Telecommunication Union, ITU-T: Telecommunication Standardization Sector

# Kapitel 3

# Authentifizierung und SSO

Zur Sicherheitsstruktur eines vernetzten Rechnersystems mit Außenanbindung, also mit Zugriff vom und zum Internet gehören sehr viele verschiedene Aspekte, die beim Aufbau und im laufenden Betrieb zu beachten sind. Dieser Umstand wird verschärft, wenn zum Leistungsspektrum des Angebotes die Bereitstellung von Diensten gehört, die auch von außerhalb des abgeschlossenen Systems genutzt werden können. Erlauben Rechnersysteme und deren zugehörige Struktur eine Nutzung der Ressourcen durch verschiedene Personen innerhalb des Systems, so muss auch davon ausgegangen werden, dass eine weitere Gefährdung durch interne Nutzer vorsätzlich oder unwissentlich entstehen kann.
Im Folgenden wird am Beispiel einer Hochschule mit ihrer vielfältigen Nutzerstruktur und mit den teils lose, teils eng gekoppelten Systemen und deren unterschiedlichen Sicherheitsanforderungen analysiert, welche Ansprüche an ein Nutzerauthentifizierungs-System gestellt werden und welche Vorteile dabei eine Single Sign-On Lösung bieten kann. Das Beispiel Hochschule lässt sich gut auf andere Bereiche mit gleichem Sicherheitsbedarf abbilden. So haben viele kleine und mittelständische Unternehmen ähnliche Anforderungen für ihre Rechnernetze, den zur Verfügung gestellten Diensten und deren Zugriffsmöglichkeiten. Eines der Hauptprobleme besteht darin, technisch unversierten Mitarbeitern eine adäquate Authentisierung zu eröffnen und dabei die Möglichkeiten der Personen einzuschränken, die unbewusst oder mutwillig eine Schädigung herbeiführen könnten.
Ein Verfahren zur Authentifizierung mit Single Sign-On muss eine breite Palette möglicher Authentisierungsschnittstellen unterstützen. Es ist näher zu betrachten, wie Nutzer auf angebotene Dienste zugreifen, welche Zugangswege sie nutzen und wo sie sich, falls notwendig, authentisieren müssen.

## 3.1 Struktur der Zielsysteme

Vereinfacht man den Aufbau eines Hochschul-Netzwerkes auf die möglichen Schnittstellen zwischen Nutzern, Rechnern und den angebotenen Diensten, so entsteht die in Abbildung 3.1 skizzierte Struktur. Das Endgerät, an dem der Nutzer arbeitet, wird im Folgenden als *Rechner* bezeichnet. Es handelt sich um einen Computer der zu diesem Zeitpunkt einem Nutzer zugeordnet werden kann. Demgegenüber stehen die *Server*. Dies sind Computer, deren Dienste über ein Rechnernetz abgerufen werden können und meist mehreren Nutzern gleichzeitig zur Verfügung stehen. Eine derart stringente Unterteilung ist oft nicht möglich, da z.B. mehrere *ThinClients*[1] über den gleichen Computer betrieben werden können. Sie dient jedoch zur Verdeutlichung, ob ein Computer an der Nutzerschnittstelle oder als Lieferant für Serverdienste gemeint ist.

[1] Als ThinClient bezeichnet man einfache Endgeräte, deren Funktion im Wesentlichen auf die Ein- und Ausgabe beschränkt ist und die meist nicht autark arbeiten können.

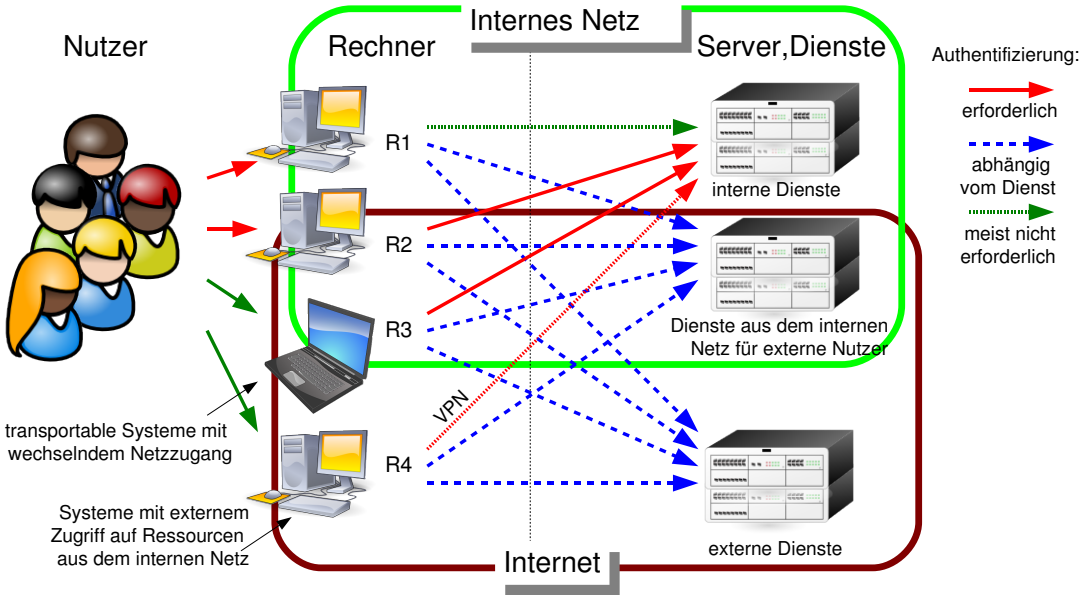



**Abb. 3.1 —** Dienste und Zugangsmöglichkeiten einer universitären Netzwerkstruktur

Die Rechner sind mit den Servern über das Rechnernetz verbunden. Die grobe Struktur des Rechnernetzes kann in zwei sich überlappende Bereiche eingeteilt werden. Der Bereich *Internet* umfasst alle Rechner und angeschlossenen Server, die eine global gültige Internetadresse besitzen und deren Dienste von anderen Rechnern aus dem Internet erreicht und genutzt werden können. Demgegenüber steht das *interne Netz*, welches Rechner, Server und zugehörige Netze der Einrichtung umfasst.
In dem überlappenden Bereich zwischen Internet und internem Netz befinden sich die von der Einrichtung betriebenen Rechner und Server und darauf befindliche Dienste, die über das Internet von außerhalb erreichbar sind. Der Zugriff kann trotzdem durch geeignete Maßnahmen, z.B. Firewalls, eingeschränkt und nur für eine Teilmenge möglicher Dienste erlaubt sein.
Den oberen internen Bereich (siehe Abbildung 3.1) zeichnet aus, dass dort keine Dienste für externe Nutzung angeboten werden. Auf diese Rechner kann nicht direkt aus dem Internet, sondern lediglich über spezielle gesicherte Verbindungen nach Authentifizierung eines berechtigten Nutzers zugegriffen werden. Hierbei kann es sich um Dienste handeln, die nur den Betrieb innerhalb des jeweiligen Subnetzes betreffen oder Dienste, die aufgrund bestimmter Vorgaben nur aus dem internen Netz genutzt werden dürfen. Da diesen Rechnern und Servern keine öffentlichen Funktionen obliegen, können deren Dienste ohne weitere Authentifizierung nutzbar sein. Aufgrund der Abschottung gegenüber dem externen Netz besitzen nur Anwender innerhalb des internen Netzes Zugriff. Trotzdem werden diese Rechner mit anderen Diensten im Internet kommunizieren. Hierbei geht die Verbindung aber vom Rechner im inneren Netz aus. Der Rechner kann demzufolge auf sämtliche Server zugreifen. Der umgekehrte Weg für externe Rechner ginge nur über eine gesicherte Verbindung (z.B. VPN[2]) in das interne Netz, welches den externen Rechner logisch zu einem Teil dieses internen Netzes macht.

Für die Server könnte eine Einteilung nach der Netzzuordnung erfolgen, Man kann unterscheiden zwischen (1) interne Server für ausschließlich interne Dienste, (2) interne Server, die darüber hinaus Dienste für externe Nutzung anbieten, und (3) externe Server, die nicht von der Einrichtung betrieben werden. Alle Server können Dienste zur Verfügung stellen, die für interne Rechner und deren Nutzer von Interesse sind.

[2] VPN: *Virtual Private Network*, Zusammenfassung physikalisch getrennter Netze zu einem gemeinsamen Netz [BK01]

Die Rechner der Nutzer lassen sich aufgrund ihrer Verbindung zu oben definierten Netzbereichen in folgende Klassen untergliedern:

**R1:** Hierbei handelt es sich um Rechner, die mit dem internen Netz „fest“ verbunden sind und sich meist in einem abgeschotteten Bereich befinden. Typische Vertreter sind Arbeitsplatz- oder Laborrechner. Ihre Nutzung ist auf einen festgelegten kleinen Personenkreis oder eine einzelne Person beschränkt. Für den Zugriff ist eine Authentifizierung erforderlich. Anschließend sind die internen Dienste meist ohne weitere Authentifizierung nutzbar.

**R2:** Diese Rechner sind ebenfalls „fest“ mit dem internen Netz verbunden, stehen aber einem größeren Nutzerkreis zur Verfügung. Typische Vertreter sind Poolrechner für Studenten. Aufgrund ihrer festen Zuordnung zum inneren Netz können diesen Rechnern ohne separate Authentifizierung bestimmte interne Dienste zur Verfügung gestellt werden, z.B. Zugriff auf Netzwerk-Filesysteme (NFS). Anders als bei Typ R1 beschränkt sich die Autorisierung aber auf Dienste, die dem Rechner und nicht dem Nutzer zur Verfügung gestellt werden. Für personalisierte Dienste ist eine Anmeldung am Dienst und nicht nur am Rechner erforderlich.

**R3:** Bei diesen Typ handelt es sich um transportable Rechner, die zu einem bestimmten Zeitpunkt mit dem inneren Netz verbunden werden oder über das Internet auf angebotene Dienste zugreifen. Notebooks von Studenten oder Mitarbeitern würden in diese Rechnerklasse fallen. Da sie vollständig unter Kontrolle der Nutzer stehen und somit auch auf eine andere Person übertragen werden können, sind Aussagen über die Zugehörigkeit der am Rechner arbeitenden Person zu einem bestimmten Personenkreis nicht möglich. Es ist vielmehr erforderlich, dass bereits während der Verbindung zum internen Netz eine Nutzerauthentifizierung vorausgeht, um den Zugriff fremder Personen zu vermeiden. Anschließend können diese Rechner analog Typ R2 oder, falls dies für den erkannten Nutzer erlaubt ist, Typ R1 aufgefasst werden.

**R4:** Diese Rechner sind nicht mit dem internen Netz „verbunden“. Ein Zugriff auf bereitgestellte Dienste erfolgt lediglich über das Internet. Für personalisierte Dienste ist eine Authentifizierung erforderlich.

Zugang zu Rechnern der Klasse R1 oder R2 verlangt eine Nutzerauthentifizierung. Der Nutzer sollte einer bestimmten Nutzergruppe angehören, so dass dem Rechner Zugriff auf einige Dienste ohne erneute Authentifizierung eingeräumt werden kann. Mit einem Anwenderkreis, z.B. bestehend aus Studenten, Mitarbeitern oder Gästen, verlangt der Zugriff auf jeden internen Dienst für Klasse R2 eine weitere Authentifizierung, insofern die Identität nicht vom Rechner an den jeweiligen Dienst über ein spezielles Protokoll (z.B. Kerberos, siehe Kapitel 4.5.2) propagiert werden kann. Für Nutzer am Rechner R3 und R4 ist eine erfolgreiche Authentifizierung gegenüber dem Rechner für die Dienste nicht ausschlaggebend. Für den Zugriff auf das interne Netz ist eine separate Authentifizierung erforderlich. Legt man diese Strukturierung des Netzwerkes zugrunde, so erkennt man mögliche Wege des Zugriffs eines Nutzers auf die angebotenen Dienste[3].

## 3.2 Authentifizierungsschnittstellen

Sind Ressourcen nicht frei zugänglich, so ist für deren Nutzung eine Authentifizierung erforderlich. Für den Nutzer gilt es nachzuweisen, dass er zu der Gruppe von berechtigten Personen gehört. Der Nutzer muss sich authentisieren.

[3] Der Zugang zum internen Netz oder darüber hinaus zum Internet kann als ein weiterer Dienst angesehen werden. Abgesehen vom Kapitel 7.2.2 wird der Netzzugang deshalb nicht als separater Dienst aufgeführt.

Im vorigen Kapitel wurden die möglichen Zugangswege eines Nutzers zu Rechner- und Netzressourcen aufgezeigt. Daraus lassen sich die Authentifizierungsschnittstellen ableiten.

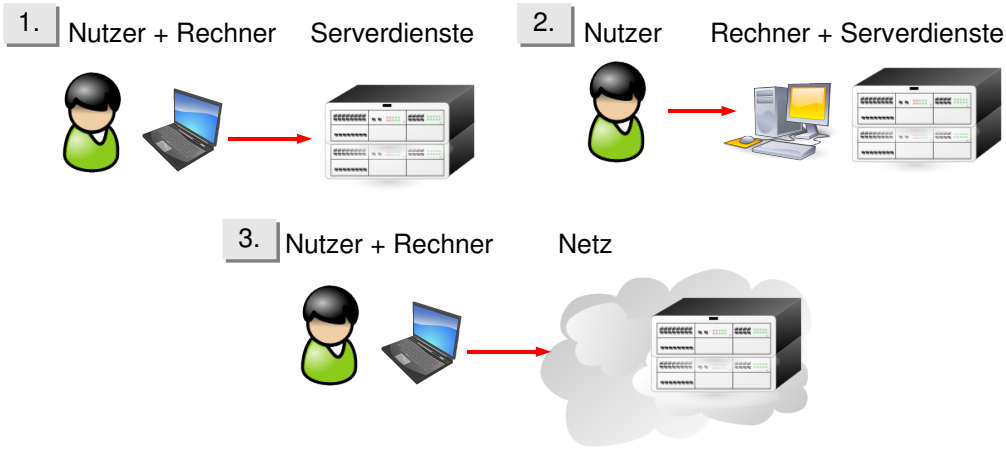



**Abb. 3.2 —** Authentifizierungsschnittstellen im täglichen Gebrauch

Nach Abbildung 3.2 kann die Authentifizierung an verschiedenen Punkten zwischen dem Nutzer und der Zielressource erfolgen.

**(1)** Ein berechtigter Nutzer authentisiert sich bei der Nutzung eines Dienstes direkt am Dienst über eine Benutzeranmeldung (Login). Somit ist die Authentifizierung eine Teilfunktion des Dienstes.
Diese Art der Authentifizierung wird bei personalisierten Diensten notwendig sein. Hierzu zählen alle Dienste, die Informationen anbieten und verarbeiten, welche gezielt einer Person zugeordnet werden. Als Beispiele seien die Bereitstellung der elektronischen Postfächer für den Abruf durch den Inhaber oder die Lehr- und Lernplattformen der Hochschule genannt.

**(2)** Eine Authentifizierung des Nutzers am Rechner kann vor dessen Verwendung notwendig sein. Dies betrifft insbesondere Rechner, die nicht einem Nutzer zugeordnet sind. Der jeweilige Nutzer weist sich vor Gebrauch dadurch aus, dass er sich gegenüber dem Rechner authentisiert (Rechner-Login). Anschließend wird für den Nutzer eine Sitzung (engl. *session*) im Rechner gestartet. Anwendungen innerhalb der Sitzung werden dem Login-Nutzer zugeordnet und diese Zuordnung bleibt bis zum Logout erhalten. Der Nutzer kann Anwendungen und Daten entsprechend seiner Berechtigung nutzen. Zusätzliche Rechte, also eine weitergehende Autorisierung erfordert eine zusätzliche Authentifizierung als Nutzer mit den entsprechenden Rechten. Mittels dieser Variante kann Zugang zu Poolrechnern und den darauf laufenden Applikationen geregelt werden.
Für Rechner unter eigener Kontrolle kann eine Authentifizierung am Rechner trotzdem sinnvoll sein. Dabei erfolgt mittels Zuordnung des Logins zu einer bestimmten Autorisierungsstufe die Festlegung, welche Ressourcen der Nutzer abrufen oder ändern darf[4]. Das Login ist also für die Rechtevergabe, die Autorisierung entscheidend. Generell sollte ein Nutzer nur die für seine Arbeit erforderlichen Rechte besitzen, um z.B. Schäden durch eigene Programme oder fremde Schadprogramme (Viren, Trojaner) zu minimieren. Demzufolge ist es sinnvoll, wenn ein Nutzer auch am eigenen Rechner mehrere Logins nutzt und sich für die Rechnernutzung „einloggt“, wobei nur zu administrativen Zwecken auf privilegierte Logins zurückgegriffen wird.

[4] Als Login wird auch der Name des Nutzers innerhalb des Systems bezeichnet. Dies ist ein mit den Authentisierungsmerkmalen verbundener Bezeichner des Nutzers, mit dem die Anmeldung am Rechner erfolgt.

**(3)** Eine Authentifizierung des Nutzers kann bei Zugriff auf das Netz erfolgen. Typisch ist diese Variante für den Zugang zum DSL-Netz eines ISP (Internet Service Providers). Innerhalb einer Hochschule ist es erforderlich, wenn ein Nutzer mittels eines Notebooks über WLAN oder kabelgebunden eine Verbindung zum internen Netz aufbaut.
Eine Authentifizierung zur Bestimmung des Nutzers ist hier aus zweierlei Gründen notwendig. Zum einen könnte der Nutzer Zugang zu Ressourcen erhalten, die lediglich für das interne Netz und die dafür berechtigte Nutzergruppe gedacht sind. Auf der anderen Seite ist der Nutzer und sein Rechner Teil des Netzwerkes und kann Daten im Internet austauschen. Aufgrund gesetzlicher Rahmenbedingungen[5] obliegen einem Provider, der den Zugang zum Internet ermöglicht, bestimmte Vorgaben über Protokollierung, insbesondere welche Nutzer zu bestimmten Zeiten einer öffentlichen Internet-Adresse zuzuordnen sind [Bre05]. Einer derartigen personenbezogenen Protokollierung muss eine Authentifizierung der Person vorausgehen.

### Authentifizierung im Hochschul-Umfeld

Für Hochschulen wird aufgrund der heterogenen Nutzerstruktur, unterschiedlicher eingesetzter Betriebssysteme und der Vielfalt eingesetzter Arbeitsplatz-, Poolrechner oder Notebooks, den darauf laufenden Applikationen sowie den angebotenen Diensten für die inner- und außeruniversitäre Anwendung an vielen Stellen die Authentifizierung des Anwenders gefordert. Somit sind alle Kombinationen nach Abbildung 3.1 denkbar. Im Allgemeinen sind sogar mehrere Varianten erforderlich. Ein Nutzer wird entweder einen universitären Rechner, z.B. einen Poolrechner nutzen und muss sich gegenüber dem Rechner authentisieren oder er nutzt den eigenen Rechner und authentisiert sich bei Zugriff auf das interne Netz.
Ohne die im Anschluss besprochenen Maßnahmen zur Reduzierung notwendiger Authentifizierungen wird es nach bereits erfolgter Anmeldung weitere Stellen geben, an denen erneut die Bestimmung der Identität erforderlich ist. Dies wird bei der Nutzung aller personenbezogenen Dienste erforderlich sein, falls die Nutzeridentität nicht vom Rechner an den Dienst über das Netz übertragen werden kann.

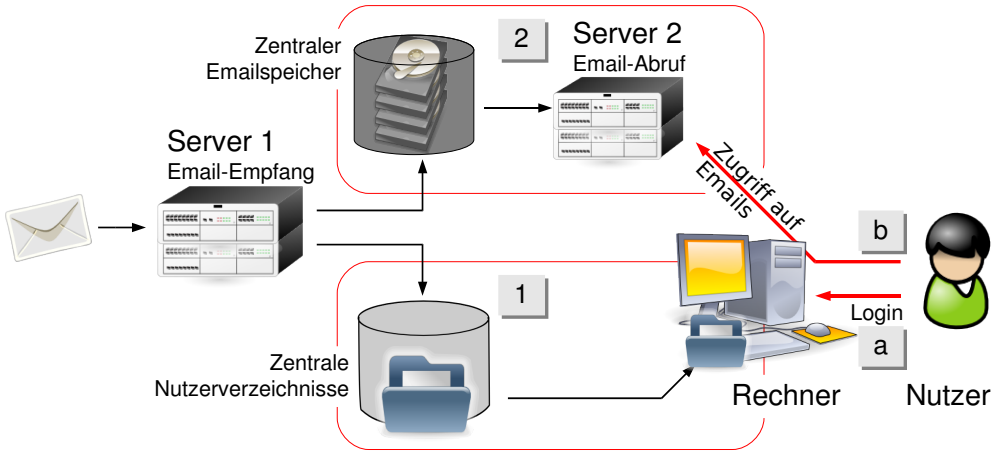



**Abb. 3.3 —** Abruf der Email

Ein Beispiel (Abbildung 3.3) soll dies verdeutlichen. Jeder Student und Mitarbeiter besitzt eine Email-Adresse. Eintreffende Emails für diese Adresse legt der empfangende Server an einen dafür vorgese-

[5] Gesetz zur Neuregelung der Telekommunikationsüberwachung und anderer verdeckter Ermittlungsmaßnahmen sowie zur Umsetzung der Richtlinie 2006/24/EG

hen Platz ab, entweder auf einen zentralen Emailspeicher [2] oder im zentralen Verzeichnis des zugehörigen Nutzers [1]. Werden die zentralen Nutzerverzeichnisse an die einzelnen Rechner verteilt, kann der Nutzer bereits nach dem Login am Rechner [a] auf seine Emails zugreifen.
Werden die Emails in einem separaten Speicher abgelegt [2], so gibt es mindestens einen weiteren Dienst für den Abruf der Email vom Server. In diesem Fall ist eine weitere Authentifizierung für den Dienst auf Server 2 erforderlich, um die Zuordnung des Nutzers zu einem Postfach zu ermitteln und im anschließenden Autorisierungsschritt seine Berechtigung zu prüfen. Die Authentifizierung ist hier ein Bestandteil des verwendeten Protokolls [b].
Beide Varianten schließen sich nicht gegenseitig aus. Server 2 könnte Email auch über die Nutzerverzeichnisse bereitstellen. Demzufolge kann es gleichzeitig mehrere Wege geben, auf eigene Emails zuzugreifen. Dabei muss an einer Stelle, der das Mailsystem vertraut, eine Authentifizierung stattfinden, bevor der Zugriff gewährt werden kann. Wird die Email nur über das Netzwerk angeboten [2], so muss der Nutzer seine Identität mehrmals nachweisen, sowohl beim Rechner-Login als auch beim Email-Abruf. Der Nutzer muss sich mehrmals authentisieren. Hoher Aufwand durch häufige Authentisierung seitens des Nutzers kann Sicherheitsprobleme bereiten, wie im Anschluss beschrieben wird. Ein möglicher Ausweg liefern *Reduced-* oder *Single Sign-On* Verfahren.

## 3.3 Reduced- und Single Sign-On

Für viele Authentifizierungsverfahren ist für den Nutzer das Verfahren selbst, also das Vorlegen der Authentisierungsmerkmale, mit Aufwand verbunden. Bei der Verwendung von Passwörtern wird für jede Anmeldung an einem Dienst die Eingabe eines Passwortes verlangt. Kommt dies häufig vor, neigt der Nutzer zu kurzen, einfachen und leicht erratbaren Passwörtern, welche die Sicherheit nachteilig beeinträchtigen.
Passwörter müssen, wie im Kapitel 4.2.1 aufgezeigt wird, entweder im Klartext oder als Hashwert beim Anbieter hinterlegt werden. Dies kann Sicherheitsprobleme verursachen, wenn die Anbieter nicht sorgsam damit umgehen. Da dieser Umstand vom Nutzer nicht beeinflusst werden kann, sollte für jeden Dienst ein eigenes Passwort gewählt werden. Durch technische Maßnahmen kann zwar eine ausreichende Qualität der gewählten Passwörter seitens der Dienste sichergestellt werden, die Nutzung verschiedener Passwörter für unabhängige Dienste lässt sich aber nicht erzwingen. Die Nutzer können also zu gleichen oder ähnlichen Passwörtern greifen, wie dies auch die in der Einleitung genannte Studie belegt. Damit hängt die Sicherheit aller Systeme, auf die ein Nutzer Zugriff hat, vom sorgsamen Umgang mit den Passwörtern seitens der Anbieters ab. Eine Kompromittierung eines Dienstes führt zur Gefährdung weiterer Dienste.
Möchte man Passwörter „sicher“ verwenden, so muss sich jeder Nutzer eine Vielzahl verschiedener Passwörter guter Qualität merken. Da dies kaum möglich ist, wird entweder nur eine geringe Anzahl Passwörter genutzt oder Hilfsmittel herangezogen. Externe Hilfsmittel, wie die verschlüsselte Ablage in einem PDA, bedeuten aber erhöhten Aufwand und sind im Endeffekt wenig komfortabel bei häufiger Verwendung. Hinzu kommen möglicherweise auch weitere Bedingungen, wie der Zwang zum regelmäßigen Wechsel von Passwörtern, der eine komfortable Rechnernutzung weiter erschwert. Andere Verfahren, z.B. eine Authentifizierung über biometrische Merkmale, besitzen ebenfalls Nachteile. So kann es für einen Nutzer kaum wünschenswert sein, wiederholt den Finger auf den Abdruckscanner legen zu müssen.
Ziel aus Sicht der meisten Nutzer wird ein System sein, bei dem man entweder die Tatsache, dass gerade die Authentifizierung stattfindet, nicht störend wahrnimmt oder zumindest die Anzahl der notwendigen Authentifizierungen eingeschränkt und im günstigsten Fall nur einmalig gefordert wird.

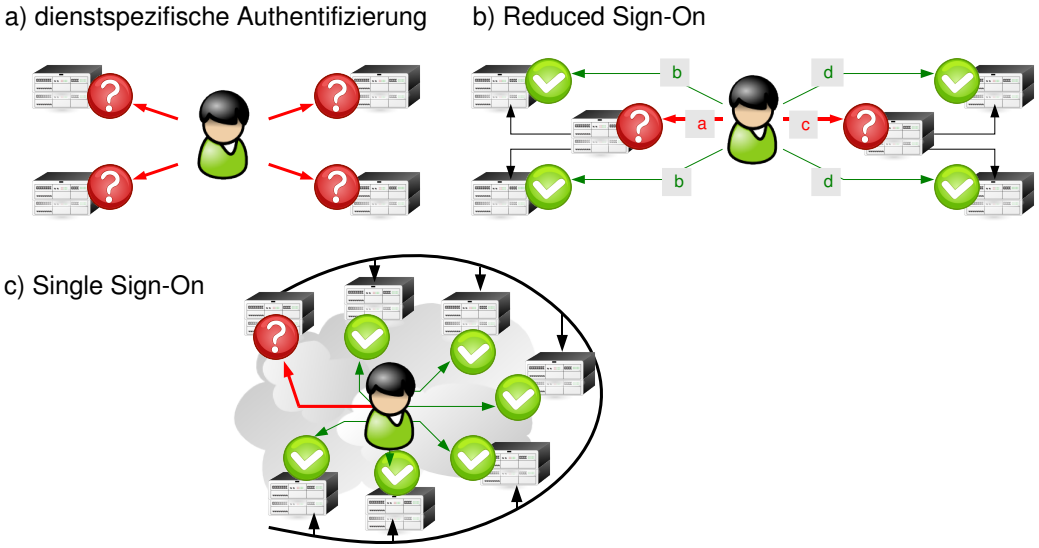



**Abb. 3.4 —** Sign-On Varianten

Abbildung 3.4 skizziert, wie die Anzahl der Anmeldevorgänge reduziert wird, indem nicht für jeden Dienst, wie es in der Abbildung oben links noch dargestellt ist, eine eigene Authentifizierung erforderlich ist. Bei *Reduced Sign-On* kann ein Nutzer nach Authentisierung gegenüber Dienst `[a]` die damit verbundenen Dienste `[b]` ohne erneute Authentifizierung nutzen. Gleiches gilt für `[c]` und `[d]`. Somit bilden die Dienste `[a]`+`[b]` und `[c]`+`[d]` jeweils einen Verbund, für den eine einmalige Authentifizierung reicht.
Noch angenehmer aus Sicht des Nutzers - ohne auf Authentifizierung zu verzichten - wäre der Fall, bei der nach einer einmaligen Anmeldung am ersten Dienst oder einem speziell dafür vorgesehenen Dienst alle weiteren Dienste ohne erneute Authentifizierung genutzt werden können, vorausgesetzt der Nutzer ist für alle Dienste autorisiert.
Der Begriff *Single Sign-On* (SSO) beschreibt eine derartige Systemumgebung, wie sie in Teil c) der Abbildung 3.4 dargestellt ist. Bei der ersten Authentifizierung, bei der ein Nutzer seine Authentisierungsmerkmale vorlegt, wird die Identität bestimmt und anschließend an die im SSO-System verbundenen Dienste propagiert. Die Stelle, welche die Authentifizierung durchführt und bescheinigt, wird als Identitätsprovider bezeichnet. Es kann sich um eine zentrale oder mehrere verteilte, mit den Diensten verbundene Stellen handeln. Die Übermittlung der Information über die Nutzeridentität sollte für den Nutzer nicht sichtbar und aus Sicherheitsgründen auch seitens des Nutzers nicht gezielt manipulierbar sein.
Ein Problem für SSO-Systeme stellt das Vertrauen dar, welches sich die Anbieter von Diensten innerhalb des Verbundes gegenseitig entgegen bringen müssen, wenn sie eine Authentifizierung bei anderen Anbietern und deren Identitätsprovidern als ausreichend ansehen sollen. Deshalb existiert momentan noch kein umfassendes SSO-System, welches mehrere Dienste im Internet abdeckt, obwohl Bestrebungen in dieser Richtung unternommen werden (siehe Kapitel 4.5.4). Aktuelle serverseitige SSO-Systeme, bei denen die Nutzeridentität zwischen den Servern propagiert wird, sind auf eine gewisse Menge von Diensten, meist innerhalb eines logisch abgeschlossenen Netzbereiches beschränkt. Damit handelt es um eine Ansammlung mehrerer Reduced Sign-On Systeme.
Einige Hochschulen oder Teile von Hochschulen verwenden bereichsübergreifende SSO-Lösungen, entweder speziell angepasste Systeme auf Basis der SSO-Federations nach dem Liberty Alliance Pro-

ject [Sch07b] oder sie bauen auf OpenID-, oder Shibbeloth-Systeme auf, wie z.B. Bibliotheken[6] oder Teile des D-Grid-Rechnerverbunds[7]. Diese SSO-Systeme werden im Kapitel 4.5.4 vorgestellt. Ein für alle deutschen Hochschulen nutzbares System könnte auf Basis der DFN-AAI[8] aufgebaut werden. Da die DFN-AAI aber keine Lizenzverträge zwischen unterschiedlichen Einrichtungen und den angebotenen Diensten vermittelt, muss hier das Vertrauensverhältnis wieder im Einzelnen geklärt und von der Einrichtung für jeden Dienst festgelegt werden.
Im Gegensatz zu den serverseitigen SSO-Systemen existieren clientseitige Lösungen, bei denen die Nutzeridentität über das Betriebssystem oder durch eine im Rechner arbeitende Software zwischen der Applikationen und dem Serverdienst ausgetauscht wird. Diese unterliegen nicht der Anbieter-Vertrauensproblematik, können aber andere Einschränkungen besitzen (siehe Kapitel 4.5).

## 3.4 Anforderungen

### 3.4.1 Anforderung an das Single Sign-On

Ziel der Arbeit ist eine SSO-Lösung für die Authentifizierung gegenüber den Diensten des internen Netzes nach Abbildung 3.1, aber auch im Hinblick auf einen externen Zugriff über das Internet. Eine Nutzung der Authentifizierung über den internen Bereich hinaus soll nicht ausgeschlossen werden. Hierfür wäre aber zu klären, inwiefern der genutzte Identitätsprovider von den externen Einrichtungen akzeptiert wird. In diesem Falle wäre die Einbindung der hier vorgestellten Lösung in ein Federation-System, z.B. der DFN-AAI denkbar, was aber nicht Gegenstand dieser Arbeit ist.
Die Dienste innerhalb eines Bereiches, welche nach einmaliger Authentifizierung übergreifend genutzt werden können, bilden den SSO-Verbund. Ein SSO-Verbund könnte derart arbeiten:

1. Authentisierung des Nutzers am Rechner mittels der nutzereigenen Authentisierungsmerkmale
2. Bestimmung der Identität im Rechner (Authentifizierung)
3. Überprüfung des Nutzerberechtigung, Freigabe für den Rechner (Autorisierung) und Speicherung einer Information über die Identität der mit dieser Sitzung verbundenen Person
4. Mitteilung der Identität an weitere Dienste, sobald der Nutzer auf diese zugreift - ggf. im Hintergrund auf dem Rechner ablaufende, für den Nutzer unsichtbare Authentifizierung
5. Löschen der Identitäts-Zuordnung beim Logout

Eine clientseitige Lösung sollte diese Umsetzung relativ einfach ermöglichen. Die Identität des Nutzers wird beim Login festgestellt und ist anschließend zwischen den Applikationen innerhalb des Clientsystems auszutauschen. Die Applikationen können die Identität anschließend an die Serverdienste übermitteln. Voraussetzung ist ein Verfahren, das mit der Information über die Identität eine Authentifizierung an den Diensten erlaubt. Ist dies möglich, so kann gleichzeitig das Problem des wechselseitigen Vertrauens der Dienstanbieter innerhalb des Verbundes vermieden werden. Allerdings wird dafür ein vertrauenswürdiges Clientsystem verlangt, die Problematik also auf den Client verlagert. Ein kompromittierter Client kann zu Sicherheitsproblemen führen, die im Kapitel 8.2 erläutert werden.
Gesucht wird demzufolge ein clientseitiges Verfahren, dass die Anmeldung gegenüber einer Vielzahl von Diensten erlaubt und trotzdem eine hohe Resistenz gegenüber einer möglichen Kompromittierung bietet. Die clientseitige SSO-Lösung sollte:

[6] `http://www.nationallizenzen.de`
[7] `http://www.d-grid.de/index.php?id=314`
[8] DFN-AAI (Authentifikation Autorisierungs Infrastruktur) `https://www.aai.dfn.de`

a) die aktuellen Serverdienste ohne größere serverseitige Anpassungen unterstützen und

b) dabei die Sicherheit gegenüber aktuell genutzten Verfahren nicht nachteilig beeinflussen,

c) auf einer Vielzahl möglicher Clientbetriebssysteme einsetzbar sein,

d) hohe Verfügbarkeit gewährleisten,

e) für eine große Anzahl gleichzeitig agierender Nutzer ohne merkliche Einschränkungen funktionieren,

f) sich in bestehende Systeme zur Nutzerverwaltung (Identity Management), zur Verwaltung der eingesetzten Software (Application Management) und zur Verwaltung von Zugriffsrechten und Sicherheitsstrukturen (Security Management) integrieren lassen,

g) die Kontrolle (Monitoring) der durchgeführten Authentisierungen ermöglichen,

h) möglichst mittels frei nutzbarer und quelloffener Software ohne größere clientseitige Anpassungen umsetzbar sein und

i) dabei nicht auf eine spezielle Hardware oder Software eines Herstellers fixiert sein.

Die Punkte a) bis g) ergeben sich aus dem Ziel, bestehende passwortbasierte Verfahren anfangs zu ergänzen und, sobald alle Nutzer das neue Verfahren akzeptieren, vollständig zu ersetzen. Dabei entstehen einmalig Kosten bei der Umstellung bestehender Authentifizierungsverfahren. Für die Pflege des SSO-Systems werden zusätzlich laufende Kosten anfallen. Diese sollen aber nicht die Wartungskosten für die Pflege bestehender Authentifizierungsverfahren übersteigen. Die Bedingung frei nutzbarer Software soll deshalb verhindern, dass durch die Umstellung weitere regelmäßige Nebenkosten entstehen. Punkt i) gewährleistet die Herstellerunabhängigkeit, um eine möglichst große Flexibilität bei der Tokenwahl zu erreichen und Folgekosten durch Bindung an ein Produkt zu vermeiden.

### 3.4.2 Sicherheitsanforderungen

Durch das SSO-System soll insgesamt ein Sicherheitsgewinn gegenüber den aktuell in vielen Hochschulbereichen genutzten passwortgestützten Authentifizierungsverfahren erzielt werden. Dazu müssen mehrere Bedingungen erfüllt werden.
Betrachtet man nur den Wunsch nach einem clientseitigen Single Sign-On, so ließe sich, abgesehen von Punkt b), ein SSO-System durch die Verwendung eines gemeinsamen Passwortes umsetzen. Der Punkt Sicherheit wäre dann aber, wie bereits im Kapitel 3.3 erläutert, nachteilig beeinträchtigt und somit eines der Kriterien für die Einführung eines SSO-Authentifizierungsverfahrens, die Verbesserung der Informationssicherheit, verletzt. Das Verfahren muss demzufolge verhindern, dass Authentisierungsinformationen an einen Dienstanbieter übermittelt werden, die dieser anschließend gegenüber anderen Anbietern missbräuchlich verwenden kann.
Aus der Sicht des Identitätsproviders und der Dienstanbieter ist eine wesentliche Anforderung, dass der Weitergabe der Authentisierungsinformationen verhindert wird. Dies erhöht den Schutz in einem Umfeld, in dem unbedarfte Nutzer zu sorglos mit den ihnen anvertrauten Zugangsmöglichkeiten umgehen. Das gewählte Authentifizierungsverfahren muss deshalb ein Kopieren der Authentisierungsmerkmale derart erschweren, dass eine kopierte Identität praktisch nicht auftreten kann.
Falls Nutzer ihre eigenen Rechner verwenden, sollte die Authentisierung bereits bei Verbindung mit dem internen Netz erfolgen. Dies stellt eine gewisse Einschränkung an ein clientseitiges SSO-System

dar, da der clientseitige Teil unter vollständiger Nutzerkontrolle steht und damit möglicherweise einfach manipulierbar ist. Das System muss somit auch robust gegen Eingriffe in die clientseitige Installation sein. Dabei würde es ausreichen, wenn eine Veränderung der clientseitig installierten Software nicht dazu führen kann, dass geschützte Authentisierungsmerkmale kopiert und von einem Dritten unberechtigt genutzt werden können.
Bei jedem Authentifizierungsvorgang werden Informationen zwischen der Person, die sich authentisiert und dem System, an dem sie sich authentisiert, ausgetauscht. Hierbei kann es sich um sensible Daten handeln. Ein Schutz dieser Information, die sich aus den Authentisierungsmerkmalen ergibt, ist insbesondere bei der Anmeldung an netzgestützten Diensten, der *Remote Authentication* zu beachten. Remote Authentication bietet aufgrund der Übertragung im Netz einem Angreifer die Möglichkeit zum Abfangen und Manipulieren der Daten, wie im Kaptitel 4.3 aufgezeigt wird.
Bei einer Authentifizierung an der lokalen Maschine besteht eine ähnliche Gefährdung. So könnten eingeschleuste Schadprogramme, z.B. Keylogger, Authentisierungsmerkmale abfangen und an unbefugte Personen weiterleiten. Auch wenn das verwendete Verfahren verhindert, dass die Authentisierungsmerkmale im Rechner kopiert werden können, besteht immer noch die Gefahr, dass Schadprogramme bei Vorliegen der Merkmale diese missbrauchen.
Die eingesetzten Authentisierungsmerkmale gehören zum Nutzer und müssen clientseitig bei jeder Authentifizierung verfügbar sein und deshalb vom Nutzer mitgeführt werden. Somit besteht der Bedarf, diese Informationen in der Hand des Nutzers vor Missbrauch zu schützen. Dieser Missbrauch kann aufgrund des Verlustes oder der Entwendung der Merkmale, z.B. bei sorglosem Umgangs des Inhabers mit den Merkmalen entstehen.

Das SSO-System und das damit eingesetzte Verfahren muss deshalb folgende Kriterien erfüllen, um im Vergleich zu Passwörtern einen Sicherheitsgewinn zu erzielen:

j) Gültige Authentisierungsmerkmale dürfen nicht bei den Anbietern hinterlegt werden.

k) Die Sicherung der Authentisierungsmerkmale vor dem Erstellen illegaler Kopien sowohl durch den Inhaber der Merkmale, als auch seitens Dritter ist erforderlich.

l) Die Übertragung der Authentisierungsinformationen ist zu schützen.

m) Die Authentisierungsmerkmale müssen gegen Verwendung durch unbefugte Programme gesichert sein.

n) Authentisierungsmerkmale müssen auch bei Verlust durch zusätzliche Sicherungsmaßnahmen vor Missbrauch geschützt sein.

Die Authentifizierung von Nutzern kann auf Objekte übertragen werden. Authentifizierung von Objekten bedeutet, dass ein System ohne konkreten Nutzerbezug sich gegenüber einem anderen System oder Dienst authentisiert. Da die gleichen Fragestellungen zu möglichen Authentifizierungsverfahren und notwendigen Sicherungsmaßnahmen auftreten, wird die Authentifizierung von Objekten nicht gesondert betrachtet.

# Kapitel 4

# Stand der Technik

Einem Rechner stehen verschiedene Verfahren zur Verfügung, um die Identität eines Nutzer zu prüfen. Im ersten Teil dieses Kapitel werden die gängigen Authentifizierungsverfahren vorgestellt und Vor- und Nachteile erörtert. Wie sich dabei herausstellt, bietet die besitzbasierte Authentifizierung mit Hardware-Token einige Sicherheitsvorteile gegenüber den häufig eingesetzten Passwort-Verfahren.
Einen Anreiz zur Umstellung von Passwörtern auf Hardware-Token soll die Einführung von Single Sign-On geben. Der zweite Teil dieses Kapitels zeigt gängige SSO-Verfahren und liefert eine Begründung für die Wahl der zertifikatsbasierten Authentifizierung, die sich gut mit Hardware-Token verbinden lässt.

## 4.1 Authentifizierung

Ein Authentifizierungsverfahren hängt im Wesentlichen von den verwendeten Authentisierungsmerkmalen ab. Die Art und Weise, mit der ein Nutzer sich gegenüber einem System authentisiert, lässt sich anhand der Merkmale einer der folgenden Kategorien zuordnen. Da diese Methoden sowohl für die Authentisierung von Personen als auch für Objekte zutreffen, sprechen wir von der Authentisierung eines Subjekts gegenüber einem System.

**Wissen** („*Something to know*“):
Ein Subjekt authentisiert sich durch Nachweis über die Kenntnis einer Information, die nur dem Subjekt und dem System bekannt sind, in dem die Authentifizierung erfolgt. Typischer Vertreter dieser Methode ist die Authentifizierung mit Passwörtern.

**Besitz** („*Something to have*“):
Das Subjekt besitzt etwas, das bei der Authentisierung vorzulegen oder anzuwenden ist. Ein Vertreter dieser Methode ist die Authentifizierung mit Ausweisen. Am Rechner kann ein kryptographischer Schlüssel genutzt werden, der auf einem Speichermedium hinterlegt ist. Die Kombination aus Speichermedium und dem enthaltenem Schlüssel stellt den Besitz dar.

**Persönliches Merkmal** („*Something to be*“):
Die Authentisierung erfolgt durch Überprüfung der Anwesenheit des Subjekts. Bei Personen kann dies durch die Kontrolle eines biometrischen Merkmals erfolgen. Ein bekannter Vertreter dieser Methode ist die Authentifizierung über einen Fingerabdruck.

## 4.2 Verfahren zur lokalen Nutzerauthentifizierung

Systeme mit lokaler Nutzerauthentifizierung sind Arbeitsplatzrechner, PDAs, Mobiltelefone und andere Systeme, bei denen der Vorgang der Identitätsbestimmung im lokalen System erfolgt.

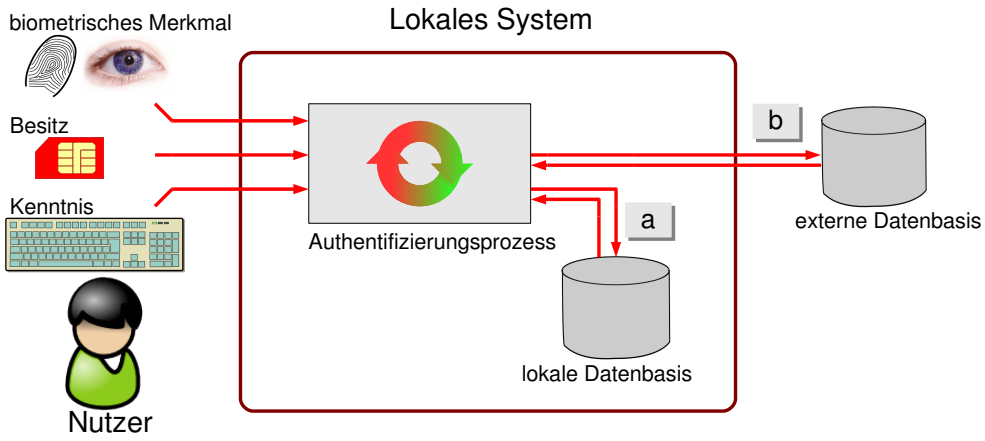



**Abb. 4.1 —** Lokale Nutzerauthentifizierung

Abbildung 4.1 zeigt den Vorgang. Eine Datenbasis speichert die Beziehung zwischen den Authentisierungsmerkmalen und der Identität der Person im System. Für die lokale Authentifizierung spielt dabei keine Rolle, ob die Datenbasis auf dem lokalen Rechner `[a]` oder auf einem entfernten System `[b]` abgelegt ist. Es zählt, dass der eigentliche Vorgang der Authentifizierung im lokalen System erfolgt. Variante `[b]` ermöglicht eine verteilte Datenbasis und die Nutzung durch mehrere Rechner, stellt aber auch besondere Sicherheitsanforderungen an das Verfahren, was den Transport der Informationen von und zur Datenbasis betrifft.

### 4.2.1 Authentifizierung mittels Kenntnis

Zu den gängigsten Verfahren zur Nutzerauthentifizierung auf Basis von Wissen zählen alle passwortbasierten Verfahren, also Verfahren, die auf Kenntnis einer geheimen Information beruhen. Bei der geheimen Information handelt es sich um ein statisches, geheimes Kennwort (engl. *password*) oder allgemein, eine Folge von Zeichen, auch mehrere Wörter, die über eine Tastatur eingegeben werden können. Manchmal werden aus mehreren Wörtern bestehende Passwörter auch als Mantra (engl. *passphrase*) bezeichnet. Im Folgenden werden diese allgemein unter dem Begriff Passwort zusammengefasst. Bei den Verfahren stellt die Kombination aus dem Login-Namen des Nutzers innerhalb des Systems und dem Passwort das Authentisierungsmerkmal des Nutzers dar, das bei einer Authentifizierung vorzulegen ist.
Die Authentifizierung erfolgt jeweils durch Vergleich des eingegebenen Passwortes mit einer in der Datenbasis hinterlegten Fassung für den zugeordneten Login-Namen. Aus Sicherheitsgründen sollte die hinterlegte Version aber nicht dem Klartext des einzugebenden Passwortes entsprechen. Andernfalls könnten administrative Nutzer mit Zugriff auf die Datenbasis oder Personen, die sich durch einen erfolgreichen Angriff auf das System Zugriff beschafft haben, die Datenbasis auslesen und sich in Folge als besagte Nutzer authentifizieren. In diesem Zusammenhang besteht eine noch größere Gefahr, wenn ein Nutzer das gleiche Passwort für mehrere Systeme verwendet. Sollte es einem Angreifer gelingen, ein Klartext-Passwort bei der Eingabe oder einer Übertragung abzufangen, so besteht Gefahr

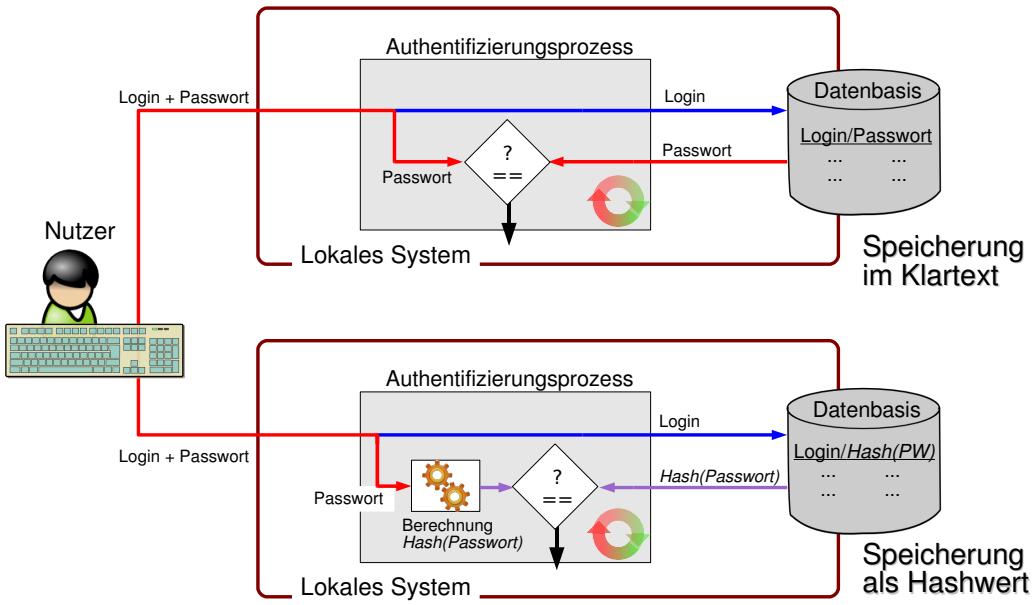



**Abb. 4.2 —** Passwort-Authentifizierung: Serverseitige Speicherung im Klartext oder als Hashwert

für weitere Systeme. Dies könnte zu einer Kettenreaktion führen, bei der durch einen Fehler in einem System weitere, davon unabhängige Systeme kompromittiert werden können.
Um eine Klartextspeicherung der Passwörter in der Datenbasis zu vermeiden, sollte nach Möglichkeit eine Einwegfunktion auf das Passwort angewendet und dessen Ergebnis gespeichert werden, so wie dies im der unteren Teil der Abbildung 4.2 dargestellt ist. Die Einwegfunktion soll sicherstellen, dass die Berechnung einer Umkehrung praktisch unmöglich ist. Dazu muss folgendes gelten. Für ein Passwort $w_x$ aus der Menge aller Passwörter $W$ wird die Berechnung $c = f(w_x)$ ausgeführt, um den zu speichernden Zielwert $c$ zu erhalten. Hierbei entspricht $W$ sinnvollerweise der Menge aller Zeichen-Kombinationen mit einer gewissen Mindestlänge, die sich über eine Tastatur eingegeben lassen[1]. Die Umkehrung $w_y = f^{-1}(c)$, also die Gewinnung eines gültigen Passwortes $w_y$, das bei Anwendung den gleichen Wert $f(w_y) = c$ wie bei Eingabe von $w_x$ liefert[2], sollte jedoch verhältnismäßig schwer gegenüber dem Nutzen sein, der sich für einen Angreifer im Anschluss bietet. Dies erschwert einem Angreifer mit Zugriff auf die Datenbasis die Entwendung eines nutzbaren Passwortes.
Als Einwegfunktion für $f$ bietet sich eine Hashfunktion mit starker Kollissionsresistenz an. Die starke Kollissionsresistenz (oftmals fälschlicherweise als Kollissionsfreiheit bezeichnet) besagt, dass der Hashwert für zwei verschiedene frei wählbare Eingaben ebenfalls unterschiedlich sein soll. Für $w_1, w_2 \in W$ mit $w_1 \neq w_2$ soll gelten: $H(w_1) \neq H(w_1)$. Natürlich ist dies aufgrund der bei Hashfunktionen begrenzten Größe des erzeugten Hashwertes nicht für die Menge aller Passwörter möglich, bei denen die Eingabe länger als der Hashwert sein kann. Starke Kollisionsresistenz bedeutet, dass zu einem frei wählbaren Passwort $w_1$ in der Praxis kein Passwort $w_2$ gefunden werden kann, dass den gleichen Hashwert liefert. Eine derartige Einweg-Hashfunktion bezeichnet man als starke oder kryptographisch sichere Hashfunktion [Eck06].

[1] Für rechnerunabhängige Passwörter sollte man sich auf die Zeichen des ASCII-Codes beschränken. Andere Zeichen, z.B. Umlaute, können je nach Landeseinstellung unterschiedlich kodiert sein.
[2] Da es sich bei gängigen Verfahren nicht um eine injektive Abbildung handelt, können verschiedene Passwörter $w_1, \ldots w_n$ den gleichen gespeicherten Wert $c = f(w_x)$ liefern, sind also gleichwertige Authentisierungsmerkmale.

In der Praxis angewendet werden *Secure-Hash-Algorithm* in Version `SHA-1`, `SHA-256`, `SHA-384` oder `SHA-512` [RFC4634]. Bei den letzten drei Varianten entspricht die Zahl der Bitlänge des erzeugten Hashwertes. Die ältere Version `SHA` und die ebenfalls oft genutzten `MD5`-Hashwerte oder die Verwendung des Enigma-Algorithmus (auch als `Crypt` bezeichnet und praktisch eingesetzt für die Ablage von Passwörtern in älteren UNIX/Linux-Versionen [MT79]) genügen hingegen aktuell den Anforderungen bezüglich der Kollissionsresistenz nicht mehr [WLF+05, LL07].
Auf Basis einer Kenntnis als Authentisierungsmerkmal existieren auch Verfahren, die nicht mit statischen, sondern mit kurzzeitig oder einmalig gültigen Passwörtern operieren. Sie werden jedoch für lokale Authentifizierung eher selten genutzt und werden deshalb im Kapitel 4.3.1 unter Remote Authentication näher erläutert.

### 4.2.2 Besitz als Authentisierungsmerkmal

Authentifizierung über Besitz bedeutet eine Authentisierung des Nutzers durch Vorweisen oder Anwenden eines Merkmals, welches der Nutzer beim Authentifizierungsvorgang „besitzen“ muss. Dieses Besitzmerkmal sollte nicht einfach kopierbar sein bzw. die Erstellung einer Kopie soll einen derart hohen Aufwand erfordern, dass es sich für mögliche Angreifer als nicht lohnenswert erweist. Der „Besitz“ wird oftmals durch ein weiteres Merkmal an einen Besitzer gebunden. Ein Beispiel ist der Reisepass als Besitz-Merkmal. Die Verbindung zum rechtmäßigen Inhaber ist durch die biometrische Übereinstimmung mit dem eingetragenen Bild gegeben.
In elektronischen Systemen ist ein vergleichbarer Besitz mit derartigen Merkmalen schwerer zu realisieren, da eine elektronische Schnittstelle zur Prüfung des Besitzes existieren muss. Die über diese Schnittstelle transportierten Informationen könnten abgefangen und modifiziert oder leicht kopiert werden, was den gestellten Anforderungen widerspräche.
Ein negatives Beispiel wäre die Authentisierung durch Vorlegen einer Magnetkarte, also einer Plastikkarte mit Magnestreifen, von dem die Informationen elektronisch ausgelesen werden können. Eine derartige Karte lässt sich mit relativ geringem Aufwand kopieren und die Kopie wäre von einem System, welches nur die Information des Magnetstreifens als Authentisierungsmerkmal nutzt, nicht vom Original zu unterscheiden. Um dies zu verhindern, wird bei Magnetkarten für Geldautomaten (EC-Card) neben dem Vorweisen des Besitzes (Auslesen des Magnetstreifens) ein weiteres Merkmal vom Nutzer gefordert, in diesem Fall die Eingabe der PIN.
Zur Authentifizierung mittels Besitz zählen alle Varianten, die eine Information auf dem „Besitz“ speichern und bei denen diese Information zur Authentisierung ausgelesen und analog einem Passwort überprüft wird. Typische Vertreter sind die bereits genannten Magnetkarten, Chipkarten mit Speicherchip oder USB-Sticks mit meist PIN-geschütztem Speicherbereich. Zur Sicherheit arbeiten die meisten Verfahren mit einem weiteren Merkmal (Abbildung 4.3), das entweder parallel zur Information an den Authentifizierungsprozess übergeben wird (Bsp. EC-Karten-PIN) oder das bereits vor dem Zugriff auf die Information einzugeben ist (Bsp. Chipkarten-PIN).

#### Mifare - kontaktlose Authentifizierung auf Basis eines Besitzes

Für bestimmte Anwendungsfälle ist eine besitzbasierte Authentifizierung eines Nutzers erwünscht, die ohne weitere Merkmale auskommt. Dazu wird oftmals die *Mifare*-Technologie[3] oder deren Nachfolger genutzt. Mifare erlaubt die berührungslose bzw. kontaktlose Authentifizierung von Personen auf Basis einer Kurzstrecken-Funktechnologie (siehe Kapitel 5.1.2). Hierbei enthält ein kontaktloser Chip, meist in einer Plastikkarte integriert, die Authentisierungsinformation, welche in der Nähe des

[3] Mifare ist eine von Philips Semiconductors eingeführte Technologie für kontaktlose Chipkarten.

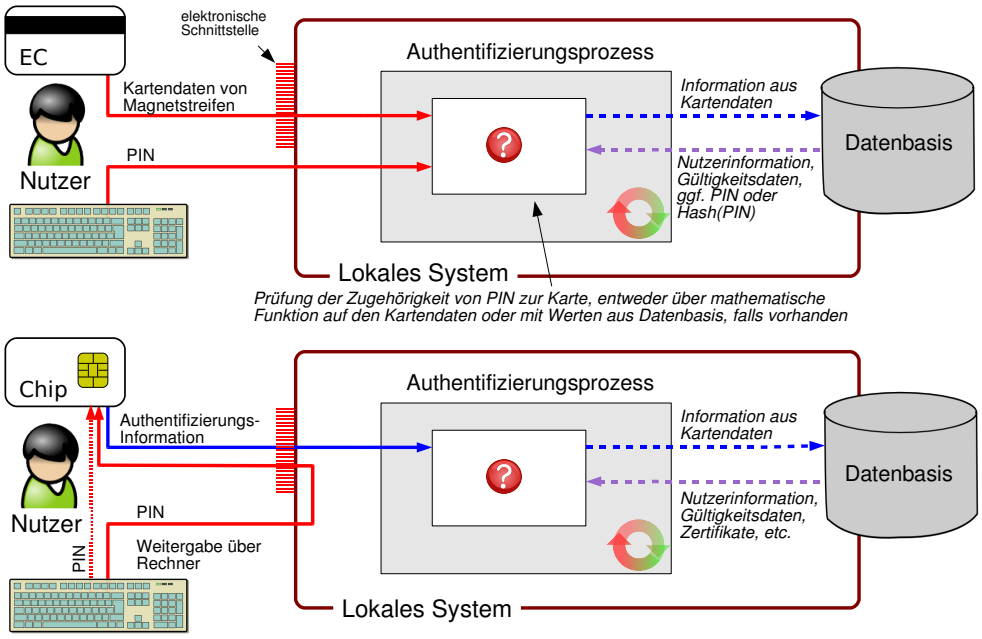



**Abb. 4.3 —** Authentifizierung mit Besitz - Beispiel EC-Karte oder Chipkarte

Lesegerätes von diesem ausgelesen und ggf. verändert werden kann. Zugriff und Übertragung erfolgen über einen verschlüsselten Kanal. Die Lesegeräte und die von ihnen lesbaren Karten haben den zugehörigen Schlüssel hinterlegt. Allerdings sind gerade bei der häufig eingesetzten Variante Mifare-Classic Sicherheitsprobleme bei dem verwendeten Crypto-1-Verfahren und dessen Implementierung bekannt [CNO08, dKGHG08, Plö08]. Es gibt jedoch Mifare-Erweiterungen, denen dieser Mangel nicht anhaftet und die in Folge auch weiterhin eingesetzt werden können.

Prinzipiell wäre es bei den Mifare-Systemen möglich, die Authentisierungsinformation auszulesen und zu vervielfältigen. Allerdings wird der Schlüssel zur Kommunikation zwischen den Kartenlesern und Karten vom Hersteller oder Provider unter Verwahrung gehalten. Nach Initialisierung kann er - abgesehen von der genannten Sicherheitsproblematik - unauslesbar innerhalb der Karten und den abgestimmten Lesegeräten abgelegt werden. So können nur berechtigte Lesegeräte auf die Karteninhalte zugreifen und Informationen an das Backend-System weitergeben. In der Firmware des Lesers und für die Software des Systems muss Sorge getragen werden, dass die zu schützende Informationen nicht nach außen dringen kann. Dieses Problem betrifft natürlich auch die vorher genannte Authentifizierung mit Besitz und zusätzlicher PIN. Es muss vermieden werden, dass eine unbefugte Person auf die vertraulichen Informationen aus dem „Besitz“ zugreifen und außerdem die PIN kopieren kann. Somit bietet es sich an, die Authentifizierung nicht auf das Auslesen geheimer statischer Merkmale zu beschränken.

Ein wesentlicher Nachteil von Mifare besteht in der Tatsache, dass bei Verlust oder Entwendung der Mifare-Karte eine unberechtigte Personen leicht Zugriff erhalten kann, vorausgesetzt es erfolgt keine zeitnahe Sperrung. Natürlich ließe sich Mifare zur Erhöhung der Sicherheit auch mit einer PIN kombinieren, doch diese Variante wird selten genutzt.

### Challenge-Response-Verfahren und Mikrocontroller-gestützte Systeme

Eine Alternative zur Mehrfaktor-Authentisierung oder zur Nutzung abgeschotteter Mifare-Systeme bietet die Verwendung dynamisch generierter Werte innerhalb des „Besitzes“. Die Information, die über die elektronische Schnittstelle transportiert wird, wäre nicht mehr statisch und somit auch nicht mehr leicht replizierbar. An ihrer Stelle tritt ein Wert, der in Folge einer Authentifizierungs-Anfrage berechnet wird. Das Authentisierungsmerkmal muss dafür eine Funktion

$$resp = f_{user}(cha) \tag{4.1}$$

bieten, die auf eine Anfrage $cha$ (engl. *challenge*) mit einer passenden Antwort $resp$ (engl. *response*) reagiert.

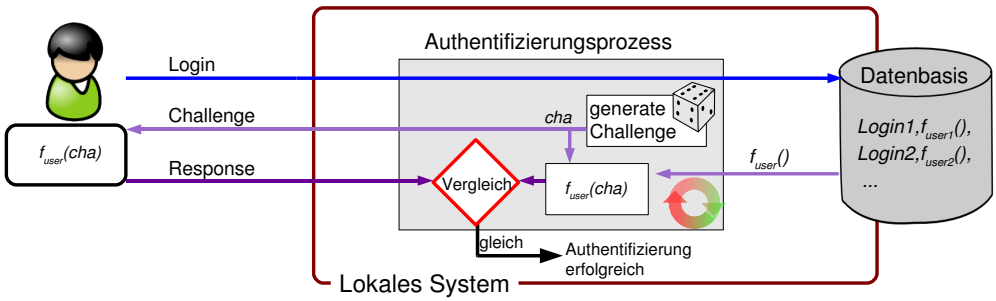



**Abb. 4.4 —** Challenge-Response Authentifizierung

Derartige, als *Challenge-Response-Authentication* bezeichnete Verfahren lassen sich auch für Authentifizierungen auf Basis eines Wissens verwenden, wenn eine Kenntnis über etwas erfragt wird, das im betreffenden Anwendungsfall nur der Person mit der zu prüfenden Identität zur Verfügung stehen sollte. Ein Beispiel wäre die Frage nach dem Geburtsnamen der Mutter oder bestimmten, nur der Person kenntlichen kalendarischer Daten. Allerdings ist hier die Sicherheit im Sinne der Authentizität, also der korrekten Zuordnung des Antwortenden zur behaupteten Identität fraglich, da zutreffende Antworten meist leicht recherchierbar sind. Auch die einfache Frage nach einem Passwort mit der passenden Antwort - dem richtigen Passwort - wäre ein Challenge-Response-Verfahren, bei dem jedoch immer die gleiche Frage gestellt wird und somit immer die gleiche Antwort gültig ist. Um zu vermeiden, dass eine Antwort wiederholt genutzt werden kann, sollte die Anfrage variiert, im besten Fall niemals wiederholt werden.
Für Authentifizierung in elektronischen Systemen basierend auf Challenge-Response-Verfahren kann, wie in Abbildung 4.4 dargestellt, ein Generator herangezogen werden, der zufällige Werte als Eingabe der Nutzer-Funktion $f_{user}$ würfelt. Sind beiden an der Authentifizierung beteiligten Parteien, der Person, die sich am System authentisiert, und dem Authentifizierungsprozess die Funktion $f_{user}$ bekannt, können beide das gleiche Ergebnis auf die Anfrage berechnen und der Authentifizierungsprozess kann sein berechnetes Ergebnis mit dem der Gegenpartei vergleichen. Übereinstimmung bedeutet erfolgreiche Authentifizierung. Werden jeweils neue Anfragen gestellt und liefert $f_{user}$ jeweils abweichende Antworten, so ist eine Wiederholung und damit der wesentliche Nachteil einer Authentifizierung auf Basis statischer Informationen vermeidbar.
An das System sind aber weitere Anforderungen zu stellen. So muss sichergestellt sein, dass nur dem berechtigten Nutzer und dem Authentifizierungsprozess die zugrunde liegende Funktion $f_{user}$ bekannt sind und aus den Wertepaaren $cha$ und $resp$ die Funktion nicht rekonstruiert werden kann.

Da die Geheimhaltung der Funktion auf das Prinzip „Security by Obscurity“ hinausläuft, verletzt sie das Kerckhoffs'sche Prinzip, welches sinngemäß lautet: *„Die Sicherheit eines Systems darf nicht von dessen Geheimhaltung abhängen.“* [Ker83]. Dieses Prinzip, welches ursprünglich eine Maxime für kryptographische Verfahren im militärischen Umfeld darstellte und darauf abzielte, dass die Sicherheit nur auf Geheimhaltung des Schlüssels basieren darf, trifft auf viele sicherheitskritische Bereiche zu. Insbesondere ist es auch für Authentifizierungsverfahren gültig. Auf Dauer könnte sich die Geheimhaltung der Funktion $f_{user}$ als problematisch erweisen, insbesondere wenn der Algorithmus nur einer begrenzten Anzahl von Personen zur Prüfung vorgelegt wird. So können Implementierungsfehler unerkannt bleiben oder der zugrunde liegende Algorithmus hat Schwächen, die bei der eingeschränkten Prüfung nicht aufgefallen sind.
Mit Hinblick auf die Sicherheit des Challenge-Response-Verfahrens ist es günstiger, wenn nicht die Funktion selber geheim zu halten ist, sondern die Sicherheit von einem einfacher geheim zu haltenden Schlüssel $K_{user}$ abhängt. Gleichung (4.1) wäre zu erweitern, so dass Nutzer und Authentifizierungsprozess

$$resp = f(cha, K_{user}) \tag{4.2}$$

auf Basis einer bekannten und geprüften Funktion $f$ berechnen. Dabei ist es natürlich erforderlich, dass aus den Wertepaaren $cha$ und $resp$ nicht auf den Schlüssel $K_{user}$ geschlossen werden kann. Somit bietet sich eine der bereits erwähnten kryptographisch sicheren Hashfunktionen, angewandt auf den mit $cha$ verknüpften Schlüssel an. Oder man nutzt symmetrische oder asymmetrische kryptographische Funktionen auf $cha$, je nach verwendetem Nutzerschlüssel (siehe Kapitel 4.3.2).
Da bei einer wiederholten Anfrage mit dem gleichen $cha$-Wert jeweils das gleiche Ergebnis berechnet wird, besteht die Gefahr einer Replay-Attacke. Dies ist eine fehlerhafte Authentifizierung, bei der ein aufgezeichnetes $(cha, resp)$-Paar erneut vorgelegt werden kann. Die wiederholte Anfrage kann dabei auch nach einem langen Zeitraum auftreten. Seitens des Authentifizierungsprozesses kann man dies einerseits durch geschickte Verwaltung der bereits getätigten $cha$-Anfragen vermeiden oder man nutzt einen weiteren Wert, z.B. einen Zähler, dessen Betrag die Einmaligkeit über den Verwendungszeitraum garantiert. Ein derartiger Wert wird als $nonce$ (engl. ***number used once***) bezeichnet. Korrekte Authentifizierung erfordert die Berechnung von

$$resp = f(cha, K_{user}, nonce) \tag{4.3}$$

seitens des Nutzers und zum Vergleich innerhalb des Authentifizierungsprozesses. Dieses Verfahren wird sehr häufig eingesetzt [And08].

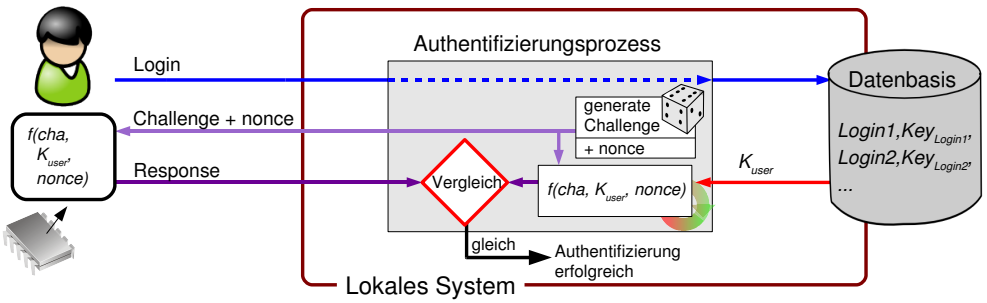



**Abb. 4.5 —** Challenge-Response basierend auf geheimen Nutzerschlüssel mit $nonce$

Für passwortbasierte Verfahren kann Gleichung (4.3) wie folgt abgeändert werden. Der Server stellt

eine Frage in Form einer $nonce$ und der Client seinerseits generiert ebenfalls einen weiteren $nonce$-Wert ($cnonce$). Zur Authentisierung überträgt der Client

$$resp = cnonce, f(nonce, cnonce, Passwort_{user}) \tag{4.4}$$

Daraufhin kann der Authentifizierungsprozess unter Kenntnis des Nutzer-Passwortes den korrekt berechneten Hashwert prüfen. $cnonce$ garantiert unabhängig vom Server-$nonce$ die Einmaligkeit der Antwort und erschwert einem kompromittierten und somit nicht berechtigten Authentifizierungsprozess (z.B. durch Umleitung des Informationsflusses über einen Dritten) die Vereinfachung der Suche nach einem gültigen Passwort unter Verwendung selbst vorgegebener $nonce$-Werte.

**Mikrocontroller-gestützte Systeme:** Challenge-Response-Verfahren bieten sich für eine Remote-Authentifizierung an Diensten im Netz an, da hier die Übertragung wiederverwendbarer geheimer Informationen vermieden werden kann. Aber auch für die lokale Authentifizierung werden sie herangezogen. Abgesehen von einfachen Frage-Antwort-Authentifizierungen sind für lokale Anmeldungen die zugrunde liegenden Response-Berechnungen nicht ohne technische Hilfsmittel lösbar und damit wären sie als Authentifizierung auf Basis einer Kenntnis nicht anwendbar. Würde die Berechnung erst im lokalen System erfolgen, so ergibt das für die lokale Authentifizierung keinen Sinn. Als Bindeglied wäre eine *Mikrocontroller-Hardware* einzusetzen, welche zur Authentifizierung mit dem lokalen System zu verbinden ist und die Berechnung der richtigen Antwort auf die Anfrage übernimmt. Diese Hardware kann so gestaltet werden, dass neben der Ausführung der Berechnung auch der geheime Schlüssel des Nutzers $K_{user}$ sicher und nicht kopierbar innerhalb der Hardware aufbewahrt werden kann.
Die Hardware stellt einen mitzuführenden Besitz dar. Es kämen folgende Mikrocontroller-gestützte Systeme in Frage:

- USB-Stick mit eingebautem Prozessor und Speicher
- Chipkarte mit integriertem Mikrocontroller, auch bekannt als Smartcard oder als Signaturkarte
- Mobiltelefon mit Rechnerschnittstelle (Bluetooth oder USB)

Die Bestimmung der Response-Werte erfordern die Berechnung kryptographischer Funktionen, die vom jeweiligen Mikrocontroller effizient auszuführen sind. Zudem muss für die sichere Aufbewahrung des Schlüssels innerhalb der Hardware gesorgt werden. Transportable Hardware, die diese Funktionalität bietet, wird als Cryptographic-Token oder kurz **Crypto-Token** bezeichnet.
Der Begriff Token wird häufig verwendet. Neben Hardware-Token gibt es die Software-Token, welche innerhalb von Dateien gespeicherte Schlüssel in Verbindung mit zugehöriger Software beschreiben. Außerdem wird die Bezeichnung Token oder Security-Token auch für andere Hardware genutzt. So werden USB-Sticks, welche als Passworttresor verwendet werden (Kapitel 4.5.1), ebenfalls als Token bezeichnet. Weitere Arten sind die in USB-Stick-Format gefertigten Generatoren von Einmal-Passwörtern, deren Funktion im Kapitel 4.3.1 näher erläutert wird. Im Folgenden wird der Begriff Crypto-Token bzw. allgemein Token nur für Hardware-Token genutzt, die zur Durchführung kryptographischer Operationen in Verbindung mit sicherer Speicherung von Schlüsseln geeignet sind.

### 4.2.3 Biometrische Verfahren

Ein wesentlicher Nachteil aller besitzgestützten Verfahren ist die Tatsache, dass das Objekt, welches den Besitz darstellt, zur Authentisierung mitgeführt werden muss. Eine Alternative hierzu stellen Verfahren dar, die biometrische Daten einer Person als Authentisierungsmerkmal verwenden. Dazu fragen

sie bestimmte biometrische Merkmale ab, die hinreichend für die Identifizierung einer konkreten Person sind. Dies kann z.B. durch Prüfung der Fingerabdrücke, der Netzhaut, der Stimme oder anderer charakteristischer körperlicher Merkmale erfolgen.

Für eine ausreichende Sicherheit ist hoher technischer Aufwand am System, an dem die Authentisierung erfolgt, erforderlich. Die Hardware muss darauf ausgelegt sein, diese Informationen hinreichend genau und fälschungssicher aufzunehmen. So muss z.B. erkannt werden, ob es sich bei dem abgefragten Merkmal um das Merkmal einer lebenden Person handelt. Außerdem ist eine eindeutige Information, z.B. aus den Fingerabdrücken zu ermitteln, die mit einer in der Datenbasis hinterlegten Information verglichen werden kann und Aufschluss über die Identität des Inhabers gibt. Die möglichen Abweichung bei Messung der analogen körperlichen Merkmale muss ebenfalls berücksichtigt werden. Dies ist gerade bei preislich günstigen Geräten meist mangelhaft umgesetzt, zu denen z.B. viele in Notebooks integrierte Fingerabdruckscanner gehören. Höherwertige Modelle können aber gute Ergebnisse erzielen [Kri07]. Zur Erhöhung der Sicherheit können auch wieder weitere Merkmale hinzugezogen werden, z.B. die Eingabe einer PIN.

Ein problematischer Punkt ist die Tatsache, dass ein biometrisches Merkmal im Gegensatz zu einem technischen Merkmal dauerhaft mit der Person verbunden ist. Im Gegensatz können Passwörter geändert oder gelöscht werden, Smartcards vernichtet oder einer anderen Person übertragen werden. Es besteht aber keine Möglichkeit, die biometrische Identität abzulegen. Sind die Merkmale einmal aufgenommen, so können sie leicht vervielfältigt und im schlimmsten Fall später zweckentfremdet genutzt werden. Bei Kenntnis dieses Missbrauchs hat der Besitzer keine Möglichkeit, das Authentisierungsmerkmal zu wechseln. Somit besteht ein enormer Schutzbedarf für die hinterlegten Informationen, um im Sinne des Datenschutzes diese nur für den ursprünglich angedachten Authentisierungszweck zu verwenden.

Dies führt im Hinblick auf bekannt gewordene Datenpannen [Spi08] - deren Vermeidbarkeit über einen längeren Zeitraum absolut sicher zu stellen dürfte sich als unmöglich erweisen - unweigerlich zu einer gewissen, nicht unberechtigten Skepsis beim Einsatz biometrischer Methoden. Bereits ein Zeitpunkt, bei dem unbefugte Anwender Zugriff auf ungeschützte biometrische Daten erhalten, würde das Gesamtsystem gefährden, da keine neuen Merkmale generiert werden können.

Dieser Punkt in Verbindung mit dem hohen technischen Aufwand für Systeme, die eine bessere Erkennungsrate liefern, und die aktuell zu hohe Fehlerrate beim Einsatz preislich günstigerer Systeme lassen die Folgerung zu, dass der Einsatz biometrischer Authentifizierung im Umfeld einer Hochschule nur in speziellen Bereichen sinnvoll ist. Zweckmäßig ist er bei der Zugangskontrolle in Sicherheitsbereichen mit höherer Sicherheitsanforderung, bei denen durch qualitativ hochwertige Scanner in Kombination mit weiteren Merkmalen der Zutritt reguliert werden kann. Für den heimischen Gebrauch am privaten Rechner, bei dem es sich meist um ein sicherheitstechnisch eher unkritisches System handelt, reicht ein Fingerabdruckscanner als lokale Zugangskontrolle. Im Hochschulbereich mit mehreren Nutzern erfordert eine Authentisierung mit Fingerabdrücken hingegen erhöhten Aufwand für das Aufnehmen der Merkmale (ein zusätzlicher Lernschritt ist erforderlich) sowie für Absicherung und Verwaltung der biometrischen Daten innerhalb des lokalen Systems.

Für Anwendungen, die über Zutrittssysteme oder eine Anmeldung am eigenen Rechner hinausgehen, sind biometrische Verfahren somit eher ungeeignet. Um ohne zentrale Hinterlegung biometrischer Merkmale und ohne Übertragung dieser Merkmale ein Single Sign-On System zu ermöglichen, wäre zusätzlicher Aufwand erforderlich. Dies lässt den Schluss zu, dass biometrische Authentifizierung keine geeignete Grundlage für die genannten Anforderungen bildet.

## 4.3 Remote Authentication

Unter *Remote Authentication* ist die Authentifizierung an Diensten über das Rechnernetz zu verstehen. Hierbei handelt es sich um Dienste, die über das Netz erreichbar sind und eine Ermittlung der Identität des Nutzers, meist zur Bestimmung der Rechte des Nutzers erfordern.

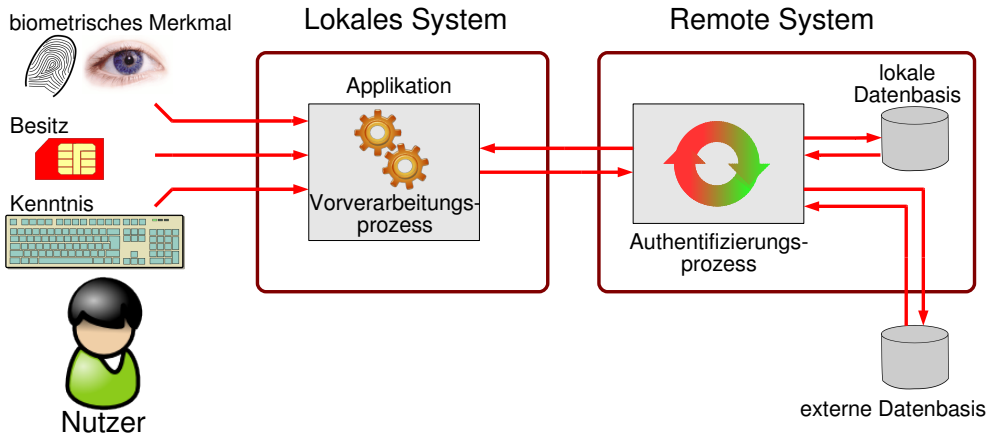



**Abb. 4.6 —** Remote Authentication

Das Prinzip der Verfahren für Remote-Authentication demonstriert Abbildung 4.6. Die Authentisierung erfolgt über eine Anwendung auf dem lokalen System, welche eine Verbindung zum Dienst auf dem Remote-System aufbaut. Zum Dienst gehört ein Authentifizierungsprozess, der mittels übertragener Authentisierungsinformationen auf die Identität des Nutzers schließt.
Der Ablauf der Authentifizierung zwischen der Anwendung und dem Dienst kann ein Prozess sein, der sich über mehrere Schritte mit zugehöriger Kommunikation erstreckt. Der Nutzer muss hierzu einer auf dem lokalen System laufenden Applikation Authentisierungsinformationen übergeben, die entweder direkt oder nach Vorverarbeitung durch die Applikation zum Authentifizierungsprozess auf dem Remote-System übertragen werden. Prinzipiell kann jede Art der Nutzerauthentifizierung gegenüber dem lokalen System auch für Remote Authentication genutzt werden, indem die Authentisierungsinformation vom lokalen System zum jeweiligen Dienst über das Netz übertragen wird. Damit ergibt sich zwangsläufig die Hürde der Absicherung gegenüber unbefugtem Abruf bzw. dem unbemerkten Kopieren der versandten Informationen. Dieser Schutz kann durch Aufbau eines durch Verschlüsselung gesicherten Kommunikationskanals erfolgen oder durch Sicherstellung, dass eine Information nur einmalig oder kurzzeitig verwendbar und damit für die meisten Angreifer wertlos ist.
Eine weitere Hürde besteht in der notwendigen Prüfung der Authentizität des Kommunikationspartners. Authentifizierung erfordert eine vorherige Prüfung der Legitimität des Empfängers, bevor Authentisierungsinformationen an ihn übertragen werden. Andernfalls könnte es sich bei dem vermeintlich richtigen Kommunikationspartner um einen Angreifer handeln. Diese Gefahr besteht auch bei Verwendung gesicherter Kanäle. Abbildung 4.7 verdeutlicht dieses Problem. Der obere Teil zeigt die Übertragung der Daten zum Authentifizierungsprozess über einen gesicherten Transportkanal, so wie es der Nutzer erwarten würde. Prüft der Nutzer aber nicht die Authentizität des Remote Systems, kann ein Angreifer die Verbindung auf das eigene System umleiten, wie es der unter Teil der Abbildung darstellt. Ein über den vermeintlich gesicherten Kanal übertragenes Passwort würde anschließend dem Angreifer vorliegen.

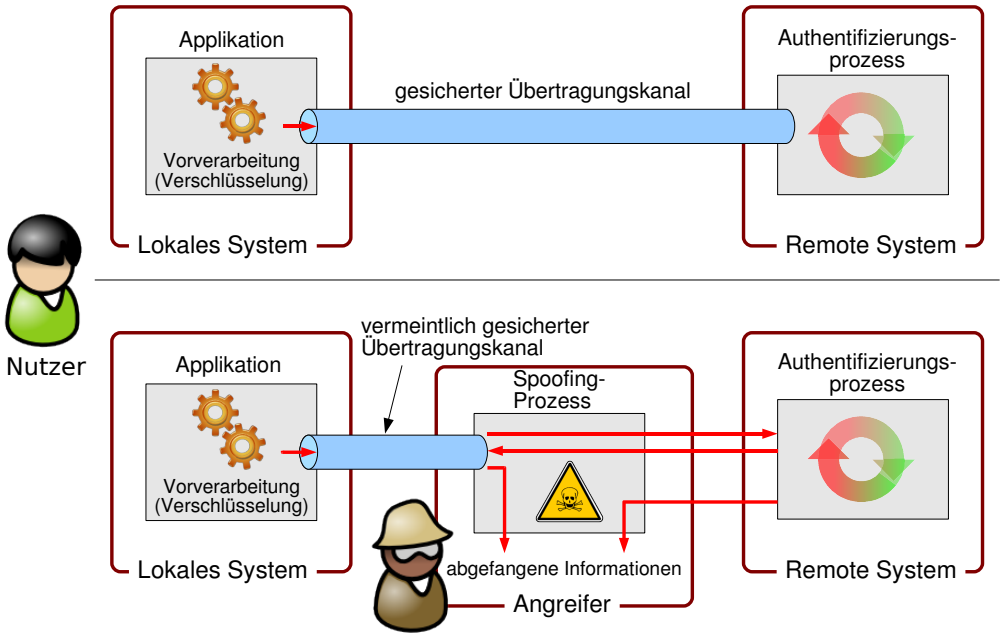



**Abb. 4.7 —** Remote Authentication mit verschlüsselten Kommunikationskanal

Ein ähnliches Problem existiert auch für die lokale Authentifizierung. Bei der Authentifizierung werden sensible Informationen an einen Prozess übertragen. Wenn es sich bei diesem vermeintlichen Systemprozess aber um eine Fälschung eines Angreifers im lokalen System handelt, so gelangt er ebenfalls an diese Informationen. Die Gefährdung bei der Nutzung über das Netz ist aber sicherlich höher, da es eine breitere Angriffsfläche bietet. Allgemein wäre es von Vorteil, wenn die gewählte Authentifizierungsvariante auch am lokalen System vor diesem Szenario Schutz bietet.

### 4.3.1 Kenntnis als Authentisierungsmerkmal

Die Nutzung einer Kenntnis als Authentisierungsmerkmal arbeitet wieder auf Basis eines gemeinsamen geheimen Wissens, in diesem Fall zwischen dem Serverdienst und dem Nutzer.

#### Statische Passwörter

Statische Passwörter können analog der Authentisierung am lokalen System genutzt werden, wenn ihre Übertragung gegen Abhören gesichert ist. Trotzdem ist die Verwendung von Passwörtern in Netzwerken problematischer und als Zugriffsschutz für hochsensible Informationen sind sie ungeeignet. Der Grund dafür ist, dass statische Passwörter für Netzdienste für einen Angreifer ein lohnenswertes Ziel darstellen. Mit einer entwendeten Login-Passwort-Kombination kann ein Angreifer über das Netz Zugriff auf den Dienst erhalten und benötigt demzufolge im Gegensatz zur lokalen Authentifizierung keinen physikalischen Zugriff auf den Rechner. Außerdem wird aufgrund des Nutzerverhaltens, bei dem ein Passwort gern für mehrere Dienste genutzt wird, der Anreiz für einen Angriff erhöht. Mit Methoden des Social Engineerings (siehe Phishing und Pharming in Kapitel 8.2) versuchen Angreifer deshalb oftmals gezielt Passwörter zu ergattern, insbesondere wenn diese letztendlich finanzielle Transaktionen ermöglichen.

### Einmalpasswörter

Eine Verbesserung gegenüber statischen bieten einmalig verwendbare Passwörter (*One-Time-Password* oder -*Code*).

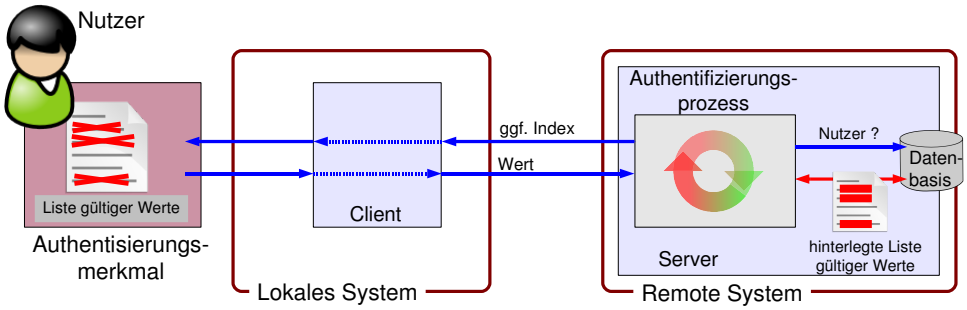



**Abb. 4.8 —** Verwendung einmalig gültiger Passwörter

Die Funktion von Einmalpasswörtern skizziert Abbildung 4.8. Bei diesem Verfahren haben Authentifizierungsprozess und Nutzer eine Liste von jeweils nur einmalig gültigen Passwörtern, bei der jedes Passwort nach Verwendung gestrichen wird. Eine praktische Anwendung finden Einmalpasswörter in Form von TANs[4]. Eine TAN dient als elektronische Bestätigung einer finanziellen Transaktion und stellt den Ersatz für die Unterschrift des Überweisenden dar.
Ein erfolgreicher Angriff während der Übertragung setzt voraus, dass der Angreifer die TAN kopieren und die Verbindung zum Server rechtzeitig unterbrechen kann. Der Angreifer besitzt dann ein einmalig gültiges Authentisierungsmerkmal. Im Rahmen finanzieller Transaktionen ist dies natürlich immer noch ein unbefriedigender Zustand. Eine gewisse Verbesserung bieten iTANs (indizierte TANs) als Form von Einmalpasswörtern, bei der nicht eine beliebige TAN auf Seiten des Nutzers ausgewählt werden kann, sondern der Authentifizierungsprozess den Index innerhalb der TAN-Liste vorgibt. Für einen Angriff muss das Opfer zur Übermittlung der richtigen iTAN motiviert werden. Außerdem muss der Angreifer diese iTAN sofort in der aktuellen Sitzung verwenden, da bei einer späteren, durch den Angreifer initiierten Sitzung seitens des Bank-Servers ein anderer Index erfragt werden sollte.
Die Ersetzung verbrauchter Passwörter ist ein Hindernis bei der Einführung von Einmalpasswort-Systemen. Passwort-Listen müssen zwischen Server und Anwender über einen (möglicherweise unsicheren) Kanal übertragen werden. Dies verursacht ähnliche Probleme, wie die Verteilung und Übertragung symmetrischer Schlüssel ohne die Nutzung asymmetrischer kryptographischer Verfahren. Werden nur selten neue Passwörter benötigt, so kann die Übertragung in Form von SMS-Mitteilungen auf das Mobiltelefon des Nutzers erfolgen. Praktischer Einsatz an einer Hochschule ist kaum möglich, da die Verteilung einen hohen organisatorischen Aufwand erfordert, verbunden mit Kosten für Versand auf dem Postweg oder per SMS. Außerdem wird angepasste Software benötigt, da viele Serverdienste meist nicht für Einmalpasswörter ausgelegt sind.

Eine weitere Variante der Einmalpasswörter besteht in der Generierung neuer Passwörter innerhalb eines Hardwaremoduls. Mittels einer vorgegeben mathematischen Vorschrift erzeugt das Modul jeweils einmalig gültig Werte. Modul und Dienst müssen dazu synchronisiert sein und einen gemeinsamen Schlüssel und Zähler zur Berechnung verwenden. Die Berechnung des Passwortes kann z.B. auf Knopfdruck erfolgen und das Ergebnis in einem kleinen Display angezeigt werden. Anschließend

[4] TAN: *Transaction Authentication Number*, dezimale Werte aus sechs bis acht Ziffern, organisiert in TAN-Listen von 25 bis 100 TANs.

wird beidseitig der Zähler erhöht, so dass die erneute Verwendung des alten Passwortes nicht möglich ist. Praktischen Einsatz findet diese Lösung z.B. für die Funktüröffner von Auto- oder Garagentüren. Allerdings zeigten die verwendeten Codes und eingesetzten Geräte technische Schwächen, die bereits erfolgreiche Angriffe mit geringem Aufwand ermöglichten [EKM+08]. Diese Angriffe beruhen aber auf hardwareseitig mangelhafter Umsetzung und zu kurzen Herstellerschlüssellängen, welche sich zukünftig beseitigen ließen.
Für den Einsatz im Hochschulbereich wären Passwort-Generatoren durchaus geeignet, allerdings stellt sich die Frage nach den Kosten für die Hardware und die Anpassung der Dienste. Für Single Sign-On sind Einmalpasswörter allein nicht ausreichend. Eine zusätzliche Erweiterung wäre erforderlich, die eine Information über die erfolgreiche Anmeldung an einem Dienst den anderen Diensten des SSO-Verbundes übermittelt.

### Zeitbeschränkte Passwörter

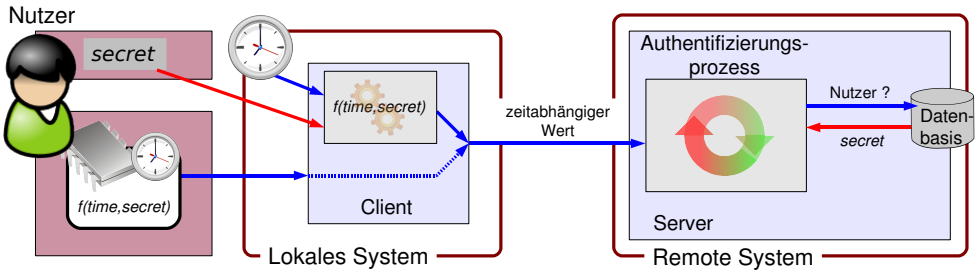



**Abb. 4.9 —** Verwendung zeitlich begrenzt gültiger Passwörter

Abbildung 4.9 illustriert ein weiteres Verfahren, welches sich für Remote-Authentication-Zwecke eignet. Grundlage bilden zeitlich beschränkt gültige Passwörter. Jedem Nutzer wird ein geheimer Wert - z.B. ein Kennwort *secret* - zugeordnet, der während der Authentifizierung auch dem Authentifizierungsprozess vorliegen muss. Unter der Voraussetzung, dass sowohl Authentifizierungsprozess und Client über die gleiche Zeit *time* bezüglich einer gemeinsamen Zeitbasis verfügen, kann die Authentifizierung durch Berechnung eines Wertes $f(time, secret)$ auf beiden Seiten mit anschließender Übertragung vom Client zum Authentifizierungsprozess erfolgen.
Die clientseitige Berechnung kann nach Eingabe von *secret* durch den Nutzer innerhalb des lokalen Systems mit der Systemuhr erfolgen. Die andere Variante besteht in der Nutzung eines Tokens mit integrierter Uhr und hinterlegtem Kennwort *secret*. Für dieses Token würde als Nutzerschnittstelle eine Taste und ein Display ausreichen. Bei einer erforderlichen Authentifizierung drückt der Nutzer die Taste, woraufhin das Token den für diesen Zeitraum gültigen Wert berechnet und im Display anzeigt. Vorteile der zweiten Variante sind die autarke Nutzung des Tokens ohne erforderliche Schnittstelle zum lokalen System und die Tatsache, dass die geheime Information *secret* das Token nicht verlassen kann und somit das Kopieren der Authentisierungsmerkmale nicht möglich ist.
Hinsichtlich der Sicherheit entspräche dieses System etwa einem Einmalpasswort-System, vorausgesetzt der berechnete Wert erlaubt praktisch gesehen keinen Rückschluss auf den geheimen Wert *secret* und der Gültigkeitszeitraum ist entsprechend knapp bemessen. Gängige Varianten arbeiten mit Zeiträumen von weniger als 5 Minuten, oftmals nur 30 Sekunden. Ein Angreifer hätte nur in diesem Intervall die Möglichkeit, abgefangene Werte missbräuchlich zu verwenden.
Im Gegensatz zu Einmalpasswortlisten ist das Verteilungsproblem auf das initiale Vereinbaren des

geheimen Wertes *secret* bzw. das Ausgeben der Token beschränkt. Ein anderes Problem entsteht bei einer zeitlichen Differenz zwischen der lokalen bzw. der Token-internen Uhr und dem Serversystem, da dies eine Authentifizierung prinzipiell verhindern würde. Deshalb ist eine Zeitsynchronisation mit einer Zeitquelle erforderlich. Hinsichtlich Single Sign-On ist das Verfahren vergleichbar mit der Einmalpasswort-Variante, benötigt also ebenfalls eine Erweiterung der Serverdienste.

### 4.3.2 Besitzbasierte Verfahren im Rechnernetz

Das bereits im vorangegangenen Kapitel vorgestellte Verfahren unter Nutzung von Token entspräche einem besitzgestützten Verfahren. Zusätzlich können die clientseitig agierenden besitzgestützten Verfahren auch für Remote Authentication genutzt werden, wenn die Daten zwischen dem serverseitig laufenden Authentifizierungsprozess und der Hardware über das Netzwerk mit der Clientanwendung ausgetauscht werden.

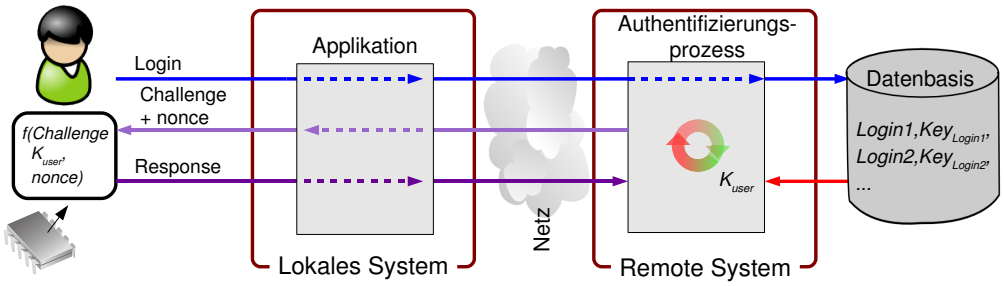



**Abb. 4.10 —** Challenge-Response mit Nutzerschlüsseln über ein Netzwerk

So stellt die Abbildung 4.10 eine einfache Erweiterung der in Abbildung 4.5 gezeigten Challenge-Response-Variante unter Verwendung von Hardware-Token dar. Die Authentifizierung, welche als Teil des Serverprozesses läuft, kommuniziert über die Client-Applikation im lokalen System mit der Hardware, die anschließend die Berechnung der clientseitigen Antwort unter Anwendung des Nutzerschlüssels durchführt.
Dieses Verfahren bietet sich für Remote Authentication insbesondere dann an, wenn bereits eine entsprechende Hardware verfügbar ist. Da nur einmalig verwendbare Datenkombinationen zwischen Server und Client ausgetauscht werden, ist die Gefahr der missbräuchlichen Nutzbarkeit abgefangener Informationen gering, vorausgesetzt das genutzte kryptographische Verfahren gilt als sicher. Für den Anwender auf Client-Seite besteht aber noch die bereits erwähnte Gefahr, dass Daten zu einem kompromittierten Server übertragen werden könnten. Deshalb muss die Authentizität des Servers, z.B. über Zertifikate geprüft werden.

#### Zertifikatsbasierte Authentifizierung

Für zertifikatsbasierte Authentifizierung stellt ein Schlüsselpaar den Besitz dar, Dieses muss sich zum Transport auf einem geeigneten Medium befinden. Die Zuordnung zum Inhaber bescheinigt ein Nutzer-Zertifikat, welches zum öffentlichen Schlüssel ausgestellt wurde.
Benötigt wird eine Infrastruktur für die Zertifikatsausstellung und -verwaltung, eine PKI. Neben Nutzer-Zertifikaten können auch Server-Zertifikate ausgestellt werden. Sie ermöglichen dem Anwender die Prüfung der Authentizität der Serverdienste. Einen möglichen Kommunikationsablauf für die wechselseitige Authentifizierung von Server und Client zeigt Abbildung 4.11. Die Werte *cnonce*

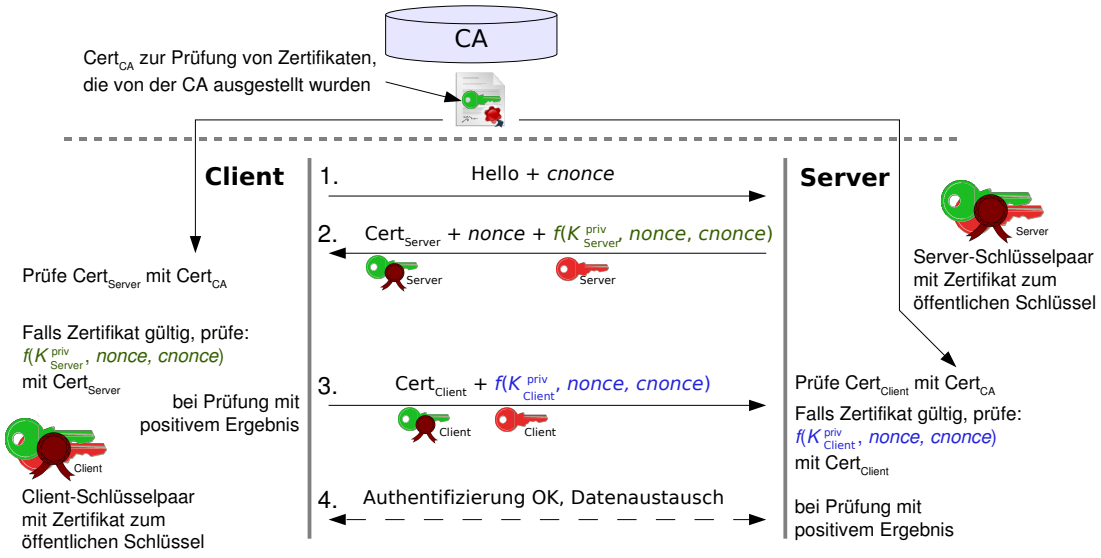



**Abb. 4.11 —** Wechselseitige Authentifizierung mit Zertifikaten

und *nonce* stellen die Challenge vom Client zum Server und umgekehrt dar, welche zusammen mit dem privaten Schlüssel als Eingabe der Funktion $f$ verwendet werden. Ein standardisiertes Verfahren, welches eigentlich dem Aufbau eines verschlüsselten Transportkanals zwischen Client-Applikationen und Serverdiensten dient, dabei aber diese Art der wechselseitigen Authentifizierung erlaubt, ist das SSL/TLS-Protokoll.

**SSL/TLS:** SSL (Secure Sockets Layer) aktuell in der Version 3, und TLS (Transport Layer Security), aktuell in Version 1.2, unterscheiden sich inhaltlich geringfügig, das Handshake-Protokoll und damit die Authentifizierung sind aber identisch [RFC2246, RFC5246]. Abbildung 4.12 stellt den Ablauf dar.

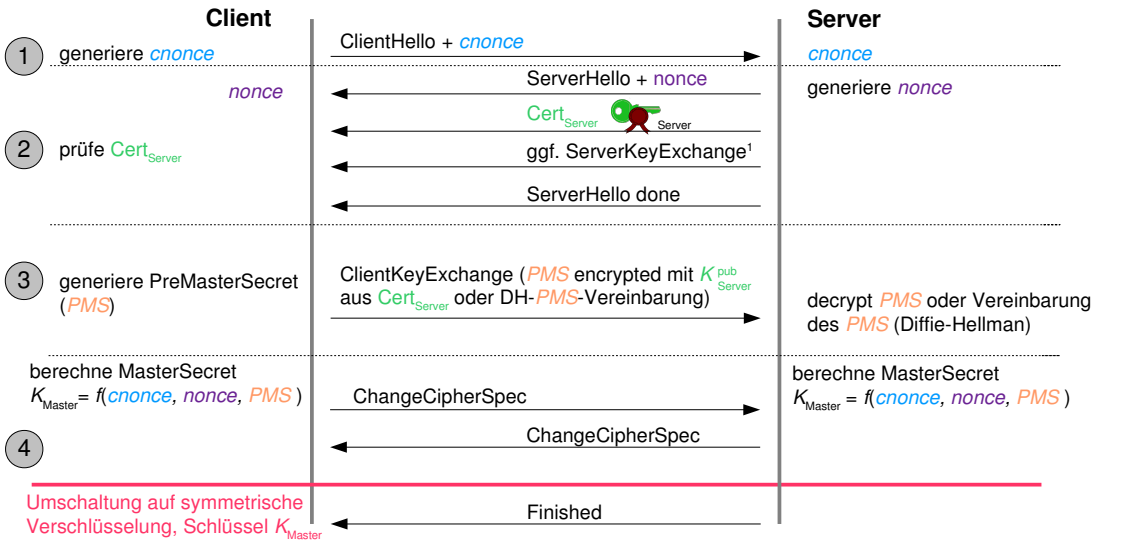



**Abb. 4.12 —** TLS-Handshake nach RFC5246, nur Authentifizierung der Serverseite

Das Handshake-Protokoll läuft in mehreren Schritten ab und dient neben der Authentifizierung dazu, den gemeinsamen symmetrischen Schlüssel für die folgende Kommunikation zu berechnen. Die Clientanwendung startet im Schritt 1 den Handshake mit einer Hello-Nachricht, welche einen *cnonce*-Wert enthält. Der Serverdienst antwortet im Schritt 2 mit seinem *nonce*-Wert und übermittelt sein Zertifikat. Clientseitig kann anschließend das Zertifikat geprüft und entschieden werden, ob die Kommunikation abgebrochen oder fortgesetzt wird. Das Zertifikat enthält den öffentlichen Schlüssel des Servers. Nur der Serverdienst, der auch den zugehörigen privaten Schlüssel besitzt, kann die folgenden Schritte durchführen und die vom Client übertragenen Werte dekodieren. Somit authentifiziert sich der Server gegenüber dem Client. In Schritt 3 wird clientseitig ein weiterer Wert (PreMasterSecret *PMS*) generiert und zum Server übertragen bzw. mit dem Server vereinbart. Aus den auf beiden Seiten vorliegenden Werten *PMS*, *cnonce* und *nonce* wird im letzten Schritt der symmetrische Ausgangsschlüssel für die anschließend folgende Kommunikation berechnet und die Verschlüsselung aktiviert.

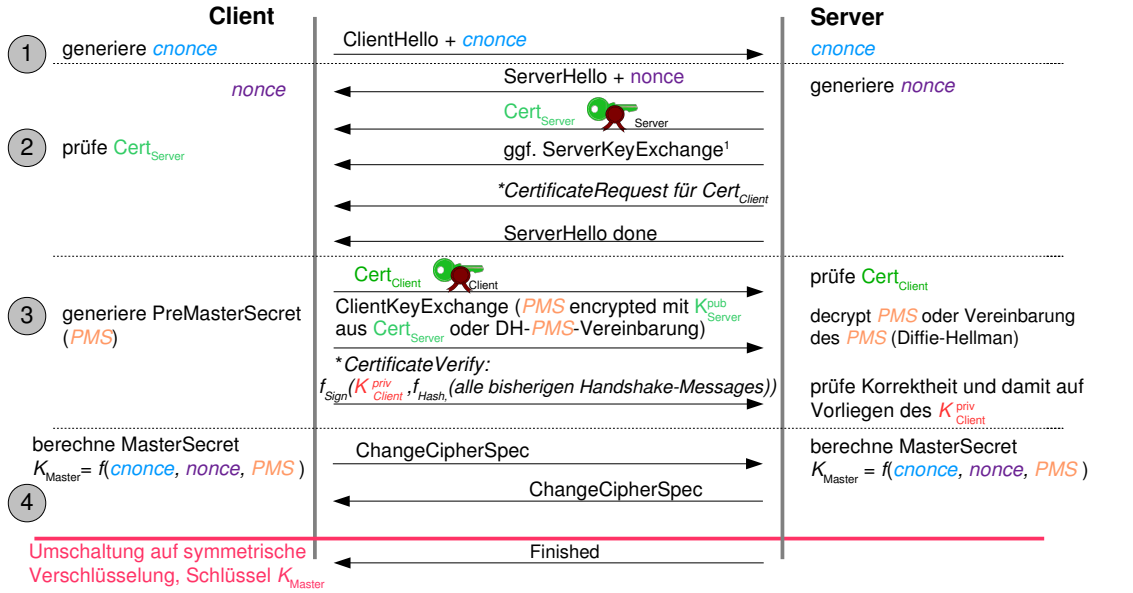



**Abb. 4.13 —** TLS-Handshake, Erweiterung zur wechselseitigen Authentifizierung

Bei diesem Handshake weist nur der Server seine Authentizität nach. Das Protokoll sieht aber auch die wechselseitige Authentifizierung vor (Abbildung 4.13). Der Serverdienst fordert dafür im Schritt 2 ein Client-Zertifikat an, welches der Client im Schritt 3 überträgt. Dabei kann es sich um das Zertifikat des Nutzers handeln und wäre somit eine Nutzerauthentifizierung. Zur Prüfung der Authentizität muss serverseitig ermittelt werden können, ob der Client auch über den zugehörigen geheimen Schlüssel verfügt. Dazu wird ebenfalls im Schritt 3 ein Wert an den Server übermittelt, der eine Signatur mit dem geheimen Schlüssel des Client enthält. Diese Signatur umfasst einen Hashwert über alle vorher ausgetauschten Handshake-Nachrichten. Dieser Hashwert muss auf beiden Seiten berechnet werden. Der Server kann die Nachricht mittels des im Client-Zertifikat enthaltenen öffentlichen Schlüssels prüfen, insofern das Client-Zertifikat, welches der Server vorher prüft, als gültig befunden wird.
Die wechselseitige Authentifizierung mittels Zertifikaten ist theoretisch in allen Applikationen möglich, die eine Datenübertragung über SSL- oder TLS-gesicherte Kanäle unterstützen. Ob dies für eine konkrete Applikation auch wirklich funktioniert, hängt aber von der jeweiligen Implementierung ab.
SSL und TLS sind im ISO/OSI-Referenzmodell in den oberen Schichten angesiedelt. Die Sicherung

der Datenübertragung kann auch in darunter liegenden Schichten erfolgen. Dies wäre der Fall, falls eine VPN- bzw. IPsec-Verbindung zur Übertragung gekapselter Daten über ein unsicheres öffentliches Netz verwendet wird. Dafür existieren analoge Verfahren zur wechselseitige Authentifizierung unter Verwendung von Zertifikaten [Lip07, BK01].
Neben der Authentifizierung des Nutzers über ein clientseitiges Nutzer-Zertifikat besteht natürlich auch die Möglichkeit, dass in einem gesicherten Kanal weiterhin Passwörter oder andere Authentisierungsinformationen übertragen werden. Dies kann von Vorteil sein, da verschlüsselte Kanäle gleichzeitig eine Fallback-Lösung zur Authentifizierung ermöglichen, wenn keine Nutzer-Zertifikate vorhanden sind.

### 4.3.3 Biometrische Verfahren

Die lokale Authentifizierung auf Basis biometrischer Sensoren verlangte einen erhöhten Aufwand im Vergleich zu Passwörtern. Bei einer Anwendung für Client-Server-Authentifizierung über ein Netzwerk kommen weitere Hürden hinzu. Aufgrund des besonderen Schutzbedarfs biometrischer Merkmale muss die Übertragung gesichert werden, so dass ein Abhören und Kopieren der Information verhindert wird. Wichtig wäre die Verwendung verschlüsselter Kanäle und eine Prüfung der Authentizität des Servers.
Bei biometrischen Merkmalen ist es außerdem leicht möglich, eine einmalig aufgezeichnete Information erneut zu verwenden. Werden diese Informationen übertragen, so ergibt sich das Problem der kopierbaren Identität. Eine Abhilfe mittels Challenge-Response-Verfahren ist wesentlich schwieriger. So muss aus der analogen, fehlerbehafteten biometrischen Information ein eindeutiger digitaler Wert generiert werden, falls man diesen als *secret* für die Response-Berechnung nutzen möchte. Da die Gewinnung eines eindeutigen Wertes schwer möglich ist, werden die Informationen entweder direkt zum Authentifizierungsprozess übertragen, oder die Authentifizierung wird auf das lokale System oder besser auf das biometrische Lesegerät verlagert. In diesem Falle könnte der Leser bereits über die Authentifizierung entscheiden und die Information an den Server übertragen. Der Server hätte aber in beiden Fällen allein durch Erhalt der Information vom Client keine Garantie, dass es sich bei dem übertragenen Ergebnis um eine authentische Information handelt, da die Daten kopiert oder der Reader manipuliert oder ausgetauscht sein könnte. Um dies zu verhindern, muss zuerst ein Schritt der wechselseitigen Authentifizierung von Reader und Serverdienst erfolgen. Der Reader weist sich als vertrauenswürdiges Gerät gegenüber dem Serverdienst aus [WSE04].
Da bereits für die lokale Authentifizierung ein hoher Aufwand beim Einsatz biometrischer Varianten entsteht und dieser für Remote Authentication aufgrund der Absicherung übertragener Information weiter erhöht wird, scheint biometrische Authentifizierung derzeit keine geeignete Variante für den Hochschuleinsatz zu sein. Sie könnte jedoch als Ergänzung in Bereichen mit höherer Anforderung an die Authentifizierung genutzt werden. Bei verbesserter preislicher Attraktivität qualitativ hochwertiger Lesegeräte mit hoher Erkennungsrate könnte sie in Zukunft eine stärkere Rolle spielen.

## 4.4 Mehrfaktor-Authentisierung

Als Mehrfaktor- oder Multifaktor-Authentisierung werden Verfahren bezeichnet, bei denen zur erfolgreichen Authentifizierung mehrere Authentisierungsmerkmale erforderlich sind. Eine typische Anwendung besteht in der Kombination von besitzbasierten Methoden in Verbindung mit der Kenntnis eines geheimen Wertes. Die bereits erwähnte Kombination von EC-Karten mit PIN wäre ein Beispiel. Eine im biometrischen Umfeld für lokale Authentifizierung nutzbare Zwei-Faktor-Kombination ist die

Verknüpfung biometrischer Authentifizierung mit kontaktlosen Smartcards. Dies würde die Speicherung der biometrischen Daten in der Karte und somit in den Händen des Karteninhabers ermöglichen. Bei der Authentifizierung müssen sich Karte und Inhaber am Leser befinden. Der Leser prüft die biometrischen Daten gegen die Referenzdaten aus der Karte, eine Speicherung im Leser oder an zentraler Stelle wäre unnötig. Das gewählte Kartensystem muss dabei verhindern, dass die enthaltenen Referenzdaten nach dem Enrollment-Prozess von unbefugter Stelle ausgelesen oder verändert werden können. Eine verloren gegangene Karte würde dann in den Händen einer fremden Person keine Authentifizierung ermöglichen.
Eine andere Mehrfaktor-Authentisierung auf Basis eines Besitzes ist die erwähnte zertifikatsbasierte Authentifizierung, bei der zum Schlüsseltransport ein Crypto-Token, z.B eine Smartcard genutzt wird. Zum Schutz vor missbräuchlicher Nutzung nach Entwendung ist ein zusätzliches Authentisierungsmerkmal notwendig, das gegenüber der Karte vorzulegen ist. Dazu könnte wieder ein geheimes Wissens, also eine persönliche PIN genutzt werden. Um ein erschöpfendes Austesten verschiedener PINs, bei denen man sich meist auf einen vier- bis sechsstelligen numerischen Wert beschränkt, zu verhindern, müssen weitere Schutzmaßnahmen getroffen werden. Dies kann durch eine Begrenzung der möglichen PIN-Eingabe-Versuche seitens der Karte erfolgen.

## 4.5 Single Sign-On

Ein Single Sign-On Verfahren soll Anwender in die Lage versetzen, nach einmaliger Anmeldung an einem Dienst des SSO-Verbundes sämtliche weiteren Dienste ohne erneute Authentifizierung nutzen zu können, vorausgesetzt, der Anwender besitzt die nötige Berechtigung.
Umfassen die Dienste des SSO-Verbundes viele verschiedene Anwendungen und damit verschiedene Protokolle, dann erfordert dies speziell angepasste SSO-Software, entweder auf Seiten der Dienstanbieter oder auf Seiten der Nutzer oder bei beiden. Für eine Hochschule darf die SSO-Lösung nicht nur einen Applikationsbereich abdecken, sondern sollte die häufig genutzten Dienste unterstützen und zugehörige Client-Applikationen (Webbrowser, Emailreader, SecureShell u.v.m.) einbinden.

### 4.5.1 Passwort-Tresor

Unter dem Begriff Passwort-Tresor versteht man die gesicherte Verwahrung der Passwörter. Dies kann in einer verschlüsselten Datei auf einem Speichermedium wie der lokalen Festplatte oder einem USB-Stick erfolgen. Die Passwörter können auch innerhalb eines Hardware-Tokens oder verschlüsselt in einem Netzdienst, z.B. einem LDAP-Server abgelegt sein.
Mittels eines Passwort-Tresors kann Single Sign-On in Applikationen mit passwortgestützter Authentifizierung eingebaut werden. Dazu wird bei jeder Passwort-Authentifizierung das Passwort-Feld automatisch durch den SSO-Service nach Zugriff auf die im Tresor gespeicherten Werte ausgefüllt.
Ein Passwort-Tresor kann bereits innerhalb einer einzelnen Applikation, z.B. den Webbrowsern Firefox oder dem Internet-Explorer integriert sein. Der Nutzer muss sich nach dem Browser-Start einmalig, meist durch Eingabe eines Masterpasswortes gegenüber dem Browser authentisieren. Bei Abruf einer HTTP-geschützten Webseite[5] bzw. einer Webseite mit einem Passwort-Eingabefeld greift der Browser auf den Tresor zu und füllt die Eingabefelder aus, so dass der Nutzer diese Eingabe nur noch bestätigen muss. Diese Art des Tresors erlaubt jedoch nur die Verwendung innerhalb der jeweiligen Applikation, wäre also nicht dienstübergreifend nutzbar.

[5] Bei HTTP-geschützten Webseiten wird der Zugriff auf die Inhalte durch eine Authentifizierungsfunktion im HTTP-Übertragungsprotokoll kontrolliert. Die Browser öffnen hierzu ein weiteres Fenster für die Login/Passwort-Eingabe.

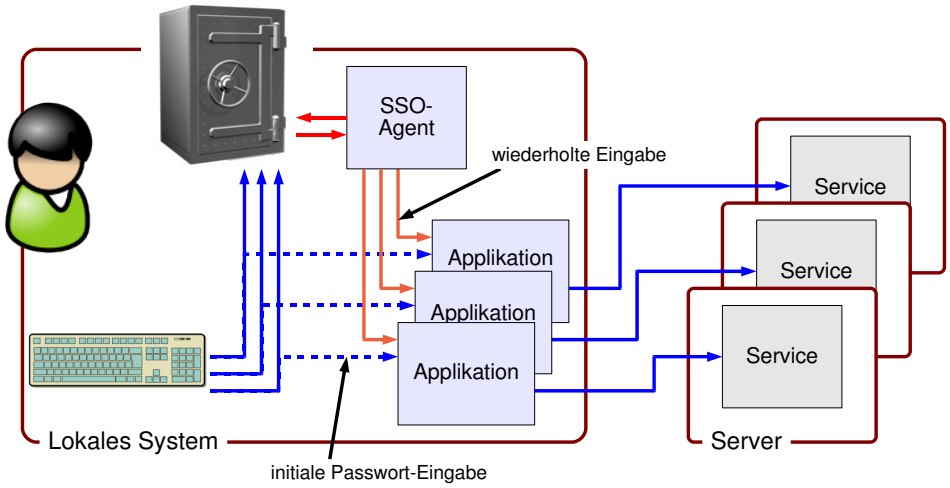



**Abb. 4.14 —** Passwort-Tresor

Andere Tresor-Lösungen arbeiten auf Basis einer clientseitigen Software, welche automatisch Passworteingabefenster von Applikationen erkennt, diese ausfüllt und bestätigt. Die Tresor-Software läuft als Dienst im Hintergrund und wird als SSO-Agent bezeichnet. Der SSO-Agent ist applikationsunabhängig und kann applikationsübergreifend verwendet werden. Im Prinzip handelt es sich um eine clientseitige Lösung, die bereits die gewünschte SSO-Funktionalität ermöglicht, allerdings noch auf der Basis von Passwörtern arbeitet. Die Nutzer müssen ein Passwort nur einmal für den Dienst und für den Lernprozess des SSO-Agenten eingeben. Sie könnten somit „sichere“ Passwörter verwenden. Die SSO-Lösung bietet aber keinen Sicherheitsgewinn, wenn die Qualität der Passworte nicht geprüft wird oder das gleiche Passwort für verschiedene Applikationen genutzt werden kann.

Ein Nachteil aller Passwort-Tresor-Lösungen besteht darin, dass eine Änderung des Passwortes am Dienst meist ein Neu-„Lernen“ für die SSO-Software erfordert, insofern sie nicht in der Lage ist, das Passwort bei der Änderung automatisch zu übernehmen. Während die Übernahme für applikationsinterne Tresor-Lösungen noch relativ gut funktioniert, stellt es sich bei applikationsübergreifenden SSO-Agenten als Problem dar [WP07]. Diese können die Felder, in denen das Passwort automatisch eingefügt werden soll, anhand des Fenster- oder Widget-Namens oder dessen Position innerhalb des Anwendungsfensters erkennen. Eine Änderungen der Anwendung kann somit dazu führen, dass der Agent nicht wie erwartet arbeitet. So ist bei jedem Programm-Update auch die SSO-Agentsoftware zu prüfen, was zu erhöhtem Wartungsaufwand führt.

Sollen die im Tresor gespeicherten Passwörter an mehreren Rechnern nutzbar sein, so ist für sicheren Transport zu sorgen, insofern der Tresor nicht bereits als Netzdienst ausgelegt ist. Kommerziell verfügbare Systeme mit SSO-Agenten nutzen Smartcards oder USB-Token, welche nach Eingabe der PIN den Zugriff auf den enthaltenen Tresor und dessen Daten gestatten. Dazu muss der SSO-Agent auf den verwendeten Rechnern installiert werden. Da es sich um proprietäre Anwendungen handelt, die untereinander keinen Austausch der Tresore ermöglichen, bindet man sich bei der Wahl an einen Anbieter und dessen Lösung. Als Beispiel seien hier die Produkte *Aladdin eToken Single Sign-On*[6] oder *SafeNet/Entrust Borderless Security SSO*[7] genannt. Letzteres bietet zusätzlich auch zertifikatsbasierte Authentifizierung, ist aber auf Token des Herstellers SafeNet beschränkt.

[6] `http://www.aladdin.com/etoken/single-sign-on.aspx`
[7] `http://www.safenet-inc.com/partners/partner_briefs/entrust.asp`

Für den hochschulweiten Einsatz sind diese Systeme ungeeignet. Sie sind preislich höher angesiedelt (Zukauf von Token und SSO-Agent-Software des jeweiligen Herstellers) und sehr pflegeintensiv. Das Prinzip eines lokal vorliegenden Authentisierungsmerkmals, welches von einem Agenten bei Bedarf vorgelegt wird, kann jedoch aufgegriffen werden. Dazu wäre es erforderlich, die Passwörter durch andere, geeignetere Merkmale zu ersetzen. Diese sollten sich vor Duplizierung schützen lassen. Wenn ein einzelnes Merkmal dann auch noch ohne Sicherheitseinschränkung für verschiedene Dienste angewendet werden kann und nur selten geändert, also neu gelernt werden muss, kann daraus ein clientseitiges SSO-Systems aufgebaut werden. Geeignet wären Zertifikate in Verbindung mit einem durch den Agenten kontrollierten Zugriff auf den zugehörigen Schlüssel. Dann blieben nur noch die Fragen des Schlüsseltransports und der Einbindung in bestehende Applikationen zu klären.

### 4.5.2 Kerberos

Tresor-Lösungen basieren darauf, bestehende passwortgestützte Authentifizierung durch einen clientseitigen Automatismus zu ergänzen. Im Gegensatz dazu ist Kerberos-Authentifizierung als Netzwerk-Authentifizierungsprotokoll mit einer zentralen „vertrauenswürdigen“ Instanz ausgelegt. Gegenüber dieser Instanz muss sich der Nutzer anfänglich authentisieren. Diese Instanz stellt somit den Identity Provider dar. Nach erfolgreicher Authentifizierung können alle beteiligten „kerberisierten“ Dienste des Kerberos-Verbundes ohne erneute Authentifizierung genutzt werden, was ein applikationsübergreifendes SSO ermöglicht. Hierfür ist jedoch sowohl die client-, wie auch die serverseitige Auslegung der Applikation für Kerberos erforderlich.

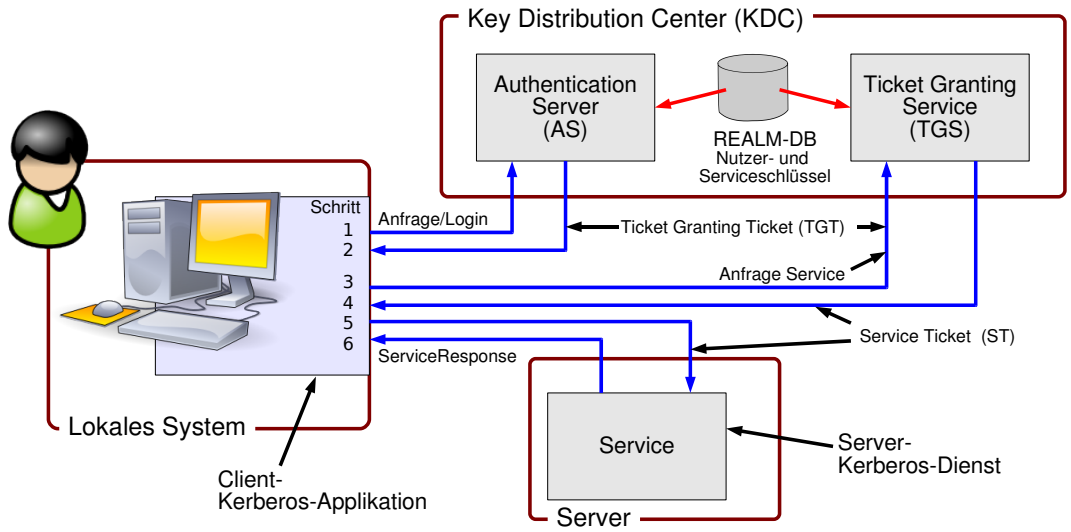



**Abb. 4.15 —** Funktionsweise Kerberos

Den Aufbau des Kerberos-System skizziert Abbildung 4.15. Die zentrale Rolle des Identity Providers ist Aufgabe des *Key Distribution Centers* (KDC).
In Kerberos (aktuell in Version 5 [RFC4120]) erfolgt die Übertragung der Informationen in Form von Tickets, deren Inhalt durch Verschlüsselung geschützt ist. Der Ablauf der Authentifizierung erfolgt in mehreren Schritten, bei denen Tickets zwischen KDC und der lokalen Applikation oder zwischen der lokalen Applikation und den Diensten übertragen werden.

1. Der Nutzer meldet sich bei dem *Authentication Server* (AS), einem Teil des KDC für eine Session an.
2. Der AS antwortet mit einem generierten *Ticket Granting Ticket* (TGT), verschlüsselt mit dem TGS-Key (Key des *Ticket Granting Service*, dem anderen Teil des KDC) und einem weiteren Ticket, welches den für die Sitzung gültigen TGS-SessionKey, verschlüsselt mit dem Schlüssel des Nutzers enthält. Im Standard vorgesehen ist die Gewinnung des Nutzer-Schlüssels aus dem Nutzer-Passwort.
3. Für die Authentifizierung gegenüber einem Dienst erfragt der Client vom TGS ein Ticket für diesen konkreten Serverdienst unter Nutzung des TGT. Der TGS kann den Inhalt des TGT entschlüsseln, da dieses mit dem TGS-Key geschützt war, und entnimmt daraus den Nutzer.
4. Falls es die Policy erlaubt, generiert der TGS ein *ServiceTicket* (ST) für den Nutzer.
5. Mit diesem ST kann sich der Client bei dem konkreten Dienst anmelden.
6. Der Serverdienst antwortet dem Client und - falls der Server die Anfrage positiv beantwortet - kann im Anschluss die Kommunikation beginnen.

Nach einer einmaligen Authentifizierung am AS (Schritt 1 und 2) können für die Dauer der Sitzung Anmeldungen an verschiedenen Diensten (Schritte 3 bis 6) ohne erneute Nutzerauthentifizierung erfolgen.
Schlüssel spielen für das System eine zentrale Rolle. Der Nutzer-Schlüssel, gewonnen aus dem Passwort, dient der Verschlüsselung des Session-Key und damit auch der Authentifizierung. Der Nutzer kann den Session-Key nur bei Kenntnis des Passwortes entschlüsseln. Dieser TGS-Session-Key, welcher im Schritt 2 mit dem TGT übermittelt wird, dient für die Dauer der Sitzung der Sicherung sämtlicher Kommunikation zwischen TGS und Client. Ein weiterer Service-Session-Key wird im Schritt 4 vom TGS vergeben und dient zur geschützten Kommunikation zwischen Client und dem Serverdienst. Die Informationen im TGT wurden mit dem TGS-Key verschlüsselt und dieser muss somit sowohl dem AS, als auch dem TGS bekannt sein. Außerdem gibt es den Service-Key, mit dem das ServiceTicket (ST) für Schritt 4 und 5 verschlüsselt wird. Dieser Service-Key muss sowohl dem TGS, als auch dem Service bekannt sein. Somit werden neben den temporären TGS- und Service-Session-Keys auch statische Schlüssel für die Nutzer und die Dienste benötigt. Diese müssen im KDC in einer gut geschützte Datenbank verwahrt werden.
Die ausgestellten Tickets sind mit einem Zeitstempel versehen und zeitlich begrenzt, was eine Zeit-Synchronisierung aller beteiligten Parteien erfordert. Dies verhindert einen unbegrenzten Missbrauch gestohlener Tickets bzw. erschwert einen Angriff auf die zugrunde liegende Verschlüsselung, da dieser innerhalb des Zeitraumes Erfolg haben muss.
Wesentlicher Vorteil von Kerberos ist die einheitliche Lösung für beliebige Netzwerk-Applikationen, welche allerdings vorher für Kerberos angepasst werden müssen. Die Kommunikation kann über ungeschützte Kanäle erfolgen, da im Protokoll die ausgetauschten Authentisierungsinformationen durch symmetrische Verschlüsselung geschützt werden. Seit der Version 5 bietet Kerberos neben der initialen Passwort-Authentifizierung weitere Möglichkeiten, z.B. eine Hardware-gestützte Authentifizierung mit Crypto-Token.
Kerberos hat aber auch Nachteile. So ist neben der notwendigen Anpassung der Server- und Client-Applikationen für die Authentifizierung ein weiteres sensibles System, das TGC mit ständiger Verfügbarkeit erforderlich. Da hiervon die Möglichkeit zur Authentifizierung innerhalb des gesamten Kerberos-Realm[8] abhängt, muss das TGC gegen Ausfall geschützt werden. Gegebenenfalls müssen

[8] Ein Kerberos SSO-Verbund wird als *Realm* (engl. für Bereich oder Königreich) bezeichnet und umfasst den Bereich, für den ein TGC die notwendigen Schlüssel enthält. Meist entspricht dies einer Internet-(Sub-)Domain.

sämtliche Teile inklusive der Datenbank redundant und unabhängig voneinander aufgebaut sein. In der Datenbank des TGS liegen alle Authentifizierungs-Schlüssel bzw. Klartext-Passwörter aller Nutzer des Realm. Da das TGC die zentrale vertrauenswürdige Instanz - ähnliche der CA innerhalb einer PKI - ist, muss der Nutzer auf den Schutz dieser Daten vertrauen können. Anders als bei einer CA müssen jedoch AS und TGS über das Netz erreichbar sein und bieten somit eine gewisse Angriffsfläche. Ist die Erreichbarkeit des TGC, z.B. aufgrund eines DoS-Angriffs[9] eingeschränkt, sind davon alle Nutzer und Dienste betroffen.

Viele Applikationen in der Unix- und Windows-Welt wurden bereits für Kerberos angepasst. So nutzt Microsoft Windows für die Netzwerk-Authentifizierung ein Kerberos-System, welches die Nutzerdaten im *Active-Directory*, einer Datenbasis ähnlich einem LDAP-Server speichert. Allerdings wurden Protokollerweiterung hinzugefügt, was eine Interoperabilität mit anderen Betriebssystemen und Applikationen, die z.B. auf Basis der freien Kerberos-Implementierung des MIT laufen, erschwert [WD01].

### 4.5.3 SSO mittels Zertifikaten und PKI

Die im Kapitel 4.3.2 gezeigten Verfahren zur Authentifizierung mittels Zertifikaten können in Verbindung mit Single Sign-On genutzt werden. Die Serverdienste müssen dazu die Anmeldung mit Zertifikaten erlauben. Das Client-Zertifikat (in diesem Fall wäre es das Nutzer-Zertifikat) und das Server-Zertifikat müssen von einer Instanz ausgestellt werden, der vom jeweiligen Kommunikationspartner vertraut wird. Dafür ist eine PKI erforderlich. Nach der Übermittlung des Zertifikats kann die jeweilige Gegenseite dessen Gültigkeit mit den CA-Zertifikaten prüfen und, falls gültig, sich unter Nutzung des eigenen privaten Schlüssel authentisieren. Dieser Ablauf kann im Rahmen eines SSL/TLS-Handshakes erfolgen und wird von vielen Diensten unterstützt.

Sind diese Bedingungen erfüllt, wäre ein applikationsübergreifendes SSO möglich, vorausgesetzt die Client-Applikationen haben Zugriff auf das Client-Zertifikat und den zugehörigen privaten Schlüssel. Erfolgreiche Authentifizierung hängt somit von der Bereitstellung von Zertifikat und Schlüssel ab. Da der Schlüssel besonders geschützt werden muss, kann er mit einem Passwort gesichert sein. Die initiale Authentifizierung erfolgt somit durch Freischaltung des Schlüssels mittels Eingabe des Passwortes, so dass anschließend alle Client-Applikationen darauf zugreifen können.

Im Gegensatz zu Kerberos wird kein drittes Online-System zur Authentifizierung benötigt. Der Vorteil gegenüber Tresor-Lösungen besteht in der Vermeidung von Passwörtern, da somit weder triviale Passwörter, noch die Mehrfach-Nutzung eines Passwortes möglich sind. Ein weiterer Vorteil ist, dass serverseitig keine geheimen Informationen abgelegt werden werden müssen.

Der Nachteil besteht in der sicheren Aufbewahrung und dem sicheren Transport des geheimen Schlüssels. Außerdem muss sichergestellt werden, dass nur autorisierte Client-Applikationen Zugriff erhalten. Weiterhin ist die Frage zu klären, welchem Aussteller von Zertifikaten vertraut werden kann. Für eine abgeschlossene Einrichtung kann dieses Problem meist durch eine lokale PKI gelöst werden. Verlässt man aber den Bereich und fokussiert auf eine globale Lösung, so kann die Frage nach dem passenden Identity Provider (in diesem Falle die passende CA) ein schwer lösbares Problem werden. Hier wurden verschiedene Ansätze entwickelt, die im folgenden Kapitel auszugsweise vorgestellt werden.

[9] DoS: *Denial-of-Service* ist ein Angriff auf einen Rechner oder Dienst, mit dem Ziel, die Erreichbarkeit einzuschränken oder zu verhindern.

### 4.5.4 Verteilte Authentifizierung und Federated Identity

Unter Nutzung von Identity Providern existieren neben Kerberos weitere applikationsübergreifende Authentifizerungsverfahren mit Unterstützung von SSO und ohne Abhängigkeit von einem Betriebssystem. Einige nehmen sich des Problems der Übertragung von Identitätsinformationen über den Bereich der Heimateinrichtung hinaus an. Diese werden als *Federation*[10] bezeichnet [Win05, MA08].

Vertreter dieser Federations nutzen häufig die *Security Assertion Markup Language* (SAML) zur Kommunikation. SAML wurde vom OASIS[11] *Security Services Technical Committee* als Standard für den Austausch von Authentifizierungs- und Autorisierungsdaten auf Basis einer XML-Syntax entwickelt [HJK08]. SAML-Systeme unterscheiden zwischen *Identity Providern* (Stelle, welche die behauptete Identität des Nutzers prüft und bescheinigt) und den *Service Providern* (Dienstanbieter, welche die ausgestellten Bescheinigungen akzeptieren und verwenden). Man kann dies mit dem KDC und den Services unter Kerberos vergleichen. Allerdings müssen für SAML die Services nicht zwangsläufig vorher mit dem Identity Provider abgeglichen werden.

Eine aktuell auf dem SAML 2.0 Standard arbeitende Federtion ist Shibboleth [MA08]. Shibboleth bietet die Möglichkeit der Übertragung einer vom Identity Provider der lokalen Einrichtung bescheinigten Identität auf Service Provider verschiedener Einrichtungen. Die Service Provider können Informationen vom Identity Provider über die Berechtigungen der Person (z.B. zugehörige Nutzergruppen) und damit deren Autorisierung erfragen. Daraus ergibt sich ein SSO-System, welches nicht nur die mit der Person verbundene Identität an die Dienste übermittelt, sondern auch deren Berechtigung propagieren kann. Durch die Trennung von Personendaten und Services wird ein verteiltes, organisationsübergreifendes (föderatives) Identity Management ermöglicht. Allerdings ist auch hier wieder die Frage zu klären, inwiefern ein Service Provider dem Identity Provider einer fremden Einrichtung trauen kann [HKN+09]. Für Shibboleth lässt sich dies über Policies im jeweiligen Service Provider festlegen.

Shibboleth ist aber nicht die einzige Federation-Lösung. Ein zur Zeit sehr erfolgreicher Vertreter ist OpenID [RR06]. OpenID benötigt OpenID-Provider, welche die Identität des Nutzers bestätigen und diese Information an die Dienstanbieter weiterleiten. Genutzt wird das von Webbrowsern verwendete HTTP-Protokoll, in das Identitäts- und Weiterleitungs-Informationen in Form von Cookies und URLs eingebettet und zwischen Nutzer, Provider und Dienst übermittelt werden (siehe Kapitel 6.1 und 7.2.1). Als bekannte Provider wären Google, VeriSign, AOL, Yahoo uvm. zu nennen. Aber auch für OpenID ergibt sich die Frage, ob ein Dienst, laufend bei einem Provider, der von einem anderen Provider bescheinigten Identität vertraut [SR09].

Ein weiteres Problem aller verteilten Lösungen besteht in der Frage, wie die initiale Authentifizierung, also die erstmalige Anmeldung am Identity Provider erfolgt. Greift man auf Passwörter zurück, so ergeben sich automatisch wieder deren Nachteile. Außerdem müssen vorhandene Dienste an das Federation-System angepasst werden. Darüber hinaus sind ständig verfügbare Services erforderlich, z.B. die Identity Provider der Heimateinrichtungen bei Shibboleth oder die OpenID-Provider.

Ist bereits eine föderative Authentifizierungslösung im Einsatz, so kann die Kombination mit zertifikatsbasierter Authentifizierung durchaus sinnvoll sein, um z.B. Passwörter für die initiale Anmeldung zu ersetzen.

---

[10] *Federated Identity* oder kurz *Federation* umfasst Standards, Technologien und Vereinbarungen, die notwendig sind, um den Austausch von Identitäts- und Berechtigungsinformationen zwischen autonomen Bereichen zu ermöglichen [MA08].

[11] Die OASIS (*Organization for the Advancement of Structured Information Standards*) ist eine internationale Organisation zur Entwicklung von E-Business- und Web-Service-Standards.

### 4.5.5 Die Wahl der SSO-Lösung

In der folgenden Übersicht sind die wesentlichen Vor- und Nachteile der genannten SSO-Systeme dargestellt. Federations werden dabei nicht berücksichtigt, da diese Arbeit auf einen abgeschlossenen Hochschulbereich abzielt.

| | *Vorteile* | *Nachteile* |
|---|---|---|
| **Tresor** | • keine Anpassungen der Dienste auf Serverseite erforderlich<br>• einfache Installation | • eine Alternative zur Verwendung von Passwörtern ist nicht möglich<br>• Erkennung von Passwortänderungen unzuverlässig<br>• hoher administrativer Aufwand |
| **Kerberos** | • Schutz der Authentisierungsmerkmale<br>• etablierter und allgemein akzeptierter Standard | • erfordert ausfallsicheres zusätzliches System (KDC)<br>• Software ist client- und serverseitig anzupassen |
| **Zertifikate** | • keine Übertragung oder serverseitige Speicherung sensibler Informationen<br>• ein Merkmal kann für mehrere Dienste genutzt werden<br>• Sicherung gegen anbieterseitigen Missbrauch der Merkmale<br>• während der Authentifizierung kein zusätzliches System online erforderlich | • benötigt PKI<br>• erfordert sicheren Transport des privaten Schlüssels<br>• benötigt Unterstützung durch client- und serverseitige Software |

Zertifikatsbasierte Authentifizierung bietet einen Sicherheitsgewinn gegenüber Passwörtern und ist gleichzeitig während der Authentifizierung unabhängig von zentralen Systemen (Ausfallschutz). Außerdem können Zertifikate auch anderweitig (Verschlüsselung, Signatur) eingesetzt werden. Bei vorhandener PKI bietet sich somit zertifikatsbasierte Authentifizierung an, wenn folgenden Punkte erfüllt werden können:

1. Der Transport des privaten Schlüssels muss derart gesichert werden, dass der Schlüssel sowohl vor Verlust, als auch vor unberechtigter Anfertigung einer Kopie geschützt ist.

2. Die Unterstützung durch gebräuchliche Applikationen und zugehörige Dienste muss gegeben sein. Die Aktivierung der zertifikatsbasierten Authentifizierung sollte ohne Anpassung der Software, z.B. durch eine Rekonfiguration möglich sein.

Wie sich das Schlüsseltransportproblem lösen lässt, beschreibt das folgende Kapitel. Die Applikations-Unterstützung wird im Kapitel 7.2 erläutert.

# Kapitel 5

# Kryptographische Token und Standards

Zertifikatsbasierte Authentifizierung in einem PKI-Umfeld in Verbindung mit einer SSO-Einbettung erfüllt die meisten der in Kapitel 3.4 aufgeführten Anforderungen. Für das Verfahren ist ein beglaubigtes Nutzer-Zertifikat und als Besitz der zugehörige geheime Schlüssel erforderlich, auf den besonders Acht gegeben werden muss.
Prinzipiell wäre eine Speicherung der Zertifikate und Schlüssel auf einem transportablen Datenträger denkbar. Zur Sicherung geheimer Schlüssel könnten diese mittels eines symmetrischen Verfahrens verschlüsselt werden, wobei als Schlüssel ein Passwort zum Einsatz käme. Derartige Sicherung bietet z.B. die Secure Shell für die benötigten Schlüssel[1] (siehe Kapitel 7.2.1, SSH). Das Dateiformat der öffentlichen Schlüssel ist festgelegt [RFC4716], für die privaten Schlüssel hingegen existieren mehrere implementierungsabhängige Varianten. Die unter der BSD-Lizenz vertriebene quelloffene OpenSSH Implementierung legt diese, gesichert mittels einer 3DES-Verschlüsselung in einer Datei ab. Auch andere allgemein gebräuchliche Verschlüsselungssoftware, z.B. PGP oder GnuPG/GPG[2] nutzen Dateien zur Ablage der Schlüssel. Diese, als „Schlüsselring“ bezeichneten Dateien können mehrere Schlüssel aufbewahren. Schlüsselringe für private Schlüssel sind dabei ebenfalls mit einem symmetrischen Verfahren gesichert.
Eine bessere Variante bietet jedoch die Verwendung kryptographischer Hardware in Form von Crypto-Token. Derartige Token, meist als kontaktbehaftete Smartcards oder in Form eines USB-Sticks verfügbar, speichern Zertifikate und zugehörige Schlüssel. Diese Hardware-Token besitzen einen für die Sicherheit des Authentifizierungssystems entscheidenden Vorteil. Sie ermöglichen die sichere und nicht kopierbare Aufbewahrung des geheimen Schlüssels. Dieser Schlüssel kann lediglich innerhalb des Tokens für kryptographische Funktionen verwendet, aber nicht aus dem Token ausgelesen werden. Ein Token mit gültigem Zertifikat und zugehörigem privaten Schlüssel erlaubt in Verbindung mit der Challenge-Response-Authentifizierung nach Kapitel 4.2.2 die zuverlässige und sichere Authentifizierung am lokalen Rechner oder an den Diensten im Netz.

## 5.1 Crypto-Token

Die Verwendung von Token als Authentisierungsmerkmal ist mit symmetrischen und asymmetrischen kryptographischen Verfahren möglich. Die meisten Applikationen unterstützen asymmetrische Ver-

[1] RFC 4252 Abschnitt 7: *Public Key Authentication Method*: beschreibt ein für die SSH fakultatives Verfahren zur Authentifizierung auf Basis asymmetrischer Schlüssel [RFC4252].
[2] GnuPG *GNU Privacy Guard*: Software nach dem OpenPGP-Standard [RFC3156, RFC4880], veröffentlicht unter der GNU Public Licence. Entspricht von der Funktionalität PGP, verwendet aber ausschließlich patentfreie Algorithmen.

fahren, wie im Kapitel 7.2 gezeigt wird. Die Speicherung des geheimen Schlüssels außerhalb des Tokens ist deshalb nicht notwendig. Demzufolge sind als Authentisierungsmerkmal Crypto-Token mit Unterstützung asymmetrischer Verfahren erforderlich. Welche Token sich dafür eignen, wird in diesem Kapitel aufgezeigt.
Die Anwendung der Token durch Client-Applikationen als Authentisierungsmerkmal erfolgt lokal oder über das Netz nach Abbildung 4.5 oder 4.10. Dabei werden kryptographische Operationen mit dem gespeicherten geheimen Schlüssel innerhalb des Tokens ausgeführt. Welche Schnittstellen zum Aufruf dieser Operationen existieren und wie Zertifikate oder andere Daten im Token abgelegt und wieder abgerufen werden können, wird ebenfalls in diesem Kapitel erläutert.

### 5.1.1 Chipkarte, Smartcard

Der Begriff Chipkarte (engl. *Chipcard*) wird allgemein für Plastikkarten mit eingebetteten Halbleiterchips verwendet. Als Smartcard (engl. *smart* für schlau oder clever) bezeichnet man Chipkarten mit eingebautem Mikrocontroller, der Funktionalitäten über die reine Datenspeicherung hinaus ermöglicht [RE02]. Einfache Speicherkarten, wie die 1994 in Deutschland eingeführte erste Version der Krankenversichertenkarte [KVK06], würden nicht unter diesen Begriff fallen.
Token in Form von Smartcards sind als Plastikkarten in verschiedenen Bauformen nach ISO/IEC-7810 Norm erhältlich, z.B. das ID-000 Format einer SIM-Karte für Mobiltelefone oder das ID-1 Kreditkartenformat [ISO7810]. Innerhalb der Karte ist ein Mikrocontroller enthalten, welcher über außen aufgebrachte elektrische Kontakte die Verbindung zum Rechner über ein Lesegerät ermöglicht. Der Mikrocontroller in der Karte besitzt einen Festspeicher (ROM) für die Ablage des Betriebssystems und den für die Programmausführung erforderlichen flüchtigen Speicher (RAM). Oftmals ist auch ein wiederbeschreibbarer Speicher (EEPROM) eingebaut, der abgelegte Daten auch nach Trennung von der Betriebsspannung behält. Den prinzipiellen Aufbau zeigt Abbildung 5.1.

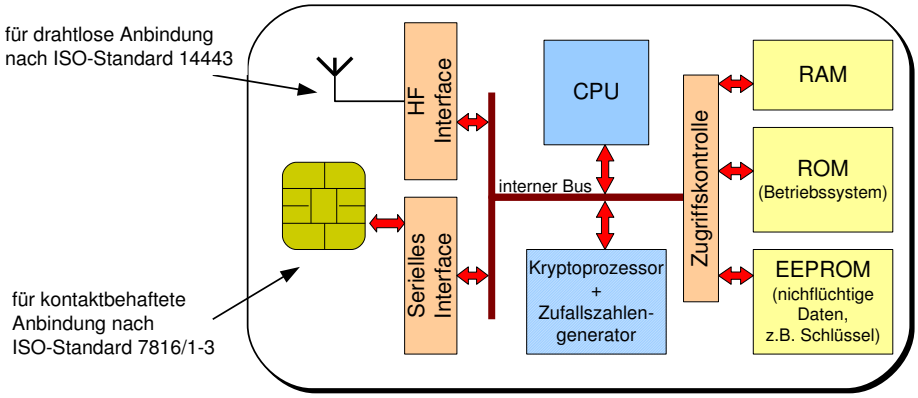



**Abb. 5.1 —** Blockschaltbild zum Aufbau von Smartcards

Das Smartcard-Betriebssystem ermöglicht dem Controller die Ausführung verschiedener Operationen. Über das Interface übertragene Daten können in der Karte zum späteren Abruf gespeichert oder direkt verarbeitet werden. Dabei können die im EEPROM enthaltenen Daten, z.B. Schlüssel, genutzt werden. Das Betriebssystem wird entweder bei der Herstellung fest im ROM hinterlegt oder ist ganz oder teilweise im EEPROM abgelegt. Letzteres ermöglicht Updates nach Herstellung.
Für die Verwendung als Crypto-Token muss das Betriebssystem der Karte die kryptographischen Algorithmen auf den übertragenen Werten und den in der Karte enthaltenen Schlüssel ausführen, ohne

dass die Schlüssel aus der Karte ausgelesen werden können. Es gibt Karten für symmetrische oder für asymmetrische Algorithmen. Einige Karten unterstützen auch gleichzeitig beide Arten. Darüber hinaus können die Karten kryptographische Hashwerte berechnen (Message-Digest Funktion) und bieten meist auch einen Zufallszahlengenerator. Dieser kann zur Erzeugung von asymmetrischen Schlüsselpaaren innerhalb der Karte genutzt werden, so dass der private Schlüssel niemals außerhalb der Karte vorliegt, was als besonders sichere Variante gilt. Allerdings hätte es den Nachteil, dass bei einer Beschädigung oder Verlust der Karte verschlüsselte Daten unwiederbringlich verloren sind, wenn sie nur mit diesem Kartenschlüssel entschlüsselt werden können.
In letzter Zeit wurden vermehrt Karten entwickelt, welche die Ausführung von nachladbarem Programmcode ermöglichen und somit sehr vielseitig einsetzbar sind [RE02]. Dabei gibt es Karten, die nur Programmcode für den jeweiligen Controller erlauben, sowie Karten, die controllerunabhängigen und damit herstellerunabhängigen Code verwenden können. Die herstellerunabhängige Programmierung erlauben *Java-Cards*. Der vom Java Card Forum[3] verabschiedete Standard ermöglicht die Nutzung der objektorientierten typisierten Sprache Java für die Erstellung von Bytecode, der von einem Interpreter auf der Karte ausgeführt wird. Die Implementierungen arbeiten aus Geschwindigkeits- und Speicherplatzgründen mit einem außerhalb der Karte liegenden (*offcard virtual machine*) und einem innerhalb der Karte liegenden Teil (*oncard virtual machine*) des Interpreters. Der mit dem offcard-Teil erstellte und geprüfte Programmcode kann auf die Karte übertragen und dort ausgeführt werden. Die Verwendung elektronischer Signaturen schützt vor Manipulation des offcard-Teils. Die Karten akzeptieren nur signierten Code, um eine Aushebelung der Sicherheitsmechanismen des offcard-Teils zu verhindern. Programmierbare Karten lassen sich unter Verwendung spezieller Programme in Crypto-Token mit der gewünschten Funktionalität umwandeln. Dabei muss für diese Karten sichergestellt sein, dass geladener Programmcode nicht manipuliert oder ausgetauscht werden kann, ohne dass dabei die im EEPROM liegenden Schlüssel gelöscht werden.

### 5.1.2 Kontaktlose Karten

Neben kontaktbehafteten existieren kontaktlose Smartcards, bei denen durch Induzierung einer Spannung der enthaltene Controller mit Energie versorgt wird. Die Kommunikation erfolgt über ein Kurzstreckenfunkprotokoll. Viele kontaktlose Karten arbeiten dazu nach dem ISO/IEC 14443 Standard für Transponderkarten[4], der die drahtlose Kommunikation mit dem Lesegerät spezifiziert [ISO14443].
Im Kapitel 4.2.2 wurden unter dem Begriff Mifare bereits kontaklosen Smartcards erwähnt. Diese dienen als Speichermedium für die geschützte Ablage von Daten. Ihre Funktionalität beschränkt sich auf das Lesen und Schreiben von Daten und das Erhöhen oder Dekrementieren enthaltener numerischer Werte [MIS50, Plö08]. Die Kommunikation erfolgt verschlüsselt und kann auf Lesegeräte begrenzt werden, die über einen passenden Schlüssel zur Authentifizierung gegenüber der Karte verfügen. Für die enthaltenen Daten kann der Zugriff durch verschiedene Autorisierungsstufen auf eine Kombination aus lesenden, schreibenden oder dekrementierenden Zugriff festgelegt werden. Der enthaltene Controller und die zugehörige Firmware erlaubt aber nicht die Ver- und Entschlüsselung übertragener Daten mit enthaltenen Schlüsseln. Eine Anwendung für Crypto-Token mit sicherer Schlüsselverwahrung kommt nicht in Frage, da zur Verwendung der geheime Schlüssels aus der Karte ausgelesen werden müsste.

[3] JFC *Java Card Forum*: Vereinigung vieler Chipkartenhersteller und der Firma SUN Microsystems, dem Hauptentwickler und Inhaber des eingetragenen Warenzeichens Java zur Standardisierung der Java Card.
[4] Transponderkarte oder engl. *Proximity Integrated Circuit Card* (PICC) ist ein Oberbegriff für kontaktlose Mikrocontroller-gestützte Systeme.

Neuere Entwicklungen, wie z.B. die Karten der Mifare ProX oder SmartMX Serie, erlauben die Programmierung. Crypto-Token-Funktionalität kann deshalb über nutzerseitig in die Karte eingebrachten Code ergänzt werden. Diese Karten gibt es auch mit Dual-Interface, so dass sie sowohl in kontaktlosen, wie auch in kontaktbehafteten Lesern verwendet werden können.
Gegenüber kontaktbehafteten Smartcards liefert das kontaklose Interface einen Vorteil. Abnutzungserscheinungen seitens des Lesegerätes oder der Karte durch häufige Kontaktierungen wären nicht gegeben. Ihr Einsatz bietet sich bei höher frequentierten Systemen an.
Nachteil ist der hohe Preis im Vergleich zu kontaktbehafteten Karten. Dieser Preis betrifft sowohl die Mifare ProX/SmartMX-Karten, als auch die passenden Lesegeräte. Außerdem sind auf diesem Gebiet viele Hersteller-Standards nicht offen gelegt, so dass keine frei verfügbare Software zur Nutzung dieser Karten als Crypto-Token existiert. Über den Umweg einer Java-Card-Software wäre es aber trotzdem möglich.

### 5.1.3 USB-Token

Crypto-Token existieren auch in Form von USB-Token. Deren Funktionalität entspricht im Wesentlichen einer kontaktbehafteten Smartcard, bei der außer der geänderten Bauform nur das elektrische Interface zur Verbindung der Smartcard mit dem Lesegerät gegen eine USB-Schnittstelle[5] ausgetauscht wurde, so dass ein Lesegerät entfallen kann. Ein Nachteil der USB-Schnittstelle besteht in der geringeren garantierten Anzahl möglicher Steckzyklen. USB-2 garantiert 1500 Zyklen [Com00]. Im Gegensatz dazu erlauben Smartcards und deren Lesegeräte meist $> 10^5$ Kontaktierungen [RE02]. Eine Anwendung von USB-Token für höher frequentierte Rechnersysteme mit oftmaligem Tokenwechsel wäre damit ausgeschlossen.

### 5.1.4 Standards für Smartcards und USB-Token

#### ISO/IEC-7816

Die meisten kontaktbehafteten Chipkarten basieren auf Teil 1 bis 4 des ISO/IEC-7816 Standards, einer Erweiterung des bereits erwähnten ISO/IEC-7810 Standards, welcher Eigenschaften von Identitätsdokumenten spezifiziert [ISO7816/01].
ISO/IEC-7816 legt neben physikalischen Aufbau der Karte und des Anschlusses (Teil 1 und 2) den Kommunikationsablauf auf der elektronischen Ebene fest (Teil 3). Teil 4 beschreibt die Organisation des Speichers analog einem Dateisystem, den Aufbau der Dateien und die zugehörigen Zugriffsberechtigungen und somit eine Sicherheitsstruktur für Smartcards.
Das Dateisystem ist hierarchisch aufgebaut, ausgehend von einer festen Startdatei (MF: *Master File*). Man unterscheidet zwischen DF- (*Dedicated Files*) und EF-Dateien (*Elementary Files*). DF-Dateien bilden ein Verzeichnis, EF-Dateien enthalten die Nutzdaten. Auf diese Art wird der sequentiell adressierbare Speicher der Chipkarten in Bereiche zerlegt, die jeweils eine DF- oder EF-Datei darstellen. Die DF-Dateien können in weitere DF- und EF-Dateien unterteilt werden. Der Freispeicher wird durch eine FAT (*File Allocation Tabelle*) verwaltet.
Der Aufbau einzelner EFs ist ebenfalls festgelegt. Chipkarten-Dateien besitzen eine eigene Struktur, abhängig vom jeweiligen Typ. Diese, für die enthalte Nettoinformation (dem Datei-Body) notwendige Festlegung ist abhängig vom Verwendungszweck. Man unterscheidet zwischen *transparenten*

[5] USB *Universal Serial Bus*: diese Schnittstelle für serielle Verbindungen zu peripheren Geräten wird ca. seit dem Jahr 1998 im PC-Sektor, nach 2000 mit Verabschiedung der USB 2.0 Spezifikation auch für andere Systeme serienmäßig eingebaut [Com00].

Dateistrukturen, bei denen innerhalb der Datei keine weitere Unterstrukturierung vorhanden ist (wird z.B. für nachladbare Programme verwendet), oder *Record*-Strukturen. Für die Aufbewahrung von Schlüsseln und Zertifikaten, bei denen der Aufbau und die Position einzelner Werte bekannt ist, würde sich eine Record-Struktur anbieten. So kann bei einem Zertifikat der erste Record das Subject, der Zweite den Schlüssel und der Dritte die Signatur enthalten. Das vereinfacht den Zugriff für den Controller. Für die Anwendung als Crypto-Token ist der vorgegebene Datei-Header ebenfalls von Interesse. Der Header enthält neben einem *Identifier* und der Typbeschreibung auch die *Access Conditions*. Diese regeln Zugriffsbedingungen und sind vom Smartcard-Betriebssystem einzuhalten.
Ebenfalls durch ISO/IEC-7816-4 festgelegt ist der Kommunikationsablauf auf Anwendungsebene in Form einzelner APDUs. Eine APDU ist ein Datencontainer, der für den Austausch von Daten zwischen dem Terminal[6] bzw. dem dahinter angeschlossenen Computer und der Chipkarte dient und somit im ISO/OSI-Referenzmodell auf der obersten, der Anwendungsebene angesiedelt wäre. Somit beschreibt eine APDU unabhängig von der zugrunde liegenden physikalischen Kommunikation den Datenaustausch zwischen dem Controller der Chipkarte und dem Prozessor des Computers. Man unterscheidet zwischen Kommando- (*command*) und Antwort- (*response*) APDUs. Eine Kommando-APDU übermittelt ein Kommando mit den zugehörigen Parametern an den Controller der Chipkarte und ruft damit eine Funktion in der Chipkarte auf. Der Controller antwortet mit einer APDU, die den Erfolgsstatus und das Ergebnis der aufgerufenen Funktion enthält (Abbildung 5.2).

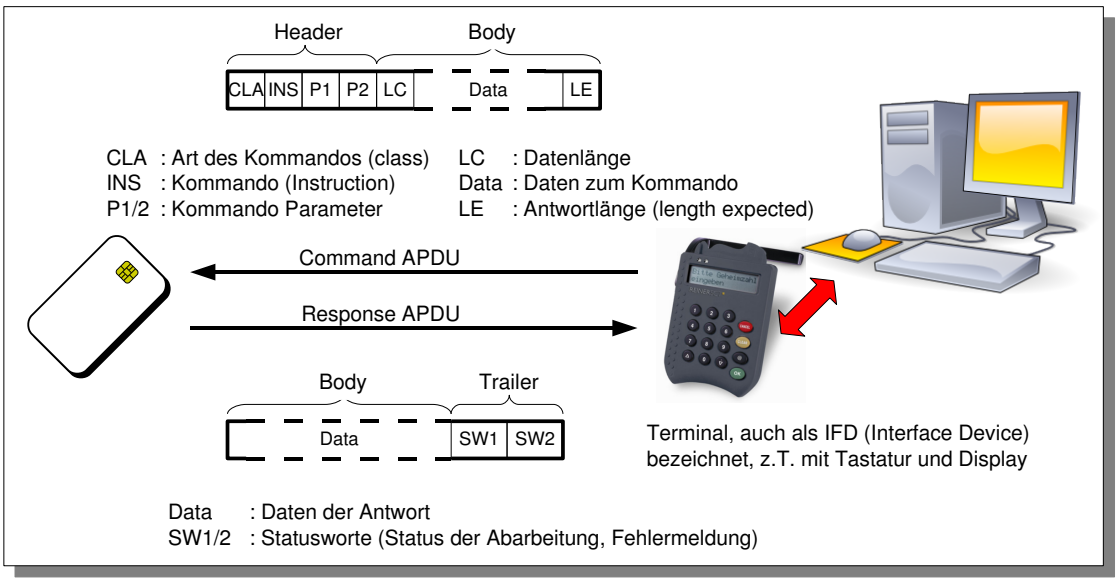



**Abb. 5.2 —** Kommando- und Antwort-APDUs für einen Funktionsaufruf

Eine Verschlüsselung durch die Karte erfolgt, indem der Controller die dafür vorgesehene Kommando-APDU erhält, deren Parameter ein *Identifier* des in der Karte hinterlegten Schlüssels und die zu verschlüsselnden Daten sind. Hatte sich der Nutzer zuvor gegenüber der Karte legitimiert, berechnet daraufhin der Controller das Ergebnis und sendet dieses in der Antwort-APDU an das aufrufende System zurück.

[6] Terminal bezeichnet das Gerät zum Einlegen der Chipkarte. Es versorgt die Karte mit Strom und verbindet die elektrische Kartenschnittstelle mit der Computerschnittstelle, z.B. über einen seriellen Port oder den USB-Anschluss. Der häufig verwendete Begriff „Smartcard-Reader“ für ein Lesegerät ist unzureichend, da die Funktionalität über reines Lesen/Schreiben hinausgeht. Es gibt Terminals mit eigener Tastatur und Display und eigenem Betriebssystem, welches autarke Kommunikation mit den Karten ermöglicht.

Unter ISO/IEC-7816-4 werden APDUs für grundlegende Kommandos zum Lesen, Schreiben, Verändern oder Löschen von binären Daten und Records sowie zur Authentifizierung gegenüber der Karte festgelegt. Diese Kommandos ermöglichen eine Basisfunktionalität, die herstellerunabhängig grundlegende Funktionen mit jeder Karte ermöglichen. Kryptographische Operationen gehören nicht zu diesem Umfang. Sie werden erst im Teil 8 in Form von Hash-, Signatur- und Verschlüsselungs-Operationen ergänzt [ISO7816/08]. Teil 15 beschreibt eine Chipkartenstruktur für kryptographische Anwendungen. Darin ist die Speicherung, Nutzung und Abfrage kryptographischer Daten auf der Karte sowie der Einsatz verschiedener kryptographischer Algorithmen definiert [ISO7816/15]. Hersteller von Smartcards nach ISO/IEC-7816 (Teil 1-4), die sich als Crypto-Token eignen, müssen aber nicht den Teil 8 und 15 des Standards umsetzen, sondern können einen eigenen Satz von Kommandos für die erforderliche kryptographische Funktionalität definieren. Sie etablieren dann eine Softwareschnittstelle innerhalb des Rechners, über die Applikationen die kryptographischen Funktionen aufrufen können. Insofern es sich nicht um Karten für Speziallösung handelt, kann der Hersteller die universelle Einsetzbarkeit durch Bereitstellung einer standardisierten Softwareschnittstelle ermöglichen, wie im Kapitel 5.2 gezeigt wird.

USB-Token können ebenfalls die im Teil 4 des ISO/IEC-7816 Standards beschriebene Kommunikation umsetzen. ISO/IEC-7816-12 ergänzt hierzu die unter ISO/IEC-7816-3 beschriebene Chipkarten-Schnittstelle um einen USB-Anschluss [ISO7816/12]. Allerdings existieren momentan noch keine Karten, die diesen Standard umsetzen. Jedoch existieren einige herstellerspezifische Lösungen mit USB-Anschluss, welche durch mitgelieferte Software den Tokenzugriff analog einer Smartcard ermöglichen (Beispiel: *Aladdin eToken Pro*, *Aladdin eToken Anywhere*[7] oder *SafeNet iKey*[8] ). Somit können Smartcards und USB-Token bei Vorliegen einer entsprechenden Softwareschnittstelle, welche das Hardwareinterface verbirgt, gleichartig verwendet werden.

### PKCS#15 (*Cryptographic Token Information Standard 15*) und ISO/IEC-7816-15

Die *Public Key Cryptographic Standards* (PKCS) umfassen eine Reihe von Spezifikationen für kryptographische Anwendungen, von denen einige für Crypto-Token von Bedeutung sind.

So beschreibt der *Cryptographic Token Information Format Standard* (PKCS Teil 15) den Aufbau und die geschützte Ablage von Objekten in Crypto-Token [PKCS15]. Dabei kann es sich um Schlüssel, Zertifikate oder andere Objekte handeln. Speziell für Chipkarten, basierend auf dem Standard ISO/IEC-7816 (Teil 1-4 und 6) definiert PKCS#15 die hierarchische Organisation der Objekte entsprechend ihrer Funktion, z.B. Kombination eines Zertifikats mit dem zugehörigen privaten Schlüssel. Diese kann in einer DF/EF-Verzeichnisstruktur der Chipkarten umgesetzt werden. Dabei wird der Schutz sensibler Daten, wie z.B. das verbotene Auslesen privater Schlüssel berücksichtigt. PKCS#15 bietet sich für Beschreibung und Organisation der Datenobjekte in Hardware-Token an und hat sich für viele Hersteller als Norm zur Organisation von Token mit kryptographischer Funktionalität etabliert.

Die PKCS#15 Spezifikation ist mittlerweile als Bestandteil in den ISO/IEC-7816-15 Standard aufgenommen worden. Mittelfristig gesehen ist PKCS#15 nicht mehr notwendig, da diese Spezifikation vollständig durch den ISO-Standard abgedeckt wird [Cha06]. Aktuell bildet sie aber noch die Grundlage gebräuchlicher kryptographischer Middleware zur Nutzung von Chipkarten, wie im Kapitel 5.3.3 gezeigt wird.

[7] `http://www.aladdin.com/etoken/`
[8] `http://www.safenet-inc.com/products/tokens/ikey.asp`

### Token-Rechner-Anbindung

Der Einsatz von Smartcards oder USB-Token im Rechnerumfeld verlangt eine Verbindung zwischen Terminal und Rechner, sowie eine Schnittstelle zum Andocken der Software. Diese Token-Interface-Software arbeitet nach Abbildung 5.3 innerhalb des Rechners zwischen Anwendungssoftware und der Hardwareverbindung zum Terminal bzw. zum USB-Token.

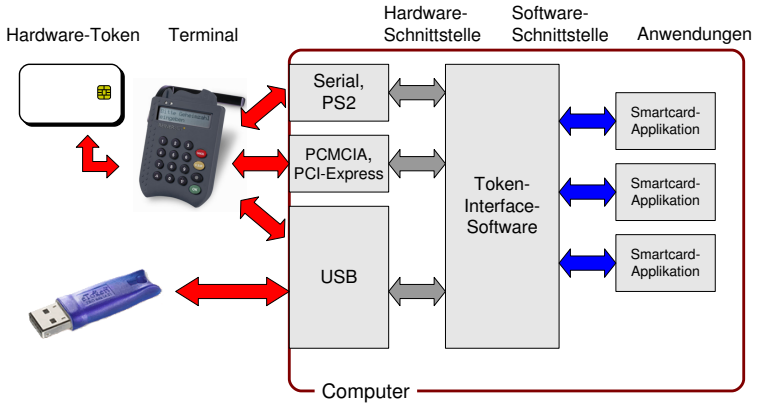



**Abb. 5.3 —** Token-Interface-Software

Für Token-Interface-Software existieren mehrere Spezifikationen in Form von Industriestandards. Sie stellen terminal- bzw. hardwareunabhängige Software-Schnittstellen für Applikationen zur Verfügung. Als Software-Andockpunkt wird jedoch eine kryptographische Schnittstelle im Rechner benötigt, also eine weitere Zwischenschicht. Dabei sollte es sich um eine PKCS#11-Schnittstelle handeln, wie im Kapitel 5.3 gezeigt wird. Diese Schnittstelle wird vom Hersteller des jeweiligen Tokens in Form einer Bibliothek angeboten und greift über die Token-Interface-Software auf das Token zu. Da Token-Hersteller aber nicht jede Token-Interface-Software unterstützen, muss bei der Wahl dieser Software auf größtmögliche Überschneidung für unterschiedliche PKCS#11-Anbieter geachtet werden. Dadurch kann eine Bindung an spezielle Token und deren Hersteller vermieden werden.

- Das **„Open Card Framework“ (OCF)**, entwickelt für den betriebssystemunabhängigen Einsatz von Smartcards, wurde hauptsächlich im Java-Umfeld eingesetzt. Inzwischen bietet die aktuelle Java-Versionen (SUN-JDK ab Version 6) eine eigene Smartcard-API[9] und nutzt die im Anschluss vorgestellte PC/SC-Schnittstelle, so dass OCF nicht weiter entwickelt wird.

- Weit verbreitet ist die **PC/SC („PersonalComputer/SmartCard“)**-Schnittstelle, welche von einer Gruppe namhafter PC-Hardware, Software-, Terminal- sowie Smartcardhersteller entwickelt wurde und ursprünglich für die Einbindung von Smartcards in PC-Systeme gedacht war. Die unter dem Namen „*Interoperability Specification for ICCs and Personal Computer Systems*“ veröffentlichte Spezifikation[10] besteht in der aktuellen Version 2.01.5 aus zehn Teilen. Diese beschreiben detailliert die Einbindung von Hard- und Software zur Nutzung von Smartcards und definieren die notwendigen Schnittstellen. Im Prinzip lassen sich darüber beliebige Smartcards, aber auch USB-Token einbinden, vorausgesetzt für Terminal oder Token existiert ein passender Treiber.

[9] Eine API (engl. *application programming interface*) definiert eine Softwareschnittstelle für die Einbindung in externe Programme auf Quelltextebene.

[10] `http://www.pcscworkgroup.com/specifications/overview.php`

Abbildung 5.4 verdeutlicht die Funktionsweise von PC/SC. Auf die Chipkarte (ICC: *Integrated Chip Card*), welche mit dem Terminal (IFD: *Interface Device*) verbunden ist, wird über den als IFD-Handler bezeichneten Terminal- oder Tokentreiber zugegriffen. Für ein angeschlossenes USB-Token steuert dieser Treiber die Tokenhardware über die USB-Schnittstelle des Rechners an.

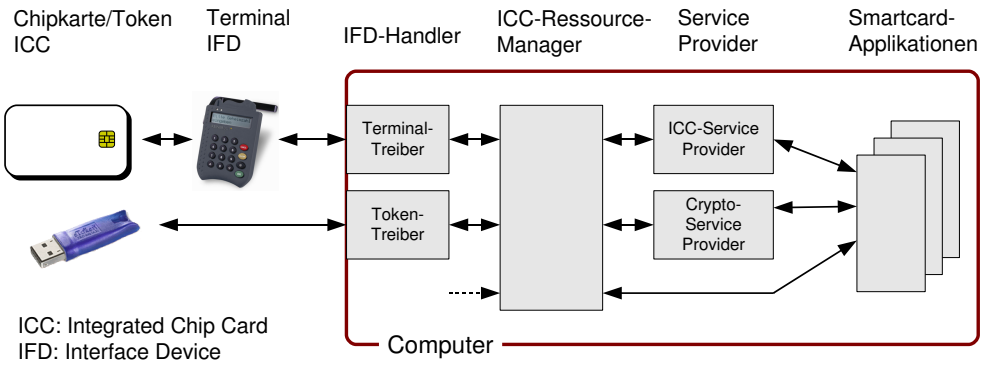



**Abb. 5.4 —** PC/SC-Architektur

Der ICC-Ressource-Manager verwaltet auf Seite der Hardware die angeschlossenen IFDs und die damit verbundenen ICCs. Auf Seite der Software regelt er den Zugriff auf die jeweilige Hardware. Außerdem stellt er einfache Kommandogruppen bereit, über die bestimmte zusammengehörige Kommandos zur Karte übertragen werden können. Dadurch wird eine Transaktion für eine Applikation abgeschlossen, bevor eine weitere Applikation die ICC- und IFD-Hardware nutzen kann. Dieses Prinzip ermöglicht die parallele Benutzung der sequentiell anzusteuernden Hardware durch mehrere gleichzeitig agierende Applikationen.

Die Softwareschnittstelle stellen Service-Provider bereit. Über deren APIs können unabhängig vom ICC Kommandos aufgerufen werden, zu denen Dateioperationen, wie Lesen, Schreiben und Löschen gehören. Falls ICC-Hersteller spezielle Kommando-APDUs nutzen, die über den ISO/IEC-7816-Standard hinausgehen oder davon abweichen, so können diese ICCs innerhalb eines PC/SC-Systems eingebunden werden, wenn der Hersteller den passenden ICC-Service-Provider liefert. Eine API für kryptographische Operationen wurde in einen separaten Provider (CSP: *Cryptographic Service Provider*) ausgelagert, um gesetzliche Regelungen zur Beschränkung kryptographischer Funktionen einhalten zu können. Applikationen können zur Nutzung der ICCs auf die Service-Provider zugreifen. Dies ist jedoch keine starre Vorgabe, da auch das Andocken am ICC-Ressource-Manager möglich ist.

Aufgrund seiner Entstehung im PC-Umfeld ist PC/SC die Basis für den Einsatz von Chipkarten in Microsoft-Betriebssystemen und wird über die Smartcard-Client-API in Windows verfügbar gemacht. Es existiert aber auch eine lizenzfreie Implementierung unter dem Namen *PC/SC-lite*[11], die große Teile der PC/SC-Spezifikation abdeckt und für verschiedene Betriebssysteme angeboten wird.

Microsoft definiert für PC/SC auch einen CSP mit zugehöriger API, welcher im Kapitel 5.2.1 beschrieben wird. Andere, von der PC/SC-CSP API abweichende kryptographische Schnittstellen bauen meist ebenfalls auf PC/SC auf und docken direkt am ICC-Ressource-Manager an. Sie arbeiten somit parallel zu den Cryptographic Service Providern von PC/SC. Ein Vertreter wäre die bereits erwähnte PKCS#11-Schnittstelle.

[11] `http://pcsclite.alioth.debian.org/`

Neben PC/SC und OCF existieren weitere Token-Schnittstellen.

• Die **MKT-Spezifikation**[12] ist in Deutschland aufgrund des Einsatzes im medizinischen Bereich verbreitet. Sie wurde bisher jedoch nicht als Basis für die Anbindung von Crypto-Token verwendet.

• Ursprünglich für Linux entwickelt wurde das **MUSCLE-Framework**[13]. Dieses Framework, das trotz des Namens nicht allein auf Linux beschränkt ist, unterstützt verschiedene Schnittstellen zum Einsatz von Smartcards. Es wird zwar nicht als Andockpunkt für PKCS#11 genutzt, liefert aber Applets für Java-Cards zum Einsatz dieser Karten als Crypto-Token.

• Eine weitere quelloffene und freie Schnittstelle bietet **OpenCT**[14]. Obwohl OpenCT für mehrere Betriebssysteme umgesetzt wurde, nutzen nur wenige PKCS#11-Anbieter diese Schnittstelle.

• Die einfache **CT-API**[15] ist ebenfalls eine Token-Schnittstelle, welche rudimentäre Funktionen zur Übertragung der APDUs an die Token bietet. Auf der CT-API bauen umfangreichere APIs, wie z.B. MKT auf. Damit liefert sie die einfachste Schnittstelle zwischen Software und Token, ist aber nicht ICC-unabhängig und bietet auch kein Ressourcen-Management.

Abgesehen von der PKCS#11-Implementierung *OpenSC*, welche neben PC/SC-lite auch die OpenCT- und CT-API-Schnittstelle unterstützt, basieren die meisten anderen PKCS#11-Implementierungen der ICC-Hersteller auf der PC/SC-Spezifikation. Deshalb und aufgrund der freien Verfügbarkeit von *PC/SC-lite* bot sich dieses Interface an und wurde als Basis für die Implementierung der SSO-Lösung herangezogen.

## 5.2 Token-Schnittstellen für kryptographische Funktionen

Im vorhergehenden Kapitel wurden Software-Schnittstellen beschrieben, die Applikationen den Zugriff auf Token ermöglicht. Abgesehen von der CT-API erlauben sie grundlegende Funktionen auf Token über eine einheitliche API und somit weitestgehende Token-Unabhängigkeit. Außer beim PC/SC-CSP beschränken sich die APIs aber auf die filesystemartige Verwaltung der Objekte und auf die Authentifizierung gegenüber dem Token. Für die Verwendung im kryptographischen Umfeld sind andere Applikations-Schnittstellen erforderlich. Diese sollten auf einer vom Token abstrahierten Ebene den Zugriff auf dessen kryptographische Funktionalität bieten. Damit ließen sich beliebige Crypto-Token einbinden, insofern deren Hersteller diese Schnittstelle unterstützen.
Ein weiterer Vorteil einer tokenunabhängigen API für kryptographische Funktionen besteht darin, dass unkritische Funktionen, welche nicht direkt vom Token unterstützt werden, in der zugehörigen Software ergänzt werden können. Das dürfen natürlich keine Methoden unter Nutzung geheimer Schlüssel sein, da diese unauslesbar im Token verwahrt werden müssen. Möglich wäre aber die softwareseitige Umsetzung der Hashwertberechnung (Digest-Funktionen) oder die Ausführung von Operationen unter Nutzung öffentlicher oder daraus abgeleiteter Schlüssel.

**Kryptographische APIs** Als kryptographische API für Anwender-Programme mit Unterstützung von Hardware-Token haben sich drei Schnittstellen etabliert, die Verwendung der *Cryptographic-Service-Provider* von *PC/SC*, die Verwendung des *Public Key Cryptography Standard #11* oder die Einbindung der *Network Security Services* (NSS).

[12] MKT: *Multifunktionale Karten-Terminals*
`http://www.teletrust.org/publikationen/spezifikationen/mkt/`
[13] MUSCLE: *Movement for the Use of Smart Cards in a Linux Environment*, `http://www.linuxnet.com/`
[14] `http://www.opensc-project.org/openct/`
[15] CT-API: *CardTerminal Application Programming Interface*

### 5.2.1 Microsoft Crypto-API

Die Microsoft Cryptography API[16] (Microsoft CAPI) steht für alle 32- und 64-Bit Windows-Systeme seit Windows 95 zur Verfügung. Inzwischen wurde sie mehrmals erweitert und in Windows Vista unter dem Namen *Cryptography API: Next Generation* (CNG) mit erweiterter Funktionalität ausgestattet. Damit unterstützt sie auch aktuelle Methoden aus dem Bereich der *Elliptic Curve*-Kryptographie.

Entsprechend der Abbildung 5.5 besteht die Microsoft Cryptography API aus zwei Teilen. Der Cryptographic Service Provider stellt die Schnittstelle zum darunter liegenden Crypto-System dar, welches aus Hard- und Software bestehen kann. Microsoft liefert eine Software-Variante (Base Provider), der kryptographische Funktionen in Software umsetzt. Zur Einbindung von Hardware-Token ist ein CSP des jeweiligen Hardware-Herstellers erforderlich. Dies entspräche der Implementierung eines CSP nach Teil 6 der PC/SC-Spezifikation.

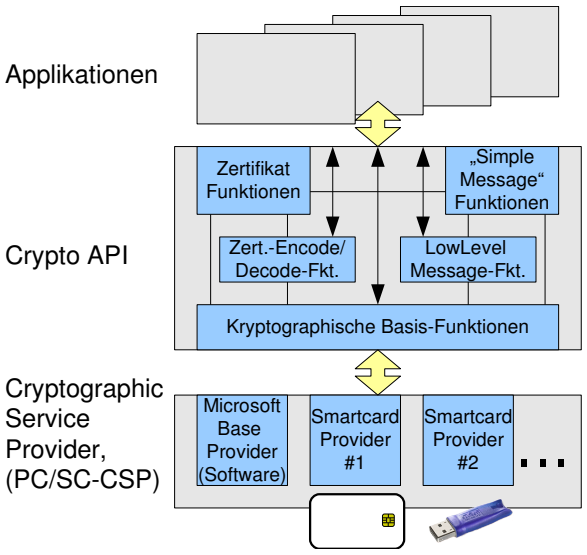



**Abb. 5.5 —** Microsoft Crypto-API

Der zweite Teil, welcher oft als die Crypto-API oder CAPI bezeichnet wird, stellt das Funktions-Interface für Applikationen bereit und greift dazu auf die darunterliegenden CSP zurück. Er bietet Funktionen zum Zertifikatsmanagement, zur Ver- und Entschlüsselung zur Signatur u.v.m.

Die Microsoft CAPI wird bisher nur für Microsoft-Windows-Betriebssysteme angeboten. Die Nutzung dieser CAPI lässt sich deshalb nicht mit der Forderung nach einer clientseitigen betriebssystemübergreifenden Einsetzbarkeit vereinen.

### 5.2.2 PKCS#11

Der *Public Key Cryptography Standard* Teil 11 (PKCS#11) beschreibt eine als *Cryptoki*[17] bezeichnete API für kryptographische Hard- oder Software-Systeme. Diese umfasst die Speicherung von Objekten wie Schlüssel und Zertifikate in Token und die Ausführung kryptographischer Operationen auf diesen gespeicherten Objekten und der Eingabe. Cryptoki definiert somit auf hardwarunabhängiger Basis eine logische Sicht auf Ressourcen und Funktionen von Token [Cha06].

Für Applikationen bietet PKCS#11 den einheitlichen Zugriff auf beliebige Token. Dabei wird vom realen Token und damit auch von der Hardware abstrahiert. Der Funktionsumfang des jeweiligen Tokens, wie z.B. die unterstützten kryptographischen Methoden (in PKCS#11 als *Mechanism* bezeichnet) können über Cryptoki erfragt und anschließend genutzt werden.

PKCS#11 erlaubt die Nutzung mehrerer Token durch eine Applikation. Hierzu werden die Token auf Slots abgebildet und der Applikation zur Verfügung gestellt (Abbildung 5.6). Hinter jedem Slot verbirgt sich meist ein Hardware-Token. Es können aber auch mehrere logische Token in einer Hardware existieren, die auf unterschiedliche Slots abgebildet werden.

[16] `http://msdn.microsoft.com/en-us/library/aa380255(VS.85).aspx` (Stand 15.6.2009)
[17] Cryptoki: **Cryp**tographic **Tok**en **I**nterface [PKCS11]

Die PKCS#11-Implementierungen verwalten meist nicht selber den Zugriff auf die Slots, sondern greifen hierzu auf eine Token-Interface-Software zurück, z.B. auf den PC/SC-Ressourcen-Manager oder OpenCT.
PKCS#11-Unterstützung für Applikationen erfolgt durch Einbindung einer Bibliothek, die vom Hersteller eines Tokens zur Verfügung gestellt wird. Die PKCS#11-Bibliothek wird als PKCS#11-*Provider* bezeichnet. PKCS#11-fähige Applikationen erlauben meist das Einbinden mehrerer PKCS#11-Provider und können somit gleichzeitig mit verschiedenen Token agieren, auch wenn für jedes Token ein anderer Hersteller die zugehörige Bibliothek liefert. Außerdem stehen Token- und somit Hersteller-übergreifende Implementierungen, wie die unter einer freien Lizenz stehende OpenSC-Bibliothek zur Verfügung.

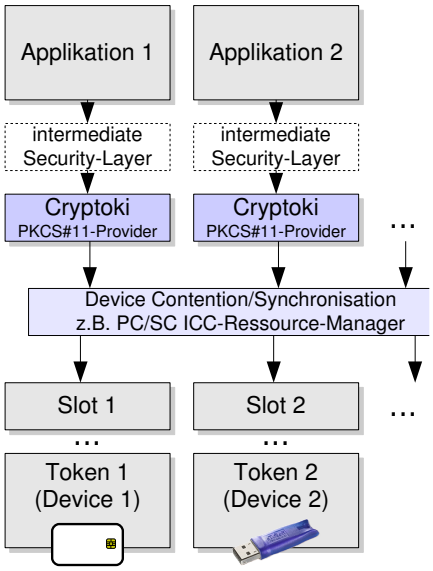



**Abb. 5.6 —** PKCS#11: Cryptoki-Modell

### 5.2.3 NSS

Eine weitere API für die Einbindungen kryptographischer Funktionen liefern die ***N**etwork **S**ecurity **S**ervices* (NSS) [18]. Auf Basis der *Netscape Portable Runtime* (NSPR) API, einer API für plattformunabhängige Aufrufe von Betriebssystemfunktionen, wurde NSS entwickelt. NSS ist eine Bibliothekssammlung für die plattformübergreifende Entwicklung von Client-Server-Applikationen zur Unterstützung sicherer Kommunikation.
Die NSS nutzen eine Vielzahl von Sicherheits- und Kommunikations-Standards, zu denen neben SSL/TLS, S/MIME [RFC3851] und X.509v3 auch diverse PKCS-Spezifikationen gehören. Die NSS-API bietet Applikationen die Möglichkeit, diese Standards über eine gemeinsame Schnittstelle zu nutzen. Die Unterstützung kryptographischer Hardware erfolgt in den NSS über PKCS#11. Somit können alle Token eingebunden werden, für die ein PKCS#11-Provider zur Verfügung steht. Eine Applikation ohne PKCS#11-, aber mit NSS-Schnittstelle kann somit indirekt die Möglichkeiten der Hardware-Token mit PKCS#11-Provider nutzen. Für das in dieser Arbeit vorgestellte System wäre deshalb anzumerken, dass mit der SSO-Unterstützung über PKCS#11 auch eine SSO-Unterstützung für NSS-Applikationen gegeben ist.

### 5.2.4 Sonstige APIs für Crypto-Token

Eine Bibliothek zur Einbettung kryptographischer Funktionalität in Applikationen bietet **OpenSSL**[19]. Die Bibliothek wird hauptsächlich für die SSL/TLS-gesicherte Kommunikation in Applikationen verwendet. OpenSSL selber besitzt keine eigene Schnittstelle zur Einbindung von Hardware-Token oder anderer kryptographischer Hardware. Allerdings definiert OpenSSL eine Schnittstelle zu *Engines*, an die sämtliche Funktionen delegiert werden können. So erlaubt die *PKCS#11-Engine*[20] eine Integration

[18] `http://www.mozilla.org/projects/security/pki/nss/`
[19] `http://www.openssl.org`
[20] `http://www.opensc-project.org/engine_pkcs11`

der PKCS#11-Provider in OpenSSL und damit den Zugriff auf Hardware-Token.
Weitere Applikations-Schnittstellen werden entweder regional begrenzt eingesetzt oder stehen nur eingeschränkt zur Verfügung. Zu erwähnen wäre das **eCard-API-Framework**[21] vom *Bundesamt für Sicherheit in der Informationstechnik* (BSI). Dessen Ziel ist die Bereitstellung einer betriebssystemunabhängigen einfachen und homogenen Applikations-Schnittstelle für eine einheitliche Nutzung unterschiedlicher Smartcards, die als eCards bezeichnet werden. Das eCard-API-Framework ist in einer vier-Schichten-Architektur aufgebaut (Application-, Identity-, Service-Access- und Terminal-Layer). Dabei werden Schnittstellen für die Anbindung der Hardware und die Einbindung in Applikationen definiert. Eingesetzt werden eCards und deren API für die elektronische Version der Gesundheitskarte (eGK), des Personalausweises (ePA) und Reisepasses (ePass) sowie für die elektronische Steuererklärung (ELSTER) und den Einkommensnachweis (ELENA). Momentan ist der Einsatz aber auf Deutschland beschränkt, weshalb die eCard-API hier nicht weiter betrachtet wurde, auch wenn das Ziel besteht, die Nutzung im Rahmen einer europaweiten eID-Infrastruktur auszuweiten.

## 5.3 Verwendung von Token mit PKCS#11

Die in Kapitel 5.2 gezeigten Schnittstellen erlauben die einfache Einbindung von Crypto-Token in Applikationen über eine standardisierte API. Die Hersteller-unabhängige PKCS#11-Schnittstelle bietet hierbei neben der breiten Unterstützung seitens der Token-Anbieter den Vorteil, dass sie plattformübergreifend einsetzbar ist. Viele Applikationen mit Bedarf an kryptographischen Funktionen - insbesondere aus dem Linux-Umfeld, aber auch unter Windows - verfügen bereits über eine PKCS#11-Anbindung. Weitere Applikationen erlauben die PKCS#11-Einbindung über OpenSSL-Engines.
Auf dieser Grundlage haben wir entschieden, dass PKCS#11 eine ideale Basis für die angestrebte besitzbasierte Authentifizierung mit Hardware-Token darstellt. Außerdem erlaubt PKCS#11 über Authentifizierung hinaus gehende Anwendungen der kryptographischen Schnittstelle. Zu klären bliebe jedoch, ob eine Nutzung in Verbindung mit clientseitiger SSO-Unterstützung möglich ist. Dazu wird im Folgenden ein kurzer Überblick über die Spezifikation gegeben und gezeigt, wie PKCS#11 zur Nutzerauthentifizierung eingesetzt werden kann.

### 5.3.1 Ein Überblick zur PKCS#11-Spezifikation

Wie bereits im Kapitel 5.2.2 beschrieben, bietet PKCS#11 eine Applikations-API *Cryptoki* zur Nutzung von Crypto-Token. Mehrere Token sind möglich und können in Cryptoki über die Abbildung auf *Slots* adressiert werden. Die Verwaltung der physikalischen Ressource „Token“ kann an die Token-Interface-Software, z.B. *PC/SC* delegiert werden (siehe Abbildung 5.6). Somit können mehrere Programme gleichzeitig ein Token als gemeinsame Ressource verwenden. Dies ist ein wesentlicher Punkt, wenn die Token in SSO-Umgebungen mit verschiedenen, gleichzeitig aktiven Applikationen eingesetzt werden sollen. Dabei wird für jedes Programm über Cryptoki das Token - oder genauer der Slot - immer in einem zum jeweiligen Programm zugehörigen Zustand dargestellt. Die Anwendungen werden somit logisch voneinander getrennt. Dies ist ein Punkt, der sich für die SSO-Nutzung später als bedeutend herausstellen wird.

**Applikation:** Für Cryptoki stellt jeder Prozess des Computers eine separate Applikation dar. Für die Nutzung des Tokens muss sich ein Prozess zum Token verbinden und ruft dazu die Cryptoki-Funktion `C_Intitialize` auf. Cryptoki ordnet diese Verbindung einer Applikation bis zum finalen Aufruf

[21] BSI TR-03112: `http://www.bsi.de/literat/tr/tr03112/index.htm`

von `C_Finalize` oder dem Entfernen des Tokens zu. Demzufolge stellt jeder Prozess auf dem Computer, und damit auch jedes Anwendungsprogramm, eine eigene Cryptoki-Applikation dar. Mehrere von einem Nutzer innerhalb einer Rechner-Sitzung gestarteten Programme können deshalb nicht zu einer PKCS#11-Applikation zusammengefasst werden. Dieser Umstand ist auch für gestartete Kindprozesse zu beachten, für die jeweils eine weitere Token-Verbindung erforderlich ist. Diese sind somit separate Cryptoki-Applikationen.
Cryptoki-Applikationen agieren getrennt voneinander. Somit kann jede Applikation ein Token als eine von ihr allein genutzte Ressource ansehen. Trotzdem können Änderungen an den Daten innerhalb des Tokens jederzeit durch andere Applikationen erfolgen. Dabei werden aber die restlichen Applikationen nicht von der Änderung informiert, auch wenn diese global genutzte Daten betrifft. Folgendes Szenario ist möglich:

*Eine Applikation verbindet sich zum Token, authentifiziert sich gegenüber diesem und liest ein enthaltenes Zertifikat aus, um sich damit gegenüber einem Server zu authentifizieren. Um die folgende Challenge zu beantworten, benötigt die Applikation den ebenfalls im Token enthaltenen zugehörigen privaten Schlüssel. In dieser Zeit verbindet sich eine zweite Applikation zum Token, authentifiziert sich mit Schreibberechtigung und löscht Zertifikat und Schlüssel. Wenn nun die Challenge bei der ersten Applikation eintrifft, fehlt der private Schüssel zur Berechnung der passenden Response, obwohl er nicht von der ersten Applikation gelöscht wurde.*

Derartiges Verhalten tritt allerdings selten auf, da Objekte wie Token-Schlüssel und -Zertifikate eher von statischer Natur sind und nur bei der Initialisierung geändert werden. Trotzdem müssen PKCS#11-Applikationen mit diesem Umstand rechnen.

**Objekte:** Für eine Applikation stellt sich ein Token als Gerät zur Speicherung von Objekten dar, welches kryptographische Operationen auf diesen Objekten ausführen kann.
Die Objekte bilden eine hierarchische Struktur nach Abbildung 5.7. Neben nicht näher spezifizierten Daten, unter denen eine Applikation beliebige Informationen ablegen kann, gibt es Zertifikate und Schlüssel, die mit definierter Darstellung der Anwendung zur Verfügung stehen.
Für jedes Objekt lassen sich Zugriffsrechte festlegen. Aufgrund der Sicherheitsanforderungen bietet Cryptoki die Möglichkeit, Schlüssel gesondert zu schützen. So verhindert das Attribut `sensitive` ein Auslesen aus dem Token im Klartext und `unextractable` das Auslesen egal in welcher Form. Private Schlüssel werden auf diese Weise gesichert. Somit sind private Schlüssel nicht kopierbar, insofern es sich um Hardware-Token handelt. Software-Token, für welche die Cryptoki-API ebenfalls bereitgestellt werden kann, können diese Art eines „Kopierschutzes“ jedoch nicht bieten.

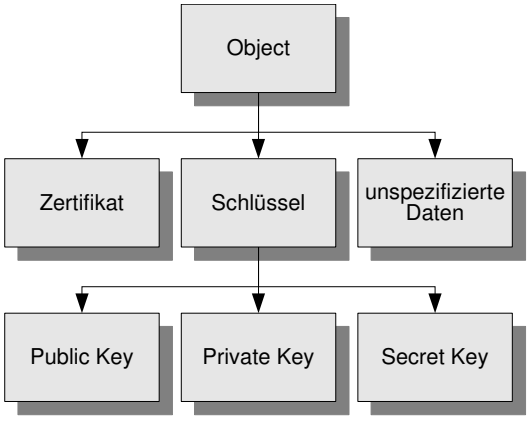



**Abb. 5.7 —** PKCS#11: Objekt Hierarchie

**Login:** Das PKCS#11 zugrunde liegende Rechtesystem erlaubt Objekte, die jeder lesen kann (*public objects*) und geschützte Objekte (*private objects*), vor deren Benutzung eine Berechtigung nachgewiesen werden muss. In Folge dessen muss sich ein Nutzer bzw. die Applikation des Nutzers gegenüber

dem Token authentifizieren. Dies kann durch Eingabe einer PIN oder auf andere, durch Token und Terminal bestimmte Art und Weise erfolgen. Einige Token erlauben z.B. die biometrische Authentifizierung über Fingerabdrücke. Die Authentisierungsmerkmale müssen im Schritt der Personalisierung, während des Enrollment-Prozesses im Token gespeichert werden. Dazu sind spezielle PIN-Objekte vorzusehen, die analog zu den privaten Schlüsseln nicht ausgelesen werden dürfen. Ein Nutzer sollte aber die eigene PIN ändern können.

**Nutzer:** Cryptoki erlaubt das Lesen der *public objects* ohne Anmeldung eines speziellen Nutzers am Token. Für die Verwendung der kryptographischen Funktionen auf den *public objects* benötigen die meisten PKCS#11-Implementierungen eine vorherige Nutzerauthentifizierung. Schreibende Änderungen für bestimmte Objekte und die Verwendung der *private objects* erfordern in jedem Fall eine vorherige Authentifizierung. Cryptoki sieht zwei Arten von Nutzern vor:

- Der „normale“ Nutzer (`User`) ist der Inhaber des Tokens. Er besitzt eine PIN, um sich gegenüber dem Token zu authentisieren.
- Der *Security Office* (`SO`) ist ein spezieller Nutzer, welcher die Initialisierung des Tokens durchführt und sich dafür mit der SO-PIN gegenüber dem Token authentisieren muss. Der SO darf die User-PIN initialisieren.

Je nach Token und PKCS#11-Provider können SO und Nutzer auch die selbe Person sein, so dass kein spezieller SO-Status erforderlich ist. Für den Hochschuleinsatz wäre dies aber von Nachteil, da eine Nutzer-Token Zuordnung, die von zentraler Stelle vergeben wird, nicht durch den Nutzer gelöscht und das Token anschließend zweckentfremdet verwendet werden sollte.
Die PKCS#11-Spezifikation besitzt eine Einschränkung gegenüber anderen Token-Schnittstellen, z.B. der von SIM-Karten in Mobiltelefonen[22]. SIM-Karten werden mit PIN und PUK (*Personal Unblocking Key*) für einen Nutzer initialisiert. Bei mehrmaliger Fehleingabe der PIN wird die Karte gesperrt und der gleiche Nutzer kann sie mit Eingabe der richtigen PUK wieder entsperren. Die PUK ist jedoch keine SO-PIN, denn sie liegt in den Händen des Nutzers. Für die Initialisierung der Karte existiert eine weitere PIN, die sich aber nur in Händen der Mobilfunkprovider befindet. Die PKCS#11-Spezifikation sieht keine PUK vor, was als Nachteil angesehen werden kann. Zwar unterstützt PKCS#11 selber kein drei-PIN-System, aber falls das Token bzw. das Betriebssystem des Tokens mehrere PINs erlauben, so kann die PUK-Funktionalität unter Beibehaltung der PKCS#11-Kompatibilität integriert werden. Dazu wird ein weiteres Programm benötigt, das bei Authentifizierung mittels PUK die Nutzer-PIN der Karte setzen kann. PKCS#11 und dessen API bieten diese Funktion nicht [Cha06].

**Session:** Die Berechtigung einer Applikation gegenüber dem Token basiert auf dem aktuellen Zustand, in dem sich die Applikation befindet. Dieser Zustand wird durch Sessions definiert, die von den Applikation nach Verbindung zum Token geöffnet werden. Man unterscheidet zwischen `R/O` (*read only*) und `R/W` (*read/write*) Sessions von nicht angemeldeten (*public session*) und angemeldeten Nutzern (*user* oder *SO session*). Session-Übergänge sind nach Abbildung 5.8 möglich.
Eine Applikation kann gleichzeitig mehrere Sessions öffnen, der Anmeldestatus (Login/Logout-Status) ist aber für die Applikation von globaler Natur. Meldet sich ein Nutzer bzw. der SO in einer Session an, so ändert sich auch der Zustand der anderen Sessions einer Applikation auf den beim Login

[22] Smartcards in Mobiltelefonen werden als SIM-Karte (*Subscriber Identity Module*) bezeichnet, da sie der Bestimmung des Mobilfunkteilnehmers im Mobilfunknetz dienen.

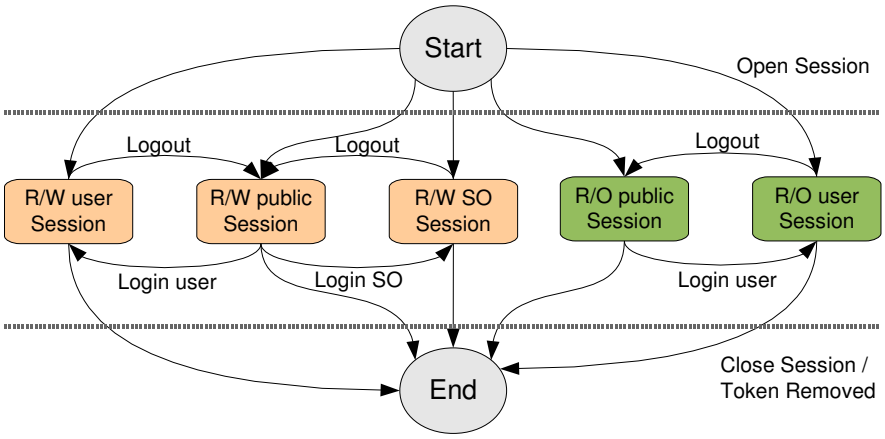



**Abb. 5.8 —** Mögliche Zustandsübergänge für Cryptoki-Sessions

festgelegten Typ, also z.B. von `R/W-public` auf `R/W-user`, falls dies möglich ist. Läuft bereits eine Session des SO, so kann die zugehörige Applikation keine `R/O`-Session öffnen, da es keine `R/O`-Sessions für den SO gibt.
Eine neue Session kann durch Aufruf der Funktion `C_OpenSession` gestartet werden. Die erste Session startet immer im *public* Status, weitere im jeweils aktuellen Status der Applikation. Ein Übergang zu einem anderen Anmeldestatus erfolgt durch Aufruf der Cryptoki Funktion `C_Login` mit Angabe der entsprechenden PIN oder durch `C_Logout`. Beenden lassen sich eine oder alle Sessions durch `C_Close(All)Session(s)` oder durch Entfernen des Tokens vom System.
Durch den Status wird der Zugriff auf die Objekte (dazu zählen die im Token gespeicherten und die temporären Objekte, die nur für die Dauer der Session existieren) geregelt, ebenso die Möglichkeit bestimmte Cryptoki-Funktionen aufzurufen. Aufgrund dieser Zugriffsbeschränkungen und der Tatsache, dass Applikationen getrennt voneinander auf das Token zugreifen, kann es erforderlich sein, dass die PIN mehrmals eingegeben werden muss. Alle Nutzerprogramme, welche das Token für die Authentifizierung entsprechend Kapitel 4.3.2 einsetzen, müssen jeweils separat ein Login durchführen und dabei die PIN angeben. Dieser Umstand muss bei der Nutzung von PKCS#11 als SSO-Basis beachtet werden.

### 5.3.2 Cryptoki-Funktionen

Die Cryptoki API unterstützt eine Vielzahl kryptographischer Funktionen. In Abhängigkeit vom Provider und den verwendeten Schlüsseln werden Teil- oder komplette Funktionen auf dem Prozessor des Tokens ausgeführt. Andere Funktionen führt die Provider-Bibliothek auf dem angeschlossenen Rechner aus. Tabelle 5.1 zeigt die Cryptoki-Kategorien entsprechend der Beschreibung der PKCS#11-Spezifikation.
Die Spalte Management umfasst Funktionen zur Initialisierung der Token und Verwaltung der Daten. Dazu zählen das Anlegen der User-PIN, das Lesen und Speichern von Objekten und die Funktionen zur Session-Verwaltung. Die Schlüssel werden als gesonderte Objekte behandelt. Sie können entweder im Token selber erzeugt werden (was bei asymmetrischen Schlüsseln den Vorteil hat, dass der private Schlüssel das Token nicht verlassen muss) oder zum Token übertragen werden. Zum Key-Management zählen auch kryptographische Funktionen zum Ver- und Entschlüsseln von Schlüsseln (`C_WrapKey` und `C_UnwrapKey`) sowie zum Ableiten von Unterschlüsseln `C_DeriveKey`. Die

| Management | Crypto-/Hash-Functions | Others |
|---|---|---|
| General purpose | Encryption / Decryption | Random number generation |
| Slot/Token management | Message Digest | Callback functions |
| Session mangement | Signature and MACing | |
| Object management | Verify signatures and MACs | |
| Key management | Dual purpose | |

**Tabelle 5.1 —** Kategorien der Cryptoki Funktionen

`Wrap`-Funktionen dienen dem Auslesen oder Schreiben von Schlüsseln, die außerhalb des Tokens nur in verschlüsselter Form vorliegen sollen. Das Ableiten dient dazu, aus einem Basisschlüssel weitere Schlüssel für die aktuelle Sitzung zu generieren. So kann z.B. der aus einer Diffie-Hellman-Merkle Schlüsselvereinbarung gewonnene geheime Wert als Basis zur Ableitung eines symmetrischen Schlüssels genutzt werden.

Die Crypto- und Hash-Funktionen umfassen die breite Palette zur Ver- und Entschlüsselung mit symmetrischen und asymmetrischen Verfahren, zur Berechnung von Hashwerten auf Daten (Message Digest) oder auf geheimen Schlüsseln (Digest Key) sowie zur Berechnung von elektronischen Signaturen oder des Message Authentication Codes[23] bzw. des HMAC-Wertes[24] einer Nachricht. Die Umkehrung, also die Überprüfung einer Signatur oder des (H)MAC-Wertes gehören ebenfalls zum Funktionsumfang. Außerdem gibt es zusammengesetzte Funktionen, z.B. zur gleichzeitigen Berechnung einer Signatur und der Verschlüsselung einer Nachricht.

Außerdem bietet Cryptoki die Möglichkeiten zum Generieren zufälliger Werte und den Einbau zusätzlicher Funktionen, die durch den Token-Hersteller definiert werden können.

### 5.3.3 PKCS#11-Provider

Für die Benutzung von Cryptoki werden für die eingesetzten Token passende PKCS#11-Provider benötigt. Neben Providern für spezielle Smartcards oder USB-Token, existieren auch Token-unabhängige Provider. Weiterhin gibt es PKCS#11-Provider für Software-Token, die kryptographische Funktionen allein in Software auf dem Rechner umsetzen und Dateien anstatt einer Hardware zur Speicherung aller Objekte verwenden. Die Nutzung von Hardware-Token und die damit verbundene Absicherung gegen Vervielfältigung stellt aber einen Eckpfeiler für die Sicherheit des SSO-Systems dar, auf den nicht verzichtet werden kann. Deshalb werden PKCS#11-Soft-Token nicht weiter betrachtet.

Die NSS (siehe Kapitel 5.2.3) können ebenfalls als PKCS#11-Provider genannt werden. Die NSS-Spezifikation greift nicht nur selber auf PKCS#11-Schnittstellen für die Token-Einbindung zu, sondern bietet Applikationen eine eigene PKCS#11-Schnittstelle. Allerdings wird momentan nur ein Teil von Cryptoki angeboten. Nach Entwickleraussagen ist es eine Behelfslösung für die Verwendung von NSS in PKCS#11-fähigen Applikationen ohne eigene NSS-Unterstützung[25]. Demzufolge soll NSS hier auch nicht näher betrachtet werden.

[23] Als *Message Authentication Code* (MAC) bezeichnet man einen Prüfwert einer Nachricht mit dem Ziel, Integrität und Authentizität zu wahren. Vor Manipulation ist er mittels symmetrischer Verschlüsselung gesichert.

[24] HMAC: *keyed-Hash Message Authentication Codes* sind spezielle MAC-Verfahren, bei denen Hashwerte der Nachricht in Verbindung mit einem geheimen Schlüssel zur Berechnung des MAC-Wertes genutzt werden [RFC2104, RFC2202].

[25] `https://developer.mozilla.org/en/PKCS11`

**OpenSC:** Das OpenSC-Projekt[26] umfasst eine Sammlung aus Bibliotheken und Programmen für den Zugriff auf Crypto-Token. OpenSC wird als Open Source unter der GNU LGPL[27] entwickelt und steht neben dem Betriebssystem Linux auch für Solaris, Microsoft Windows, MacOS-X und auf weiteren Systemen zur Verfügung. OpenSC setzt einen Großteil der PKCS#11-Spezifikation um (aktuell Version 2.20). Unterstützt wird eine Vielzahl verschiedener Smartcards und USB-Token. Die Token und die enthaltenen Daten werden von OpenSC nach dem PKCS#15 Standard[28] initialisiert und verwendet. Dadurch sind die von OpenSC initialisierten Token auch mit anderen PKCS#11-Providern einsetzbar, insofern diese ebenfalls dem PKCS#15-Standard folgen.
Die meisten für diese Arbeit erforderlichen Tests und Implementierungen erfolgten mittels OpenSC. Anschließend wurde auf Kompatibilität mit anderen PKCS#11-Providern getestet. Dabei zeigte sich, dass die SSO-Lösung unter allen getesteten Providern funktioniert.

### 5.3.4 Authentifizierung mit Crypto-Token und PKCS#11

Eine Authentifizierung unter Nutzung von PKCS#11 und der Cryptoki-API kann auf einer der drei in Abbildung 5.9 gezeigten Methoden erfolgen.

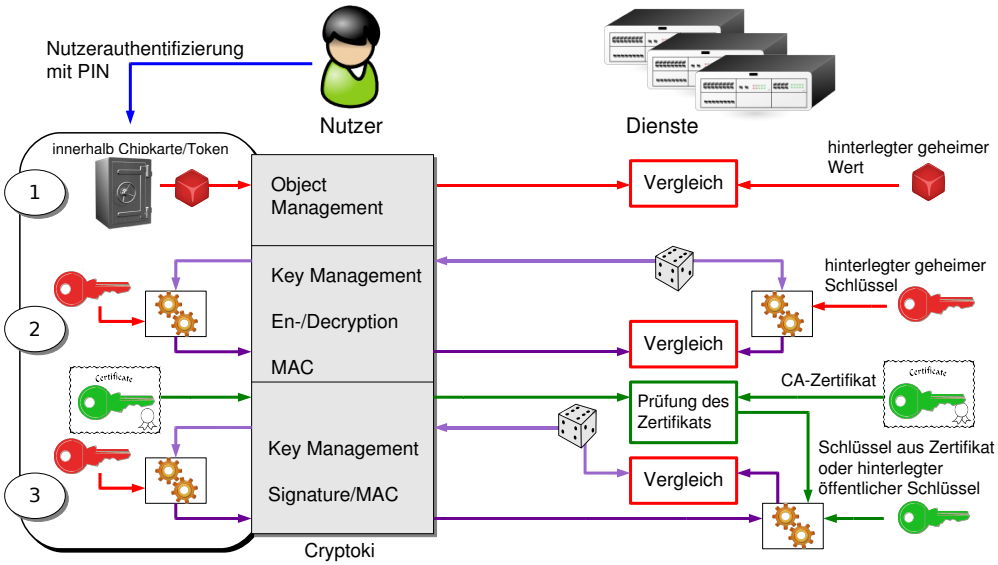



**Abb. 5.9 —** Authentifizierung mit Token und PKCS#11

1. Im Token wird ein Geheimnis (ein geschütztes Datenobjekt) hinterlegt, welches auch auf Seiten des Dienstes vorhanden ist. Ein gegenüber dem Token authentifizierter Nutzer kann dieses Objekt auslesen und als Authentisierungsmerkmal zum Dienst übertragen.
   Mittels Cryptoki wäre diese Art der Authentifizierung durch Verwendung geschützter Datenobjekte umsetzbar. Es entspräche von der Arbeitsweise der Verwendung von Passwörtern und hätte den Nachteil von auslesbaren und somit kopierbaren Merkmalen.

[26] OpenSC: *Open SmartCard*, `http://www.opensc-project.org/`
[27] LGPL: *Lesser General Public License*, `http://www.opensource.org/licenses/lgpl-2.1.php`
[28] PKCS#15 wird verwendet, insofern dies mit der Hardware des Tokens möglich ist. Einige Token unterstützen z.B. keine Speicherung von Zertifikaten, können aber trotzdem mit OpenSC genutzt werden. Dabei wird ein für das Token speziell angepasstes Format verwendet.

2. Unter Verwendung symmetrischer Kryptographie kann in einem Challenge-Response-Verfahren das Vorhandensein des passenden Schlüssels im Token erfragt werden. Dazu kann ein Response-Wert durch Berechnung des HMAC-Wertes der übertragenen Challenge innerhalb des Tokens generiert und anschließend zum Dienst übertragen werden.
   Cryptoki bietet hierfür die Funktionen aus dem Bereich der Schlüsselverwaltung und Berechnung von MAC-Werten, vorausgesetzt das Token unterstützt symmetrische Verschlüsselung. Der Nachteil dieses Verfahrens besteht darin, dass es sich um ein gemeinsames Geheimnis handelt, das auch auf Seiten der Dienste vorhanden sein muss und damit das Vertrauen der Anbieter untereinander erfordert. Es ließen sich auch mehrere Schlüssel für unterschiedliche Dienste verwenden, was aber die Frage nach Aufwand und Wartbarkeit aufwirft.
3. Asymmetrische kryptographische Verfahren umgehen das Problem der externen Aufbewahrung geschützter Authentisierungsmerkmale. In einem Challenge-Response-Verfahren entsprechend Kapitel 4.2.2 und 4.3.2 wird innerhalb des Tokens mit dem enthaltenen privaten Schlüssel der Response-Wert berechnet. Das kann z.B. durch eine Signaturoperation auf dem Challenge-Wert erfolgen. Die Prüfung des Response-Wertes seitens des Dienstes erfolgt mittels des öffentlichen Schlüssels, der entweder dem Dienst vorliegt oder aus dem Zertifikat entnommen werden kann. Dazu wäre das Token-Zertifikat vorher zu übertragen und vom Dienst zu prüfen.
   Der Vorteil dieses Verfahrens liegt darin, dass die Service Provider lediglich die öffentlichen und damit frei verfügbaren Schlüssel benötigen. Werden statt öffentlicher Schlüssel Zertifikate verwendet (siehe Kapitel 4.3.2 und 2.3) so kann auf einer serverseitigen Speicherung der Schlüssel verzichtet werden. Dazu müssen die Dienst-Anbieter den Identitätsprovidern, in diesem Fall den Zertifikatsausstellern vertrauen.
   Cryptoki unterstützt diese Authentifizierung durch Funktionen für asymmetrische Verfahren (Encryption, Signature)und verwaltet Schlüssel und Zertifikate im Token.

Als universelles Authentifizierungsverfahren zur Ablösung von Passwörtern bietet sich die dritte Variante an. Der wesentliche Vorteil ist die nicht mehr zwingend erforderliche Speicherung geheimer Merkmale außerhalb des Tokens. Werden Token-Schlüssel neben der Authentifizierung auch zur Verschlüsselung verwendet, dann ist aus Key-Recovery-Gründen die Erzeugung und Speicherung der Schlüssel außerhalb der Token trotzdem sinnvoll. So können bei physikalischer Token-Beschädigung die Daten weiterhin entschlüsselt werden. Die Schlüsselhinterlegung sollte aber auf eine zentrale vertrauenswürdige Stelle beschränkt werden.
Die Wahl dieser Stelle kann sich als problematisch erweisen. Dazu muss der Schutz der hinterlegten Daten geklärt werden und wer den Zugriff auf die Daten erhält. Eine mögliche Variante besteht in einer verschlüsselten Speicherung, nach dem Vier- oder Mehr-Augen-Prinzip. Nur wenn mehrere Personen ihren Teil des Schlüssels zusammenfügen, kann der hinterlegte Schlüssel wiederhergestellt werden. Welche Gruppe von Personen ein Recovery erlaubt, muss für die jeweilige Einrichtung in einer Policy (PKI-Policy) festgelegt und veröffentlicht werden. Nur wenn die Nutzer davon ausgehen, dass nicht gegen ihren Willen ein Zugriff auf ihre persönlichen Schlüssel erfolgt, kann eine breite Akzeptanz erreicht werden. Andernfalls bestünde die Gefahr, dass Nutzer den Einsatz von Token auf den Bereich der Authentifizierung (Passwortersatz) beschränken und die Verwendung für Verschlüsselung und Signatur nicht in Erwägung ziehen.
Können die Fragen der Schlüsselhinterlegung gelöst werden, so steht ein Hardware-Crypto-Token-gestütztes Verfahren für die Nutzerauthentifizierung auf Basis von Zertifikaten mit Potential für weitere Anwendungen zur Verfügung. Damit bliebe noch zu klären, wie sich Hardware-Token in ein SSO-System mittels PKCS#11 unter Wahl der dritten Authentifizierungsvariante integrieren lassen.

# Kapitel 6

# Clientseitiges Token-gestütztes SSO

PKCS#11 wurde zur Vereinfachung kryptographischer Applikationen bei Verwendung von Hardware-Crypto-Token entwickelt. Die Token können als Authentisierungsmerkmal für asymmetrische Verfahren in Verbindung mit Zertifikaten genutzt werden. Der Ablauf zertifikatsbasierter Authentifizierung wurde im Kapitel 4.3.2 beschrieben. Abweichungen sind möglich, aber im Kern ähneln sich die Verfahren. Als Ersatz für den Passwort-Einsatz an Hochschulen bietet sich zertifikatsbasierte Authentifizierung mit Einbindung von Crypto-Token an. Damit lassen sich die Probleme des sicheren Transports und der Aufbewahrung privater Schlüssels in den Händen unbedarfter Endanwender lösen. Zertifikatsbasierte Authentifizierung kann außerdem gut mit SSL/TLS-geschützter Kommunikation verbunden werden, was den Vorteil der gesicherten Datenübertragung im Netzwerk bietet.
Voraussetzung für den Einsatz des Verfahrens ist die Unterstützung durch Software und Dienste. Für die Einbindung von Crypto-Token ist außerdem eine clientseitige Token-Schnittstelle erforderlich. Sind beide Voraussetzungen erfüllt, kann zertifikatsbasierte Authentifizierung mit Crypto-Token umfassend eingesetzt werden. Aus Sicht eines Anwenders stellt sich allerdings die Frage der Akzeptanz, insbesondere wenn die Sicherheitsprobleme der bisher eingesetzten Passwortverfahren nicht bekannt sind oder gezielt ignoriert wurden. War bisher die Kenntnis eines Passwortes ausreichend, so ist für zertifikatsbasierte Authentifizierung erhöhter Hardwareaufwand (Smartcard-Terminal oder freier USB-Anschluss) und die Mitführung des Tokens erforderlich. Deshalb soll als weiterer Anreiz die Authentifizierung in Kombination mit Token über die Einführung von Single Sign-On erleichtert und darüber hinaus weitere Nutzeffekte ermöglicht werden.
In diesem Kapitel wird gezeigt, wie sich die zertifikatsbasierte Authentifizierung unter Verwendung von Hardware-Token und der PKCS#11-Schnittstelle clientseitig mit Single Sign-On Fähigkeiten erweitern lässt. Ziel soll es sein, nach dem Login am Rechner keine weiteren Eingaben für Authentifizierungen vom Nutzer zu fordern. Die Nachteile von Passwörtern und deren SSO-Lösungen dürfen dabei nicht wieder einfließen.

## 6.1 Server- und clientseitiges SSO

Wie im Kapitel 3.3 erläutert, umfasst Single Sign-On einen Verbund von Diensten, bei denen nur eine einmalige Authentifizierung bei der Anmeldung am ersten oder einen speziell dafür vorgesehenen Dienst erforderlich ist. Alle weiteren Dienste im Verbund können ohne erneute Authentifizierung genutzt werden. Betrachtet man die technische Umsetzung von SSO-Verfahren, so kann man zwischen server- und clientseitigem SSO unterscheiden.

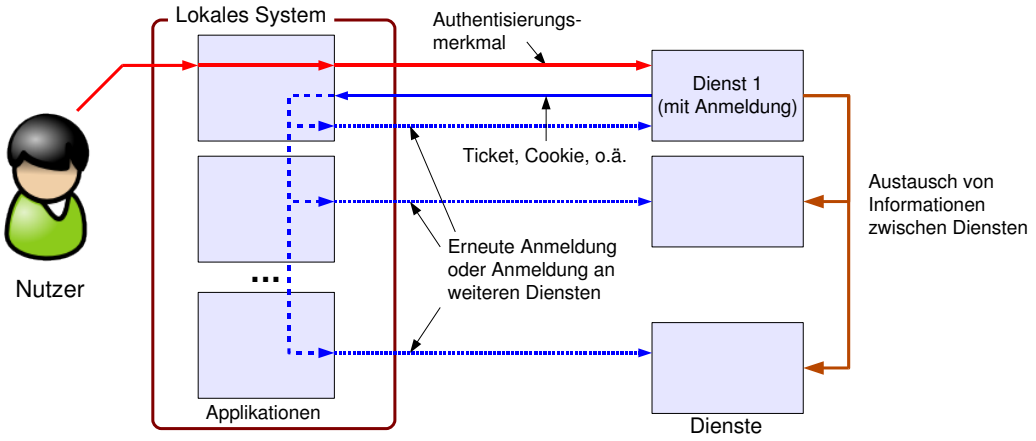



**Abb. 6.1 —** SSO mit serverseitiger Kommunikation

**Serverseitiges SSO:** Eine Möglichkeit besteht in einer Anpassung der Serverdienste im Zusammenspiel mit der Client-Software. Der Ablauf startet mit der Authentifizierung gegenüber einem Serverdienst. Ob es sich hierbei um einen zentralen Server, wie z.B. der AS bei Kerberos handelt und auf welche Art diese Authentifizierung abläuft, ist dabei unerheblich. Bei Erfolg wird die Information über den Nutzer allen Diensten im SSO-Verbund mitgeteilt. Dies kann durch eine Propagierung der Information im Verbund oder durch Hinterlegung an zentraler Stelle erfolgen. Zum Client wird hierbei eine Information übertragen (bei Kerberos das TGT), mit der die clientseitige Applikation sich bei anderen Diensten im Verbund anmelden kann. Bei Nutzung verschiedener Applikationen muss diese Information auch innerhalb des Clients propagiert werden. Sie sollte deshalb gegen Entwendung oder Manipulation z.B. kryptographisch geschützt sein, um Missbrauch zu verhindern.

Ein Beispiel für servergestütztes SSO ist die häufig anzutreffende SSO-Verbindung von Webservern. Die Authentifizierung des Nutzers erfolgt mit dem Browser gegenüber einem Webserver-Dienst oder einem speziellen Identitätsprovider. Bei erfolgreicher Authentifizierung erhält er ein Cookie[1] mit einer zeitlich begrenzten Information und speichert es im Webbrowser. Serverseitig wird eine Nutzer-Sitzung gestartet. Da im Normalfall keine dauerhafte Verbindung zwischen Browser und Dienst besteht, muss sich der Nutzer bei folgenden Anfragen erneut ausweisen. Hierfür genügt die Übertragung des Cookies zum Dienst, welcher daran die Sitzung und somit den Nutzer erkennen kann. In Abhängigkeit vom Geltungsbereichs kann das Cookie auch von anderen Servern erfragt werden. Dazu müssen die Dienste untereinander die Cookie-Nutzer-Zugehörigkeit austauschen, eine serverseitige Vernetzung wäre erforderlich.

Mittels geeigneter kryptographischer Methoden kann auf die Verbindung der Dienste untereinander verzichtet werden. Hierzu hinterlegt der erste Server eine Information über den authentisierten Nutzer im Cookie. Diese Information wird derart verschlüsselt, dass die anderen Dienste sie aus dem abgefragten Cookie nach Entschlüsselung entnehmen können. Der Nutzer hingegen, über dessen Rechner diese Cookies übertragen werden, kann aufgrund fehlender Schlüssel die Information nicht modifizieren, ohne sie gleichzeitig ungültig zu machen. Die Server untereinander müssen im Vorfeld einen gemeinsamen Schlüssel vereinbart haben. Für den Apache-Webserver wurde diese Funktionalität in

[1] *Cookie* (engl. für Keks) stammt vom Begriff *Fortune Cookie* (Glückskeks) ab und ist eine Datenkapsel, welche eine kleine, meist nicht im Klartext lesbare Information enthält, die auf Rechnern hinterlegt werden kann und zwischen Rechner und Diensten ausgetauscht wird.

DACS[2] implementiert. DACS ermöglicht SSO-Federations für Browserdienste (siehe Kapitel 4.5.4). OpenID nutzt ebenfalls clientseitig übertragene Cookies zur Übermittlung der Identität.

Eine Nutzung der Cookies durch andere Client-Applikationen für weitere Dienste, z.B. für den Email-Versand im Emailreader ist aber nicht vorgesehen. Lediglich Kerberos ist mit Austausch des TGT zwischen den einzelnen Client-Applikationen dafür ausgelegt.

Anzumerken wäre, dass für Kerberos, OpenID und bei Nutzung von DACS eine serverseitige Anpassung durch Zusatzsoftware erforderlich ist, bei Kerberos sogar der Einsatz angepasster Client-Software.

**Clientseitiges SSO:** Die andere Möglichkeit zur Umsetzung von SSO besteht in einer ausschließlich clientseitigen Lösung. Voraussetzung ist, dass der Nutzer für alle Dienste die erforderlichen Authentisierungsmerkmale besitzt. Durch die Client-Applikation sind die erforderlichen Merkmale bei Bedarf automatisch zu ermitteln und an den Dienst zu übertragen, ohne dass der Nutzer hierfür zusätzliche Eingaben tätigen muss.

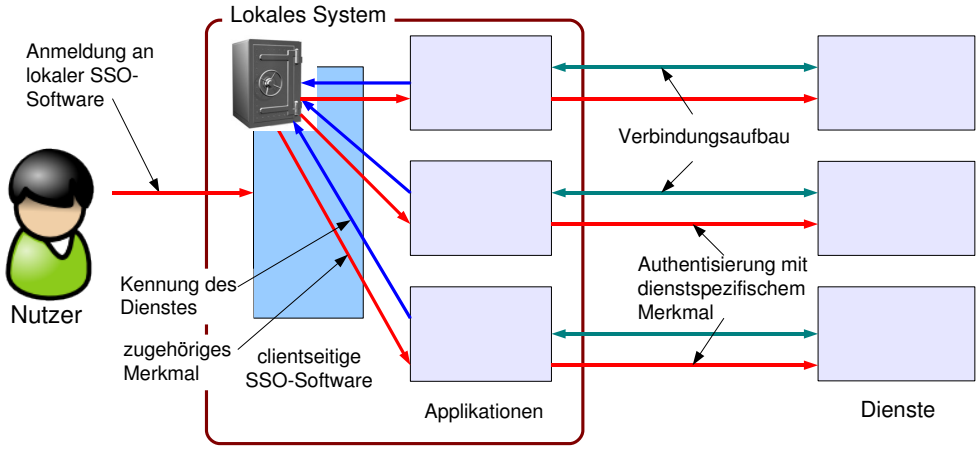



**Abb. 6.2 —** Clientseitiges SSO

Im Gegensatz zur servergestützten Lösung, bei der nach einer erfolgreichen Authentifizierung nur speziell dafür vorgesehene Informationen (Cookies oder TGTs) übertragen werden, arbeitet die clientbasierte Lösung bei jeder Verbindung mit den diensteigenen Authentisierungsmerkmalen des Nutzers. Bei Einsatz von Passwörtern bedeutet es, das für jeden weiteren Dienst das jeweils erforderliche Passwort übertragen wird. Für den Server unterscheidet es sich dabei nicht, ob der Nutzer selber die Authentisierungsmerkmale vorlegt oder es durch einen clientseitigen Automatismus erfolgt. Ein Vorteil clientseitiger Lösungen besteht somit darin, dass keine Anpassungen für die Server erforderlich sind.

Eine praktische Umsetzung zeigt die Verwendung des im Kapitel 4.5.1 vorgestellten Passworttresors in Kombination mit passwortbasierter Authentifizierung. Da sich Nutzer nicht mehr alle Passwörter merken müssen, können diese strenge Kriterien erfüllen. Anderseits besitzen Tresorlösungen Nachteile, wenn es um die transportable Aufbewahrung von Passwörtern geht. Außerdem können die Passwörter trotz Tresor weiterhin durch die Nutzer vorgegeben und damit auch kopiert werden. Zertifikatsbasierte Authentifizierung in Verbindung mit Crypto-Token hätte diesen Nachteil nicht.

[2] DACS: *Distributed Access Control System*, `http://dacs.dss.ca`

### 6.1.1 Zertifikatsbasierte Authentifizierung mit clientseitigem SSO

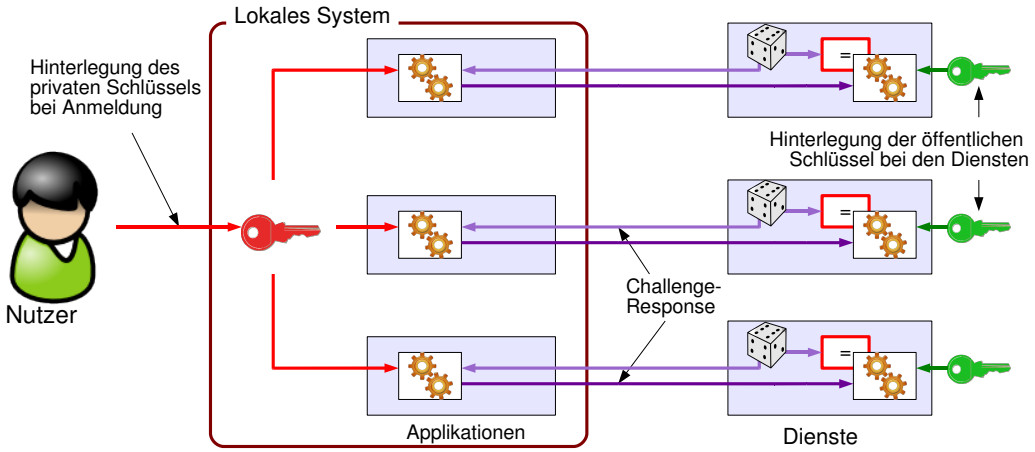



**Abb. 6.3 —** SSO mit Public-Key-Authentifizierung

Eine einfache Umsetzung für clientseitiges SSO mit asymmetrischer Kryptographie ohne Zertifikate zeigt Abbildung 6.3. Diese als SSO mit Public-Key-Authentifizierung bezeichnete Variante entspricht einem Challenge-Response-Verfahren unter Nutzung asymmetrischer Verschlüsselungs- bzw. Signaturverfahren. Der Nutzer übergibt das Schlüsselpaar, insbesondere den als Authentisierungsmerkmal erforderlichen privaten Schlüssel an das lokale System. Die Applikationen können selbständig darauf zugreifen und diesen zur Beantwortung der Challenge nutzen.
Der Nachteil ist, dass der öffentliche Schlüssel des Nutzers bei allen Diensten hinterlegt sein muss. Wird dieser Schlüssel erneuert, z.B. weil der private Schlüssel kompromittiert wurde, so muss er an alle Dienste übertragen werden. Bei Verwendung vieler Dienste führt das zu hohen Aufwand.

Einfacher ist es, wenn clientseitiges SSO mit asymmetrischer Kryptographie und Zertifikaten, wie in Abbildung 6.4 dargestellt, eingesetzt wird. Die Funktionsweise der Authentifizierung entspricht

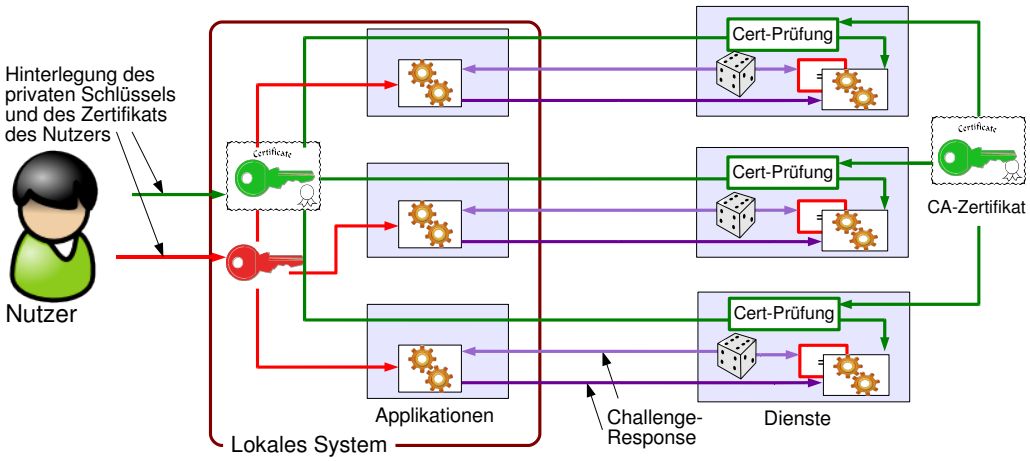



**Abb. 6.4 —** SSO mit zertifikatsbasierter Authentifizierung

Punkt 3 aus Kapitel 5.3.4. Die im lokalen System laufenden Applikationen haben Zugriff auf das Nutzer-Zertifikat, welches sie den Diensten präsentieren. Diese prüfen die Gültigkeit mit dem CA-

Zertifikat und starten bei Erfolg eine Challenge. Die Applikationen können diese wiederum mit dem privaten Schlüssel erfolgreich beantworten.
Für SSO hinterlegt der Nutzer Zertifikat und Schlüssel im System, so dass die Applikationen bei Bedarf darauf zugreifen können. Für den Fall eines kompromittierten privaten Schlüssels kann durch weitere Maßnahmen, wie der Verwendung von Zertifikatssperrlisten oder der Online-Abfrage gültiger Zertifikate (OCSP[3] oder SCVP[4]) auf Seiten der Dienste eine Sperrung des Schlüssels vor Ablauf des Zertifikats erreicht und das System gegen größeren Missbrauch gesichert werden. Dabei muss der Nutzer gegenüber den Diensten nicht aktiv werden, er muss lediglich rechtzeitig die Sperrung des Zertifikats an einer zentralen Stelle veranlassen. Ist es aus anderen Gründen erforderlich, ein neues Schlüsselpaar zu erzeugen, muss kein neuer Schlüssel bei den Diensten hinterlegt werden, vorausgesetzt zum öffentlichen Schlüssel wird wieder ein Zertifikat vom gleichen Aussteller vergeben.

Bei den Verfahren mit clientseitigem SSO auf Basis von Public-Key- oder zertifikatsbasierter Authentifizierung kann der gleiche Schlüssel für mehrere Provider genutzt werden. Die Provider selber besitzen nur den öffentlichen Schlüssel bzw. das CA-Zertifikat, so dass ein Missbrauch der Provider untereinander ausgeschlossen ist. Somit lässt sich mit relativ geringem Aufwand ein clienstseitiges SSO umsetzen, allerdings unter der Voraussetzung, dass die Serverdienste eine Authentifizierung mit Zertifikaten oder zumindest mit Public-Key-Verfahren erlauben.
Für die Sicherheit des Systems ist das asymmetrische Verfahren und seine Implementierung, die Ausstellung der Zertifikate und insbesondere auch auf der Sicherung des privaten Schlüssels entscheidend. Dieser Schlüssel befindet sich in den Händen der Nutzer. Provider müssen darauf vertrauen, dass er bei den Nutzern sicher verwahrt ist. Da immer mit unvorsichtigen Nutzern oder kompromittierten Clientsystemen zu rechnen ist, müssen technische Maßnahmen zum besseren Schutz der privaten Schlüssel ergriffen werden. Eine Lösung mit relativ hohem Sicherheitsgewinn bietet die Einbindung von Crypto-Token in das clientseitige SSO.

## 6.2 SSO mit Crypto-Token und PKCS#11

Bieten die verwendeten Applikationen und Dienste zertifikatsbasierte Authentifizierung, so lässt sich SSO in Kombination mit Crypto-Token umsetzen, wenn die jeweiligen Applikationen Schnittstellen zum Zugriff auf die Token besitzen und diese auch im Rahmen der Authentifizierung verwenden.
Abbildung 6.5 skizziert die Einbindung des Tokens in die SSO-Umgebung. Die Authentifizierung des Nutzers am lokalen System beginnt mit Verbindung des Tokens zur Rechner-Schnittstelle. Es handelt sich um eine Zwei-Faktor-Authentisierung. Deshalb ist zusätzlich die Eingabe des zweiten Merkmals, der geheimen Information erforderlich. Für die meisten Token wird es sich um eine PIN handeln. Würde diese Freischaltung für alle im System agierenden Applikationen bestehen bleiben, so wäre das für ein clientseitiges SSO mit Public-Key- oder zertifikatsbasierter Authentifizierung ausreichend. Es könnte aber die Sicherheit einschränken, da auch Schadprogramme unbemerkt den Schlüssel verwenden könnten. Außerdem erlaubt PKCS#11 keine applikationsübergreifende Nutzung der Schlüssel.

In Verbindung mit PKCS#11 sind für eine SSO-Lösung zwei Punkte erforderlich. Der eine Punkt betrifft den Zugriff der lokalen Applikation auf das Token und damit den Schutzmechanismus von PKCS#11 vor unbefugten Applikationen. Wie im Kapitel 5.3 beschrieben, erfordert Cryptoki vor Zugriff auf geschützte Objekte eine Authentifizierung jeder einzelnen Applikation gegenüber dem

[3] OCSP: *Online Certificate Status Protocol*, ein Online-Validierungsdienst für X.509-Zertifikate [RFC2560]
[4] SCVP: Das *Server-based Certificate Validation Protocol* kann als dedizierter Nachfolger von OCSP mit erhöhtem Funktionsumfang bezeichnet werden [RFC5055]. SCVP kann z.B. Zertifikatsketten auflösen. Allerdings konnte sich SCVP im Gegensatz zu OCSP noch nicht weiträumig etablieren [Sch07a].

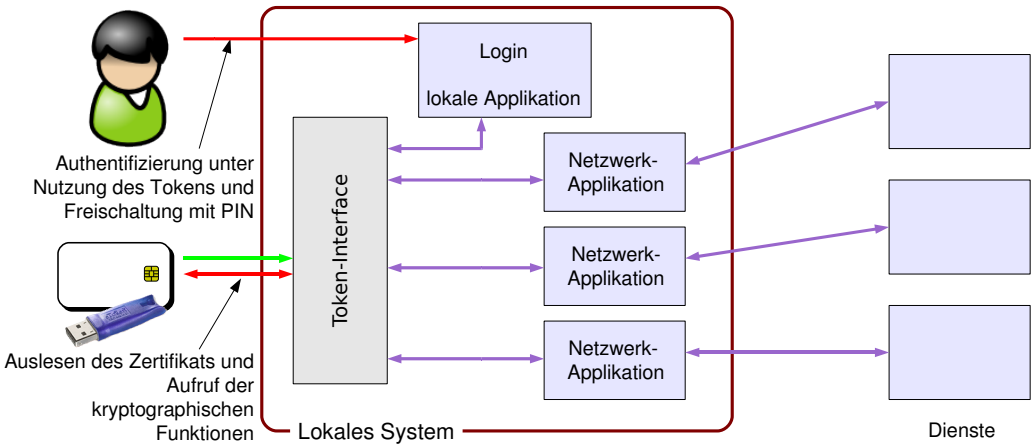



**Abb. 6.5 —** Einbindung von Token in clientseitiges SSO

Token. Hier muss ein Weg gefunden werden, sich auf eine einmalige PIN-Eingabe bei Anmeldung am Rechner zu beschränken. Der zweite Punkt betrifft die Absicherung. Wenn für mehrere Applikationen nur eine PIN-Eingabe erforderlich ist, muss durch technische Maßnahmen sichergestellt werden, dass nur berechtigte Applikationen Zugriff auf die Token erhalten. Zur Suche einer technischen Lösung für diese Punkte muss man sich die Nutzerauthentifizierung von PKCS#11 genauer anschauen.

### 6.2.1 PKCS#11: Nutzerauthentifizierung am Token

SSO mit Token wäre nach Abbildung 6.4 einfach umsetzbar, wenn der Zugriff auf Funktionen mit privaten bzw. geheimen Schlüsseln bereits in einer öffentlichen PKCS#11 Sitzung, also ohne Nutzerauthentifizierung möglich wäre. Das Vorhandensein des Tokens würde in diesem Fall für clientseitiges SSO ausreichen. Vom Prinzip her stellt es eine besitzbasierte Einfaktor-Authentisierung dar. Die Schlüssel können trotzdem nicht kopiert und die Token somit nicht dupliziert werden. Dies wäre zwar ein Vorteil gegenüber Passwörtern, der Nachteil besteht aber darin, dass ein entwendetes Token sofort missbräuchlich einsetzbar wäre. Ist die zertifikatsbasierte Authentifizierung für eine Vielzahl von Diensten möglich, würde mit der Tokenentwendung ein zentraler Angriffspunkt entstehen.
Demzufolge ist eine Lösung anzustreben, bei der Token als Authentisierungsmerkmal nur bei vorheriger Nutzerauthentifizierung gegenüber dem Token eingesetzt werden können. Die Nutzerauthentifizierung soll für alle Applikationen bestehen bleiben, bis entweder der Nutzer diese Bindung auflöst, das Token gesperrt wird oder Token und Rechner getrennt werden. Die erforderliche Authentifizierung soll aber nur einmalig, vorzugsweise beim Rechner-Login erfolgen.
Die Authentifizierung der Applikation des Nutzers gegenüber dem Token erfolgt bei PKCS#11 durch Aufruf des Befehls `C_Login` in einer Session der Applikation. Hierbei ist die PIN zu übergeben[5]. War dieser Aufruf erfolgreich, so gilt der Login-Status für alle Sessions dieser Applikation. Die PIN selber ist in Form einer Zeichenkette zu übermitteln. Der Datentyp entspricht einer UTF8-kodierten Zeichenkette, so dass theoretisch alle Unicode-Zeichen und somit beliebige Eingaben möglich sind. Es ist aber abhängig vom zugrunde liegenden Token, welche Zeichen letztendlich zugelassen sind und in welchem Bereich die Länge der PIN liegen muss.
Die PKCS#11-API Cryptoki ab Version 2.11 unterstützt Verfahren zur Authentifizierung, bei denen keine PIN übermittelt werden muss. Hintergrund ist der Umstand, dass es sicherer ist, die PIN direkt

[5] Anhang B zeigt das Protokoll der Funktionsaufrufe über die Cryptoki-Schnittstelle für eine konkrete Anwendung.

am Token einzugeben und nicht über Software zu übertragen. Innerhalb des Rechners ist die PIN der Gefahr einer Entwendung ausgesetzt. So existiert für Smartcards die Möglichkeit, bei speziellen Terminals die PIN mit einer integrierten, meist numerischen Tastatur direkt am Terminal einzugeben. Die PIN wird vom Terminal in die Übertragung der Kommando-APDU zum Token Login eingeschleift, sie liegt somit niemals im Rechner vor. Dafür wird ein Terminal mit Tastatur benötigt. Gemäß der Spezifikationen des Zentralen Kreditausschusses[6] wäre es ein Reader der Klasse 2 oder höher. Mit Reader wird das Smartcard-Terminal bezeichnet, welches beim elektronischen Zahlungsverkehr über HBCI[7] für die Verbindung zur Smartcard dient. Dabei handelt es sich um ein Terminal, das Karten nach ISO/IEC-7816 Standard verarbeiten können.
Die Token könnten aber auch andere Authentifizierungsmöglichkeiten besitzen, z.B. einen Fingerabdruckscanner direkt auf der Smartcard. Durch Setzen eines Flags signalisiert PKCS#11 dann der Applikation, dass die Authentifizierung während der Login-Funktion außerhalb von Cryptoki erfolgt. Als PIN-Parameter kann dann von der Applikation ein NULL-Pointer übergeben werden. Dieser Umstand kann indirekt für SSO ausgenutzt werden.

### 6.2.2 Erweiterung der PKCS#11 Nutzerauthentifizierung für SSO

Die PKCS#11-Spezifikation verlangt für jede Applikation eine Authentifizierung, wenn die enthaltenen privaten Schlüssel genutzt werden sollen. Für Single-Sign-On wäre eine technische Lösung notwendig, die diese Authentifizierung nur von der ersten Applikation fordert.
Erreicht werden kann dies z.B. über eine Erweiterung der PKCS#11-Spezifikation. Dafür müssten aber die Applikationen angepasst werden. Sinnvoller scheint eine Variante zu sein, bei der unter Beibehaltung der bestehenden PKCS#11-Spezifikation keine wiederholte Authentifizierung des Nutzers gegenüber dem Token erforderlich wäre. Die Herausforderung ist, dass seitens der Applikation dieser Umstand auch erkannt werden muss. So dürfen keine weiteren PIN-Abfrage erfolgen und trotzdem muss das Token uneingeschränkt genutzt werden können. Soll dies ohne Veränderung der Spezifikation und ohne Verzicht auf geschützte Objekte bzw. Schlüssel möglich sein, so muss die Authentifizierung durch einen Automatismus im Hintergrund erfolgen. Ohne Anpassungsbedarf für Applikationen geht dies nur durch einen modifizierten PKCS#11-Provider. Dieser Provider muss an Stelle des zum Token gelieferten eingebunden werden.

#### PKCS#11: automatisierte Nutzerauthentifizierung

Als Basis dient ein PKCS#11-Provider des Token-Herstellers. Dessen Funktionalität ist aber fest vorgegeben und ein Login-Automatismus kann dort nicht eingebaut werden. Für die Applikationen soll aber weiterhin eine Bibliothek mit der Cryptoki-Schnittstelle zur Verfügung stehen und diese muss den Automatismus beinhalten.
Das führt zu einem Systemaufbau, bei dem zwischen dem PKCS#11-Provider des Herstellers und den Applikationen eine Zwischenschicht, analog einem Netzwerk-Proxy eingefügt wird (Abbildung 6.6). Bei einem Netzwerk-Proxy handelt es sich um einen Vermittler zwischen zwei Kommunikationspartnern. Er nimmt Anfragen eines Partners entgegen und übermittelt sie an den anderen Partner, wobei er als Stellvertreter des Absenders agiert. Dabei kann er Modifikationen in der Anfrage oder der zum Absender übertragenen Antwort durchführen.

[6] Der Zentrale Kreditausschuss (ZKA) gibt Vorgaben für Komponenten und Systeme des elektronischen Zahlungsverkehrs. `http://www.hbci-zka.de`

[7] HBCI (*Home Banking Computer Interface*) beschreibt eine Softwareschnittstelle zur Verbindung einer lokalen Applikation mit einem Dienst auf einem Bank-Rechner.

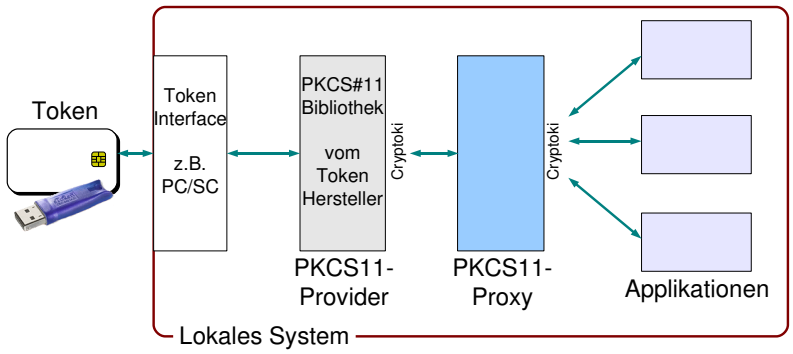
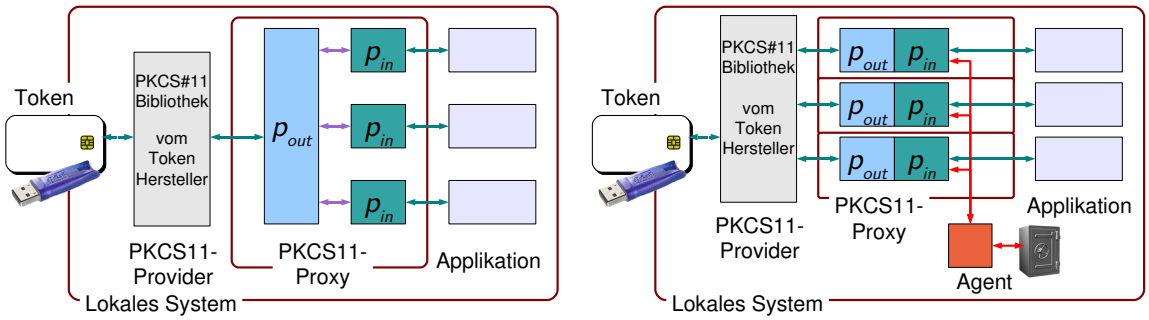



**Abb. 6.6 —** Einbindung von Token in clientseitiges SSO

Ein PKCS#11-Proxy muss eine passende Cryptoki-Applikations-Schnittstelle besitzen und die Anfragen über die Cryptoki-API an die Hersteller-Bibliothek weiterleiten. Dabei sind Modifikationen in den übertragenen Daten, z.B. über einen bestehenden Login-Status, durchzuführen, so dass Applikationen letztendlich keine eigene Authentifizierung durchführen. Vorteil dieser Proxy-Lösung gegenüber einem eigenen Token-Provider besteht in der Nutzung beliebiger Token, insofern deren Hersteller einen PKCS#11-Provider liefern.
Unter Vorgabe der PKCS#11-Spezifikation ist die Implementierung eines PKCS#11-Proxies auf zwei Arten möglich (Abbildung 6.7). Auf Seite der Applikationen besitzen beide die bekannte Cryptoki-Schnittstelle ($p_{in}$).




(a) Zusammenfassung in einer Applikation

(b) Abbildung auf getrennte Applikationen

**Abb. 6.7 —** Varianten eines PKCS#11-Proxy-Systems

In Variante (a) baut der Proxy eine Verbindung zum PKCS#11-Provider über $p_{out}$ auf. Die Anfragen aller Applikationen werden über diese gemeinsame Verbindung weitergeleitet. Dazu werden alle Sessions von $p_{in}$ auf Sessions in $p_{out}$ abgebildet. Für die Hersteller-Provider entspricht es dem Zugriff durch eine Applikation $p_{out}$. Die Nutzerauthentifizierung in der ersten Session schaltet das Token für alle Applikationen frei. Die gewünschte SSO-Funktionalität ist erreicht.
Eine Implementierungen von Variante (a) müsste die Sessions der Applikationen von $p_{out}$ und ihre Zuordnung in $p_{in}$ verwalten. Ein Problem dabei ist, dass viele Provider nur eine begrenzte Session-Anzahl zulassen. Für getrennt agierende Applikationen wird diese Schranke ausreichend hoch sein. Durch die Akkumulation der Sessions aller Applikationen könnte diese aber schnell überschritten werden. Ein anderes, sicherheitsrelevantes Problem besteht darin, dass keine Trennung zwischen den Applikationen und den von ihnen temporär erzeugten PKCS#11-Objekten gegeben ist. Temporäre

Objekte werden im Speicher des Providers (im RAM) und nicht auf dem Token erzeugt. Da mit dem Proxy aber alle Applikationen auf die selbe Instanz der Hersteller-Bibliothek und somit auf den gleichen Speicher zugreifen, sind die temporären Objekte für alle Applikationen sichtbar. Bei Variante (a) wäre deshalb eine gegenseitige Beeinflussung der Applikationen möglich. Ein Login des Security Officers müsste ebenfalls berücksichtigt werden. So darf ein SO-Login durch eine Applikation sich nicht auf andere Applikationen auswirken. Eine technische Umsetzung, ohne den SO-Zugriff prinzipiell zu verhindern, dürfte sich als schwierig erweisen. Variante (a) birgt deshalb Risiken für Entwurfsfehler, die sich zusammen mit global sichtbaren Objekte negativ auf die Systemsicherheit auswirken. Ein Sicherheitsvorteil besteht aber darin, dass die initiale Authentifizierung direkt am Token, z.B. mit einem Klasse-2 Reader erfolgen kann.

Variante (b) verwendet ebenfalls eine Schnittstelle $p_{in}$ für jede Applikation, die aber jeweils eine eigene Verbindung $p_{out}$ zum nachgeordneten Provider aufbaut. Jede Instanz von $p_{out}$ bedient genau eine Applikation. Damit wäre die Trennung temporärer Objekte kein Problem.
Allerdings ermöglicht Variante (b) ohne eine zusätzliche Erweiterung kein SSO, da jede Applikation eine eigene Instanz für den Provider bildet und damit für die Authentifizierung selbst zuständig ist. Deshalb kommt der Agent zum Einsatz. Authentifiziert sich die erste Applikation gegenüber dem Token, so wird die Authentisierungsinformation (die PIN) im Agent hinterlegt. Erfordert in Folge der Provider von einer weiteren Applikation eine Authentifizierung, so wird der Agent kontaktiert. Die Aufgabe des Agenten ist es, im Speicher einen möglichst sicheren Container für die Aufbewahrung der sensiblen PIN zu bilden und diese bei Bedarf an die Proxy-Bibliothek $p_{in,out}$ zu übergeben. Vorher ist zu prüfen, ob es sich um eine berechtigte Applikation handelt. Wie diese Prüfung erfolgen kann, wird im Kapitel 7.1.1 beschrieben.
Zur Implementierung von Variante (b) ist es notwendig, dass Applikationen die `C_Login`-Funktion mit einer leeren PIN-Angabe aufrufen können. Diese Funktion muss von $p_{out}$ derart modifiziert werden, dass die PIN vom Agenten erfragt und bei Bedarf eingetragen wird. Der Nachteil ist, dass die PIN nicht direkt über einen Klasse-2 Reader eingegeben werden kann, da sie bereits im Rechner vorliegen muss.
Bei Vergleich der beiden Varianten lässt sich (b) mit geringeren Eingriffen in die Funktionsweise von Cryptoki umsetzen. Die Trennung einzelner Applikationen kann beibehalten werden. Der Overhead der Session-Verwaltung ist ebenfalls geringer. Wenn außerdem der Agent nur selten kontaktiert werden muss, ist diese Variante insgesamt auch performanter. Variante (b) wäre deshalb als Grundlage eines SSO-Systems auf Token-Basis geeignet, vorausgesetzt die PIN-Abfrage ließe sich auf die erste Applikation beschränken.

### PKCS#11: Unterdrückung der PIN-Abfrage

Aufgrund der PKCS#11-Spezifikation müssen Applikationen von einer notwendigen Authentifizierung vor Zugriff auf sensible Informationen im Token ausgehen. Sie könnten deshalb den Nutzer nach einer PIN fragen, auch wenn es aufgrund des Login-Automatismus gar nicht erforderlich wäre. Somit muss die unnötige PIN-Abfrage der PKCS#11-Applikationen unterdrückt werden.
Um dies zu erreichen, ist ein gezielter Eingriff der Proxy-Bibliothek in die Kommunikation zwischen den Applikationen und dem Provider erforderlich. Hierzu ist die Kommunikation einer Applikation mit dem Token genauer zu untersuchen.
Im Anhang B wird die PKCS#11-Kommunikation am Beispiel eines Email-Abrufs gezeigt. Allgemein läuft die Kommunikation nach folgendem Schema ab, bei dem einzelne Punkte übersprungen werden können, wenn die Funktion nicht benötigt wird:

1. Initialisierung des PKCS#11-Providers [C_Initialize] und Abfrage der unterstützen Funktionen [C_GetFunctionList] (Anhang B: Schritt 0 und 1)
2. Abfrage der Slots [C_GetSlotList] und [C_GetSlotInfo],
   Slot ≡ Token ⇒ verfügbare Token werden erfragt (Anhang B: Schritt 3 bis 5)
3. Abfrage der Token-Informationen [C_GetTokenInfo] und der vom Token unterstützten Methoden [C_GetMechanismList] (Anhang B: Schritt 6 bis 8)
4. Öffnen einer oder mehrerer Sitzungen zum Slot [C_OpenSession]
   (Anhang B: Schritt 9, 17, ...)
5. Durchführung verschiedener Operationen auf öffentlichen Objekten, z.B. Suchen und Lesen von Zertifikaten [C_FindObjectsInit und C_FindObjects]
   (Anhang B: Schritt 10 bis 12, ...)
6. Erfragen von Informationen über die aktuelle Sitzung [C_GetSessionInfo] und, falls erforderlich, Authentifizierung gegenüber dem Token [C_Login] (Anhang B: Schritt 50)
7. Aufruf kryptographischer Funktionen unter Nutzung öffentlicher oder geschützter Objekte
8. Logout [C_Logout] und Schließen der Session [C_CloseSession] oder aller Sessions [C_CloseAllSession] gegenüber dem Token
9. Trennung der Providerbindung [C_Finalize]

Die erste Maßnahme betrifft die Modifikation der Rückmeldung vom Provider, falls eine Applikation durch Aufruf der Management-Funktion C_GetTokenInfo Informationen vom Token erfragt. Als Antwort liefert Cryptoki eine Token-Info-Struktur. Diese enthält Informationen über den Typ des Tokens (z.B. manufacturerID und serialNumber) und kodiert dessen Fähigkeiten in einer Bitmaske flags. Eines der Bits (CKF_PROTECTED_AUTHENTICATION_PATH) signalisiert, dass die Authentifizierung außerhalb von PKCS#11, z.B. direkt am Token erfolgt. Setzt der Proxy dieses Bit in der Rückantwort an die Applikation, wird daraufhin keine PIN erfragt. Dieses Flag ist von der Proxy-Bibliothek zu setzen, wenn für das genutzte Token eine PIN im Agent hinterlegt wurde, also bereits eine erfolgreiche Authentifizierung stattfand (siehe Anhang B: Schritt 6).

Eine weitere Modifikation betrifft die Rückantwort auf Abfrage von Informationen über die aktuelle Sitzung mittels C_GetSessionInfo. Die in der Antwort gelieferten Flags über den Status der Sitzung können derart modifiziert werden (z.B. durch Setzen von CKS_RO_USER_FUNCTIONS), dass die anfragende Applikation in der Session davon ausgeht, dass bereits der Zustand einer erfolgreichen Nutzeranmeldung besteht (Anhang B: Schritt 50). Infolgedessen wird die Applikation kein Login durchführen. Liegt intern noch eine *public session* vor, so ist bei Bedarf ein automatisches Login durchzuführen. Dies führt zur letzten Modifikation.

Über die Objekt-Management-Funktion C_FindObjectsInit mit anschließendem Aufruf von C_FindObjects erhält eine Applikation Informationen über die Token-Objekte. Wenn kein Nutzer angemeldet ist, werden keine Informationen über geschützte Objekte geliefert. Benötigt die Applikation Informationen über geschützte Objekte, so setzt sie beim Aufruf von C_FindObjectsInit das Flag CKO_PRIVATE_KEY oder CKO_SECRET_KEY. Findet Sie dabei keine privaten Objekte, wird sie entweder ein Login durchführen, also die PIN erfragen, oder davon ausgehen, dass keine privaten Objekte vorhanden sind und das Token letztendlich nicht verwenden. Um dies zu verhindern und trotzdem die PIN-Abfrage zu unterdrücken, muss vor der Suche nach geschützten Objekten

eine Anmeldung am Token stattfinden. Es muss also automatisch ein Login gegenüber dem Token durchgeführt werden, bevor vom Proxy der Aufruf von `C_FindObjectsInit` an den Provider weitergeleitet wird (Anhang B: Schritt 126).

Aufgrund dieser Maßnahmen führt keine der getesteten Applikationen eine wiederholte PIN-Abfrage durch. Das eigentliche Login muss nun noch im Hintergrund mit Hilfe des Agenten erfolgen.

### PKCS#11: Automatisches Login mit hinterlegter PIN

Der Login-Automatismus erfordert die Modifikation der `C_Login`-Funktion. Aufgrund der signalisierten externen Authentifizierung wird `C_Login` von den Applikation ohne Angabe einer PIN aufgerufen. Der Proxy kann sie aber mit Hilfe des Agenten automatisch ergänzen.
Erfolgte kein Login vor Aufruf der Funktion `C_FindObjectsInit` zur Suche nach privaten Objekten, so muss die modifizierte Login-Funktion automatisch aktiviert werden. Deshalb muss der Proxy den Zustand eines erfolgreichen Logins erkennen. Dazu muss er den Rückgabewert von `C_Login` auswerten. Bei Erfolg ist die Applikation im Zustand einer User-Session. Für folgende Funktionsaufrufe und deren Datenaustausch mit dem Provider müssen keine weiteren Modifikationen erfolgen. Allerdings ist zu beachten, dass laut Spezifikation mit dem Schließen der letzten Session der Login-Status verloren geht. Da der Proxy diesen Fall erkennen muss, sind intern die Anzahl der Sitzungen zu verwalten. Dazu reicht ein Zähler aus, der in Abhängigkeit von `C_Open`-, `C_Close`- und `C_CloseAllSession`-Aufrufen modifiziert wird (Anhang B: Schritt 9, 17, 162).
Abschließend bliebe noch zu klären, wie der Agent initial die benötigten Informationen, also die PIN erhält. Denkbar wäre, dass er diese von der ersten PKCS#11-Applikation, für die der Automatismus noch nicht aktiv war, übernimmt. Alternativ kann eine Befehlsschnittstelle integriert werden. Ein Nutzerkommando kann dann die PIN des eingelegten Token an den Agenten übergeben. Die bequemste SSO-Einbettung ist aber die Verbindung des Rechner-Logins mit der Hinterlegung der PIN im Agenten, so wie in Abbildung 6.8 dargestellt.

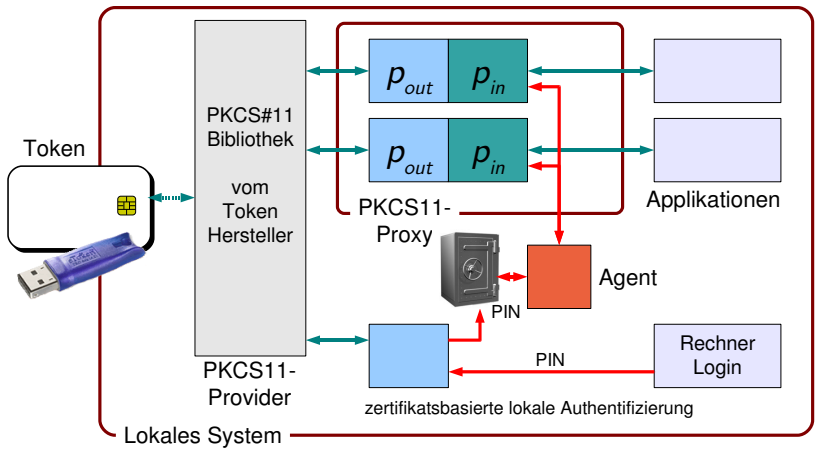



**Abb. 6.8 —** PIN-Hinterlegung bei Rechner-Login

Dazu wäre der Login-Mechanismus, über den sich der Nutzer am Rechner anmeldet, auf zertifikatsbasierte Authentifizierung umzustellen und als Token-Schnittstelle PKCS#11 zu verwenden. Gleichzeitig muss mit dem Login der Agent als Rechner-Dienst im Hintergrund gestartet und eine Information für den Nutzer hinterlegt werden. Diese Information dient den Applikationen zur Kontaktaufnahme mit der Proxy-Bibliothek.

# Kapitel 7

# Realisierung

Für clientseitiges SSO mittels Crypto-Token ist eine Proxy-Bibliothek nach Variante (b) aus Kapitel 6.2.2 erforderlich. Diese Bibliothek ist in Verbindung mit dem ebenfalls erforderlichen PKCS11-Agenten für eine automatische Authentifizierung des Nutzers gegenüber dem Token verantwortlich. Der Agent dient zur Verwahrung der Authentisierungsinformation. Für die meisten Token wird eine PIN genutzt, biometrische oder andere Varianten können aber ebenfalls integriert werden.
Der erste Teil dieses Kapitels beschreibt die entwickelte clientseitige Testsoftware, bestehend aus PKCS#11-Provider und Agent. Dazu gehört die Sicherung der Kommunikation zwischen Provider und Agent, sowie der Schutz der hinterlegten PIN im Hauptspeicher vor unbefugtem Auslesen oder Verwendung durch unautorisierte Applikationen.
Provider und Agent sind nur ein Teil des SSO-Systems. Für zertifikatsbasierte Authentifizierung werden clientseitig arbeitende Anwendungen mit PKCS#11-Schnittstelle benötigt. Diese Schnittstelle muss von den Applikationen für die Authentifizierung gegenüber Diensten genutzt werden. Dabei sollte ein möglichst großer Bereich alltäglicher Anwendungen überdeckt werden. Für ausgesuchte Applikationen und Client-Server-Protokolle wurde deshalb geprüft, ob sie sich für zertifikatsbasierte Authentifizierung einsetzen lassen. Die Ergebnisse werden im zweiten Teil vorgestellt.

## 7.1 Implementierung als Proof-of-Concept

Für einen Praxistest des Ansatzes zum clientseitigen SSO ist die Software für Proxy und Agent erforderlich. Der Proxy wird als Ersatz vor einen bestehende PKCS#11-Provider geschaltet und ist somit als dynamische Bibliothek zu implementieren. Der Agent muss eine Schnittstelle für die Proxy-Bibliothek bieten und Maßnahmen zur Sicherung der PIN ergreifen.

### 7.1.1 SSO-Client-Software

Die Client-Software besteht insgesamt aus drei Teilen, dem PKCS#11-Provider in Form der Proxy-Bibliothek, dem Agenten zur PIN-Verwahrung und Software zur Steuerung des Agenten durch den Nutzer.
Die Proxy-Bibliothek benötigt für ihre volle Funktionalität die Verbindung zum Agenten. Es handelt sich um eine dynamische Bibliothek und kein eigenständiges Programm. Da eine Bibliothek von der Applikation zur Laufzeit eingebunden wird, arbeitet sie mit den Rechten der Applikation und in deren Speicherbereich. Eine gesicherte Kommunikation zwischen Bibliothek und Agent muss unter dieser Bedingung möglich sein. Hierfür sind mehrere Ansätze denkbar:

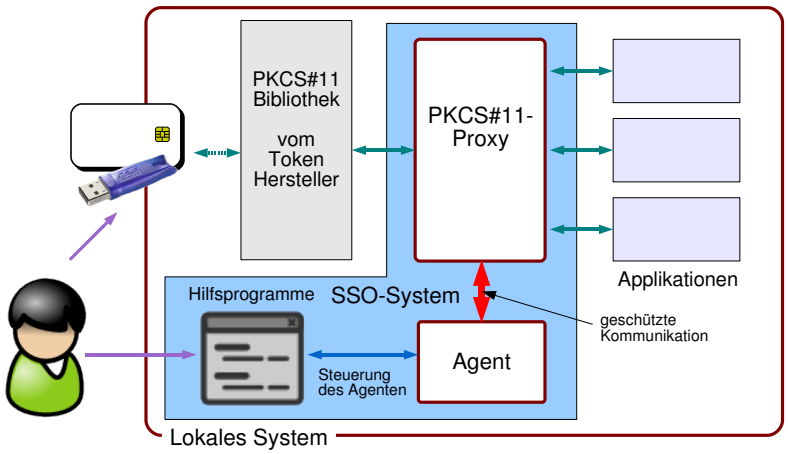



**Abb. 7.1 —** Aufbau des Testsystems

- Die Kommunikation könnte über einen **gemeinsamen Speicherbereich** (*shared memory*) erfolgen. Da verschiedene Applikationen auf diesen Speicherbereich zugreifen, ist entsprechender Synchronisationsbedarf und Maßnahmen zur Absicherung gegen fehlerhafte Programme zu treffen. Hier hängt es von den Möglichkeiten des zugrunde liegenden Betriebssystems ab. Da es hier größere Unterschiede bei verschiedenen Betriebssystemen gibt und die Kommunikation demzufolge jeweils anzupassen wäre, wurde diese Lösung nicht weiter in Betracht gezogen.

- Dahingegen wäre die Nutzung eines **Netzwerkprotokolls** unabhängig vom Betriebssystem, wenn es auf Grundlage eines TCP- oder UPD-Transportprotokolls[1] läuft. TCP/UDP-Protokolle werden von allen gängigen Betriebssystemen unterstützt. Ein systemübergreifendes Agent-Protokoll könnte darauf aufbauen. Allerdings wäre es für den Agenten unmöglich, die aufrufenden Programme zu identifizieren und erforderliche Sicherheitsprüfungen durchzuführen.

- Im Gegensatz dazu würde ein **Socket im Filesystem** als Kommunikationsschnittpunkt innerhalb der Maschine es einem Agenten ermöglichen, die sich verbindende Applikation zu identifizieren. Dem Agenten stehen neben dem Namen der Applikation weitere Informationen zur Verfügung, mit der z.B. zwischen systemweit installierter oder nutzereigener Software anhand des Verzeichnisses unterschieden werden kann. Noch wichtiger ist, dass sich der Nutzer der kontaktierenden Applikation ermitteln lässt. Dabei kann vor Erteilung der PIN-Freigabe geprüft werden, ob es sich um eine Applikation handelt, die zur aktuellen Nutzersitzung gehört.

Gängige Betriebssysteme bieten Socket-Kommunikation und nutzen dazu vergleichbare Funktionen, so dass eine Anpassung für weitere Systeme leicht möglich sein sollte. Lediglich die Implementierung der Prüfmethoden des Agenten sind systemabhängig und bedürfen einer speziellen Bearbeitung. Das Abhören übertragener Informationen bei der Kommunikation über Filesystem-Sockets durch fremde Applikationen ist, falls überhaupt möglich, nur administrativen Nutzern bzw. dem Betriebssystem vorbehalten. Ein Angriff wäre für einen normalen Systemnutzer nicht durchführbar. Ein Schutz vor administrativen Nutzern auf dem eigenen Rechner ist aber prinzipiell nicht möglich. Die Socket-Variante ist somit ein guter Kompromiss zwischen Sicherheit und Implementierungsaufwand im Hinblick auf systemübergreifenden Einsatz.

[1] TCP und UDP sind Transportprotokolle in Schicht 4 des ISO/OSI-Referenzmodells. Sie ermöglichen die Kommunikation zwischen Applikationen und werden auch als Ende-zu-Ende-Protokolle bezeichnet [RFC 793, RFC 768].

### PKCS#11-Agent-Protokoll

Auf Basis der Socket-Kommunikation wurde ein Protokoll entworfen, mittels dem die PKCS#11-Proxy-Bibliothek Informationen an den Agenten übertragen und vom Agenten erfragen kann. Dazu wurde der PKCS#11-Agent analog dem häufig eingesetzten OpenSSH-Agent implementiert. Der OpenSSH-Agent dient der Hinterlegung von privaten Schlüsseln und kann im Rahmen einer Public-Key-Authentifizierung von der OpenSSH-Client-Software bei der Verbindung zum Server kontaktiert werden (Kapitel 7.2.1, Abschnitt Remote Shell).

Das PKCS#11-Proxy-Protokoll umfasst folgende Steuerungsbefehle:

| | |
|---|---|
| ADD_IDENTITY | : Eintragen einer neuen PIN |
| REQUEST_IDENTITIES | : Erfragen der PIN zu dem als Parameter angegebenen Token |
| REMOVE_(ALL)_IDENTITY | : Löschen einer oder aller hinterlegten Informationen |
| QUIT | : Aufforderung zum Beenden des Agenten |
| (UN)LOCK | : temporäres De- und Reaktivieren des Agenten |

Die Kommunikation zwischen PKCS#11-Proxy-Bibliothek und Agent erfolgt über einen Socket, den der Agent beim Starten automatisch erzeugt. Dieser muss der Bibliothek vor Nutzung bekannt sein. Der Applikation und damit der Bibliothek wird deshalb beim Start eine Umgebungsvariable (`PKCS11AGENT`) übergeben, die von der Bibliothek bei der Initialisierung ausgewertet wird. Weiterhin muss der Proxy-Bibliothek mitgeteilt werden, welcher Hersteller-PKCS#11-Provider für das Token zu nutzen ist. Dazu wird ebenfalls eine Umgebungsvariable (`PKCS11PROV`) genutzt.

### Sicherungsmaßnahmen

Vor Freigabe der PIN sind durch den Agenten Maßnahmen zu treffen, dass die PIN nicht an unberechtigte Applikationen übergeben wird. Hierzu prüft der Agent beim Aufbau der Verbindung zum Socket, um welche Applikation es sich handelt und ob diese dem aktuellen Nutzer und dessen Sitzung zuzuordnen ist. Dazu kann eine Liste erlaubter Applikationen in der Konfiguration angegeben werden. Bei einer negativen Prüfung wird die Verbindung unterbrochen. Vor einem Login gegenüber dem Token und damit vor dessen Nutzung fordert Cryptoki den Aufruf der `C_Initialize`-Funktion. Zu diesem Zeitpunkt wird die Verbindung zwischen der Proxy-Bibliothek und dem Agenten aufgebaut und es erfolgt die Prüfung der Autorisierung. Da `C_Initialize` nur einmal pro Applikationsstart erforderlich ist, wird die relativ aufwendige Prüfung nur selten auftreten und sollte den Betrieb nicht merklich verzögern.

Die Speicherung der PIN ist auf die Laufzeit des Agenten beschränkt. Mit dem Remove-Kommando kann eine PIN vorzeitig aus dem Speicher entfernt werden. Beim Eintragen der PIN ist außerdem eine Information über das zugehörige Token zu hinterlegen, um Fehlzuordnungen und somit eine ungewollte Sperrung des Tokens zu vermeiden.

Die Eintragung der PIN kann durch eine Software erfolgen, welche die benannte Socket-Schnittstelle implementiert. Dazu wurde ein Hilfsprogramm entwickelt, welches verfügbare Token prüft, vom Nutzer die PIN erfragt und diese zusammen mit der Token-Information dem Agenten übergibt. Außerdem wurde für die lokale Rechner-Anmeldung ein PAM-Mechanismus (siehe Kapitel 7.2.2) erweitert, der eine Anmeldung mit dem Token erlaubt, automatisch den Agent für den jeweiligen Nutzer startet und die während des Logins eingegebende PIN in diesem Agenten hinterlegt.

Zum Schutz der PIN gegenüber dem Auslesen im Speicher wurden Maßnahmen zur verdeckten Ablage getroffen, die das leichte Auffinden erschweren. Dabei ist von einem Angriff durch Software anderer Nutzer auszugehen. Software mit administrativen Rechten könnte auf einfachere Methoden zum Ausspähen der PIN zurückgreifen. Der Schutz des Betriebssystems vor Zugriff auf Speicher anderer Applikationen stellt deshalb die erste Schutzmaßnahme dar.

Prinzipiell besteht auch die Gefahr, dass nach dem Ausschalten des Rechners der RAM-Inhalt erhalten bleibt und anschließend analysiert werden kann. Dafür ist ein sofort im Anschluss erfolgender physikalischer Zugriff auf den Rechner erforderlich [HSH+08]. Somit ist es wichtig, bei Verlassen und Ausschalten des Rechners mit dem Entfernen des Tokens auch die PIN im Speicher durch Überschreiben zu löschen. Dafür wurde ebenfalls ein Automatismus entwickelt.

Weitere angedachte Schutzmaßnahmen betreffen das Erzeugen eines zufälligen symmetrischen Schlüssels beim Starten des Agenten und Verschlüsselung der PIN. Die verschlüsselte PIN wird in einem getrennten Speicherbereich abgelegt. Hierzu wird ein Speicherbereich genutzt, bei dem keine Auslagerung auf Festplatte durch das Speichermanagement des Betriebssystems erfolgt. Bei einer Analyse des Speichers müssen somit zur Laufzeit beide Positionen, die von Schlüssel und PIN erkannt werden. Bei einem potentiell möglichen Angriff sollte es die Suche nach der PIN erschweren.

### 7.1.2 Aufbau eines Testsystems

Das Testsystem wurde unter Linux in Verbindung mit dem frei verfügbaren Provider *OpenSC* und unter Nutzung von *PC/SC-lite* entwickelt und getestet. Nach erfolgreichem Probebetrieb wurden weitere Tests mit den kommerziellen Provider-Bibliotheken *Aladdin eToken* und *Siemens HiPath* durchgeführt. Dabei konnten keine Probleme gegenüber dem Entwicklungssystem diagnostiziert werden, was auf Kompatibilität zur PKCS#11-Spezifikation hindeutet. Für die Nutzung mit anderen Providern werden deshalb keine Probleme erwartet, insofern sie sich ebenfalls an die Spezifikation halten. Die genutzten Rechner und Token sind im Anhang C aufgelistet.

Eine Adaption für weitere Betriebssysteme und Token sollte leicht möglich sein. Die Proxy-Bibliothek erfordert keine betriebssystemspezifischen Funktionen. Lediglich die Sicherheitsprüfung im Agenten benötigt Informationen vom Betriebssystem, die einen tieferen Einblick in dessen Arbeitsweise voraussetzen. Für den Einsatz des SSO-Sytems wäre außerdem das Vorhandensein eines Hersteller-PKCS#11-Providers für das verwendete Token unter dem jeweiligen Betriebssystem erforderlich. Für gängige Token liefern die Hersteller Provider-Bibliotheken für Microsoft Windows und Linux. Der OpenSC-Provider erlaubt darüber hinaus die Nutzung vieler Token in Unix-Systemen.

## 7.2 PKCS#11-Applikationen und -Authentifizierung

### 7.2.1 Client-Server-Applikationen

Für eine Umstellung auf Token-gestützte Authentifizierung werden Server-Applikationen mit zertifikatsbasierte Authentifizierung auf Basis von SSL/TLS oder vergleichbarer Protokolle benötigt. Hinzu kommen die Client-Applikationen, die diese Protokolle unterstützen.

Untersucht wurde anhand ausgewählter Applikationen, wie sich zertifikatsbasierte Authentifizierung mit clientseitigem SSO umsetzen lässt. Diese Applikationen decken einen Großteil möglicher Anwendungsfälle ab. Sie stehen exemplarisch für alle Applikationen, so dass sich eine komplette Infrastruktur daraus ableiten lässt.

### Browser und Webserver

Zu den am häufigsten eingesetzten Applikationen zur Übertragung von Informationen über das Netz zählt der Webbrowser (kurz Browser), welcher Informationen von einem Dienst, dem Webserver abrufen kann. Die ursprüngliche Aufgabe eines Browser bestand in der Anzeige von Informationen aus dem „World Wide Web“, die mittels der Hypertext Markup Language (HTML) in strukturierter Form vorlagen und Text mit Grafiken sowie elektronische Verweise (Hyperlinks) auf weitere Seiten umfassten. Inzwischen sind hier vielfältige Möglichkeiten zum Layout und zur Einbindung weiterer Objekte hinzugekommen. Darüber hinaus können Browser durch Plugins mit zusätzlichen Funktionen erweitert werden, so dass sie sich als universelles Werkzeug für die Nutzung in Rechnernetzen etabliert haben.
Grundlage für die Kommunikation der Browser mit den Webservern bildet das **HTTP**-Anwendungs-Protokoll [RFC2626]. Dieses Protokoll ist nicht auf einfache Kommunikation in Form der Anfrage einzelner „Web-Seiten“ und deren Auslieferung vom Server beschränkt. Vielmehr bietet es die Möglichkeit, neben gezielten Inhalts-Anfragen in Form von URLs auch eigene Daten zum Server zu übertragen. Somit liefert es eine passende Grundlage für die bidirektionale Übertragung verschiedener Inhalte. Da mit den Browsern ein passender Client auf fast jedem Computer verfügbar ist, wird HTTP gern für Portale und andere interaktive Netzdienste genutzt.

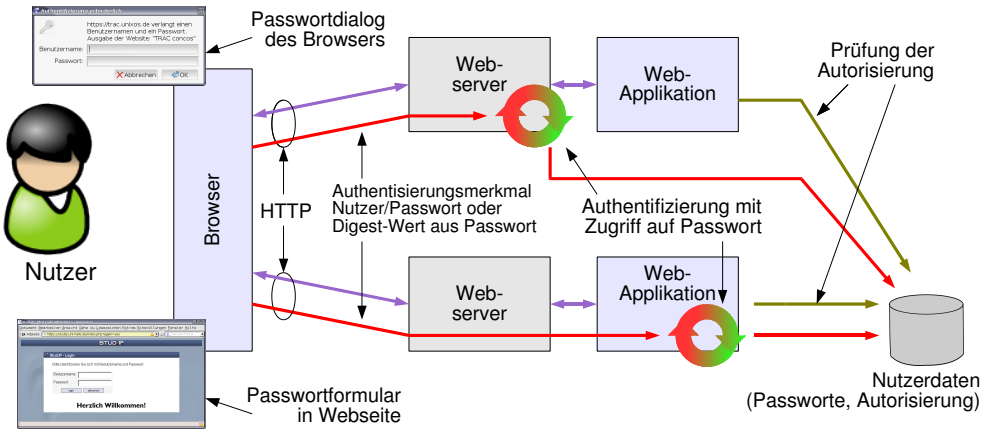



**Abb. 7.2 —** Authentifizierung über das HTTP-Protokoll

Um nutzerbeschränkte und personalisierte Dienste vor unberechtigtem Zugriff zu schützen und die Verbindung einem Nutzer zuordnen zu können, wird eine Authentifizierung vom Browser verlangt. Hierfür bietet HTTP die *Basic* und *Digest Access Authentication* [RFC2617] zur Authentifizierung auf Basis eines gemeinsamen Geheimnisses (Passwort). Dazu öffnen die Browser ein Dialog-Fenster zur Eingabe eines Nutzernamens und des Passwortes. Über speziell gestaltete Web-Seiten können in vorgefertigten Formularfeldern Passwörter erfragt werden. Die Verifizierung der Nutzer-Passwort-Kombination erfolgt durch die Webserver-Software oder durch die verbundene Web-Applikation (Abbildung 7.2). Eine Portal-Seite als Applikation auf dem Webserver kann die Authentizität gegen ihre eigene Nutzerdatenbank prüfen oder dem Web-Serverdienst die Prüfung überlassen und anschließend den übermittelten Nutzernamen verwenden, um die Autorisierung zu prüfen.
Aktuelle Browser bieten die Möglichkeit, Passwortfelder und -dialoge automatisch mittels der Daten eines integrierten Passworttresors zu befüllen. Auch die Einbindung in Kerberos-Netze ist möglich. Hierfür kann das *HTTP Negotiate Authentication Schema* [RFC4559] verwendet werden.

Die Verwendung zertifikatsbasierter Client-Authentifizierung gehört nicht zum Umfang des HTTP-Protokolls und muss entweder durch Protokollerweiterung oder durch eine clientseitig im Browser ablaufende Applikation - z.B. ein Java-Applet - oder durch Java-Script-Funktionalität ergänzt werden. Aufgrund der Besonderheiten der Browser, z.B. unterschiedlicher Java-Versionen, ist dies meist nicht einfach und betriebssystemübergreifend umzusetzbar und findet praktisch kaum Anwendung.
Um zertifikatsbasierte Client-Authentifizierung trotzdem zu ermöglichen, bietet sich die Einbettung von HTTP in SSL/TLS-Tunnel an [RFC2818]. Dazu wird neben dem Port des Webservers (Port 80) ein separater Port (443) für den SSL/TLS-Tunnel zur HTTP-Übertragung verwendet. Dieser Tunnel wird als HTTPS-Verbindung bezeichnet. Es existiert eine weitere Variante [RFC2817], mit der über ein spezielles Kommando ein „Upgrade“ der unverschlüsselten HTTP-Verbindung über Port 80 auf eine SSL/TLS-geschützte Verbindung auf dem selben Port erfolgt. Diese Variante wird aber selten verwendet.
Die verschlüsselten SSL/TLS-Tunnel bietet sich natürlich dort an, wo sensible Informationen übertragen werden. Deshalb wird HTTPS von den meisten Browsern unterstützt. Viele Web-Server und -Applikationen fordern eine HTTPS-Verbindung, wenn Passwörter zu übertragen sind. Wie im Kapitel 4.3.2 beschrieben, erlaubt SSL/TLS neben der obligatorischen Server-Authentifizierung auch die Client-Authentifizierung. Dazu muss im Browser ein passendes Zertifikat und der zugehörige private Schlüssel vorliegen. Um Client-Authentifizierung mit Crypto-Token umzusetzen, wird eine passende Token-Schnittstelle im Browser, also die Einbindung eines PKCS#11-Providers benötigt.
Für diese Arbeit wurden verschiedene Linux-Browser untersucht. Die Browser, welche der Mozilla-Browser-Serie entstammen, boten die gewünschten Funktionen. Durch Einbindung der PKCS#11-Proxy-Bibliothek anstatt des Hersteller-Providers an der Schnittstelle zum Browser (Abbildung 7.3) kann eine zertifikatsbasierte Authentifizierung ohne Eingabe der PIN erfolgen. Mittels des Browsers *Firefox 3.0* konnte dieses Verhalten erfolgreich verifiziert werden. Damit steht eine passende clientseitige Applikation mit großem Funktionsumfang zur Verfügung.

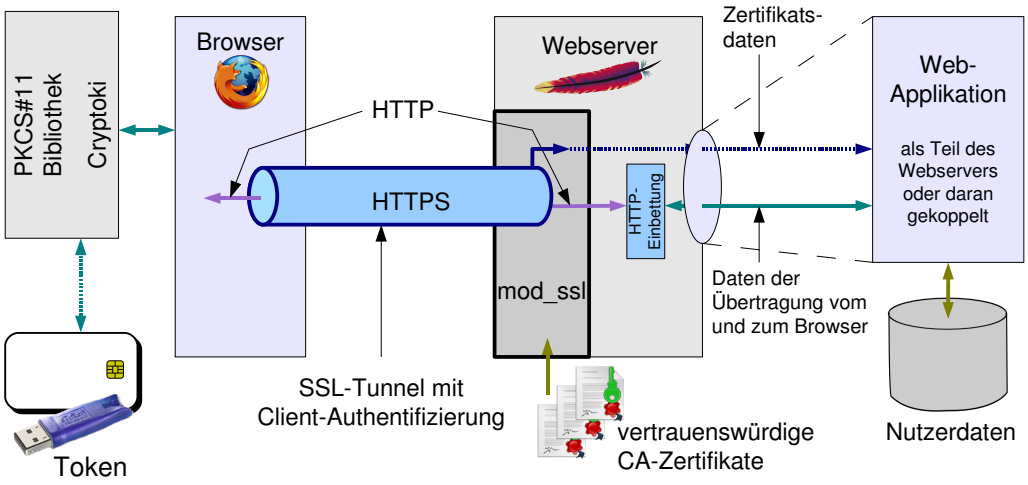



**Abb. 7.3 —** Apache mit SSL-Unterstützung und zertifikatsbasierte Client-Authentifizierung

Die SSL/TLS-Authentifizierung des Browser muss serverseitig unterstützt werden. Die hinter dem Server liegende Applikation kann diese Aufgabe nicht übernehmen, da die SSL/TLS-Funktionalität integraler Bestandteil des Webservers ist und die Authentifizierung im Rahmen des Verbindungsaufbaus erfolgt. Somit ist es die Aufgabe des Webservers. Erfolgreich getestet wurde es mit dem

*Apache-HTTP-Server*[2] in Version 2.2 und dessen SSL/TLS-Unterstützung über das *mod_ssl* [3] Modul. Durch eine Option in der Server-Konfiguration fordert der Apache-Webserver während des HTTPS-Verbindungsaufbaus zur zertifikatsbasierten Client-Authentifizierung auf. Der Client überträgt anschließend ein Signatur-Zertifikat, vorausgesetzt er besitzt ein derartiges. Man unterscheidet zwischen einer optionalen und einer erforderlichen Authentifizierung. Hat der Client keinen Zugriff auf passende Zertifikate, so sendet er eine leere Liste. Bei der erforderlichen Authentifizierung bricht der Server die weitere Kommunikation ab. Bei der optionalen wird die Verbindung fortgeführt, der Client gilt aber aus Sicht des Webservers als nicht authentifiziert. Diesen Zustand kann die angeschlossene Applikation erfragen und über weitere Schritte entscheiden.
Liegt hingegen ein Zertifikat vor, erfolgt die Prüfung durch den Webserver gegen die hinterlegten CA-Zertifikate. Ist der anschließende *CertificateVerify*-Schritt während des SSL/TLS-Handshakes ebenfalls erfolgreich, so gilt der Client als authentifiziert. Andernfalls entscheidet wieder die Konfiguration des Webservers. Bei optionaler Authentifizierung kann der Webserver den Browser auf eine Passwortgeschützte Seite umleiten. Dadurch kann eine Fallback-Strategie für Nutzer ohne oder mit ungültigen Zertifikaten implementiert werden (siehe Anhang D.1.1).
Bei erfolgreicher Authentifizierung stehen die im Zertifikat enthaltenen Informationen der Applikation zur Verfügung. Sie werden von *mod_ssl* in Umgebungsvariablen abgelegt. Falls nicht automatisch allen Inhabern von gültigen Zertifikaten der Zugang gewährt werden soll, muss im nächsten Schritt die Autorisierung geprüft werden. Dazu könnte die Applikation aus dem Zertifikats-*Subject* den Nutzer ermitteln und Autorisierungsinformationen aus einer zentralen Datenbank, z.B. über eine Schnittstelle vom Identity-Management-System der Hochschule erfragen.

### Emailversand und -empfang

Software für Versand und Abruf von Nachrichten in Form von Emails bezeichnet man als *Mail-* oder *Message User Agent* (MUA). Ein Dienst, welcher zu versendende Emails vom MUA in Empfang nimmt, nennt man *Mail Submission Agent* (MSA). Der MSA gibt die Emails zwecks Weiterleitung an einen *Mail Transfer Agent* (MTA). Jede Email wird ggf. über mehrere MTAs in einer Kette zu dem für den Empfänger zuständigen MTA weiterleitet. Am Ende der Kette steht der *Mail Delivery Agent* (MDA), der die Nachricht in einem dafür vorgesehenen Speicher, dem Postfach ablegt. Meist sind MSA, MTA und MDA innerhalb einer Software integriert und es hängt von der Konfiguration dieser Software ab, welche Dienste angeboten werden. Bekannte Vertreter sind *sendmail*, *postfix* und *exim*. Zu den MUAs gehören u.a. *Outlook*, *Thunderbird*, *Pegasus Mail* oder *Sylpheed*. MUAs werden allgemein als (E-)Mailprogramme bezeichnet. Abbildung 7.4 illustriert den Ablauf der Emailübertragung und des Abrufs.
Für Email-Versand und -Abruf sind verschiedene Anwendungsprotokolle zuständig. Für den Versand und die Weiterleitung der Email wird das **SMTP**-Protokoll[4] genutzt. Der Abruf empfangener Emails hängt vom System des Empfängers ab, wie im Kapitel 3.2 bereits gezeigt wurde. MDAs können ein Verzeichnis des betreffenden Nutzers als Postfach verwenden, so dass Zugriff für den Empfänger-MUA über das Filesystem besteht. Sollen empfangene Emails über das Netz abrufbar sein, so werden die Protokolle **POP3**[5] und **IMAP**[6] genutzt.
Die Notwendigkeit der Authentifizierung bei Versand von Emails hängt von der Forderung des MSA

[2] `http://httpd.apache.org/`
[3] `http://httpd.apache.org/docs/2.2/mod/mod_ssl.html`
[4] SMTP: *Simple Mail Transfer Protocol* [RFC 821, RFC2821, RFC5321]
[5] POP3: *Post Office Protocol*, Version 3 [RFC1939]
[6] IMAPv4 *Internet Message Access Protocol*, Version 4 [RFC3501]

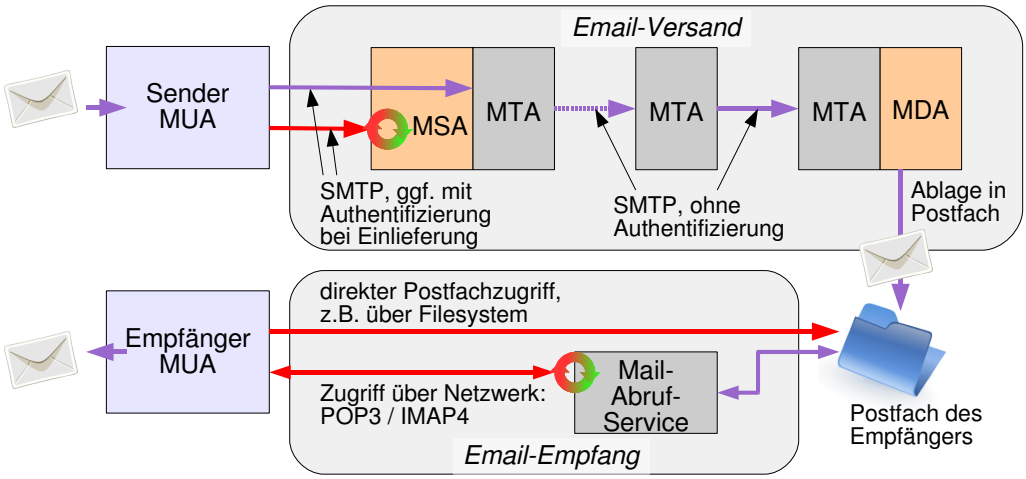



**Abb. 7.4 —** Email-Infrastruktur

ab. Prinzipiell kann natürlich ein MUA auch den MSA überspringen und mittels des SMTP-Protokolls direkt die Email an den nächsten MTA übermitteln. Viele MTAs nehmen jedoch Emails nicht entgegen, wenn sie aus dem Bereich der dynamisch von Internet Service Providern vergebenen Internet-Adressen eintreffen. So soll das Einbringen unerwünschter Nachrichten in das Mailsystem verhindert werden. Aus diesem Grund ist es für Nutzer am heimischen PCs oder mit Notebooks sinnvoll, einen MSA mit angebundenem MTA zu nutzen, der seinerseits die Email ausgehend von einer statischen Adresse weiterleitet. MSAs sind meist für ein Teilnetz zuständig und nehmen daraus Emails ohne Authentifizierung entgegen. Befindet sich der MUA des Senders jedoch nicht im MSA-Teilnetz und zielt die Empfängeradresse auch nicht auf die Domain des MSA, ist eine Authentifizierung erforderlich. Derart wird ein *Open-Relay* vermieden und das Einschleusen unerwünschter Emails über den MSA wird verhindert.

Ursprünglich war Authentifizierung in SMTP nicht vorgesehen, wurde aber aufgrund wachsender Probleme mit SPAM- bzw. Junk-Mails[7] erforderlich. Das SMTP-Protokoll nach dem RFC 821 Standard wurde mittels *Extended SMTP* (ESMTP) erweitert. ESMTP erlaubt zusätzliche SMTP-Kommandos, die je nach Fähigkeit des SMTP-Dienstes angeboten werden können [RFC1869]. Später gingen viele Erweiterungen in die Basis des SMTP-Standards ein [RFC2821, RFC5321]. Eine Erweiterung betrifft die Authentifizierung, für die zusätzlich das `SMTP-AUTH` Kommando eingeführt wurde [RFC2554, RFC4954]. `SMTP-AUTH` ermöglicht den Einsatz von SASL-Mechanismen[8]. SASL ist eine separate Authentisierungs-Schicht, die verschiedene Authentifizierungsverfahren vereint. Dazu gehören neben der Übertragung von Passwörtern (`PLAIN` [RFC4616]) auch verschiedene Challenge-Response-Verfahren (`CRAM-MD5` [RFC2195], `DIGEST-MD5` [RFC2831]), ebenfalls auf Basis von Passwörtern, die aber in gehashter Form übertragen werden. Weiterhin gehört die Kerberos-Anbindung und Verfahren auf Basis asymmetrischer Kryptographie [RFC3163] zu den angebotenen Mechanismen. Für Einsatz von Zertifikaten zur Authentifizierung wären asymmetrische Algorithmen geeignet, wie das in RFC 3163 enthaltene `9798-U-RSA-SHA1-ENC`. Aber nur wenige Emailprogramme unterstützen diese Mechanismen und es mangelt an der Einbindung von Hardware-Token.

Ein Ausweg besteht wieder in der Verwendung von SMTP über SSL/TLS mit zertifikatsbasierter Client-Authentifizierung. SMTP sieht die Verwendung von TLS vor. Die Aktivierung erfolgt über

[7] Bezeichnung für unerwünschte elektronische Nachrichten, meist in Form von Werbemails.
[8] SASL: ***S**imple **A**uthentication and **S**ecurity **L**ayer* [RFC2222, RFC4422, RFC4752]

das ESMTP-Kommando `STARTTLS` innerhalb einer SMTP-Verbindung [RFC2487, RFC3207]. Aber auch die SMTP-Tunnelung über einen separaten, verschlüsselten Kanal ist möglich. Dies wird als SMTP über SSL (SMTPS) bezeichnet und läuft auf einem separaten Port[9]. Ein MSA zur Email-Annahme kann einen weiteren Port nutzen[10] und dort ebenfalls das `STARTTLS`-Kommando anbieten. Dies wird jedoch selten genutzt, da die meisten MUAs ihre Emails über SMTP oder SMTPS einliefern. Einige MUAs unterstützen für SSL/TLS auch die Verwendung von Client-Zertifikaten in Verbindung mit Hardware-Token. Erfolgreich getestet werden konnte es mittels des Emailprogramms *Thunderbird*[11], welches für verschiedene Systeme zur Verfügung steht.
Auf Serverseite wird ein MSA bzw. ein MTA benötigt, der die zertifikatsbasierte Authentifizierung bei Aufbau einer Verbindung unterstützt. Hierfür bot sich *Postfix*[12] an, ein unter der freien *Mozilla Public License*[13] verfügbares Mailsystem, welches MSA, MTA und MDA-Funktion vereint. Die Konfiguration eines bestehenden Postfix-Mail-Servers kann derart ergänzt werden, dass im ersten Schritt von jedem MUA außerhalb der Domain der Aufbau eines verschlüsselten SSL- oder TLS-Kanals in Verbindung mit einem optionalen Client-Zertifikat gefordert wird (Anhang D.1.3). Eine Fallback-Strategie lässt sich integrieren, die bei nicht vorhandenem oder ungültigem Client-Zertifikat auf einen anderen Authentifizierungsmechanismus umschaltet, z.B. `SMTP-AUTH`. Ein Autorisierungsschritt nach Authentifizierung ist in Postfix ebenfalls möglich.

Für Protokolle zum Email-Abruf wurde im Gegensatz zum SMTP-Protokoll bereits beim Entwurf die Authentifizierung eingeplant. Die ersten Spezifikationen des *Post Office Protokoll Version 3* (**POP3**) sehen hierfür nach Aufbau der Verbindung eine Authentifizierung mittels Passwort vor [RFC1081, RFC1225]. In späteren Versionen (ab RFC 1460) wurde das `APOP`-Kommando hinzugefügt, ein Challenge-Response-Verfahren, bei dem ein Hashwert aus Servertimestamp und dem Nutzerkennwort anstatt des Klartext-Passwortes übertragen wird [RFC1460, RFC1725]. In der aktuellen Version [RFC1939] unterstützt POP3 viele weitere Mechanismen [RFC1734, RFC1731]. Dazu gehören Kerberos in Version 4, die GSSAPI[14] und S/Key One-Time Passwörter [RFC1760, RFC2289]. Mit den Möglichkeiten zur Erweiterung von POP3 nach RFC 2449 wurde später auch SASL hinzugefügt [RFC2449, RFC5034].
Das *Internet Message Access Protocol Version 4* (**IMAP**) sieht ebenfalls eine Authentifizierung vor, erlaubt aber auch die Anmeldung als anonymer Nutzer mit beliebigem Passwort [RFC1730, RFC2060]. Für öffentliche oder gemeinsam genutzte Postfächer kann es sinnvoll sein, jedoch nicht für den Zugriff auf persönliche Postfächer. Zur Authentifizierung sind neben Nutzername/Passwort die bereits erwähnten Mechanismen Kerberos, GSSAPI und S/Key-OTP möglich [RFC1731]. Die aktuelle Protokoll-Version erlaubt darüber hinaus die Einbindung von SASL [RFC3501].
SSL/TLS-Verschlüsselung ist ebenfalls in POP3 und IMAPv4 vorgesehen [RFC2595], eine zertifikatsbasierte Authentifizierung des Nutzers wurde jedoch nicht in die Standards aufgenommen. Trotzdem unterstützen einige Mailprogramme diese Funktionalität und erlauben die Nutzung von Zertifikaten in Verbindung mit Hardware-Token. Ein Problem stellt jedoch die in den POP3/IMAP-Standards vorgeschriebene Authentifizierung dar. Selbst wenn sich der Client bereits während des Aufbaus der SSL/TLS-Verbindung authentifiziert hat, wird weiterhin eine Authentifizierung im POP3-, bzw.

[9] SMTP nutzt Port 25. Für SMTPS hat sich Port 465 etabliert, obwohl dieser von der zuständigen Behörde IANA (*Internet Assigned Numbers Authority*) anderweitig vergeben wurde.
[10] SMTP-Submission auf Port 587 [RFC4409]
[11] `http://www.mozillamessaging.com`
[12] `http://www.postfix.org`
[13] `http://www.mozilla.org/MPL/`
[14] GSSAPI: *Generic Security Services Application Program Interface*, ein IETF-Standard für eine generische Security-API mit Unterstützung von Authentifizierungsfunktionen

IMAP-Protokollablauf vom Nutzer gefordert. Über den in SASL-Mechanismus `ANONYMOUS`, der einem Nutzer Zugriff ohne Nachweis seiner Identität gestattet, kann eine weitere Authentifizierung nach Verbindungsaufbau vermieden werden [RFC4505]. Die Server müssen hierzu beim Verbindungsaufbau dem Client mitteilen, dass lediglich dieser SASL-Mechanismus unterstützt wird. Dabei muss im Server trotz anonymer SASL-Anmeldung das Postfach des konkreten Nutzers bestimmt werden. In der POP3/IMAP-Spezifikation ist das so nicht vorgegeben, da die Identität während des Authentifizierungsschrittes zugeordnet wird. Der Server muss deshalb bei der Anmeldung aus dem übertragenen Zertifikat den Nutzer bestimmen und dem Postfach zuordnen können.

Während des Tests konnte eine lauffähige Version mittels des *Dovecot*-Email-Servers[15] aufgebaut werden. Dovecot hat in der getesteten Version 1.1.13 eine SSL/TLS-Option, die clientseitige Authentifizierung fordert. Da Dovecot auch Pluggable Authentication Module unterstützt (PAM, siehe Kapitel 7.2.2), können nachfolgende überflüssige Authentifizierungen mittels einer speziellen PAM-Konfiguration immer positiv beantwortet und damit ignoriert werden (siehe Anhang D.1.2). Aus dem Zertifikats-*Subject* lässt sich ein Attribut auswählen, das zur Anfrage an eine Nutzerdatenbank dient. Diese Datenbank liefert das zugehörige Postfach. So lässt sich einstellen, dass eine im *Subject*-Feld `x500UniqueIdentifier` kodierte User-ID zur Anfrage an eine LDAP-Datenbank verwendet wird. Falls ein Eintrag für den Nutzer vorliegt, was gleichzeitig als Autorisierung angesehen werden kann, liefert der LDAP-Eintrag die notwendigen Informationen.

Insgesamt entsteht ein Mailsystem nach Abbildung 7.5 bei der ein MUA in Verbindung mit einem Hardware-Token zum Versand und zur Abholung von Emails verwendet werden kann. Thunderbird bietet analog Firefox die Einbindung von Hardware-Token über PKCS#11-Provider. Mit dem SSO-PKCS#11-Proxy ist dann weder eine mehrmalige PIN-Eingabe, noch die Eingabe eines Passwortes erforderlich.

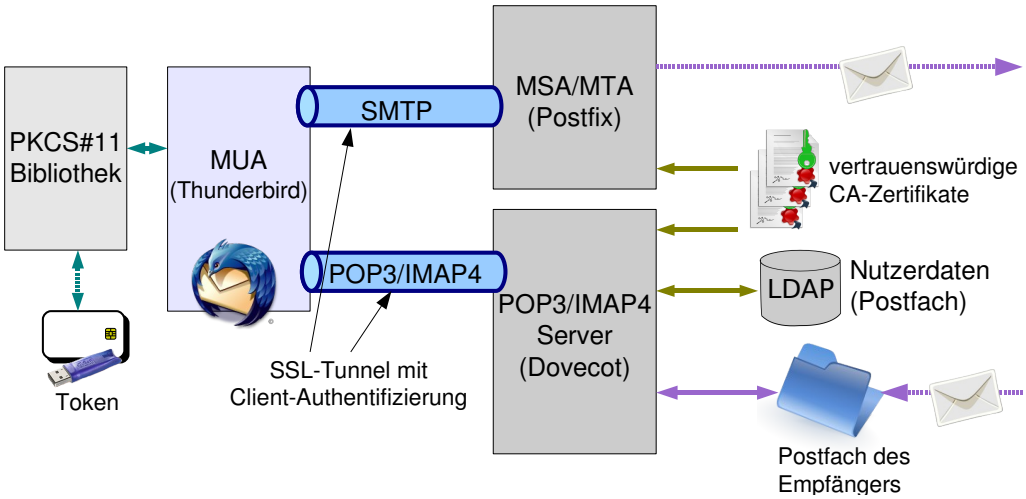



**Abb. 7.5 —** Zertifikatsbasierte Authentifizierung für Email-Anwendungen

[15] Dovecot ist eine weit verbreitete Mailserver-Applikation. Laut Aussagen der Entwickler wurde der Fokus nicht auf Funktionsumfang, sondern auf Sicherheit gesetzt. `http://www.dovecot.org`

### Remote Shell

Zu den häufig genutzten Anwendungen gehört der Zugriff auf Dateien, die Nutzung des Befehlsinterpreters (Shell-Zugriff) oder der Oberfläche (Remote-Desktop) entfernter Rechner. Für jedes dieser Anwendungsfelder wurden spezielle Netzprotokolle entwickelt, z.B. FTP, RCP für Dateiübertragung, TELNET, RSH für den Shell-Zugriff und X11 oder RDP für die Nutzung der grafischen Oberfläche [ES05]. Zur Authentifizierung dienen Passwörter, die z.B. in einem über X11 dargestellten Login-Dialog einzutragen sind und zum Server übertragen werden. Zunehmender Bedarf zur gesicherten Authentifizierung und zum Schutz der transportierten Daten ließen die Protokolle schnell an ihre Grenzen stoßen. Deshalb wurden neue Protokolle eingeführt, die zur Sicherheit auf Datenverschlüsselung setzen.

Die *Secure-Shell* (**SSH**) ist eines der Client-Server-Programme, welches mit verschlüsselter Kommunikation arbeitet. Das ursprüngliche SSH-Protokoll, welches einige Implementierungsschwächen besaß, wurde 1996 durch die als **SSH-2** bezeichnet Version ersetzt. SSH-2 wurde mit kleineren Änderungen und Erweiterungen im Jahre 2006 durch mehrere RFCs als Standard untermauert [RFC4250]-[RFC4254]. Für die meisten Betriebssystem sind SSH-2-Implementierungen für Client und Server unter einer freien Lizenz erhältlich, z.B. die *OpenSSH*[16].

SSH-2 ist trotz des Begriffes „Shell“ im Namen nicht auf die Funktion einer Remote Shell beschränkt, sondern bietet Möglichkeiten zur Übertragung von Daten (**SFTP** als Ersatz von FTP) oder zur Weiterleitung beliebiger Netzprotokolle durch einen SSH-Tunnel. Letzteres erlaubt die Nutzung des X11- oder RDP-Protokolls und damit Zugriff auf die Oberfläche des Rechners oder die Tunnelung weiterer Protokolle, z.B. HTTP. Abbildung 7.6 demonstriert das Prinzip. Erforderlich sind zwei Applikationen, der SSH-Server und ein Client. Die Verbindung erfolgt, indem der Client einen Transportkanal zum Server aufbaut, über den er sich anschließend authentifiziert. Bei Erfolg können über diesen Transportkanal mehrere Transportströme parallel arbeiten, die vom Server und Client weiter verteilt werden. Die Nutzung der Shell und die Dateiübertragung gehört zum SSH-Funktionsumfang. Für die anderen getunnelten Datenströme sind die weitere serverseitige Dienste und clientseitige Applikationen erforderlich.

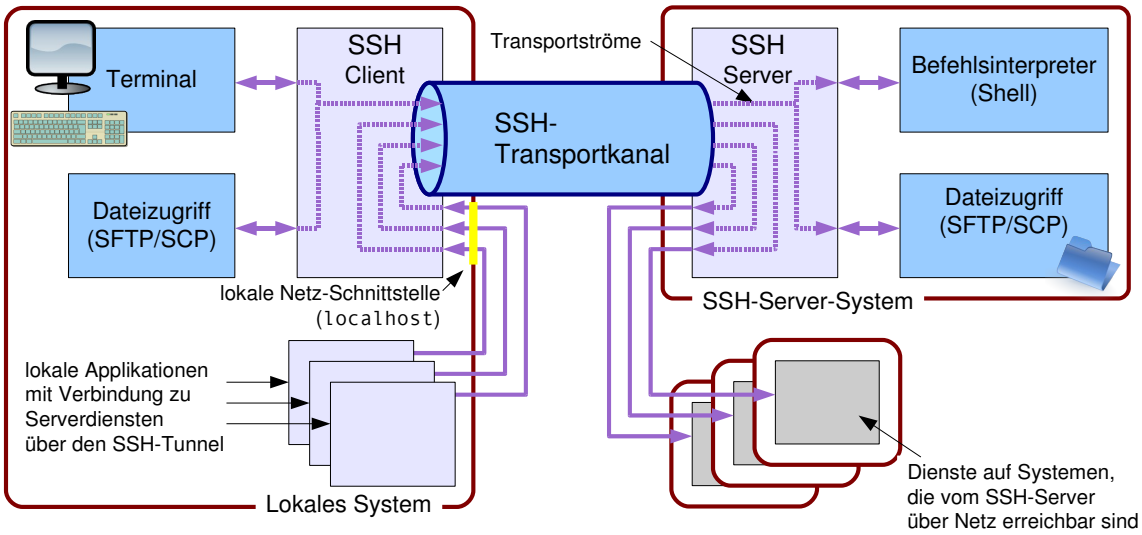



**Abb. 7.6 —** SSH: Datentunnel für mehrere Applikationen

[16] `http://www.openssh.com`

Aufgrund dieser vielfältigen Möglichkeiten wird die SSH-2 gern genutzt. Sichere Authentifizierungsverfahren sind hier besonders wichtig, da durch die Tunnelung andere Schutzmaßnahmen, wie z.B. Firewalls umgangen werden können. Die Authentifizierung wird in RFC 4252 geregelt. Neben Passwörtern gehört die Verwendung von Nutzerschlüsseln zu den Authentifizierungsmechanismen. Ergänzt werden diese durch Erweiterungen zur GSSAPI-Unterstützung und damit die Möglichkeit zur Nutzung von Kerberos [RFC4462].

Für die Ankopplung der Token und die Einbindung in das PKCS#11-System ist insbesondere der als „`publickey`“ bezeichnete Authentifizierungsmechanismus interessant. Dieser erlaubt dem Nutzer die Anmeldung mit dem privaten Schlüssel, wenn dem Server der zugehörige öffentliche Schlüssel bekannt ist. Dazu übersendet der Client neben dem Nutzernamen zuerst den öffentlichen Schlüssel, den der Server mit dem Hinterlegten vergleicht. Bei Akzeptanz übersendet der Client anschließend eine Signatur über Daten aus der bisherigen Kommunikation. Dazu muss der Client den privaten Schlüssel verwenden, der meist passwortgesichert in einer Datei abgelegt wird. Anhand der Signatur kann der Server die Authentizität prüfen.

Das Verfahren ähnelt der SSL/TLS-Client-Authentifizierung, hat aber Unterschiede. Insbesondere werden keine Zertifikate und damit keine PKI genutzt, sondern lediglich Schlüsselpaare. Die meisten Serverimplementierungen erwarten den berechtigten öffentlichen Schlüssel in einem speziellen OpenSSH- oder RFC 4716-Format als Datei in einem Unterverzeichnis des Nutzers [RFC4716]. Für deren Hinterlegung auf dem SSH-Server ist der Nutzer oder ein Administrator zuständig. RFC 4819 beschreibt eine Protokollerweiterung zur Verwaltung hinterlegter Schlüssel auf dem Server. Sie zielt jedoch ebenfalls auf eine Schlüsselverwaltung in Eigenverantwortung der Nutzer ab [RFC4819]. Andererseits verbietet die SSH-2-Spezifikation auch nicht die Verwendung von zentral vergebenen Zertifikaten. Das wird aber nur von wenigen SSH-2-Implementierungen unterstützt.

In Verbindung mit der `publickey`-Authentifizierung bietet die OpenSSH einen Automatismus, der von vielen Anwendern gern genutzt und als wesentliche Erleichterung bei häufigem Einsatz empfunden wird. Hierzu dient ein SSH-Agent, der die erforderlichen privaten Schlüssel im Speicher vorhält und bei Bedarf kontaktiert werden kann. Nach einmaliger Freigabe des Schlüssels erlaubt er den Aufbau neuer Verbindungen ohne erneute Eingabe des Passwortes zur Schlüsselfreigabe, vorausgesetzt, der gleiche öffentliche Schlüssel ist bei den Servern hinterlegt. Dies ermöglicht SSO für SSH-Anwendungen auf Basis eines clientseitigen Agenten. Der PKCS#11-Agent basiert auf der gleichen Idee, allerdings verwaltet er PINs und keine im Klartext vorliegenden privaten Schlüssel.

Die OpenSSH erlaubt die Verwendung des Smartcard-Frameworks OpenSC für die Nutzung der Schlüssel innerhalb von Crypto-Token. Im Rahmen dieser Arbeit wurde in einem ersten Schritt die bestehende OpenSC-Schnittstelle derart erweitert, dass der OpenSSH-Agent zur Hinterlegung der PINs von Crypto-Token genutzt werden kann. Diese Erweiterung wurde obsolet, nachdem für den clientseitigen Teil der OpenSSH ein Modul zur Einbindung von PKCS#11 zur Verfügung stand[17]. Dieses Modul erlaubt die SSH-`publickey`-Authentifizierung mit Token unter der Voraussetzung, dass die erforderlichen Signaturverfahren vom Token unterstützt werden. Die Verbindung mit dem PKCS#11-Proxy ermöglicht eine vollständige Integration in das clientseitige SSO-System. Ein laufender SSH-Agent oder eine serverseitige Veränderungen sind nicht erforderlich.

Anpassungen des SSH-Servers sind jedoch erforderlich, wenn auf eine serverseitige Hinterlegung von Schlüsseln verzichtet werden soll. Ein Patch für die OpenSSH erlaubt die Verwendung von Zertifikaten und somit die Hinterlegung des Zertifikat-*Subjects* anstatt eines Schlüssels[18]. Der Server prüft das Client-Zertifikat anhand von CA-Zertifikaten und bei Gültigkeit autorisiert er den Nutzerzugriff,

[17] `http://alon.barlev.googlepages.com/openssh-pkcs11`
[18] `http://roumenpetrov.info/openssh`

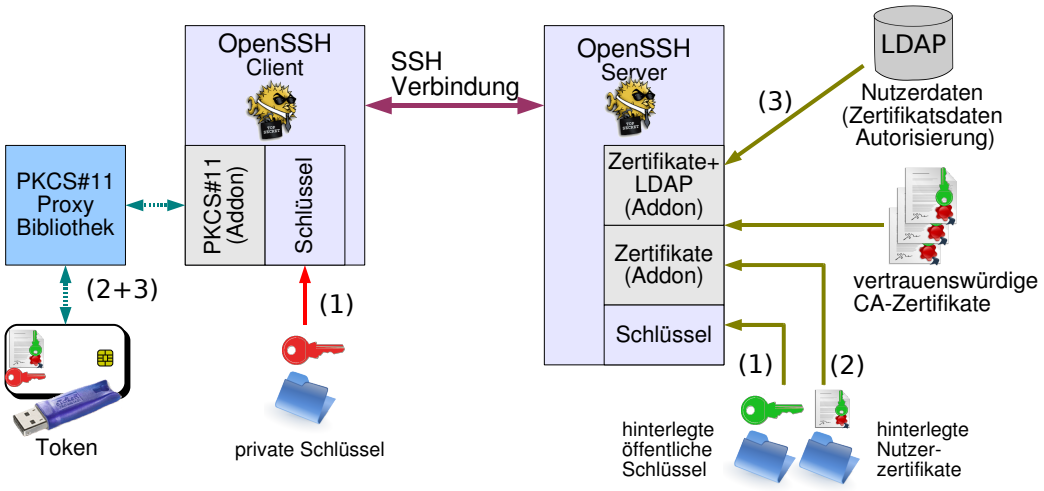



**Abb. 7.7 —** Einbindung der OpenSSH in clientseitiges SSO
(1) Standard ist die Nutzung von Schlüsselpaaren
(2) Einbindung PKCS#11, Verwendung von Zertifikaten, Hinterlegung im Server-Filesystem
(3) Hinterlegung der Daten im LDAP-Server, keine Ablage im Filesystem erforderlich

falls in der `publickey`-Datei das Zertifikats-*Subject* steht. Damit wurden Nutzerschlüssel durch Zertifikats-Einträge ersetzt, für deren Hinterlegung aber immer noch der Nutzer selber zuständig ist. In einem zweiten Schritt wurde für diese Arbeit eine OpenSSH-Erweiterung entwickelt, die in Verbindung mit dem Zertifikats-Patch die Anbindung des OpenSSH-Servers an eine LDAP-Datenbank ermöglicht. Für berechtigte Nutzer können die erforderlichen Zertifikate aus der zentralen LDAP-Datenbank entnommen werden. Dies verringert den serverseitigen Verwaltungsaufwand und kann mögliche Fehlerquellen reduzieren, da keine Schlüssel in Eigenverantwortung der Nutzer verwaltet werden.
Mit den Erweiterungen bilden OpenSSH-Client und -Server ein System nach Abbildung 7.7. Dieses lässt sich in die SSO-Lösung integrieren und ermöglicht gleichzeitig weiteren Client-Server-Applikationen die Nutzung gesicherter Transportkanäle.

### 7.2.2 Lokale Anwendung der PKCS#11-gestützten Authentifizierung

Die Beispiele Browser, Emailprogrammen und SSH zeigen, dass der Aufbau eines clientseitigen SSO-Systems für eine breite Palette von Client-Server-Diensten möglich ist. Neben netzgestützten Diensten ist eine Authentifizierung am lokalen Rechner erforderlich. Wie sich lokale Authentifizierung mit dem SSO-System vereinen lässt, wird im Folgenden gezeigt.

#### Rechner-Authentifizierung

Bei Einsatz eines Rechners im Mehrnutzerbetrieb ist eine Anmeldung am Rechner dann erforderlich, wenn entweder der Rechnerzugang eingeschränkt ist oder nutzerspezifische Ressourcen, z.B. ein Home-Verzeichnis zugewiesen werden sollen. Diese, als *Login* bezeichnete lokale Anmeldung des Nutzers am Rechner kann ebenfalls durch die zertifikatsbasierte Authentifizierung in Verbindung mit Crypto-Token erfolgen. Ein Token, welches das Authentisierungsmerkmal darstellt, muss zum Login mit dem Rechner verbunden werden. Das Login am Rechner selber darf natürlich nicht automatisch

über das SSO-System erfolgen, sondern hier muss der Nutzer die PIN selbständig eingeben. Das Login stellt den Ausgangspunkt und die Anknüpfung für das SSO-System dar, da gleichzeitig die PIN zur späteren Nutzung im Agenten hinterlegt werden kann. Die Umstellung auf zertifikatsbasierte Authentifizierung wäre deshalb auch für einen privaten Rechner von Vorteil, da sie einerseits den Agenten mit den notwendigen Informationen versorgt und anderseits einen gewissen Schutz des Rechners vor Missbrauch durch fremde Personen geben kann.
Um diese Idee praktisch zu testen, wurde ein bestehendes PKCS#11-PAM-Modul[19] erweitert. PAM bietet in UNIX-Systemen die Möglichkeit, Authentisierungsaufgaben an spezielle Bibliotheken, die PAM-Module zu delegieren [XSS97]. Dabei können mehrere Module mit unterschiedlichen Authentifizierungsverfahren parallel oder nacheinander genutzt werden. Neben der Authentifizierung gehören zur PAM-API Methoden zum Account- und Session-Management. Das Account-Management erlaubt die Trennung von Authentifizierung und Autorisierung, da für bereits erfolgreich authentifizierte Nutzer separat festgelegt werden kann, ob der Zugriff gewährt wird. Über das Session-Management kann zum Zeitpunkt des Starts oder beim Beenden der Nutzer-Sitzung weitere Aktionen ausgelöst werden. Außerdem bietet PAM unter dem Begriff Password-Management eine Funktion zur Änderung der Authentisierungsmerkmale an, die aber nicht auf Passwörter beschränkt ist. Ein PAM-Modul für Token, welches das Password-Management anbietet, kann z.B. die PIN aktualisieren.
Für die Einbindung des SSO-Systems bot sich PAM an, da bereits im OpenSC-Framework ein Modul *PAM-PKCS11*[20] vorlag. Dieses nutzt PKCS#11-Token als Authentisierungsmerkmal. Hierzu muss der Inhaber das Token mit dem Rechner verbinden und die PIN an das Modul übergeben. Dieses prüft anschließend die Zertifikate des Tokens auf Gültigkeit. Wird ein gültiges Zertifikat gefunden, erfolgt ein Token-Login mit der angegebenen PIN. Die Existenz des zugehörigen privaten Schlüssels im Token wird mittels eines Challenge-Response-Verfahrens geprüft. War dies erfolgreich, so wird in einem zweiten Schritt eine Zuordnung des Zertifikatsinhabers zum Systemnutzer durchgeführt. Dieser *Mapping*-Schritt kann verschiedene Quellen einbinden, um mittels der Informationen des Zertifikates den Systemnutzer zu bestimmen. Eine im Rahmen dieser Arbeit entstandene Erweiterung eines „LDAP-Mappers“ erlaubt die Einbindung mehrerer, redundanter LDAP-Server, um über eine LDAP-Abfrage mittels des Zertifikat-*Subjects* oder anderer Zertifikats-Daten den Nutzer aus dem LDAP-Datenbestand zu bestimmen.
Eine zweite Ergänzung des PAM-PKCS11-Moduls ermöglicht es, die während der Authentifizierung eingegebene PIN direkt an den Agenten zu übermitteln. Hierzu wird die PIN bis zum Start der Sitzung im PAM-Speicher gehalten. Sie wird im folgenden PAM-Schritt des Session-Starts ausgelesen, anschließend der PKCS#11-Agent gestartet und an diesen die PIN übergeben. Somit werden bereits alle notwendigen Information inklusive der PIN im Agenten hinterlegt. Nach Ablauf der Sitzung, beim `Logout`, wird der Agent über die PAM Session-Verwaltung beendet. Mit dieser Erweiterung und in Kombination mit der LDAP-Anbindung ergibt sich das System aus Abbildung 7.8. Nach dem Systemstart mit aktiviertem Agent stehen den Applikationen die notwendigen Informationen zur Verfügung. Damit wäre Single Sign-On vom Login bis zum Logout für alle zertifikatsbasierten Authentifizierungen ohne erneute PIN-Eingabe möglich.

Neben dem Betriebssystem-Login können weitere Applikationen zur Authentisierung auf die PAM-API zurückgreifen und somit ebenfalls eingebunden werden. Ein Beispiel wäre ein Bildschirmschoner, der nach Aktivierung die Konsole sperrt. Diese als *Screen-Lock-* oder *X-Lock* bezeichneten Applikationen erfordern eine Authentifizierung, bevor sie das System wieder für den Nutzer freigeben. Um Missbrauch vorzubeugen, ist die Tokenentnahme bei kurzer Arbeitsunterbrechung und Verlassen des

[19] PAM: Die *Pluggable Authentication Module* bieten eine modular konfigurierbare Authentisierungs-API.
[20] `http://www.opensc-project.org/pam_pkcs11`

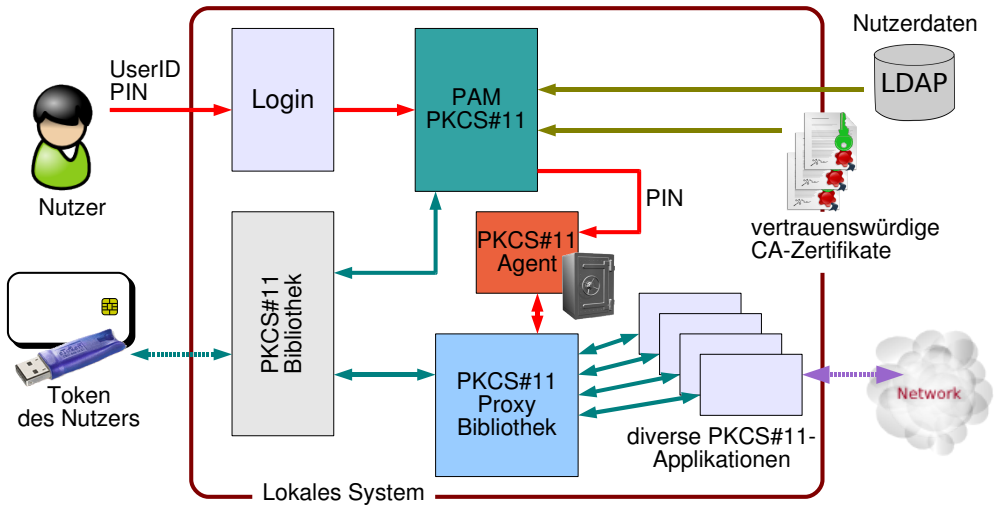



**Abb. 7.8 —** Lokale Rechneranmeldung mit PIN-Hinterlegung

Rechners sinnvoll. In Verbindung mit dem PAM-PKCS#11-Modul kann durch einen im Hintergrund laufenden Prozess bei Entfernung des Tokens automatisch der Screen-Lock aktiviert und die gespeicherten PINs aus dem Hauptspeicher entfernt werden. Bei Einlegen des Tokens muss der Screen-Lock deaktiviert werden, wobei eine erneute Authentifizierung über PAM erforderlich ist. Im PAM Session-Management Schritt der Screen-Lock-Software wird erkannt, dass bereits ein PKCS#11-Agent für den Nutzer aktiv ist und die PIN wird wieder übergeben. Das SSO-System speichert somit ohne eingelegtes Token keine PIN im Hauptspeicher und reduziert damit die mögliche Gefährdung eines PIN-Diebstahls bei Verlassen des Rechners, vorausgesetzt, das Token wird entnommen.

### 802.1X - Anmeldung am Rechnernetz

Bei der Einbindung eines Rechners in ein Rechnernetz unterscheidet man zwischen dauerhafter (statischer) und temporärer (dynamischer) Verbindung. Für statische Verbindungen kann eine Authentifizierung sinnvoll sein, für dynamische ist sie meist sogar vorgeschrieben. Internet-Service-Provider verlangen bei Einwahl die Authentifizierung des Kunden. Um einem Missbrauch in drahtlosen Netzen vorzubeugen, ist auch bei Kontaktaufnahme eines Notebooks zum WLAN eine Authentifizierung erforderlich. Aber auch für statische, drahtgebundene Rechnereinbindung kann eine Authentifizierung gegenüber der Netzwerkhardware sinnvoll sein. Dies verhindert die unbefugte Nutzung eines frei zugänglichen Netzwerkanschlusses, z.B. in einem Computerpool. Es existiert somit der Bedarf zur Authentifizierung eines Rechners bei statischen oder eines Nutzers bei dynamischen Verbindungen zum Netz. Diese Authentifizierung erfolgt am Netzzugangspunkt, z.B. dem *AccessPoint* bei der WLAN-Verbindung oder einem *Ethernet-Switch* bei einer auf Ethernet-Technologie beruhenden Kabelverbindung.

Im WLAN ist aufgrund der genutzten Funktechnik eine räumliche Eingrenzung schwer möglich. Deshalb wurden bereits beim Entwurf des WLAN-Standards IEEE 802.11 die gesicherte Datenübertragung und Authentifizierung berücksichtigt. Beim anfangs genutzten WEP-Verfahren konnten aber sehr früh Sicherheitsmängel erkannt werden, so dass WEP ersetzt werden musste. Die Nachfolger

WPA[21] und WPA2[22], basieren dazu auf der ursprünglich als Teil (i) bezeichneten Erweiterung von IEEE 802.11 [IEE04a]. Dieser im aktuellen WLAN-Standard [IEE07] eingeflossene Teil ersetzt nicht nur die Verschlüsselungsbasis von WLAN durch andere Verfahren, sondern bietet auch neue Authentifizierungsverfahren. Wie bei WEP können sich Nutzer eines AccessPoints mit dem gleichen Passwort oder Schlüssel (*shared secret*) anmelden. Für nutzerspezifische Abrechnung (*Accounting*) oder zur Trennung von Authentifizierung und Autorisierung genügt ein gemeinsames Geheimnis jedoch nicht. IEEE 802.11i ergänzt die Möglichkeiten durch den IEEE 802.1X Standard [IEE04b]. Dieser erlaubt eine Authentifizierung mit verschiedenen, als *Credentials* bezeichneten Merkmalen. Im einfachsten Falle kann es wieder eine Kombination aus Nutzername und Passwort sein.

IEEE 802.1X unterscheidet zwischen *Supplicant*, *Authenticator* und *Authentication Server* (AS). Der Supplicant ist der Nutzer bzw. Rechner, der sich am Authenticator anmeldet. Der Authenticator kontrolliert den Netzzugang. Für ein nicht authentifiziertes System kann vom Authenticator der Zugriff auf ein bestimmtes Netz oder einen Proxy-Server beschränkt oder ganz unterbunden werden. Nach erfolgreicher Authentifizierung kann der Netzzugriff erweitert werden. Bei einer nutzerspezifischen Lösung kann der Rechner in das Subnetz bzw. VLAN[23] des Nutzers geschaltet werden. Im WLAN übernimmt der AccessPoint die Rolle des Authenticators, im drahtgebundenen Netz der Ethernet-Switch. Die Aufgabe des AS besteht darin, die vom Supplicant übermittelten Credentials zu prüfen (Authentifizierung), die Berechtigung des Nutzer z.B. aus einer Datenbank zu ermitteln (Autorisierung) und diese anschließend dem Authenticator mitzuteilen.

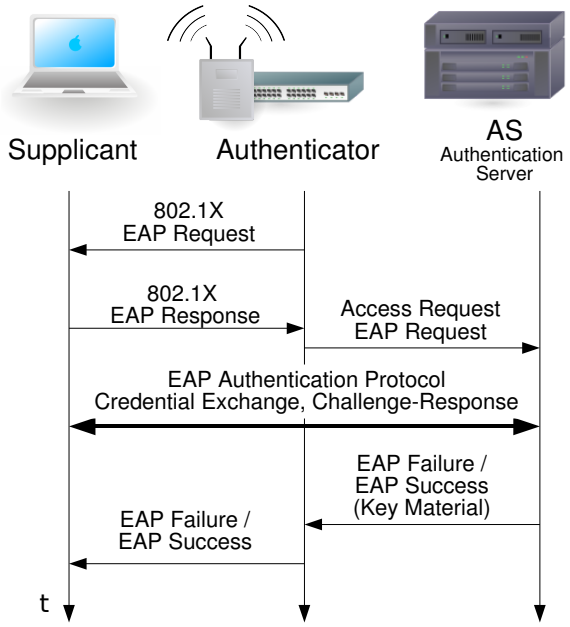



**Abb. 7.9 —** Ablauf 802.1X-Authentifizierung

Abbildung 7.9 skizziert den Ablauf. Der Supplicant verbindet sich mit dem Authenticator und löst da-

[21] WPA; *Wi-Fi Protected Access*, Teilmenge des IEEE 802.11i Standards, der unter Nutzung der bereits für WEP eingesetzten RC4-Stromchiffre durch besseres Key-Management einen akzeptablen Grad an Sicherheit erzielt.

[22] WPA2: Erweiterung von WPA, insbesondere durch Einsatz des symmetrischen AES-Verfahren, Umsetzung aller grundlegenden Funktionen von IEEE 802.11i

[23] VLAN: *Virtual Local Area Network,* logisches Teilnetz eines Netzwerkes

bei den Authentifizierungsvorgang aus. Über das genutzte EAP-Protokoll[24] fordert der Authenticator den Supplicant zur Authentifizierung auf und stellt bei korrekter Antwort eine Autorisierungsanfrage an den AS. Die anschließend vom Supplicant übertragenen Credentials werden vom Authenticator an den AS weitergeleitet. Hierbei kann es sich um bidirektionalen Datenaustausch, z.B. ein Challenge-Response Verfahren handeln. War die Authentifizierung erfolgreich, bestimmt der AS die Autorisierung und teilt diese dem Authenticator mit. Für WLAN werden auch initiale Schlüssel für die Verwendung in WPA bzw. WPA-2 übertragen. Anschließend gewährt der Authenticator dem Supplicanten Zugriff. Im WLAN-Umfeld werden die Verfahren auf EAP-Basis mit dem Begriff *WPA/WPA2 Enterprise* bezeichnet.

Für die Einbindung in die zertifikatsbasierte Authentifizierung müssen Supplicant und AS Zertifikate unterstützen und mittels des EAP-Protokolls die notwendigen Informationen übertragen können. Eine EAP-Einbettung gibt es für verschiedene Authentifizierungsverfahren. Mittels `EAP-PSK` *Pre-Shared Key* [RFC4764] kann ein gemeinsamer Schlüssel, abgeleitet aus einem Passwort verwendet werden. `EAP-TTLS`[25] erlaubt über einen TLS-Tunnel, bei der sich der AS gegenüber dem Supplicanten mit einem Zertifikat ausweisen muss, die gesicherte Übertragung der Credentials. Gern genutzt wird `EAP-TTLS` in Verbindung mit `PAP`[26], wobei Nutzerkennung und Passwort über den gesicherten Tunnel im Klartext übertragen werden. Somit kann für WLAN-Authentifizierung ein nutzereigenes Passwort anstatt eines gemeinsamen Geheimnisses eingesetzt werden. Das Passwort liegt dem AS im Klartext vor und kann gegen weitere Datenquellen, z.B. einen LDAP-Server geprüft werden. Das erlaubt die Nutzung eines zentralen Passwortes für mehrere Dienste.

Neben den genannten Verfahren bietet EAP mit `EAP-TLS` [RFC5216] auch die Unterstützung für clientseitige zertifikatsbasierte Authentifizierung. Das Verfahren ähnelt `EAP-TTLS`. Es wird aber kein TLS-Tunnel zur Übertragung weiterer Credentials aufgebaut, sondern das TLS-Handshake selber zur wechselseitigen Authentifizierung genutzt. Das bietet sich natürlich für die Einbettung in das SSO-System an. Dafür muss nicht nur der AS, sondern auch der Supplicant ein Zertifikat besitzen. Die Überprüfung von Client-Zertifikat und Schlüssel erfolgen nach dem SSL/TLS-Verfahren. Die Umsetzbarkeit der 802.1X-Authentifizierung auf Basis von Nutzer-Zertifikaten für den Einsatz in heterogenen Netzstrukturen konnte bereits im Rahmen einer Diplomarbeit verifiziert werden [Spa07].

Seitens des AS ist ein System erforderlich, das anhand von Zertifikaten die Nutzer und deren Autorisierung aus einer Datenbank ermitteln kann. Für das Testsystem wurde diese Funktion über den RADIUS[27]-Server *FreeRADIUS*[28] in Verbindung mit einer LDAP-Datenbank umgesetzt. Aus dem übertragenen Zertifikat wird bei Gültigkeit für den Nutzer anhand des *Subjects* ein LDAP-Eintrag ermittelt. Dieser Eintrag kann neben der Autorisierung eine Information über das zugeordnete Subnetz in Form einer VLAN-ID nach dem IEEE 802.1Q-Standard enthalten [IEE03]. Ein VLAN-fähiger AccessPoint als Authenticator kann somit den Rechner nach Authentifizierung des Nutzers in das zugeordnete Subnetz schalten. Das erweist sich als praktisch für eine vereinfachte Zugriffsregelung auf Arbeitsgruppen-Ressourcen, z.B. für den Zugriff auf zentralen Speicherplatz. Deshalb wurde im Rahmen dieser Arbeit *FreeRADIUS* erweitert, so dass die Nutzerkennung direkt aus einem Attribut des Zertifikats ermittelt werden kann. Die LDAP-Anfragen werden dadurch wesentlich beschleunigt.

Für die Token-Einbindung benötigt der Anwender einen Supplicant mit EAP-TLS-Unterstützung,

[24] EAP: ***E**xtensible **A**uthentication **P**rotocol*, ein Framework zur Nutzung verschiedener Authentisierungsverfahren in Netzwerken [RFC3748, RFC5247]

[25] `EAP-TTLS`: *EAP-Tunneled Transport Layer Security* [RFC5281]

[26] `PAP`: ein *Password Authentication Protocol* aus dem PPP-Standard [RFC1334]

[27] RADIUS: *Remote Authentication Dial-In User Service*, ein Dienst für Authentifizierung, Autorisierung und das Accounting von Nutzern in Rechnernetzen (AAA-System) [RFC2865]

[28] FreeRADIUS, Implementierung eines RADIUS-Servers unter einer freien Lizenz `http://freeradius.org`

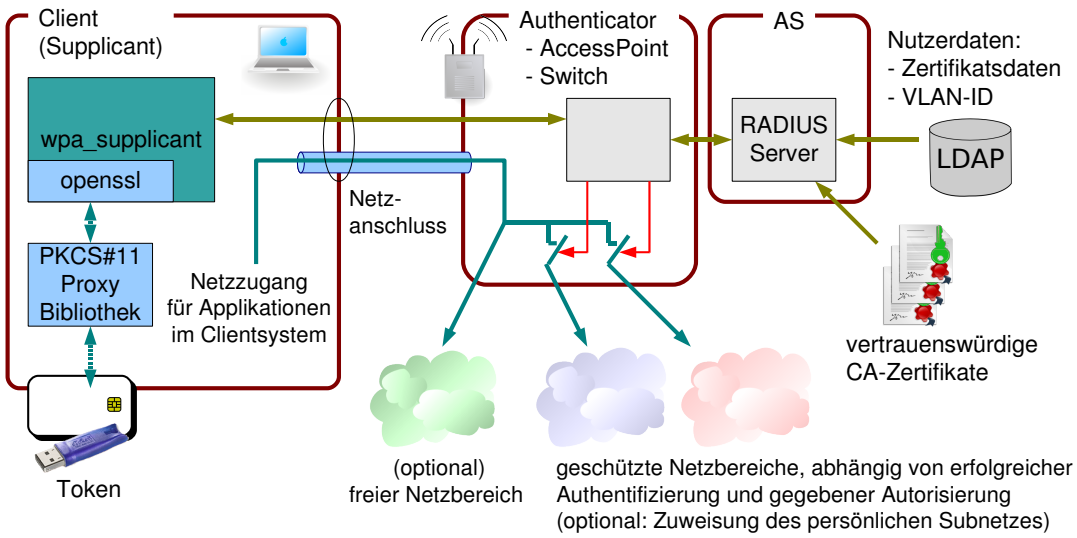



**Abb. 7.10 —** Netzzugangskontrolle über 802.1X und SSO-System Einbindung

der eine Schnittstelle zu Crypto-Token bietet. Supplicanten mit eingebauter PKCS#11-Unterstützung unter einer freien Lizenz existieren momentan noch nicht. Der verfügbare *wpa_supplicant*[29] unterstützt EAP-TLS und greift für die erforderlichen kryptographischen Funktionen auf die OpenSSL-Bibliotheken zurück. OpenSSL wiederum erlaubt den Zugriff auf Token über die PKCS#11-Engine. Über diesen Umweg war es möglich, mit freier Software einen Supplicanten mit Zugriff auf Token aufzubauen, der sich gleichzeitig in das clientseitige SSO-System integrieren lässt (Abbildung 7.10). Die Konfiguration ist im Anhang D.2.3 beschrieben.

### 7.2.3 Infrastruktur

Die komplette Einbindung des als Proof-of-Concept entwickelten Systems stellt Abbildung 7.11 dar. Das clientseitige System besteht aus der Token-Schnittstelle (1,2), der SSO-Erweiterung (3,3a) und den darauf aufbauenden Applikationen (8). Für die lokale Anmeldung mit Agent-Aktivierung dient das Login-System auf Basis von PAM-PKCS#11 (5), welches auch bei einer wiederholten Anmeldung, z.B. bei aktiviertem Screen-Lock genutzt werden kann. Hinzu kommt die für den Netzzugriff des Rechners erforderliche 802.1X-Supplicant-Software (6), welche über die OpenSSL-Engine (7) ebenfalls das Token nutzen kann. Über OpenSSL können auch weitere, hier bisher nicht erwähnte Applikationen um Token-Unterstützung ergänzt werden. Der PKCS#11-Eventmanager (4) dient zur Reaktion auf die Entnahme des Tokens, aktiviert den Screen-Lock und fordert den Agent zur Löschung der PIN auf.

Für die Etablierung 802.1X-Authentifizierung wird eine passende Hardware für den Authenticator benötigt. WLAN-AccessPoints müssen dazu die Enterprise-Funktionalität bieten. Für kabelgebundene Anschlüsse liefern VLAN-fähige Switches oft die erforderliche 802.1X-Funktionalität. In Verbindung mit einem RADIUS-Server und dessen Anbindung an die Nutzerverwaltung kann mit dem Token die Netzwerk-Anmeldung erfolgen.

Die serverseitigen Anpassungen beschränken sich auf Konfiguration der Dienste für die zertifikatsbasierte Authentifizierung. Um eine lokale Nutzerverwaltung für den jeweiligen Dienst zu vermeiden

[29] `http://hostap.epitest.fi/wpa_supplicant`

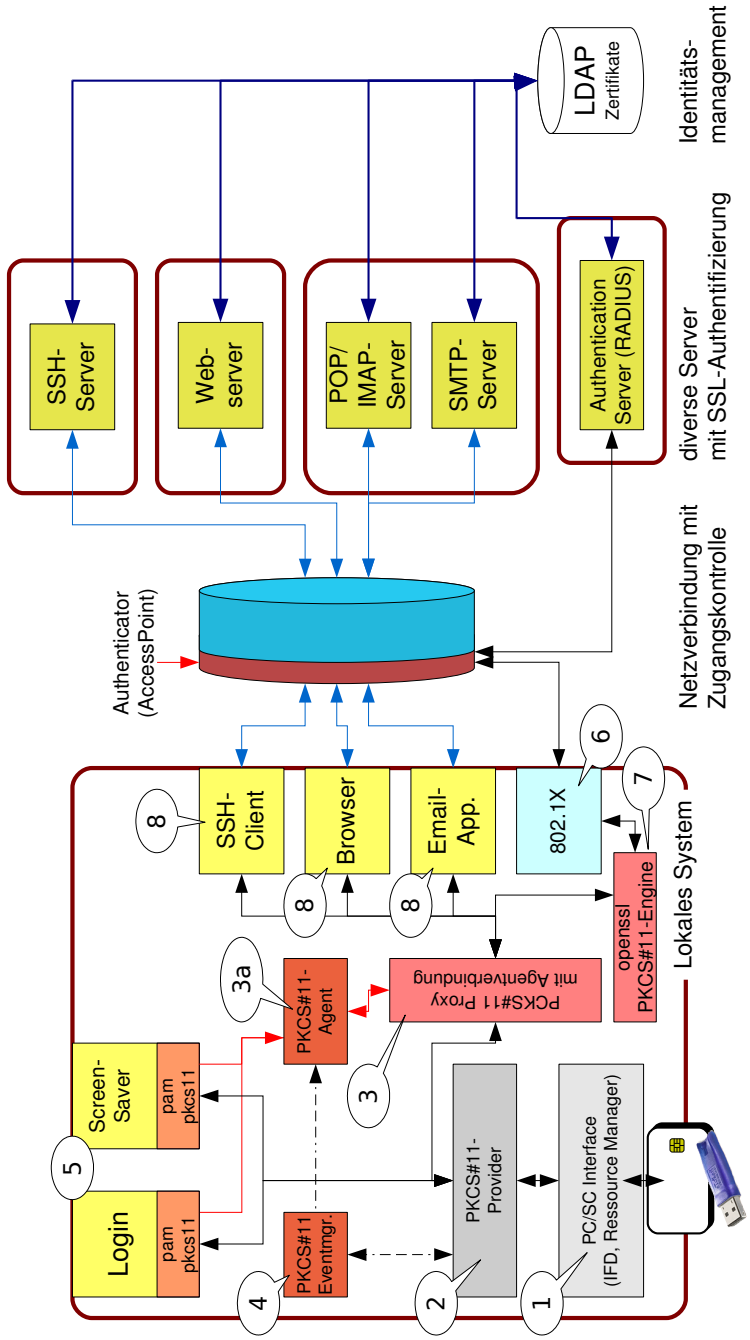



**Abb. 7.11** — Struktur des SSO-Systems für zertifikatsbasierte Authentifizierung

und weitere Synergieeffekte zu erzielen, sollte die Zuordnung der Dienst-Logins (auch als Nutzer-Account des Dienstes bezeichnet) zu den Zertifikaten über ein zentrales System erfolgen. Aus diesem System kann auch der RADIUS-Server für die Netzwerkanmeldung seine Daten beziehen. Poolrechner können die Zuordnung des Zertifikats zum Nutzer-Account darüber durchführen. Neben den Authentisierungsdaten können auch Informationen über die Nutzerrechte (Autorisierung) für die jeweiligen Dienste hinterlegt werden. Für alle getesteten Server-Applikationen einschließlich RADIUS konnte ein LDAP-Server genutzt werden, dessen Informationen aus dem zentralen Nutzer- und Identitätsmanagement der Einrichtung stammen und somit von zentraler Stelle gepflegt werden.

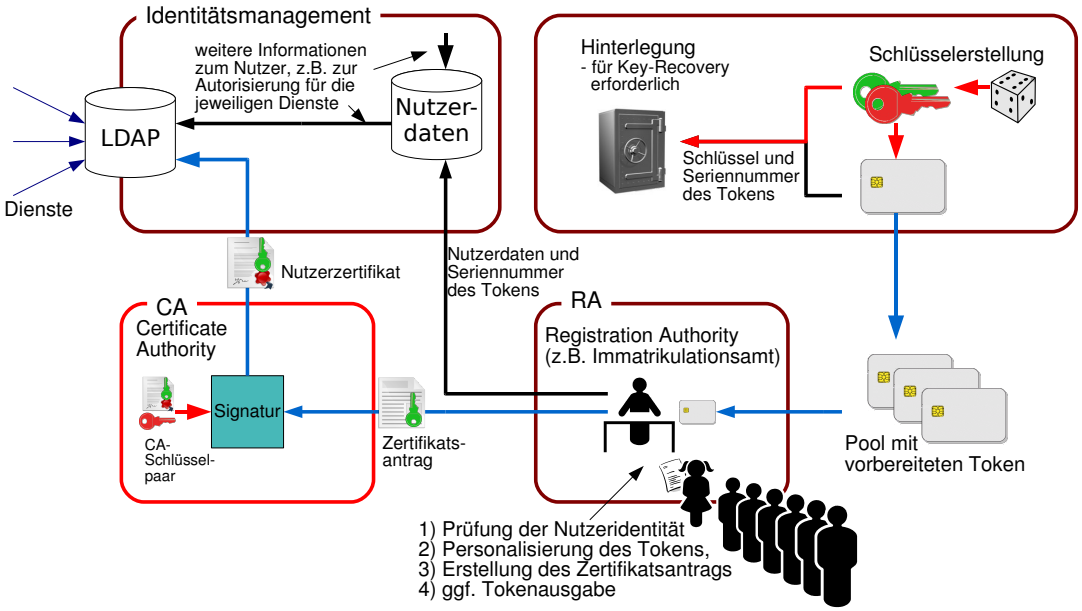



**Abb. 7.12 —** Organisatorischer Ablauf der Ausstellung und Hinterlegung von Zertifikaten

Für die Nutzer-Zertifikate der PKI wird eine ausstellende Certificate Authority (CA) benötigt. Die Ausstellung der Zertifikate kann mit der Verwaltung der Einrichtung gekoppelt sein. Für eine Hochschule wäre die Verbindung von Immatrikulationsamt mit einer Registration Authority (RA) sinnvoll (Abbildung 7.12). Bei der Immatrikulation werden die Token ausgegeben. Diese enthalten bereits Schlüsselpaare. Ein Zertifikatsantrag wird bei der Ausgabe an eine konkrete Person mit dem Tokenschlüssel erstellt und anschließend an die CA weitergeleitet. Nach Ausstellung werden diese im LDAP-Server oder in anderen Nutzerdatenbanken und nachträglich auf dem Token, z.B. an einem Selbstbedienungs-Terminal hinterlegt. Oder man wartet mit der Tokenausgabe, bis die Zertifikate erstellt und auf dem Token abgelegt wurden.

Das skizzierte System (Abbildung 7.11 und 7.12) beschreibt eine umfassende homogene Authentifizierungslösung auf Basis von Nutzer-Zertifikaten, welche die meisten Bereiche der täglichen Arbeit abdecken sollte. Die Voraussetzung zur Inbetriebnahme ist damit im Wesentlichen beschränkt auf das Vorhandensein einer PKI zur Ausstellung und Verwaltung der Zertifikate und der Bereitschaft der Nutzer zur Verwendung der Token.

## 7.3 Weitere Anwendungsmöglichkeiten der PKCS#11-Schnittstelle

Das System wurde vordergründig entwickelt, um bestehende passwortbasierte Authentifizierungslösungen mit einer sicheren Variante zu ergänzen oder komplett abzulösen. Werden Token eingesetzt, so eröffnen sich jedoch weitere Anwendungsmöglichkeiten. Diese können die Attraktivität der Tokennutzung steigern und somit zusätzliche Anreize zu deren Nutzung schaffen.

### 7.3.1 Verschlüsselung und Entschlüsselung

**Email:** In Verbindung mit dem Emailversand über das Internet ist zur Übermittlung vertraulicher Daten eine Verschlüsselung erforderlich. Hierfür bieten sich verschiedene Verfahren an.
Für die Verwendung von X.509-Zertifikaten existiert mit *S/MIME*[30] ein Standard, der es erlaubt, Nachrichten gezielt für einen oder mehrere Empfänger zu verschlüsseln. Zur Verschlüsselung wird ein gültiges Zertifikat des Empfängers benötigt. Dieses könnte in einem LDAP-Server (analog Abbildung 7.11) vorliegen und vom Mailprogramm erfragt werden. In der getesteten Applikation Thunderbird kann die Abfrage der Empfänger-Zertifikate aus LDAP-Servern über Plugins[31] aktiviert werden. Für andere Emailprogramme existieren ähnliche Lösungen. Dies ermöglicht das komfortable Versenden verschlüsselter Nachrichten, da Zertifikate von Empfängern innerhalb der Hochschule automatisch anhand der Empfängeradressen aus dem LDAP-Datenbestand ermittelt werden können. Zur Entschlüsselung benötigt der Empfänger Zugriff auf den privaten Schlüssel. In Verbindung mit dem Token ist hierfür die PKCS#11-Schnittstelle zu nutzen. Wurde Thunderbird bereits für die zertifikatsbasierte Authentifizierung konfiguriert, sind keine weiteren Anpassungen notwendig. Die Entschlüsselung erfolgt automatisch und im Hintergrund mit Hilfe des Tokens.
Gern genutzt werden außerdem PGP oder die freie Implementierung GnuPG. Sie bieten Verschlüsselung und erlauben Signaturen auf dem Vertrauensmodell des Net-Of-Trust. Sie sind als eigenständige Applikationen oder als Plugins für Emailprogramme verfügbar. Mit *gnupg-pkcs11-scd*[32] existiert eine PKCS#11-Schnittstelle für GnuPG, die zur Entschlüsselung mittels Token genutzt werden kann. Zur Verschlüsselung wird sie nicht benötigt, lediglich der öffentliche Schlüssel des Empfängers muss verfügbar sein. Dieser kann lokal im Filesystem liegen oder aus einem Key-Server bezogen werden. Der öffentliche Schlüssel kann natürlich auch aus einem X.509-Zertifikat des Empfängers, falls verfügbar, entnommen und innerhalb des Net-Of-Trust durch wechselseitige Unterschrift bestätigt werden. Dies ist prinzipiell möglich, da sowohl S/MIME, als auch die Protokolle von GnuPG und PGP auf Basis asymmetrischer Kryptographie mit den gleichen Verfahren arbeiten. Deshalb kann sowohl mit S/MIME-, als auch mit GnuPG/PGP in Verbindung mit dem PKCS#11-Agent eine Integration in das SSO-System erfolgen. Eine verschlüsselte Nachricht wird dann bei Empfang automatisch entschlüsselt.

**Dateien, Festplatten und Festplattenpartitionen:** Neben dem Emailversand können die benannten Programme auch zur Verschlüsselung vertraulicher Daten für die Ablage im Filesystem in Verbindung mit dem Token ohne Zusatzaufwand genutzt werden. Für die Verschlüsselung von Festplatten gibt es viele Varianten. Zur Einbindung von Token wurde LUKS[33] gewählt.
Wie die Grafik 7.13 zeigt, besteht eine LUKS-verschlüsselte Partition aus einem Header und dem mittels eines symmetrischen Verfahrens verschlüsselten Datenteil. Der Header besitzt bis zu acht

[30] S/MIME: *Secure/Multipurpose Internet Mail Extensions* [RFC3851]
[31] *Card Viewer Extended* `http://www.trustedbird.org/tb/Card_Viewer_Extended`
*Multi-LDAP* `http://www.trustedbird.org/tb/Multi-LDAP`
[32] `http://gnupg-pkcs11.sourceforge.net`
[33] LUKS: *Linux Unified Key Setup*, `http://code.google.com/p/cryptsetup/`

Key-Slots, die zur Entschlüsselung des verschlüsselten symmetrischen Masterkeys dienen. Jeder Slot enthält den Masterkey und sichert ihn vor Zugriff entweder mit einem Passwort oder einem weiteren Schlüssel, dem *Slot-Key*. Zur Erschwerung von Brute-Force-Angriffen wird das PBKDF2-Verfahrens eingesetzt [RFC2898].

Die Slot-Aktivierung mittels eines Slot-Keys erlaubt den Zugriff auf die Partition mit Hilfe des Tokens. Dieser Slot-Key kann im Token abgelegt werden. Einfacher wäre aber die verschlüsselte Ablage im Filesystem, wobei das Token nur zu dessen Entschlüsselung dient. Somit bleibt der Schlüssel auf dem Rechner und im Token wird kein Platz verbraucht. Dazu wird der Slot-Key mittels des eigenen öffentlichen Schlüssels verschlüsselt und auf einer weiteren Partition abgelegt. Hierfür bietet sich die Boot-Partition an, da diese immer unverschlüsselt ist. Bei der Freischaltung wird der Schlüssel aus der Boot-Partition ausgelesen und über die Tokenschnittstelle mit Hilfe des Tokens entschlüsselt. Der entschlüsselte Slot-Key wiederum dient zur Aktivierung eines Slots für den Erhalt des symmetrischen Masterkeys, der dann zur Entschlüsselung der Partition dient. Die Token-Schnittstelle kann über verschiedene Systeme umgesetzt werden. Erfolgreich getestet wurde das Tool *pkcs15_crypt* aus dem OpenSC-Framework zusammen mit Token auf Basis der PKCS#15-Spezifikation und die OpenSSL-Tools in Verbindung mit der Engine-PKCS#11 und diversen PKCS#11-Providern. *pkcs15_crypt* zeichnet sich dabei durch seine geringe Größe gegenüber OpenSSL aus.

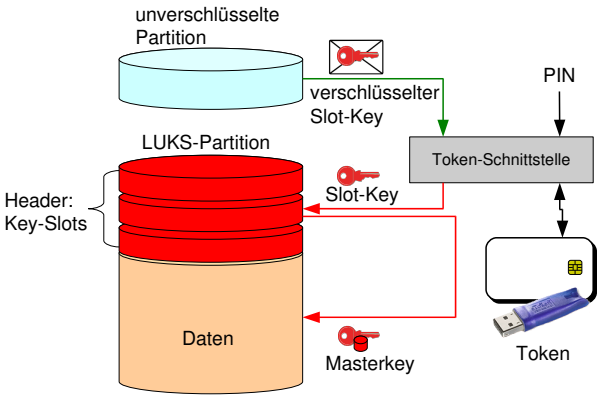



**Abb. 7.13 —** LUKS-Festplattenverschlüsselung

Die Einbindung in das SSO-System ist ebenfalls möglich. Wird jedoch die Betriebssystem-Partition verschlüsselt, so ist das System zu erweitern, da bereits vor dem Nutzer-Login zum Starten des Betriebssystems auf das Token zugegriffen werden muss. Dafür wurde eine Erweiterung des SSO-Systems gebaut, welche die Festplattenentschlüsselung durchführt und anschließend eine Nutzer-Sitzung mit aktivem Agenten startet. Für Poolrechner an der Hochschule ist dies weniger sinnvoll. Für wissenschaftliche Arbeitsplatzrechner und insbesondere für Notebooks kann eine derart verschlüsselte Festplatte aber den Schutz sensibler Daten bei Diebstahl gewährleisten.

### 7.3.2 Elektronische Signatur

Ein weiteres Einsatzfeld für Token bietet die elektronische Signatur. Diese kann zur Bestätigung der Authentizität elektronischer Nachrichten oder Dokumente genutzt werden. Das für die Token-Schlüssel ausgestellte Zertifikat muss die Verwendung zur Signatur erlauben. Die Zertifikats-Policy kann ein Zertifikat derart einschränken, dass es lediglich zur Verschlüsselung, aber nicht zur Signatur genutzt werden kann. In einem X.509v3-Zertifikat wird das über Flags in den Extensions geregelt. Beachtet der Empfänger diese Flags, so wird er eine unerlaubte Signatur nicht anerkennen.

In Verbindung mit einer zentralen Hinterlegung der Zertifikate in einem LDAP-Server können diese zur Prüfung von Signaturen herangezogen werden. So erlaubt das Emailprogramm Thunderbird mittels bereits erwähnter Plugins bei Empfang einer signierten Email mit unbekanntem Absender-Zertifikat den automatischen Abruf des Zertifikats aus einem der konfigurierten LDAP-Server. Bei erfolgreicher Prüfung gegen hinterlegte CA-Zertifikate wird es anschließend zur Verifizierung der

Signatur genutzt. Das erlaubt eine komfortable Prüfung von Email-Signaturen.
Aus Sicherheitsgründen sei allerdings angemerkt, dass Zertifikate zur Authentifizierung nicht gleichzeitig für die Signatur genutzt werden sollen. Hierfür gibt es zwei Gründe. Sollen über den Einsatzbereich der Hochschule hinaus rechtskräftig verbindliche Signaturen erstellt werden, so sind die Anforderungen der deutschen bzw. europäischen Signaturverordnung[34] einzuhalten. Diese schränkt für „qualifizierte elektronische Signaturen“ die Verwendung von Automatismen bei der Ausstellung ein und damit auch die Verwendung eines Agenten. Der für die Authentifizierung verwendete SSO-Agent würde prinzipiell eine Massensignatur erlauben, was der Verordnung widerspricht.
Der zweite Grund besteht in einem Sicherheitsproblem, welches sich durch die Anwendung des Signaturschlüssels ergibt, wenn dieser auch zur Verschlüsselung von Dokumenten oder zur Authentifizierung genutzt wird. Bei Anwendung des RSA-Verfahrens kann ein Angreifer aus den Informationen einer abgefangenen verschlüsselten Nachricht eine neue Nachricht erstellen und sie dem Inhaber des privaten Schlüssels übermitteln, welcher sie ggf. mit Signatur zurückschickt. Diese Antwort kann zur Entschlüsselung der ursprünglichen Nachricht genutzt werden [Eck06].

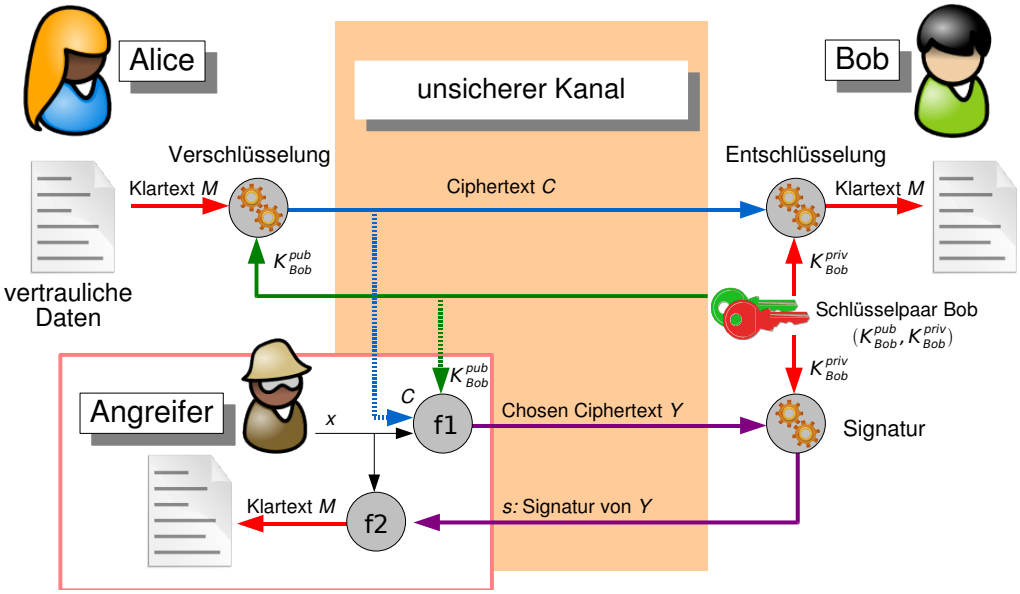



**Abb. 7.14** — Angriff mit gewähltem Kryptotext auf das RSA-Verfahren, wenn ein gemeinsames Schlüsselpaar für Verschlüsselung und Signatur genutzt wird

Den Ablauf des Angriffs skizziert Abbildung 7.14. Ver-/Entschlüsselung sowie Signatur und Verifikation mittels des RSA-Verfahrens erfolgen immer durch Berechnung von

$$\mathrm{RSA}(K, x) = x^{Exp} \mod N$$

Exponent $Exp$ und Modulus $N$ sind Teil des jeweiligen Schlüssels $K$. Im Beispiel möchte Alice eine Nachricht $M$ gesichert, d.h., verschlüsselt an Bob übermitteln. Hierzu nutzt sie den öffentlichen Schlüssel $K^{pub}_{Bob}$ von Bob, der aus dem öffentlichen Exponenten $e$ und dem Modulus $N$ besteht. Alice berechnet

$$C = \mathrm{RSA}(K^{pub}_{Bob}, M) = M^e \mod N \tag{7.1}$$

[34] Gesetz über Rahmenbedingungen für elektronische Signaturen (Signaturgesetz - SigG), Stand: 16. Mai 2001, §17: Produkte für qualifizierte elektronische Signaturen

Der Angreifer mit Zugriff auf $K_{Bob}^{pub}$ kopiert aus dem unsicheren Kanal den Wert $C$. Er wählt sich einen Wert $x < N$ und berechnet daraus den Wert $Y$:

$$\begin{aligned} X_C &= \mathrm{RSA}(K_{Bob}^{pub}, x) = x^e \bmod N \\ Y &= X_C \cdot C \mod N \end{aligned}$$

Nun verleitet der Angreifer Bob zur Unterzeichnung von $Y$, woraufhin er als Antwort die Signatur $s$ erhält:

$$s = \mathrm{RSA}(K_{Bob}^{priv}, Y) = Y^d \bmod N \tag{7.2}$$

Mit $x^{-1}$, dem multiplikativen Inversen[35] von $x \bmod N$ berechnet der Angreifer

$$\begin{aligned} x^{-1} \cdot s \pmod N &= x^{-1} \cdot Y^d \pmod N \\ &= x^{-1} \cdot {X_C}^d \cdot C^d \pmod N \\ &= x^{-1} \cdot (x^e)^d \cdot C^d \pmod N \\ &= \underbrace{x^{-1} \cdot x} \cdot \underbrace{C^d} \pmod N \\ &= 1 \cdot \mathrm{RSA}(K_{Bob}^{priv}, C) = M \end{aligned}$$

und hat anschließend Zugriff auf die geheime Nachricht $M$.
Um trotzdem Signatur und Verschlüsselung gleichzeitig mit einem Token durchzuführen, erfordert es zwei getrennte Zertifikate. Eines dient zur Ausstellung von Signaturen ($K_{Bob,sig}^{priv}$), das andere zur Verschlüsselung und für Authentifizierungszwecke ($K_{Bob,crypt}^{priv}$). Fängt ein Angreifer die Nachricht $C$ ab, in Gleichung (7.1) verschlüsselt mit $K_{Bob,crypt}^{pub}$, so kann er sie nicht mit Hilfe einer Signatur entschlüsseln, da die Berechnung (7.2) den Signaturschlüssel $K_{Bob,sig}^{priv}$ verwendet.

PKCS#11 erlaubt getrennte Slots mit dem gleichen Token (siehe Kapitel 5.2.2). Um diese Token im Einklang mit dem SSO-System nutzbar zu machen, ist die PKCS#11-Agent-Bibliothek zu erweitern. So dürfen keine Slots vom Agenten bedient werden, deren enthaltene Zertifikate für Signaturen gedacht sind. Eine Unterscheidung ist für den Agenten anhand der Zertifikats-Policy möglich.

Neben der Möglichkeit als Versand in einer signierten Email existieren zur Signatur von Dateien und Dokumenten verschiedene Standards. Prinzipiell können Dokumente mittels des S/MIME-Standards oder der PGP/GnuPG-Software signiert werden. Die Software legt hierzu neben dem Dokument eine Signaturdatei an, welche anschließend für die Prüfung der Authentizität herangezogen werden kann und demzufolge mit dem Dokument zu speichern ist. Mittels eines PKCS#11-Providers kann mit der freien S/MIME-Applikation *gpgsm*[36] in Verbindung mit gnupg-pkcs11-scd unter Einbeziehung der Token Dateien bzw. Dokumente signiert werden.
Für PDF-Dokumente existiert eine eigene Spezifikation zur Erstellung und Einbettung von Signaturen. Software verschiedener Hersteller (*Adobe Acrobat*, *Aloaha PDF*) unterstützt die Erstellung signierter PDF-Dokumente. Hier ist jedoch zu beachten, dass diese Programme meist direkt über die CT-API oder die PC/SC-Schnittstelle auf die Token zugreifen und die PKCS#11-Einbindung somit nicht möglich ist.

[35] Für teilerfremde Werte $x$ und $N$ ist das eindeutige multiplikative Inverse zu $x \bmod N$ der Wert $x^{-1}$, für den $x^{-1} \cdot x \equiv 1 \pmod N$ gilt.
[36] `http://www.gnupg.org`

# Kapitel 8

# Erfahrungen aus dem Testbetrieb und Sicherheitsaspekte

Praktische Erfahrungen mit dem clientseitig arbeitenden SSO-System konnten durch den Testbetrieb mit der entwickelten Linux-Software gewonnen werden. Dazu wurde der PKCS#11-SSO-Agent und die Proxy-Bibliothek auf mehreren Client-Systemen installiert und über einen mehrmonatigen Zeitraum im realen Einsatz getestet. Die Ausstattung der Testsysteme ist im Anhang C aufgelistet. Sie stehen exemplarisch für Rechner der unteren und mittleren Leistungsklasse. Die gewonnenen Erfahrungswerte belegen die praktische Nutzbarkeit selbst auf leistungsschwachen Rechnern.

Im Gegensatz zur passwortbasierten Authentifizierung, die in den meisten Applikationen bereits eingebaut ist, verlangt das SSO-System eine zusätzliche Software, die verschiedene Auswirkungen haben kann:

- erhöhter Speicherverbrauch und CPU-Belastung
- verändertes Laufzeitverhalten, z.B. Verzögerungen bei Anmeldungen
- Schwachstellen und Angriffsmöglichkeiten durch Verlagerung auf besitzbasierte Authentifizierung

Für eine Etablierung des SSO-Systems ist zu klären, wie sich diese Punkte bei der alltäglichen Arbeit für den Nutzer auswirken können. Um die Aussage zur praktischen Nutzbarkeit des SSO-Systems zu untermauern, wurde deshalb der auftretende Ressourcenbedarf ermittelt und mögliche Angriffspunkte analysiert. Die Ergebnisse werden in diesem Kapitel vorgestellt.

## 8.1 Ressourcenbedarf

Ein Punkt betrifft den entstehenden Bedarf an Ressourcen in Form von zusätzlichem Speicher, erhöhter Prozessor-Belastung und längerer Ausführungszeit. Dieser Overhead kann sich durch die zusätzliche Software im Clientsystem ergeben oder in Folge der Umstellung von passwortbasierter auf Token-gestützte Authentifizierung auftreten. Im Testsystem konnten folgende Werte zum Ressourcenverbrauch ermittelt werden.

### 8.1.1 Speicheranforderung und Laufzeiten für Agent und Proxy-Bibliothek

Die getesteten Applikationen erlauben die Nutzung von PKCS#11-Bibliotheken für kryptographische Anwendungen. Somit belegen sie den dafür erforderlichen Speicher unabhängig vom SSO-System. Deshalb soll hier nur der Verbrauch betrachtet werden, der durch die zusätzliche Einbindung des Client-SSO-Systems entsteht. Folgende Grafik verdeutlicht den Zusammenhang:

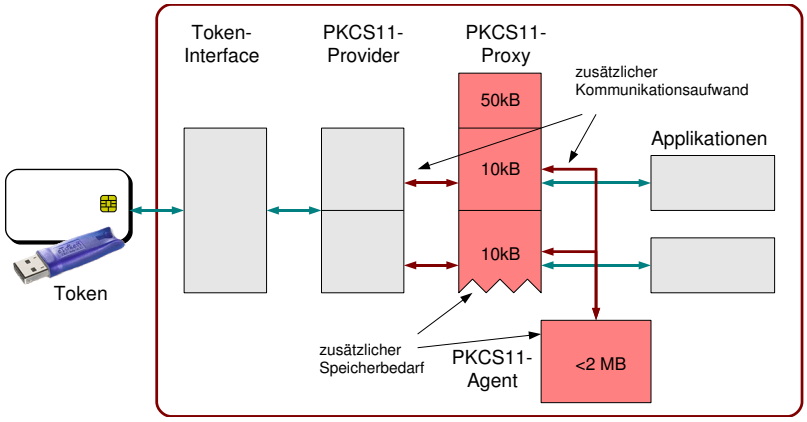



**Abb. 8.1 —** Zusätzlicher Speicherbedarf des SSO-Systems

Der Bedarf an Arbeitsspeicher liegt in einem Test mit zwei hinterlegten Token im Durchschnitt bei weniger als 2 MByte für den Agenten, wovon lediglich 320 kByte resident sind und somit im Speicher vorgehalten werden müssen. Die hinterlegte PIN wird durch technische Maßnahmen in diesem physikalischen Speicher verwahrt und kann nicht auf Festplatte ausgelagert werden.
Zusätzlichen Speicher erfordert jede Applikation für die Provider-Bibliothek, die in diesem Falle durch die PKCS#11-Proxy-Bibliothek ersetzt wird. Dieser Bedarf beziffert sich auf weniger als 60 kByte und setzt sich aus 50 kByte shared Memory für allen Applikationen und 10 kByte für die applikationseigenen Daten zusammen. Es gilt

$$\text{Speicherbedarf} \approx 50\text{kB} + 10\text{kB} \cdot |\text{Applikationen}|$$

Bei einer Schätzung von höchstens 100 verschiedenen PKCS#11-Applikationen ergibt sich ein Hauptspeicherbedarf von weniger als 3 MByte. Der Wert 100 ist dabei für ein Clientsystem sehr hoch angesetzt. Insgesamt stellt die speichermäßige Belastung für ein Endanwender-System auf einem üblichen Heim- oder Bürorechner keine größere Belastung dar.

Eine erhöhte Auslastung des Prozessors ist ebenfalls kaum bemerkbar. Abgesehen von der Initialisierung und der Login-Funktion handelt es sich meist nur um die Weiterleitung der Informationen von der Applikation zum Hersteller-Provider und umgekehrt. Beim Start einer Applikation und Initialisierung der Proxy-Bibliothek erfolgt die Prüfung der Berechtigung seitens des Agenten. Hier kommt es auf den Umfang der Prüfung an und wie diese für das jeweilige Betriebssystem umgesetzt wird. In der Testimplementierung im Linux-System ist weniger als eine Sekunde bei etwas erhöhtem I/O-Load aber ohne merkliche Prozessorbelastung messbar. Viele Applikationen lagern ihr PKCS#11-Subsystem in einen getrennten Thread oder Prozess aus. Bei Start eines Webbrowsers, der auch auf einem aktuellen Mehrkern-Prozessor im Schnitt mehrere Sekunden dauert, erfolgt die PKCS#11-Initialisierung in dieser Zeit parallel im Hintergrund. Sobald der Browser dem Nutzer zur Verfügung steht, ist auch die PKCS#11-Initialisierung abgeschlossen.

Zum Overhead durch das Client-SSO-System müssen im Hintergrund agierende Dienste hinzugerechnet werden, die bestimmte Aufgaben im Zusammenhang mit dem Token erfüllen. Dazu zählt der Dienst, welcher bei Entnahme des Tokens einen Bildschirmschoner startet. Kann der Dienst auf PKCS#11-Events reagieren, so würde er nur im Falle der Tokenentnahme aktiv, also keine dauerhafte Last verursachen. Unterstützt die Provider-Bibliothek keine derartige Event-Rückmeldung, so kann die Entnahme durch Polling-Abfragen, z.B. im Abstand von 5 Sekunden detektiert werden, was ebenfalls zu keiner spürbaren Belastung führt.

Zusammenfassend lässt sich sagen, dass die clientseitige Ressourcen-Belastung und auftretende Verzögerungen durch Hinzunahme des Agent-Systems derart gering sind, dass der Einsatz auf allen gängigen Arbeitsrechnerklassen, vom Notebook mit wenig Ressourcen bis zum gut ausgestatteten Arbeitsplatzrechner möglich ist. Jedoch ist zu bedenken, dass die Umstellung auf zertifikatsbasierte Authentifizierung und die Verwendung von Hardware-Token prinzipielle Verzögerungen gegenüber andere Verfahren mit sich bringen kann, die nicht durch das SSO-System bedingt sind.

### 8.1.2 Antwortzeiten bei Verwendung von Hardware-Token

Es ist die Frage zu klären, ob durch einen Wechsel auf zertifikatsbasierte Authentifizierung andere Beeinträchtigungen zu erwarten sind, die eine praktikable Nutzung erschweren. Dies können hohe Wartezeiten bei der Authentifizierung oder andere für den Nutzer spürbare Beeinträchtigungen sein.
Im Gegensatz zur kaum merklichen Verzögerung durch die Proxy-Bibliothek gegenüber der direkten Nutzung des PKCS#11-Providers sind bei einem Einsatz von Hardware-Token die Antwortzeit durch Aufruf einzelner PKCS#11-Funktionen, insbesondere der kryptographischen Funktionen mit dem privaten Schlüssel bemerkbar. Verantwortlich dafür ist die Abarbeitung der PKCS#11-Funktionen innerhalb der Token. In der aufgezeichneten PKCS#11-Sitzung im Anhang B kann dies bei der Authentifizierung (Signaturoperation auf Seite 152 ff.) erkannt werden.
Gründe für die Verzögerung sind bei Chipkarten die relativ langsame serielle Anbindung über das Terminal und allgemein bei allen Token die relative geringe Verarbeitungsgeschwindigkeit des Token-Prozessors im Vergleich zu einem PC-Prozessor. Das trifft insbesondere für aufwendige Algorithmen, wie z.B. bei Durchführung einer RSA-Signatur- oder Entschlüsselungsoperationen zu. Zur Bewertung dieses Umstands ist zu prüfen, wie kryptographische Operationen innerhalb des Tokens die Ausführungszeit der Authentifizierung negativ beeinflussen.

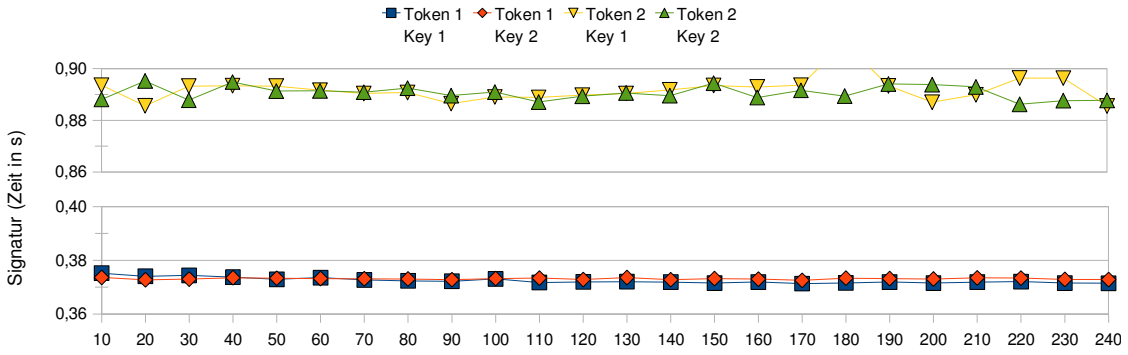



**Diagramm 8.2 —** Ausführungszeit einer Signaturoperation in Abhängigkeit von der Eingabegröße

Das Diagramm 8.2 zeigt die Ausführungszeit bei RSA-Signaturoperationen am Beispiel des *Aladdin eToken Pro 72k* (Token 1) mit dem Provider *Aladdin PKI Client* (V5.00.28) und einem Vertreter aus

der häufig anzutreffenden Serie der *CardOS 4.3B*-Smartcards (Token 2) mit dem Provider *Siemens HiPath SIcurity Card API* (V3.1B). Die Ergebnisse wurden für diese Token angegeben, da es sich um die Exemplare mit der kürzesten und längsten Ausführungszeit aus dem zur Verfügung stehenden Testpool handelt (siehe Anhang C).

Es wurden Tests mit verschiedenen Schlüsselpaaren bei einem Modulus der Größe 2048 Bit auf Eingaben zwischen 10 und 240 Byte durchgeführt[1]. Die Größe 2048 Bit wurde gewählt, da dieser Wert hinreichende Sicherheit gegen Angriffe bieten sollte [BNA08]. Für viele Token stellen 2048 Bit auch die obere Grenze dar. Getestet wurden 120 zufällig generierte Schlüssel. Stellvertretend sind die Ergebnisse der Schlüssel mit den größten Laufzeitabweichungen gegenüber dem Durchschnitt im jeweiligen Token angegeben.

Die Messung startet, nachdem die Verbindung zum Token aufgebaut wurde. Die gemessene Zeit umfasst deshalb die Kommunikation zwischen der Software und dem Token, beginnend mit dem Verpacken der Befehle und Daten in Argumente der APDUs, das Versenden, die Ausführung im Token und das Abholen der Rückgabewerte. Angegeben ist der Mittelwert über 20 Durchläufe auf zufällige Eingabewerten der angegebenen Größe. Es greift jeweils nur ein Prozess mit einem Thread auf das Token zu. Das Diagramm zeigt die Werte für Signaturoperationen, die Laufzeit für Entschlüsselungsoperationen ist aufgrund der RSA-Arbeitsweise identisch. Sie unterscheiden sich nur durch die Dateneinbettung für die Eingabe der Operation, das so genannte *Encryption scheme* bzw. *Padding*[2]. Die PKCS#1 Spezifikation in der aktuellen Version 2.1 [PKCS1, RFC3447] empfiehlt die Verwendung von `OAEP`[3], das aber vom Token 2 nicht unterstützt wird. Deshalb wurde das noch sehr häufig genutzte `RSAES-PKCS#1-v1_5` Padding nach PKCS#1 (Ver. 1.5) für die Messung gewählt. Für die Laufzeitaussagen sollte es aber kaum erkennbare Abweichungen geben.

Die Ergebnisse für Token 2 decken sich mit den Herstellerangaben von 820ms (Signatur) für den Infineon Smartcard-Controller SLE66CX322P bei einer Taktfrequenz von 5MHz unter Beachtung der Initialisierung und des Kommunikationsaufwands [Inf02a]. Dieser ist in den getesteten CardOS 4.3B-Smartcards eingebaut, für welche die längste Laufzeit gemessen werden konnte. Für den Aladdin eToken-Controller mit der schnellsten gemessenen Ausführungszeit waren keine Herstellerangaben verfügbar.

Wie am Ergebnis zu erkennen ist, hängt die Ausführungszeit nicht von der Länge der Eingabe ab. Aufgrund des gewählten Paddings ergibt sich für jede Signatureingabe zwischen 10 und 240 Byte eine Ausgabe der festen Größe 256 Byte (analog Modulus von 2048 Bit). Das Padding erfolgt im Token, es werden somit unterschiedlich viele Bytes übertragen. Die Länge, welche bei der Kommunikation mit dem Token als Faktor in Form der APDU-Größe eingeht, ist aber demzufolge vernachlässigbar gegenüber der Ausführungszeit der Operation innerhalb des Tokens. Sind längere Eingaben erforderlich, so wird die Eingabe gesplittet und in mehreren Schritten verarbeitet. Die Laufzeit ist dann mit der Schrittanzahl zu multiplizieren. Die Größe spielt somit keine Rolle, insofern die Operation in einem Schritt ausgeführt werden kann. Dies ist für gängige Challenge-Response-Authentifizierungen der Fall, da dort Eingabewerte mit einer Länge bis zu 160 Bit verwendet werden[4].

Wenn die Laufzeit nicht vom Schlüssel und der Länge der Eingabe abhängt, so könnte sie immer noch

[1] Die PKCS#11 zugrunde liegende PKCS#1-Spezifikation für RSA in Version 1.5 erlaubt für einen Modulus der Länge $k$ Eingaben mit einer Blockgröße $\leq k - 11$ (in Byte) für den genutzte Mechanismus `CKM_RSA_PKCS`, falls die Durchführung der Operation in einem Schritt erfolgen soll.

[2] Das Padding dient der Erschwerung von Angriffen auf das RSA-Verfahren mit gewähltem Klartext, für die RSA ohne Formatierung der Eingabe anfällig wäre [Ble98].

[3] OAEP; *Optimal Asymmetric Encryption Padding*, gilt als sicher vor adaptiven Chosen-Ciphertext Angriffen

[4] 160 Bit entspricht der Länge eines SHA-1 Wertes, dem aktuell meist genutzten Algorithmus zur Berechnung von Hash- bzw. Digest-Werten.

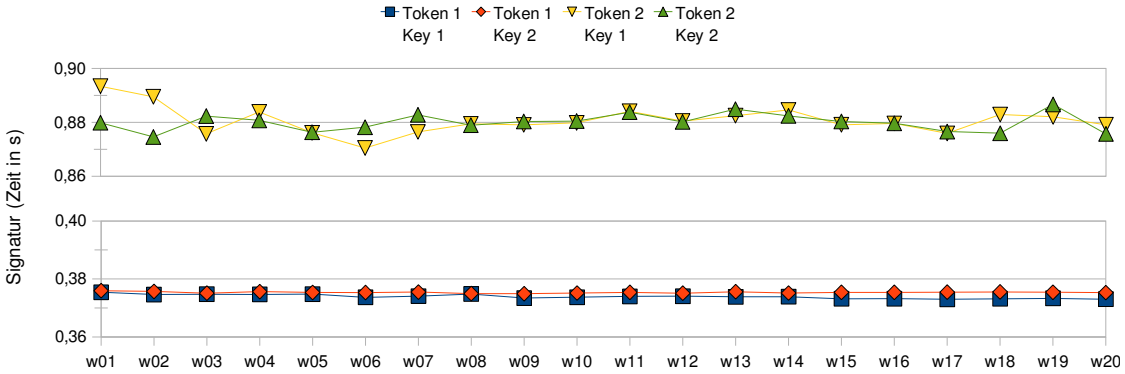



**Diagramm 8.3 —** Ausführungszeit einer Signaturoperation in Abhängigkeit von der Eingabe

durch den Inhalt der Eingabe beeinflussbar sein. Für die Messung in Diagramm 8.2 wurde der Durchschnitt über 20 zufällige Eingaben der jeweiligen Größe verwendet. Dagegen zeigt Diagramm 8.3 die Laufzeit für feste Eingabemuster der Länge 240 Byte. Es wurden wieder die Schlüsseln aus Diagramm 8.2 verwendet. Im Unterschied zur vorhergehenden Messung wurde aber der Mittelwert über 100 Messungen mit jeweils dem gleichen Eingabemuster gebildet. Die Muster bestehen aus konstanten Folgen von Nullen und Einsen (w1 und w2) und zufällig gewählten Folgen (w3 bis w20). Es zeigt sich, dass die Laufzeit nahezu unabhängig von der Wahl der Eingabe ist.
Diese Laufzeitstabilität wurde erwartet. Der Grund liegt in einer Sicherheitsanforderung an Crypto-Token. Mittels Seitenkanalanalysen kann ein Angreifer versuchen, ohne Hardwarbeschädigung z.B. durch Messung des Stromverbrauchs, elektrischer Emissionen oder der Laufzeit kryptographischer Operationen mit dem geheimen Schlüssel einen Rückschluss auf dessen Aufbau zu ziehen. Bei der Durchführung einer RSA-Signaturoperation mit dem geheimen Schlüssel auf Eingabe $M$ erfolgt die Berechnung

$$s = \mathrm{RSA}(M, K^{priv}) = \mathrm{padding}(M)^d \mod N$$

Die Laufzeit der im Token mittels modularer Exponentiation schrittweise berechneten Exponentialoperation wäre abhängig von der Eingabe $M$ und würde einen Rückschluss auf den Exponenten $d$ erlauben. Ziel der Token-Ingenieure ist es daher, diese Abhängigkeiten zu verschleiern, um die Extraktion des privaten Schlüssels zu erschweren. Neben der physikalischen Härtung der Token-Chips (verschleiertes Chip Layout, Verschlüsselung auf den Datenbussen uvm.) werden bei Ausführung der RSA-Operation zufällige Werte eingebaut. Der folgende Algorithmus erzeugt eine Laufzeit, die unabhängig von der Eingabe ist. Das Padding wird nicht berücksichtigt. Gegeben ist die Eingabe $M$ und das Schlüsselpaar $K$ bestehend aus den Exponenten $e, d$ und dem Modulus $N$.

$blind$ = zufälliger Wert, relativ prim zum geheimen Exponenten $d$;
$M = M \cdot blind$; $r = 1$;
for Bits b in $d$; do
  $r = r \cdot r \mod N$;
  if b>0 then $r = r \cdot M \mod N$; fi;
done
$r = r \cdot blind^{-e} \mod N$;

Der Rückgabewert $r$ ist die Signatur der Eingabe. Durch den zufälligen Wert $blind$ ist die Laufzeit nicht mehr direkt von der Eingabe abhängig. Das Verfahren wird als RSA-Blinding bezeichnet [QPDK04].

Die gewählten technischen Maßnahmen sorgen somit dafür, dass eine nahezu eingabeunabhängige Laufzeit erreicht wird. Laufzeiten sind als fester Wert für alle Eingaben anzusehen, für die eine Operation in einem Schritt durchgeführt werden kann. Dies trifft auch für die Signaturoperation auf der Challenge während der Authentifizierung zu. Demzufolge erlauben die gemessenen Werte eine Aussage über die Verzögerung bei der Authentifizierung. Laut der PKCS#11-Spezifikation kann eine Applikation ein Token für mehrere parallele Operationen einsetzen, z.B. über Multithreading. Agieren gleichzeitig mehrere Applikationen mit dem Token, so besteht auch eine gewisse Wahrscheinlichkeit, dass zu einem Zeitpunkt mehrere Prozesse gleichzeitig auf das Token zugreifen Abschließend wäre deshalb noch der Fall zu betrachten, bei der ein gleichzeitiger Tokenzugriff über mehrere Threads oder Prozesse erfolgt und wie sich das auf die Laufzeit auswirkt.
Dazu wurden mit den Token und Schlüsselpaaren aus Diagramm 8.2 Messungen für mehrere parallel agierenden Threads oder Prozesse auf gewürfelten Eingaben der Länge 240 Byte durchgeführt und der Mittelwert über 20 Durchgänge bestimmt.

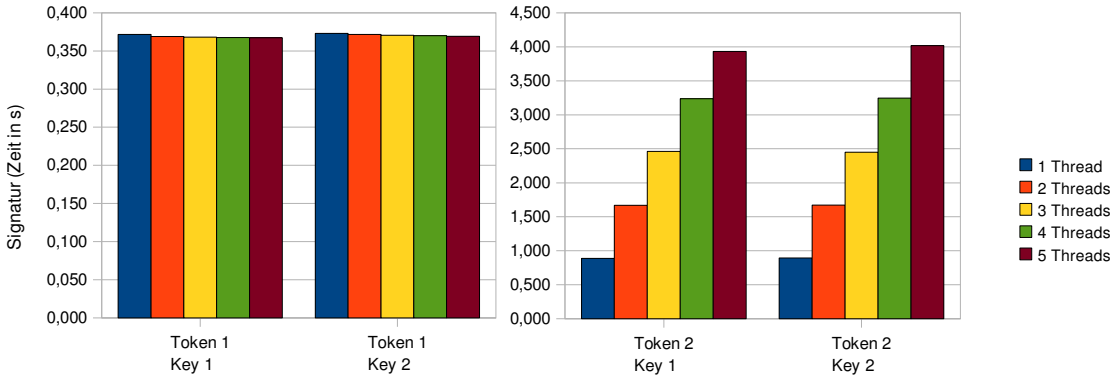



**Diagramm 8.4 —** Zeit für Signaturoperationen bei mehreren Threads

Laut Diagramm 8.4 verhalten sich die Token hierbei unterschiedlich. Während bei Token 1 keine wesentliche Änderung der Laufzeit entsteht - bei mehreren Threads kann sie sogar sinken - steigt sie bei Token 2 mit zunehmender Threadanzahl. Vermutlich ist Token 1 über die USB-Schnittstelle in der Lage, mehrere Operationen parallel auszuführen und Synerergieeffekte bei der Initialisierung der Operation zu nutzen. Bei Token 2, angeschlossen über ein Smartcard-Terminal, gibt es dagegen eine nahezu lineare Abhängigkeit der Laufzeit von der Anzahl der Threads.

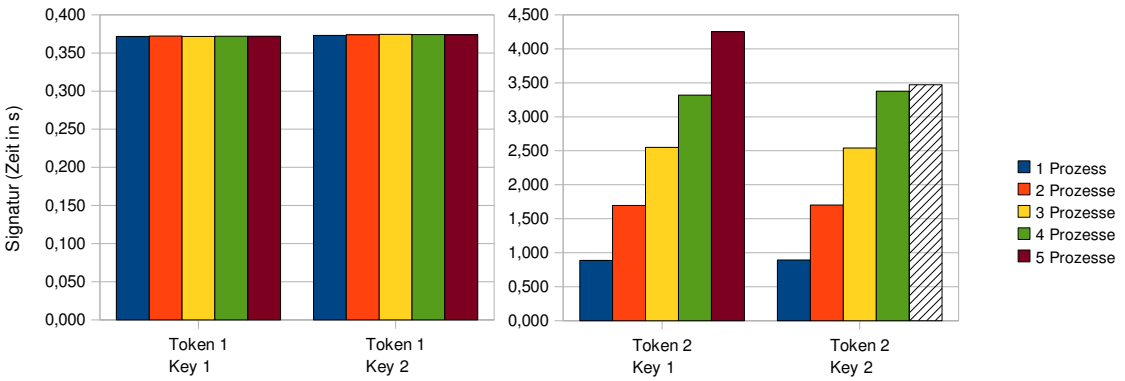



**Diagramm 8.5 —** Zeit für Signaturoperation bei mehreren Prozessen

Ähnlich verhält es sich bei mehreren Applikationen, dargestellt in Form unterschiedlicher Prozesse (Diagramm 8.5). Entsprechend der Spezifikation ist eine Trennung der Sitzungen erforderlich. Im Gegensatz zu Threads, die sich Objekte einer instanziierten Provider-Bibliothek „teilen“ können, agieren die Prozesse jeweils mit autarken Objekten. Das erfordert zusätzliche Operationen und kann damit eine weitere Steigerung der Laufzeit gegenüber Threads bedeuten. Während bei Token 1 kein signifikanter Anstieg messbar ist, kann bei Token 2 ein deutlicher Anstieg bemerkt werden, der bei parallel agierenden Prozessen noch etwas höher als beim Zugriff durch mehrere Threads ausfällt. Vermutlich kann Token 2 intern nicht parallel arbeiten und die Trennung der Objekte gewährleisten, so dass es über den PKCS#11-Provider im Basissystem erfolgen muss. Das erfordert neben Verwaltungsaufwand auch zusätzliche Kommunikation mit dem Token. Im Falle von Schlüssel 2 auf Token 2 kam es bei gleichzeitiger Nutzung durch mindestens 5 Prozesse sogar zu einem vorzeitigen Abbruch der Operation mit Fehler für einen der Prozesse.
Prinzipiell lassen sich Token aber von mehreren Prozessen oder Threads gleichzeitig nutzen. Für einige Token spielt dabei auch die parallele Nutzung bei der Ausführungszeit keine Rolle, während sie bei anderen Token mit der Anzahl gleichzeitiger Operationen ansteigt. Dieser Umstand käme während der Authentifizierung zum Tragen, sollten mehrere Applikationen gleichzeitig eine Signaturoperation erfordern. Im ungünstigsten Falle (ab 5 Applikationen) wäre die Authentifizierung zu wiederholen. Dieser Umstand sollte jedoch im Normalbetrieb eher selten auftreten und kann, z.B. im Falle eines Webbrowser durch erneuten Aufruf der Webseite behoben werden.

### 8.1.3 Auswertung des Ressourcenbedarfs

Bei den getesteten Smartcards mit RSA-Unterstützung und einer RSA-Modul-Größe von 2048 Bit belief sich die Zeit einer Signaturoperation auf dem SLE66CX322P Controller inklusive Kommunikation mit Token auf maximal 0,9 Sekunden. Für die schnelleren Aladdin-Token liegt sie bei 0,4 Sekunden. Während einer SSL-Authentifizierung kann der Nutzer somit eine Verzögerung von bis zu maximal einer Sekunde für den Verbindungsaufbau bemerken. Vergleicht man dies mit Passwörtern, so muss die Zeit zur Eingabe berücksichtigt werden, falls kein SSO-Mechanismus aktiv ist. Da die Eingabe von „nicht-trivialen“ Passwörtern meist länger als die Eingabe einer numerischen PIN für Token dauert, sollte die zusätzliche Token-Verzögerung kaum ins Gewicht fallen.
Falls die Möglichkeit besteht, sollten kryptographische Operation außerhalb des Tokens, auf dem schnelleren Prozessor des Rechners erfolgen. Bei einer verschlüsselten SSL-Verbindung müsste auf dem Tokens z.B. nur die Signatur zur Authentifizierung berechnet werden (Schritt 3 nach Abbildung 4.13). Jedoch delegieren Applikationen diverse Operationen an den PKCS#11-Provider. Dazu zählt auch die Berechnung von Digest-Werten. Im Anhang B ist dies erkennbar. Jeweils die erste Verbindung zu einem Mailordner des IMAP-Servers erfordert eine Digest-Operation (Schritt 163 und 391). Bei den getesteten Providern konnte dabei aber keine Kommunikation über das Terminal mit dem Token aufgezeichnet werden, so dass von einer Umsetzung innerhalb der PKCS#11-Providersoftware, also im Rechner und nicht im Token auszugehen ist. Für diese Operationen sollten sich keine Geschwindigkeitsnachteile ergeben.
Betrachtet man die Zeit für die SSO-Authentifizierung, so wäre das vorgestellte zertifikatsbasierte System mit den passwortbasierten SSO-Systemen zu vergleichen. Insgesamt hängen alle SSO-Lösungen von der Verzögerung im Rechnernetz und damit vom verwendeten Internetzugang und der Belastung des Netzes ab. Sie sind somit auch abhängig von Stoßzeiten. Bei einem Kerberos-SSO-Verbund ist der zusätzlich erforderliche Datenaustausch mit dem KDC und dessen Verarbeitungszeit zu beachten. Die Gesamtzeit hängt neben der Netzanbindung auch von der KDC-Belastung ab. Ansteigende Nutzerzahlen führen bei Kerberos zu Verzögerungen, während bei einer clientseitig arbeitenden

SSO-Lösung keine zusätzliche Belastung durch die Authentifizierung entsteht. Ein direkter Vergleich mit serverseitig agierenden SSO-Systemen ist aber schwer möglich, da er von zu vielen Faktoren abhängt. Eine Verzögerung von bis zu einer Sekunde ist aber auch im zu erwartenden Bereich für serverseitige Lösungen.
Auch gegenüber den getesteten clientseitigen passwortbasierten Tresor-Lösungen (*Aladdin SSO* und *v-GO SSO Logon*) ist kein Geschwindigkeitseinbruch bemerkbar. Der Grund hierfür liegt in der Arbeitsweise. Passwort-Tresor-Lösungen müssen darauf warten, dass eine Client-Applikation das Passwort-Fenster „erzeugt“ hat, bevor die Authentifizierung erfolgen kann. Das setzt meist die separate Übertragung einer speziellen Passwort-Seite voraus. Legt man Wert auf Sicherheit, so muss die Verbindung über einen gesicherten Tunnel hergestellt werden. Der Tunnel wiederum verursacht zusätzliche Übertragungszeit. Die Reaktionszeit von Tresor-Lösungen ist deshalb vergleichbar mit der des Hardware-Token-gestützten SSO-Systems, stellenweise sogar langsamer.

Zusammenfassend lässt sich sagen, Token-gestützte zertifikatsbasierte SSO-Authentifizierung benötigt nur wenige Ressourcen und stellt keine wesentliche Zusatzbelastung für das Clientsystem dar. Abgesehen von der etwas aufwendigeren Nutzer-Prüfung bei Applikationsstart ist die Geschwindigkeit im Bereich der Zeit anderer SSO-Lösungen. Die Verzögerungen sollten für Nutzer kaum störend bemerkbar sein, insofern nicht zu einem Zeitpunkt mehrere Authentifizierungen durch gleichzeitig agierende Applikationen erforderlich sind.

## 8.2 Sicherheitsaspekte

Würde die Einführung eines zertifikatsbasierten SSO-Systems die Sicherheit des Rechners oder des angeschlossenen Netzes nachteilig beeinflussen, so wäre ein wesentlicher Punkt der Anforderungen aus Kapitel 3.4 verletzt. Sicherheitsmängel könnten aufgrund der Umstellung von Passwörtern auf ein anderes Authentifizierungsverfahren oder der zusätzlichen Verwendung der clientbasierten SSO-Lösung entstehen
Die Absicherung der Authentifizierung ist im Vergleich zu passwortgestützten Verfahren zu betrachten. Dem Schutz des privaten Schlüssels ist ein besonderer Stellenwert beizumessen, da ein Schlüssel für viele Dienste nutzbar ist. Passwörter hätten den Vorteil, dass für die einzelnen Anwendungen verschiedene Passwörter gewählt werden könnten. Diese müssten allerdings auch qualitativ gut sein, was oftmals nicht der Fall ist. Ein direkter Vergleich fällt somit schwer.

### 8.2.1 Gefahrenquellen

Eine Angriffsmöglichkeit besteht in der Ausnutzung von Schwachstellen im zugrunde liegenden asymmetrischen Verfahren. Ein Angreifer kann im unsicheren Kanal übermittelte Information abfangen und analysieren. Die Information können verändert werden, um damit die Basis für einen Angriff auf das Authentisierungsmerkmal, den privaten Schlüssel des Tokens zu legen. Außerdem kann ein Angreifer versuchen, über das Token des rechtmäßigen Inhabers an den enthaltenen Schlüssel zu gelangen oder das Token selber zu missbrauchen.

#### Angriffe auf die Verschlüsselung

Die bei Passwortverfahren oftmals genutzten und häufig zum Erfolg führenden Wörterbuch-Angriffe stellen kein adäquates Werkzeug für den Angriff auf einen Schlüssel dar. Dies setzt voraus, dass die Schlüssel wirklich zufällig gewählt wurden. Ein qualitativ guter Zufallsgenerator muss deshalb bei der

Schlüsselerstellung verwendet werden. Andernfalls besteht die Gefahr, dass ein Schlüsselpaar „nachgeneriert“ wird. Außerdem sollten Zufallswerte oder Schlüsselteile nicht wiederverwendet werden.
Die betrachteten Applikationen nutzen zur Kommunikation hybride Verfahren. Ein asymmetrischen Verfahren dient zur wechselseitigen Authentifizierung mit Vereinbarung eines symmetrischen Schlüssels. Ein Angriff auf die Authentifizierung wird deshalb auf das zugrunde liegende asymmetrische Verfahren abzielen. Zur Beurteilung der Gefährdung bei zertifikatsbasierter Authentifizierung sind die Möglichkeiten der Kryptoanalyse für die eingesetzten asymmetrischer Verfahren zu betrachten. Der öffentliche Schlüssel ist frei verfügbar und kann unbemerkt für einen Angriff herangezogen werden.

- **Exhaustive Search:** Prinzipiell ist eine erschöpfende Suche des privaten Schlüssels im Schlüsselraum denkbar, was aber bei hinreichend großer Schlüssellänge unter Nutzung aktuell verfügbarer Methoden praktisch unmöglich ist.

- **Chosen-Ciphertext:** Dieser Angriff entspricht der im Kapitel 7.3.2 für das RSA-Verfahren gezeigten Methode, bei der ein gewählter Kryptotext dem Opfer vorgelegt wird. Gegen den gezeigten Angriff helfen getrennte Schlüsselpaare für Signatur und Verschlüsselung. Für die Authentifizierung führt dieser Angriff nicht zum Erfolg, da hierbei nur ein Rückschluss auf Challenge-Werte, aber nicht auf den Schlüssel möglich ist. Ansonsten gilt das RSA-Verfahren als robust gegen Chosen-Ciphertext-Attacken.

- **Known-Plaintext:** Für diesen Angriff müssen Klartext und dessen Verschlüsselung bekannt sein. Basis sind mitgeschnittene Challenge- und Response-Übertragungen. Da die Response eine Signatur enthält, also mit dem privaten Schlüssel erzeugt wurde, gibt es hier eine Verbindung zum bekannten Klartext. Die als Antwort übertragenen Datenmengen sind meist sehr gering, z.B. für SSL/TLS nach Abbildung 4.13 ist dies ein signierte Hashwert. Dem Angreifer liegt somit kaum Material für einen Angriff vor. Ohne eine hinreichende Menge an Vergleichsdaten dürfte sich ein erfolgreicher Angriff als schwierig erweisen [Eck06]. Durch das Padding fließen außerdem zufällige Werte ein, die eine systematische Analyse erschweren.

- **Chosen-Plaintext:** Hier versucht der Angreifer für einen von ihm gewählten Klartext eine gültige Verschlüsselung bzw. Signatur zu erhalten, die er seinerseits gegenüber dem Server präsentieren oder zur Analyse des Schlüssels verwenden kann. Dies setzt einen Man-In-The-Middle-Angriff (siehe unten) voraus. Verwenden die Kommunikatiosnpartner nonce-Werte, bleibt dieser Angriff wirkungslos.

Neben dem Angriff auf das asymmetrische Verfahren kann die Authentifizierung auch über den Eingriff in die Kommunikation zwischen Client und Server kompromittiert werden.

- **Spoofing by Counterfeit Servers:** In diesem Falle gibt sich der Angreifer als Serverdienst aus und leitet den Netzwerverkehr zu sich selbst um. Hier liefert aber eine Prüfung des Server-Zertifikats durch den Client Abhilfe, insofern der Angreifer nicht in den Besitz des gültigen Server-Schlüssels gelangen kann.

- **Replay Attack:** Dieser Angriff basiert auf einer erneuten Übermittlung abgefangener Übertragungen an den Server. Werden nonce-Werte in der Challenge verwendet, so wird auch diesem Angriff die Basis entzogen.

- **Man-in-the-Middle (MITM):** Dieser bereits mehrfach erwähnte Angriff zielt darauf ab, sich zwischen die Kommunikation des Nutzers und des Dienstes zu schalten. Wie in Kapitel 2.3

vorgestellt, kann als Abhilfe mit Zertifikaten die Authentizität des Kommunikationspartners geprüft werden.

Zertifikate stellen den wesentlichen Schutz vor Eingriffen in die Kommunikation dar. Für die Sicherheit des Nutzers ist deshalb wichtig, die Gültigkeit der Server-Zertifikate zu prüfen. Da genau dieser Punkt meist nicht beachtet wird [CCR09], bildet MITM die Grundlage vieler weiterer Angriffe.
Bei Einhaltung bestimmter Regeln, zu denen die Zertifikatsprüfung gehört, bieten die verwendeten asymmetrischen Verfahren in Verbindung mit Zertifikaten einen guten Schutz vor Angriffen. Der Einsatz sollte die Sicherheit kaum nachteilig beeinflussen. Im Gegenteil, im Vergleich zu Passwort-Verfahren, bei denen Klartext-Passwörter in einem kryptographisch gesicherten Tunnel übertragen und somit bei Brechen der Verschlüsselung abgefangen und kopiert werden können, bieten zertifikatsbasierte Client-Authentifizierungen weniger Angriffsfläche. Selbst bei der erfolgreichen Entschlüsselung einer einzelnen Übertragung wird nicht der private Schlüssel, sondern lediglich eine Information extrahiert, die für eine erneute Authentifizierung nicht genutzt werden kann.

**Kompromittierung des privaten Schlüssels**

Ein wesentliches Ziel für den Angreifer wäre demzufolge die Entwendung oder die Ermittlung des privaten Schlüssels. Bei der Umstellung auf zertifikatsbasierte Authentifizierung muss deshalb der Schutz des privaten Schlüssels gewährleistet sein. Dies muss auch bei Einführung des SSO-Systems beachtet werden. Der Schlüssel darf sich nur in den Händen des Inhabers befinden und sollte selbst für diesen nicht kopierbar sein.
Die Hardware-Crypto-Token wurden für diesen Zweck ausgelegt. Die zur Kompromittierung von Passwörtern oftmals eingesetzten Methoden des **Social-Engineerings** führen bei Token nicht zum Erfolg. Mittels Social-Engineering versucht ein Angreifer das Opfer zur Preisgabe des Passwortes zu verleiten. **Phishing**-Angriffe[5] sind eine Form des ungezielten Social Engineerings. Sie verführen unzureichend informierte oder sorglose Nutzer zur Übermittlung von Passwörtern oder PIN/TAN-Kombinationen [Lip09]. Private Schlüssel in Hardware-Crypto-Token können aber vom Nutzer nicht einfach ausgelesen werden. Ziel des Angreifers kann also nur die Entwendung des Tokens oder ein Fernzugriff über das Netz auf ein angeschlossenes Token sein. Aber selbst dafür bieten Token einen gewissen Schutz. Ein unbefugter Nutzer hat nur eine begrenzte Anzahl an Versuchen zur Eingabe einer gültigen PIN. Andernfalls bleibt das Token für weitere Versuche gesperrt.
Bei Token-Verlust kann der Inhaber das Zertifikat sperren lassen. Eine Authentifizierung mit dem Token ist dann nicht mehr möglich. Somit muss ein Angreifer ein gestohlenes Token kopieren, bevor der Inhaber den Verlust bemerkt. Dazu wäre eine Extraktion des privaten Schlüssels aus dem Token notwendig. Der Angreifer muss also die Schutzmechanismen des Tokens überwinden. Kommt der Angreifer in den Besitz der PIN oder ist er selber der Inhaber des Tokens, so kann er die bereits erwähnten **Seitenkanalangriffe** anwenden. Mit einer hinreichend großen Testmenge und unter Anwendung stochastischer Methoden ist ggf. ein Rückschluss auf den Schlüssel möglich [SLP05]. Moderne Token werden auf diese Angriffs-Möglichkeiten getestet und dagegen gehärtet (siehe Blinding in Kapitel 8.1.2). Außerdem kann die Anzahl der Operationen pro Zeiteinheit vom Token eingeschränkt sein und somit einem Angreifer nicht genügend Spielraum geben.
Der erfolgreiche Angriff auf gehärtete Token erfordert enormen Aufwand. Gehärtete Token werden für finanzielle Transaktionen (HBCI) vom Zentralen-Kreditausschuss als sicheres Medium empfohlen. Somit sollten sie auch eine adäquate Sicherheit im Hochschulbereich gewährleisten können. Der

[5] Phishing: Kunstwort, welches das „Abfischen“ von Passwörtern oder anderen sensiblen Informationen mit den Methoden des Social Engineerings, meist unter Nutzung der Massenkommunikation mittels Email beschreibt.

Angreifer steht außerdem unter Zeitdruck, da der Verlust des Tokens gemeldet und das Zertifikat gesperrt werden könnte[6]. Das sollte den Anreiz eines teuren und zeitraubenden Angriffs schmälern.

### Schutz der PIN

Um den Missbrauch entwendeter Token zu verhindern und Angriffen auf Token entgegen zu wirken, muss die PIN sicher verwahrt werden. Die PIN ist gefährdet, wenn Token in einem kompromittierten Umfeld, z.B. einem PC mit installiertem *Keylogger* genutzt werden. Dies setzt natürlich voraus, dass die PIN über die PC-Tastatur und nicht über einen Klasse-2-Reader direkt am Token eingegeben wird. Der SSO-Agenten erhöht die Gefahr, da der Angreifer die PIN auch aus dem Speicher extrahieren kann.
Aufgrund der gewählten Implementierung (Kapitel 7.1.1) lässt sich ein Angriff nur dann umsetzen, wenn der Angreifer über privilegierte Administrator-Rechte verfügt oder direkt im Anschluss an den Nutzer Zugriff auf den verwendeten Rechner hat, um den Speicherinhalt auszulesen und zu analysieren. Schutz vor einem Angriff durch einen Administrator ist verhältnismäßig schwer. Selbst bei Verwendung von Smartcards mit einem Klasse-2-Reader müsste sich der Nutzer vorher von dessen Unversehrtheit überzeugen, was praktisch kaum möglich ist. Vor anderen Angriffen sollte die PIN zur Laufzeit gut geschützt sein. Voraussetzung dafür ist eine fehlerfreie Implementierung der genutzten Sicherungsmaßnahmen des Betriebssystems. Schutz gegen ein nachträgliches Auslesen der PIN aus dem Speicher liefert ein Mechanismus, der bei Entnahme des Tokens die PIN im Hauptspeicher durch Überschreiben löscht. Achten die Nutzer darauf, ihre Token bei Verlassen des Systems zu entfernen und die Ausführung des PIN-Lösch-Mechanismus nicht durch ein Hardware-Reset zu verhindern, so kann der Angreifer die PIN im Anschluss nicht auslesen.
Ein Vorteil für das SSO-System ist die Tatsache, dass die PIN lediglich einmalig einzugeben ist. Ein Angreifer, der die Eingaben des Nutzers auf der Tastatur visuell beobachtet, hat somit während einer Sitzung nur einmalig beim Login eine Chance, die richtige PIN zu erkennen. Die nur einmalig notwendige Eingabe kann auch dazu führen, dass Nutzer längere PIN-Kombinationen wählen, was die Schwierigkeit für den Angreifer weiter erhöht.

### Softwarefehler

Eine weitere mögliche Gefahrenquelle sind Softwarefehler. Hier wäre zwischen Fehlern im Betriebssystem und Fehlern in der Implementierung der Client-Software zu unterscheiden.
Fehler im Betriebssystem beziehen sich darauf, dass zugesagte Eigenschaften, wie der Schutz eines angeforderten Speicherbereichs vor Auslagerung auf Festplatte, nicht eingehalten werden. Diese müssen nicht zwangsläufig auf eine fehlerhafte Implementierung zurückzuführen sein, sondern können durch eingeschleuste Schadprogramme (Viren, Trojaner) mit administrativen Rechten entstehen. Ein Betriebssystem ohne Schädlingsbefall ist also Voraussetzung für die Sicherung der PIN. Hier gilt wiederholt, bei Passwörtern wäre die Gefährdung höher, da der Angreifer das Authentisierungsmerkmal erhält. Bei der Verwendung von Token erhält er lediglich die PIN aus dem Speicher und muss weitere Maßnahmen zur Vervollständigung des Angriffs unternehmen.
Ein Fehler in der Client-Software, der das Ziel eines Angriffs zum Erhalt der PIN sein kann, wäre in drei Bereichen zu suchen: Speicherung der PIN, Überprüfung der Berechtigung eines Programms vor PIN-Übergabe, und Verwendung der PIN in der PKCS#11-Proxy-Bibliothek. Die ersten beiden Punkte sind Aufgaben des Agenten. Bei der Implementierung müssen Fehler bei der Interaktion zwischen Agent und Bibliothek vermieden werden. Zugriffe von fremden, nicht zum SSO-System gehörenden

[6] Hier wird davon ausgegangen, dass die Nutzer ihre eigenen Token nicht kompromittieren und sich damit selber schaden.

Komponenten müssen erkannt und die PIN-Übermittlung verhindert werden. Eine aussagekräftige Fehlermeldung zur Warnung des Nutzers wäre in diesem Fall sinnvoll. Da nur Programme des Login-Nutzers zur Kommunikation mit dem Agenten zugelassen sind (vorausgesetzt die Prüfung durch die Betriebssystemfunktion ist korrekt), sind Angriffe durch Fremdprogramme nur dann möglich, wenn sie vom Nutzer z.B. nach erfolgreicher Überredung (Social Engineering) zur Ausführung gebracht werden. Eine Einschränkung des Agenten auf eine Auswahl vorinstallierter Programme würde einen gewissen Schutz bieten.
Zeiträume zur unverschlüsselten Ablage der PIN sind zu minimieren. Dies verringert die Wahrscheinlichkeit, dass aufgrund eines Programmfehlers einem Angreifer sensible Speicherbereiche ungeschützt zur Verfügung stehen. Innerhalb der Proxy-Bibliothek ist die PIN nach der Übertragung vom Agenten sofort an das Token zu übermitteln und im Speicher durch Überschreiben zu löschen.
Bei der Beispiel-Implementierung wurde zwar auf Schutz der PIN geachtet, prinzipiell lässt sich aber nicht ausschließen, dass Softwarefehler in den Quellen, dem verwendeten Compiler und den Zusatzbibliotheken enthalten sind, die letztendlich doch zur Kompromittierung der PIN führen können. Eine formale Prüfung der Software auf Einhaltung der angegebenen Schutzkriterien wäre eine Möglichkeit, diese Unsicherheit zu beseitigen. Die Wahrscheinlichkeit, dass diese von einem Angreifer gezielt ausgenutzt werden, kann jedoch als gering eingestuft werden. Dies belegen Erfahrungen mit dem vielfach eingesetzten SSH-Agenten.

### 8.2.2 Abschätzung der Gefährdung des Gesamtsystems

Bei dem SSO-System handelt es sich um eine Software, welche die Komplexität des Clientsystems erhöht. Dadurch können zusätzliche Fehlerquellen entstehen, die gewisse Unsicherheitsfaktoren darstellen. Das SSO-System zielt auf Ablösung bestehender passwortgestützter Systeme ab und deshalb sind alle Faktoren im Vergleich zu passwortbasierten Verfahren zu betrachten.
Hier offenbaren sich Stärken bei der zertifikatsbasierten Authentifizierung mit Hardware-Crypto-Token. Insbesondere hervorzuheben wäre die relativ sichere Aufbewahrung der Authentisierungsmerkmale in den Händen der Nutzer. Im Gegensatz zu Passwörtern bietet sie selbst bei sorglosem Umgang guten Schutz. Erst die gleichzeitige Entwendung von Token und PIN stellt eine Gefahr dar. Der Einsatz des SSO-Systems erhöht die Gefahr der PIN-Entwendung. Im Gegensatz zu Passwörtern reicht jedoch die PIN allein nicht aus.
Zertifikatsbasierte Authentifizierung kann im Vergleich mit Passwortverfahren als sicher eingestuft werden. Sie bietet bei guter Schlüsselwahl mit hinreichender Schlüssellänge in Verbindung mit Zertifikatsprüfung einen guten Schutz vor Angriffen.
Einen messbaren Wert und somit eine quantitative Aussage über die Sicherheit des SSO-System lässt sich nicht geben. Qualitativ gesehen kann es, abgesehen von der möglicherweise besseren Wahl der PIN bei Einsatz des SSO-Systems, nicht die gleiche Sicherheit bieten, die eine Umstellung auf zertifikatsbasierte Authentifizierung mit Crypto-Token ohne Nutzung eines SSO-Agenten bringen würde. Dies gilt aber nur für den Fall, dass dort die PIN bei jeder Authentifizierung direkt am Smartcard-Terminal bzw. am Token eingegeben und nicht über den Rechner zum Token weiter geleitet wird. Weitere Erkenntnisse, die mögliche Schwachstellen offenbaren, wird der zukünftige Einsatz des Systems außerhalb des bisherigen Testumfelds liefern. Da keine systematischen Schwachstellen erwartet werden, sollten sich erkannte Probleme beheben lassen.
Insgesamt gesehen kann von einer merklichen Verbesserung gegenüber Passwortverfahren ausgegangen werden. Dies gilt insbesondere aus Sicht der Dienst-Betreiber, da mit den Token Authentisierungsmerkmale genutzt werden, die einen guten Schutz vor Entwendung oder gezielter Vervielfachung bieten.

# Kapitel 9

# Zusammenfassung und Ausblick

## 9.1 Zusammenfassung

Mit dieser Arbeit wurde das Ziel verfolgt, die Daten- und Informationssicherheit netzgestützter Systeme im Hochschulbereich zu verbessern. Der vereinfachten Einsatz kryptographischer Verfahren in den Händen der Nutzer soll dazu beitragen. Um dieses Ziel zu erreichen, wurde ein Teilgebiet aus dem Bereich der IT-Sicherheit genauer betrachtet, das Problem einer „sicheren“ Nutzerauthentifizierung. Die Authentifizierung mit Zertifikaten unter Einbindung von Hardware-Token bietet sich an, wie in den Kapiteln 4 und 5 gezeigt wurde. Die Umstellung auf dieses sichere und robuster gegenüber Fehlern der Nutzer arbeitende Verfahren erkauft man sich durch den Umstand einer besitzbasierten Authentifizierung, deren Konsequenz die erforderliche Mitführung des Tokens ist.
Für den Nutzer sind deshalb Anreize zur Nutzung der Hardware-Token zu schaffen, die nicht allein in einer Anhebung des Sicherheitsniveaus begründet sein dürfen. Vielmehr sollte sich eine Erleichterung für den Nutzer gegenüber den aktuell eingesetzten passwortbasierten Authentifizierungsverfahren einstellen. Die Verbindung der Authentifizierung mit Single Sign-On und die Nutzung der Token für Signatur und Verschlüsselung sollten genügend Anreize schaffen können. Um die breite Palette an Token, Applikationen und aktuellen Betriebssystemen zu unterstützen, muss das SSO-System auf gängigen Schnittstellen basieren. Im Kapitel 5 wurde gezeigt, dass sich hierfür allein die PKCS#11-Schnittstelle eignet.
Das Hinzufügen der SSO-Funktionalität wurde durch eine clientseitige Erweiterung der Tokenschnittstelle unter Beibehaltung der PKCS#11-Kompatibilität erreicht. Durch die Hinzunahme einer Agentsoftware mit einer Proxy-Bibliothek konnte die gewünschte SSO-Funktionalität ohne Eingriff in bestehende Schnittstelle realisiert werden. Technische Maßnahmen verhindern die wiederholte PIN-Eingabe, wodurch sich Single Sign-On aus Sicht des Anwenders ergibt, der sich nur einmalig gegenüber dem Token authentifizieren muss.
Die Implementierung eines Testsystems erwies sich als unproblematisch und lieferte die ersten Ergebnisse im realen Einsatz. Die Funktionalität der erhaltenen SSO-Lösung konnte für eine breite Palette an Software praktisch erprobt werden. Notwendige Umstellungen beschränken sich, wie im Kapitel 7 gezeigt wurde, meist auf Anpassung der Konfiguration oder einer Integration frei verfügbarer Plugins. Das Testsystem lieferte messbare Ergebnisse zur Geschwindigkeit und dem zusätzlichen Ressourcenverbrauch. Die ermittelten Werte zeigen zwar eine spürbare, aber im Vergleich mit Passwörtern für die Authentifizierung ausreichende Reaktionszeit. Der zusätzlicher Speicher- und Rechenzeitbedarf fällt hingegen für aktuelle Computersysteme kaum ins Gewicht.
Eine quantitative Aussage und damit ein messbarer Wert der Sicherheit lässt sich schwer geben. Eben-

so ist ein Sicherheitsvergleich zu anderen SSO-Verfahren schwer möglich, da hierbei Annahmen über das reale Nutzerverhalten einfließen würden. Im Vergleich zu einer rein passwortgestützten Authentifizierung lässt die bestehende Umsetzung aber folgende Aussage zu. Mit Ersetzung passwortbasierter Verfahren durch eine SSO-Lösung mit Hardware-Token wird ein Sicherheitsgewinn, sowohl aus Sicht der Anwender, als auch aus Sicht der Betreiber netzgestützter Dienste erzielt. Gleichzeitig wird dem Nutzer mit Crypto-Token ein Werkzeug in die Hand gegeben, welches ihm weitere Optionen zur Steigerung der Sicherheit durch Nutzung kryptographischer Verfahren eröffnet.
Derart untermauert und im Testsystem praktisch erprobt lässt die Arbeit den Schluss zu, dass die Umstellung auf die zertifikatsbasierte Authentifizierung und die Nutzung von Hardware-Token in Verbindung mit der SSO-Funktionalität bestehende passwortbasierte Systeme ablösen kann und gleichzeitig einen praktischen Beitrag zur Verbesserung der Sicherheit im vernetzten Umfeld und somit für den Hochschuleinsatz bietet.

## 9.2 Ausblick

Sicherheit ist ein evolutionärer Prozess, bei dem es gilt, bekannte Maßnahmen zu deren Verbesserung zu ergreifen und erkannte Schwachstellen zu beseitigen. Für das SSO-System bedeutet es, dass die SSO-Software und mit ihr die PKCS#11-Provider, Interface-Treiber und letztendlich sogar die Hardware auf mögliche sicherheitsrelevante Fehler zu prüfen sind. Hierzu kann die SSO-Software einer formalen Prüfung auf Einhaltung bestimmter Bedingungen zur Gewährleistung der Sicherheit unterzogen werden. Um belastbare Aussagen zu treffen, muss auch das Betriebssystem einbezogen werden, was praktisch nicht möglich ist. Ebenso müssen Compilerfehler berücksichtigt werden. Sinnvoller erscheint es, das System einer regelmäßigen Prüfung durch mehrere, voneinander unabhängige Personen und einer Reihe von automatisierten Tests zur möglichen Erkennung von Schwachstellen zu unterziehen. So könnte, analog zu den etablierten kryptographischen Verfahren, für die ebenfalls kein formaler Beweis der Sicherheit gegeben werden kann, trotzdem ein akzeptables Maß an Vertrauenswürdigkeit erreicht werden.

Das SSO-System wurde exemplarisch für das Betriebssystem Linux aufgebaut und in einem abgegrenzten Subnetzes praktisch an einer Reihe von Diensten sowie für Netzwerk- und lokale Rechner-Anmeldung erprobt. Dieser Test ist auf einen größeren Bereich auszuweiten. Neue Erkenntnisse und weitere Einsatzmöglichkeiten könnte der Einsatz in den Computerpools und die private Verwendung der Token durch eine Vielzahl von Anwendern liefern. Ziel sollte es sein, den Verwaltungsaufwand zu minimieren und notwendige organisatorische Abläufe von der Token-Ausgabe über die Zertifikatsausstellung und -erneuerung bis hin zum Rückruf von Zertifikaten möglichst effizient zu gestalten.

Zukünftige Entwicklung betreffen die Verbesserung des Umgangs mit dem Kartensystem. PKCS#11 unterstützt zwei PINs und zwei Nutzerklassen, den SO- und den User-Status, jeweils über die PIN definiert. Für den praktischen Einsatz ist aber eine drei-PIN Lösung, vergleichbar mit SIM-Karten von Mobiltelefonen, besser geeignet. Dadurch können dem Nutzer PIN und PUK zur Verfügung gestellt werden (siehe Kapitel 5.3.1). Unter Beibehaltung der PKCS#11-Kompatibilität kann dies durch eine eigene Kartenstruktur auf PKCS#15- bzw. ISO-7816-15-Basis oder über ein speziell angepasstes Java-Applet für Java-Cards erreicht werden.
Gegenstand weiterer Untersuchungen wäre die Einbindung kontaktloser Karten, z.B. Mifare ProX, SmartMX oder DESfire. Ein hybrides System könnte den Vorteil einer berührungslosen Authentifizierung, mit der Nutzung von Zertifikaten verbinden. Diese würde sich bei Zutritts-Kontrollsystemen anbieten. Weiterhin käme die Nutzung der zertifikatsbasierten Authentifizierung in Verbindung mit der Biometrie in Frage. Biometrische Merkmale könnten die Token-PIN ersetzen.

# Literaturverzeichnis


[AKS04] AGRAWAL, MANINDRA, NEERAJ KAYAL und NITIN SAXENA. *PRIMES is in $P$*. Annals of Mathematics, 160(2):781–793, 2004.

[AL02] ADAMS, CARLISLE und STEVE LLOYD. *Understanding PKI*. Addison-Wesley Publishing Company, 2. Auflage, 2002.

[And08] ANDERSON, ROSS. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley & Sons, 2008. URL: `http://www.cl.cam.ac.uk/~rja14/book.html`.

[BD00] BIHAM, ELI und ORR DUNKELMAN. *Cryptanalysis of the A5/1 GSM stream cipher*. In: *Progress in cryptology - INDOCRYPT 2000. 1st international conference in cryptology in India, Calcutta*, Band Lecture Notes Computer Science 1977, Seiten 43–51. Springer, 2000.

[BK01] BUCKBESCH, JÖRG und ROLF-DIETER KÖHLER. *VPN - virtuelle private Netze : sichere Unternehmenskommunikation in IP-Netzen*. FOSSIL-Verl., 1. Auflage, 2001. URL: `http://gso.gbv.de/DB=2.1/CMD?ACT=SRCHA&SRT=YOP&IKT=1016&TRM=ppn+33062024X&sourceid=fbw_bibsonomy`.

[Ble98] BLEICHENBACHER, DANIEL. *Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1*. In: KRAWCZYK, HUGO (Herausgeber): *CRYPTO*, Band 1462 der Reihe *Lecture Notes in Computer Science*, Seiten 1–12. Springer, 1998. URL: `http://dblp.uni-trier.de/db/conf/crypto/crypto98.html#Bleichenbacher98`.

[BNA08] *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung*. Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen (BNetzA), Veröffentlicht am 27.1.2009 im Bundesanzeiger Nr. 14, Seite 346, November 2008. URL: `http://www.bundesnetzagentur.de/media/archive/15549.pdf`.

[Bre05] BREYER, PATRICK. *Die systematische Aufzeichnung und Vorhaltung von Telekommunikations-Verkehrsdaten für staatliche Zwecke in Deutschland*. Rhombos-Verlag, Berlin, 2005. URL: `http://publikationen.ub.uni-frankfurt.de/volltexte/2005/500/pdf/BreyerPatrick.pdf;http://publikationen.ub.uni-frankfurt.de/volltexte/2005/500/index.html;http://deposit.ddb.de/cgi-bin/dokserv?idn=973937076`.

[BSN05] BEUTELSPACHER, ALBRECHT, THOMAS SCHWARZPAUL und HEIKE B. NEUMANN. *Kryptografie in Theorie und Praxis*. Vieweg+Teubner, 1 Auflage, 2005.

[BSW06] BEUTELSPACHER, ALBRECHT, JÖRG SCHWENK und KLAUS-DIETER WOLFENSTETTER. *Moderne Verfahren der Kryptographie. Von RSA zu Zero-Knowledge*. Vieweg, 2006.

[CCR09] CALLEGATI, FRANCO, WALTER CERRONI und MARCO RAMILLI. *Man-in-the-Middle Attack to the HTTPS Protocol*. IEEE Security & Privacy, 7(1):78 ff., Januar 2009.

[Cha06] CHALOUPKA, ALEXANDER. *Inside Smartcards - Der steinige Weg zur Interoperabilität*. IT Security, 3/2006:44 ff., Mai 2006.

[CNO08] COURTOIS, NICOLAS T., KARSTEN NOHL und SEAN O'NEIL. *Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards*. Cryptology ePrint Archive, (166), 2008. `http://eprint.iacr.org/`.

[Com00] COMPAQ, HEWLETT-PACKARD, INTEL, LUCENT, MICROSOFT, NEC, AND PHILIPS. *Universal Serial Bus Revision 2.0 specification*, 2.0 Auflage, April 2000. URL: `http://www.usb.org/developers/docs/`.

[dKGHG08] KONING GANS, GERHARD DE, JAAP-HENK HOEPMAN und FLAVIO D. GARCIA. *A Practical Attack on the MIFARE Classic.* In: GRIMAUD, GILLES und FRANÇOIS-XAVIER STANDAERT (Herausgeber): *CARDIS*, Band 5189 der Reihe *Lecture Notes in Computer Science*, Seiten 267–282. Springer, 2008. URL: `http://dblp.uni-trier.de/db/conf/cardis/cardis2008.html#GansHG08`.

[DR02] DAEMEN, JOAN und VINCENT RIJMEN. *The Design of Rijndael: AES - The Advanced Encryption Standard (Information Security and Cryptography)*. Springer, 1 Auflage, 2002.

[DSS09] NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *Digital Signature Standard (DSS), 3rd. Revision*. FIPS Publication 186-3, Juni 2009. URL: `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`.

[Eck06] ECKERT, CLAUDIA. *IT-Sicherheit*. Oldenbourg Wissenschaftsverlag, München, 4. überarb. Auflage, 2006.

[EKM+08] EISENBARTH, THOMAS, TIMO KASPER, AMIR MORADI, CHRISTOF PAAR, MAHMOUD SALMASIZADEH und MOHAMMAD T. MANZURI SHALMANI. *On the Power of Power Analysis in the Real World: A Complete Break of the KeeLoqCode Hopping Scheme.* In: WAGNER, DAVID (Herausgeber): *CRYPTO*, Band 5157 der Reihe *Lecture Notes in Computer Science*, Seiten 203–220. Springer, 2008. URL: `http://dblp.uni-trier.de/db/conf/crypto/crypto2008.html#EisenbarthKMPSS08`.

[Elg85] ELGAMAL, TAHER. *A public key cryptosystem and a signature scheme based on discrete logarithms.* IEEE Trans. Inf. Theory, 31:469–472, 1985.

[ES05] ENDRES, JOHANNES und PETER SIERING. *PC-Fernsteuerung nicht nur mit Remote Desktop*. c't Magazin für Computer Technik, (10):102, 2005.

[Fon05] FONTAINE, CAROLINE. *RC4*. In: VAN TILBORG, HENK C. A. (Herausgeber): *Encyclopedia of Cryptography and Security*. Springer, 2005. URL: http://dblp.uni-trier.de/db/reference/crypt/crypt2005.html#Fontaine05d.

[Gar96] GARFINKEL, SIMSON. *PGP: Pretty good privacy*. O'Reilly, International Thomson Verlag, Bonn, 1996.

[Gar01] GARFINKEL, SIMSON. *Database nation: the death of privacy in the 21st century*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[Han05] HANDSCHUH, HELENA. *RC6*. In: VAN TILBORG, HENK C. A. (Herausgeber): *Encyclopedia of Cryptography and Security*. Springer, 2005. URL: http://dblp.uni-trier.de/db/reference/crypt/crypt2005.html#Handschuh05a.

[HJK08] HARDING, PATRICK, LEIF JOHANSSON und NATE KLINGENSTEIN. *Dynamic Security Assertion Markup Language: Simplifying Single Sign-On.* IEEE Security & Privacy, 6(2):83–85, 2008. URL: http://dblp.uni-trier.de/db/journals/ieeesp/ieeesp6.html#HardingJK08.

[HKN+09] HOLE, KJELL J., ANDRÉ N. KLINGSHEIM, LARS-HELGE NETLAND, YNGVE ESPELID, THOMAS TJØSTHEIM und VEBJØRN MOEN. *Risk Assessment of a National Security Infrastructure*. IEEE Security & Privacy, 7(2):34 ff., Januar 2009.

[HSH+08] HALDERMAN, J. ALEX, SETH D. SCHOEN, NADIA HENINGER, WILLIAM CLARKSON, WILLIAM PAUL, JOSEPH A. CALANDRINO, ARIEL J. FELDMAN, JACOB APPELBAUM und EDWARD W. FELTEN. *Lest We Remember: Cold Boot Attacks on Encryption Keys.* In: *USENIX Security Symposium*, Seiten 45–60. USENIX Association, 2008. URL: http://dblp.uni-trier.de/db/conf/uss/uss2008.html#HaldermanSHCPCFAF08.

[IEE03] *IEEE standards for local and metropolitan area networks. Virtual bridged local area networks*. IEEE Std 802.11Q-2003, 2003.

[IEE04a] *IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, Amendment 6: Medium Access Control (MAC) Security Enhancements*. IEEE Std 802.11i-2004, 2004.

[IEE04b] *IEEE Standard for Information technology-Port Based Network Access Control*. IEEE Std 802.11X-2004, 2004.

[IEE07] *IEEE Standard for Information technology-Telecommunications and information exchange between systems-Local and metropolitan area networks-Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Std. 802.11-2007 (Revision of IEEE Std 802.11-1999), Seiten C1–1184, Dezember 2007.

[Inf02a] INFINEON TECHNOLOGIES. *Security & Chip Card ICs SLE 66C322P*, November 2002. 16-Bit Security Controller with Memory Management and Protection Unit.

[Inf02b] INFINEON TECHNOLOGIES. *Security & Chip Card ICs SLE 66C642P*, November 2002. 16-Bit Security Controller with Memory Management and Protection Unit.

[ISO14443] *ISO/IEC 14443, Part 1-4, Identification cards - Contactless integrated circuit cards - Proximity cards*, 2008.

[ISO7810] *ISO/IEC 7810:2003 Identification cards – Physical characteristics*, 2003.

[ISO7816/01] *ISO/IEC 7816-1, Identification cards - Integrated circuit(s) cards with contacts - Part 1: Physical characteristics*, 1998.

[ISO7816/02] *ISO/IEC 7816-2, Identification cards - Integrated circuit(s) cards with contacts - Part 2: Dimensions and location of the contacts*, 2007.

[ISO7816/03] *ISO/IEC 7816-3, Identification cards - Integrated circuit(s) cards with contacts - Part 3: Electrical interface and transmission protocols*, 2006.

[ISO7816/04] *ISO/IEC 7816-4, Identification cards - Integrated circuit(s) cards with contacts - Part 4: Organization, security and commands for interchange*, 2005.

[ISO7816/08] *ISO/IEC 7816-8, Identification cards - Integrated circuit(s) cards with contacts - Part 8: Security related interindustry commands*, 1999.

[ISO7816/12] *ISO/IEC 7816-12, Identification cards - Integrated circuit(s) cards with contacts - Part 12: USB electrical interface and operating procedures*, 2005.

[ISO7816/15] *ISO/IEC 7816-15, Identification cards - Integrated circuit(s) cards with contacts - Part 15: Cryptographic information application*, 2004.

[JMV01] JOHNSON, DON, ALFRED MENEZES und SCOTT A. VANSTONE. *The Elliptic Curve Digital Signature Algorithm (ECDSA).* Int. J. Inf. Sec., 1(1):36–63, 2001. URL: `http://dblp.uni-trier.de/db/journals/ijisec/ijisec1.html#JohnsonMV01`.

[Ker83] KERCKHOFFS, AUGUSTE. *La cryptographie militaire*. Journal des sciences militaires, IX:Seiten 5–38 (Januar), 161–191 (Februar), 1883.

[KR07] KUROSE, JAMES F. und KEITH W. ROSS. *Computer Networking: A Top-Down Approach (4th Edition).* Addison Wesley, 4 Auflage, 2007.

[Kri07] KRISSLER, JAN. *Feine Linien - Wie leicht sich Fingerabdrucksensoren austricksen lassen*. c't Magazin für Computer Technik, (12):102, 2007.

[KVK06] *Technische Spezifikation der Versichertenkarte*, Oktober 2006. URL: `http://www.vdak.de/vertragspartner/Telematik/download/techn_spezifik_vkarte_20061030_v2_07.pdf`.

[Lip07] LIPP, MANFRED. *VPN - Virtuelle Private Netzwerke: Aufbau und Sicherheit*. Addison-Wesley, München, September 2007.

[Lip09] LIPSKI, MARCUS. *Social Engineering – Der Mensch als Sicherheitsrisiko in der IT*. Diplomarbeit, Private FernFachhochschule Darmstadt, Januar 2009.

[Lis05] LISKOV, MOSES. *Miller-Rabin Probabilistic Primality Test.* In: VAN TILBORG, HENK C. A. (Herausgeber): *Encyclopedia of Cryptography and Security*. Springer, 2005. URL: http://dblp.uni-trier.de/db/reference/crypt/crypt2005.html#Liskov05c.

[LL07] LIANG, JIE und XUE-JIA LAI. *Improved Collision Attack on Hash Function MD5.* Journal of Computer Science and Technologie, 22(1):79–87, 2007. URL: http://dblp.uni-trier.de/db/journals/jcst/jcst22.html#LiangL07.

[MA08] MEZLER-ANDELBERG, CHRISTIAN. *Identity Management*. dpunkt.verlag, 2008.

[Mil85] MILLER, VICTOR S. *Use of Elliptic Curves in Cryptography.* In: WILLIAMS, HUGH C. (Herausgeber): *CRYPTO*, Band 218 der Reihe *Lecture Notes in Computer Science*, Seiten 417–426. Springer, 1985. URL: http://dblp.uni-trier.de/db/conf/crypto/crypto85.html#Miller85.

[Mil03] MILLER, MICHAEL. *Symmetrische Verschlüsselungsverfahren. Design, Entwicklung und Kryptoanalyse klassischer und moderner Chiffren.* Teubner, Stuttgart, 2003.

[MIS50] *MIFARE MF1ICS50 - Functional Specification*, Januar 2008. URL: http://www.nxp.com/acrobat_download/other/identification/M001053_MF1ICS50_rev5_3.pdf.

[MT79] MORRIS, ROBERT und KEN THOMPSON. *Password Security - A Case History.* Commun. ACM, 22(11):594–597, 1979. URL: http://dblp.uni-trier.de/db/journals/cacm/cacm22.html#MorrisT79.

[PKCS1] *PKCS #1: RSA Encryption Standard*, Juni 2002. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf.

[PKCS11] *PKCS #11: Cryptographic Token Interface Standard Version 2.20*, 2004. URL: http://www.rsa.com/rsalabs/node.asp?id=2133.

[PKCS15] *PKCS #15: Cryptographic Token Information Format Standard*, 2000. URL: http://www.rsa.com/rsalabs/node.asp?id=2141.

[Plö08] PLÖTZ, HENRYK. *Mifare Classic - Eine Analyse der Implementierung*. Diplomarbeit, Humboldt-Universität Berlin, Oktober 2008. URL: http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2008-21/SAR-PR-2008-21_.pdf.

[QPDK04] QUISQUATER, J.-J., PIERRE PARADINAS, YVES DESWARTE und ANAS ABOU EL KALAM. *Smart Card Technologies and Applications*, Band Vol. 153 der Reihe *IFIP Advances in Information and Communication Technology*. Springer, April 2004.

[RE02] RANKL, WOLFGANG und WOLFGANG EFFING. *Handbuch der Chipkarten*. Carl Hanser Verlag, München,Wien, 4. Auflage, 2002.

[RFC 768] POSTEL, J. *User Datagram Protocol*. RFC 768 (Standard), August 1980. URL: http://www.ietf.org/rfc/rfc768.txt.

[RFC 793] POSTEL, J. *Transmission Control Protocol*. RFC 793 (Standard), September 1981. URL: http://www.ietf.org/rfc/rfc793.txt. Updated by RFC 3168.

[RFC 821] POSTEL, J. *Simple Mail Transfer Protocol*. RFC 821 (Standard), August 1982. URL: http://www.ietf.org/rfc/rfc821.txt. Obsoleted by RFC 2821.

[RFC1081] ROSE, M.T. *Post Office Protocol: Version 3*. RFC 1081, November 1988. URL: http://www.ietf.org/rfc/rfc1081.txt. Obsoleted by RFC 1225.

[RFC1225] ROSE, M.T. *Post Office Protocol: Version 3*. RFC 1225 (Draft Standard), Mai 1991. URL: http://www.ietf.org/rfc/rfc1225.txt. Obsoleted by RFC 1460.

[RFC1334] LLOYD, B. und W. SIMPSON. *PPP Authentication Protocols*. RFC 1334 (Proposed Standard), Oktober 1992. URL: http://www.ietf.org/rfc/rfc1334.txt. Obsoleted by RFC 1994.

[RFC1460] ROSE, M. *Post Office Protocol - Version 3*. RFC 1460 (Draft Standard), Juni 1993. URL: http://www.ietf.org/rfc/rfc1460.txt. Obsoleted by RFC 1725.

[RFC1725] MYERS, J. und M. ROSE. *Post Office Protocol - Version 3*. RFC 1725 (Standard), November 1994. URL: http://www.ietf.org/rfc/rfc1725.txt. Obsoleted by RFCs 1939, NaN.

[RFC1730] CRISPIN, M. *Internet Message Access Protocol - Version 4*. RFC 1730 (Proposed Standard), Dezember 1994. URL: http://www.ietf.org/rfc/rfc1730.txt. Obsoleted by RFCs 2060, 2061.

[RFC1731] MYERS, J. *IMAP4 Authentication Mechanisms*. RFC 1731 (Proposed Standard), Dezember 1994. URL: http://www.ietf.org/rfc/rfc1731.txt.

[RFC1734] MYERS, J. *POP3 AUTHentication command*. RFC 1734 (Proposed Standard), Dezember 1994. URL: http://www.ietf.org/rfc/rfc1734.txt.

[RFC1760] HALLER, N. *The S/KEY One-Time Password System*. RFC 1760 (Informational), Februar 1995. URL: http://www.ietf.org/rfc/rfc1760.txt.

[RFC1869] KLENSIN, J., N. FREED, M. ROSE, E. STEFFERUD und D. CROCKER. *SMTP Service Extensions*. RFC 1869 (Standard), November 1995. URL: http://www.ietf.org/rfc/rfc1869.txt. Obsoleted by RFC 2821.

[RFC1939] MYERS, J. und M. ROSE. *Post Office Protocol - Version 3*. RFC 1939 (Standard), Mai 1996. URL: http://www.ietf.org/rfc/rfc1939.txt. Updated by RFCs 1957, 2449.

[RFC2060] CRISPIN, M. *Internet Message Access Protocol - Version 4rev1*. RFC 2060 (Proposed Standard), Dezember 1996. URL: http://www.ietf.org/rfc/rfc2060.txt. Obsoleted by RFC 3501.

[RFC2104] KRAWCZYK, H., M. BELLARE und R. CANETTI. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104 (Informational), Februar 1997. URL: `http://www.ietf.org/rfc/rfc2104.txt`.

[RFC2195] KLENSIN, J., R. CATOE und P. KRUMVIEDE. *IMAP/POP AUTHorize Extension for Simple Challenge/Response*. RFC 2195 (Proposed Standard), September 1997. URL: `http://www.ietf.org/rfc/rfc2195.txt`.

[RFC2202] CHENG, P. und R. GLENN. *Test Cases for HMAC-MD5 and HMAC-SHA-1*. RFC 2202 (Informational), September 1997. URL: `http://www.ietf.org/rfc/rfc2202.txt`.

[RFC2222] MYERS, J. *Simple Authentication and Security Layer (SASL)*. RFC 2222 (Proposed Standard), Oktober 1997. URL: `http://www.ietf.org/rfc/rfc2222.txt`. Obsoleted by RFCs 4422, 4752, updated by RFC 2444.

[RFC2246] DIERKS, T. und C. ALLEN. *The TLS Protocol Version 1.0*. RFC 2246 (Proposed Standard), Januar 1999. URL: `http://www.ietf.org/rfc/rfc2246.txt`. Obsoleted by RFC 4346, updated by RFC 3546.

[RFC2289] HALLER, N., C. METZ, P. NESSER und M. STRAW. *A One-Time Password System*. RFC 2289 (Standard), Februar 1998. URL: `http://www.ietf.org/rfc/rfc2289.txt`.

[RFC2315] KALISKI, B. *PKCS #7: Cryptographic Message Syntax Version 1.5*. RFC 2315 (Informational), März 1998. URL: `http://www.ietf.org/rfc/rfc2315.txt`.

[RFC2440] CALLAS, J., L. DONNERHACKE, H. FINNEY und R. THAYER. *OpenPGP Message Format*. RFC 2440 (Proposed Standard), November 1998. URL: `http://www.ietf.org/rfc/rfc2440.txt`.

[RFC2449] GELLENS, R., C. NEWMAN und L. LUNDBLADE. *POP3 Extension Mechanism*. RFC 2449 (Proposed Standard), November 1998. URL: `http://www.ietf.org/rfc/rfc2449.txt`.

[RFC2487] HOFFMAN, P. *SMTP Service Extension for Secure SMTP over TLS*. RFC 2487 (Proposed Standard), Januar 1999. URL: `http://www.ietf.org/rfc/rfc2487.txt`. Obsoleted by RFC 3207.

[RFC2554] MYERS, J. *SMTP Service Extension for Authentication*. RFC 2554 (Proposed Standard), März 1999. URL: `http://www.ietf.org/rfc/rfc2554.txt`.

[RFC2560] MYERS, M., R. ANKNEY, A. MALPANI, S. GALPERIN und C. ADAMS. *X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP*. RFC 2560 (Proposed Standard), Juni 1999. URL: `http://www.ietf.org/rfc/rfc2560.txt`.

[RFC2595] NEWMAN, C. *Using TLS with IMAP, POP3 and ACAP*. RFC 2595 (Proposed Standard), Juni 1999. URL: `http://www.ietf.org/rfc/rfc2595.txt`. Updated by RFC 4616.

[RFC2617] FRANKS, J., P. HALLAM-BAKER, J. HOSTETLER, S. LAWRENCE, P. LEACH, A. LUOTONEN und L. STEWART. *HTTP Authentication: Basic and Digest Access Authentication*. RFC 2617 (Draft Standard), Juni 1999. URL: http://www.ietf.org/rfc/rfc2617.txt.

[RFC2626] II, P. NESSER. *The Internet and the Millennium Problem (Year 2000)*. RFC 2626 (Informational), Juni 1999. URL: http://www.ietf.org/rfc/rfc2626.txt.

[RFC2817] KHARE, R. und S. LAWRENCE. *Upgrading to TLS Within HTTP/1.1*. RFC 2817 (Proposed Standard), Mai 2000. URL: http://www.ietf.org/rfc/rfc2817.txt.

[RFC2818] RESCORLA, E. *HTTP Over TLS*. RFC 2818 (Informational), Mai 2000. URL: http://www.ietf.org/rfc/rfc2818.txt.

[RFC2821] KLENSIN, J. *Simple Mail Transfer Protocol*. RFC 2821 (Proposed Standard), April 2001. URL: http://www.ietf.org/rfc/rfc2821.txt.

[RFC2831] LEACH, P. und C. NEWMAN. *Using Digest Authentication as a SASL Mechanism*. RFC 2831 (Proposed Standard), Mai 2000. URL: http://www.ietf.org/rfc/rfc2831.txt.

[RFC2865] RIGNEY, C., S. WILLENS, A. RUBENS und W. SIMPSON. *Remote Authentication Dial In User Service (RADIUS)*. RFC 2865 (Draft Standard), Juni 2000. URL: http://www.ietf.org/rfc/rfc2865.txt. Updated by RFCs 2868, 3575.

[RFC2898] KALISKI, B. *PKCS #5: Password-Based Cryptography Specification Version 2.0*. RFC 2898 (Informational), September 2000. URL: http://www.ietf.org/rfc/rfc2898.txt.

[RFC3156] ELKINS, M., D. DEL TORTO, R. LEVIEN und T. ROESSLER. *MIME Security with OpenPGP*. RFC 3156 (Proposed Standard), August 2001. URL: http://www.ietf.org/rfc/rfc3156.txt.

[RFC3163] ZUCCHERATO, R. und M. NYSTROM. *ISO/IEC 9798-3 Authentication SASL Mechanism*. RFC 3163 (Experimental), August 2001. URL: http://www.ietf.org/rfc/rfc3163.txt.

[RFC3207] HOFFMAN, P. *SMTP Service Extension for Secure SMTP over Transport Layer Security*. RFC 3207 (Proposed Standard), Februar 2002. URL: http://www.ietf.org/rfc/rfc3207.txt.

[RFC3447] JONSSON, J. und B. KALISKI. *Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1*. RFC 3447 (Informational), Februar 2003. URL: http://www.ietf.org/rfc/rfc3447.txt.

[RFC3501] CRISPIN, M. *INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1*. RFC 3501 (Proposed Standard), März 2003. URL: http://www.ietf.org/rfc/rfc3501.txt. Updated by RFCs 4466, 4469, 4551.

[RFC3748] ABOBA, B., L. BLUNK, J. VOLLBRECHT, J. CARLSON und H. LEVKOWETZ. *Extensible Authentication Protocol (EAP)*. RFC 3748 (Proposed Standard), Juni 2004. URL: `http://www.ietf.org/rfc/rfc3748.txt`.

[RFC3851] RAMSDELL, B. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification*. RFC 3851 (Proposed Standard), Juli 2004. URL: `http://www.ietf.org/rfc/rfc3851.txt`.

[RFC4120] NEUMAN, C., T. YU, S. HARTMAN und K. RAEBURN. *The Kerberos Network Authentication Service (V5)*. RFC 4120 (Proposed Standard), Juli 2005. URL: `http://www.ietf.org/rfc/rfc4120.txt`. Updated by RFC 4537.

[RFC4250] LEHTINEN, S. und C. LONVICK. *The Secure Shell (SSH) Protocol Assigned Numbers*. RFC 4250 (Proposed Standard), Januar 2006. URL: `http://www.ietf.org/rfc/rfc4250.txt`.

[RFC4251] YLONEN, T. und C. LONVICK. *The Secure Shell (SSH) Protocol Architecture*. RFC 4251 (Proposed Standard), Januar 2006. URL: `http://www.ietf.org/rfc/rfc4251.txt`.

[RFC4252] YLONEN, T. und C. LONVICK. *The Secure Shell (SSH) Authentication Protocol*. RFC 4252 (Proposed Standard), Januar 2006. URL: `http://www.ietf.org/rfc/rfc4252.txt`.

[RFC4253] YLONEN, T. und C. LONVICK. *The Secure Shell (SSH) Transport Layer Protocol*. RFC 4253 (Proposed Standard), Januar 2006. URL: `http://www.ietf.org/rfc/rfc4253.txt`.

[RFC4254] YLONEN, T. und C. LONVICK. *The Secure Shell (SSH) Connection Protocol*. RFC 4254 (Proposed Standard), Januar 2006. URL: `http://www.ietf.org/rfc/rfc4254.txt`.

[RFC4409] GELLENS, R. und J. KLENSIN. *Message Submission for Mail*. RFC 4409 (Draft Standard), April 2006. URL: `http://www.ietf.org/rfc/rfc4409.txt`.

[RFC4422] MELNIKOV, A. und K. ZEILENGA. *Simple Authentication and Security Layer (SASL)*. RFC 4422 (Proposed Standard), Juni 2006. URL: `http://www.ietf.org/rfc/rfc4422.txt`.

[RFC4462] HUTZELMAN, J., J. SALOWEY, J. GALBRAITH und V. WELCH. *Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol*. RFC 4462 (Proposed Standard), Mai 2006. URL: `http://www.ietf.org/rfc/rfc4462.txt`.

[RFC4505] ZEILENGA, K. *Anonymous Simple Authentication and Security Layer (SASL) Mechanism*. RFC 4505 (Proposed Standard), Juni 2006. URL: `http://www.ietf.org/rfc/rfc4505.txt`.

[RFC4559] JAGANATHAN, K., L. ZHU und J. BREZAK. *SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows*. RFC 4559 (Informational), Juni 2006. URL: `http://www.ietf.org/rfc/rfc4559.txt`.

[RFC4616] ZEILENGA, K. *The PLAIN Simple Authentication and Security Layer (SASL) Mechanism*. RFC 4616 (Proposed Standard), August 2006. URL: http://www.ietf.org/rfc/rfc4616.txt.

[RFC4634] EASTLAKE 3RD, D. und T. HANSEN. *US Secure Hash Algorithms (SHA and HMAC-SHA)*. RFC 4634 (Informational), August 2006. URL: http://www.ietf.org/rfc/rfc4634.txt.

[RFC4716] GALBRAITH, J. und R. THAYER. *The Secure Shell (SSH) Public Key File Format*. RFC 4716 (Informational), November 2006. URL: http://www.ietf.org/rfc/rfc4716.txt.

[RFC4752] MELNIKOV, A. *The Kerberos V5 ("GSSAPI") Simple Authentication and Security Layer (SASL) Mechanism*. RFC 4752 (Proposed Standard), November 2006. URL: http://www.ietf.org/rfc/rfc4752.txt.

[RFC4764] BERSANI, F. und H. TSCHOFENIG. *The EAP-PSK Protocol: A Pre-Shared Key Extensible Authentication Protocol (EAP) Method*. RFC 4764 (Experimental), Januar 2007. URL: http://www.ietf.org/rfc/rfc4764.txt.

[RFC4819] GALBRAITH, J., J. VAN DYKE und J. BRIGHT. *Secure Shell Public Key Subsystem*. RFC 4819 (Proposed Standard), März 2007. URL: http://www.ietf.org/rfc/rfc4819.txt.

[RFC4880] CALLAS, J., L. DONNERHACKE, H. FINNEY und R. THAYER. *OpenPGP Message Format*. RFC 4880 (Proposed Standard), November 2007. URL: http://www.ietf.org/rfc/rfc4880.txt.

[RFC4954] SIEMBORSKI, R. und A. MELNIKOV. *SMTP Service Extension for Authentication*. RFC 4954 (Proposed Standard), Juli 2007. URL: http://www.ietf.org/rfc/rfc4954.txt.

[RFC5034] SIEMBORSKI, R. und A. MENON-SEN. *The Post Office Protocol (POP3) – Simple Authentication and Security Layer (SASL) Authentication Mechanism*. RFC 5034 (Proposed Standard), Juli 2007. URL: http://www.ietf.org/rfc/rfc5034.txt.

[RFC5055] FREEMAN, T., R. HOUSLEY, A. MALPANI, D. COOPER und W. POLK. *Server-Based Certificate Validation Protocol (SCVP)*. RFC 5055 (Proposed Standard), Dezember 2007. URL: http://www.ietf.org/rfc/rfc5055.txt.

[RFC5216] SIMON, D., B. ABOBA und R. HURST. *The EAP-TLS Authentication Protocol*. RFC 5216 (Proposed Standard), März 2008. URL: http://www.ietf.org/rfc/rfc5216.txt.

[RFC5246] DIERKS, T. und E. RESCORLA. *The Transport Layer Security (TLS) Protocol - Version 1.2*. RFC 5246 (Proposed Standard), August 2008. URL: http://www.ietf.org/rfc/rfc5246.txt.

[RFC5247] ABOBA, B., D. SIMON und P. ERONEN. *Extensible Authentication Protocol (EAP) Key Management Framework*. RFC 5247 (Proposed Standard), August 2008. URL: http://www.ietf.org/rfc/rfc5247.txt.

[RFC5281] FUNK, P. und S. BLAKE-WILSON. *Extensible Authentication Protocol Tunneled Transport Layer Security – Authenticated Protocol Version 0 (EAP-TTLSv0)*. RFC 5281 (INFORMATIONAL), August 2008. URL: `http://www.ietf.org/rfc/rfc5281.txt`.

[RFC5321] KLENSIN, J. *Simple Mail Transfer Protocol*. RFC 5321 (Draft Standard), Oktober 2008. URL: `http://www.ietf.org/rfc/rfc5321.txt`.

[Rie07] RIEGER, SEBASTIAN. *Einheitliche Authentifizierung in heterogenen IT-Strukturen für ein sicheres e-Science-Umfeld*. Cuvillier, 2007.

[Riv97] RIVEST, RONALD L. *The RC5 Encryption Algorithm*. Technischer Bericht, MIT Laboratory for Computer Science, Cambridge, Massachusetts, März 1997.

[RR06] RECORDON, DAVID und DRUMMOND REED. *OpenID 2.0: a platform for user-centric identity management.* In: JUELS, ARI, MARIANNE WINSLETT und ATSUHIRO GOTO (Herausgeber): *Digital Identity Management*, Seiten 11–16. ACM, 2006. URL: `http://dblp.uni-trier.de/db/conf/dim/dim2006.html#RecordonR06`.

[RRSY98] RIVEST, RONALD L., M. J. B. ROBSHAW, R. SIDNEY und Y. L. YIN. *The RC6 Block Cipher*. In: *First Advanced Encryption Standard (AES) Conference*, 1998.

[Sch93] SCHNEIER, BRUCE. *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish)*. In: *Fast Software Encryption, Cambridge Security Workshop Proceedings*, Band 809 der Reihe *Lecture Notes in Computer Science*, Seiten 191–204, Cambridge, UK, December 1993. Springer-Verlag.

[Sch07a] SCHMEH, KLAUS. *Kryptographie - Verfahren, Protokolle, Infrastrukturen*. dpunkt.verlag, 3. Auflage, 2007.

[Sch07b] SCHÖNBERG, MARC. *Single Sign-On-Technologien für das World Wide Web*. Dissertation, Universität Hamburg, September 2007. URL: `http://vsis-www.informatik.uni-hamburg.de/getDoc.php/thesis/179/Single_Sign-On-Technologien_f%FCr_das_World_Wide_Web.pdf`.

[SKW+00] SCHNEIER, BRUCE, JOHN KELSEY, DOUG WHITING, DAVID WAGNER und NIELS FERGUSON. *Comments on Twofish as an AES Candidate.* In: *AES Candidate Conference*, Seiten 355–356, 2000. URL: `http://dblp.uni-trier.de/db/conf/aes/aes2000.html#SchneierKWWF00`.

[SLP05] SCHINDLER, WERNER, KERSTIN LEMKE und CHRISTOF PAAR. *A Stochastic Model for Differential Side Channel Cryptanalysis.* In: RAO, JOSYULA R. und BERK SUNAR (Herausgeber): *CHES*, Band 3659 der Reihe *Lecture Notes in Computer Science*, Seiten 30–46. Springer, 2005. URL: `http://dblp.uni-trier.de/db/conf/ches/ches2005.html#SchindlerLP05`.

[Spa07] SPARENBERG, JÖRG-UWE. *Einführung einer zertifikatsbasierten IEEE 802.1x Infrastruktur an der Martin-Luther-Universität Halle-Wittenberg*. Diplomarbeit, Martin-Luther-Universität Halle-Wittenberg, Juli 2007.

[Spi08] *Riesiges Datenleck - Telekom verschwieg über Jahre Sicherheitslücke*. Spiegel Online, 01.10.2008. URL: http://www.spiegel.de/wirtschaft/0,1518,581717,00.html.

[SR09] SCHMIDT, MAIK und CHRISTIANE RÜTTEN. *Eine für alles - Single Sign-on mit OpenID*. c't Magazin für Computer Technik, (18):156, August 2009.

[Tan96] TANENBAUM, ANDREW S. *Computer Networks*. Prentice Hall, Upper Saddle River, 1996.

[TWP07] TEWS, ERIK, RALF-PHILIPP WEINMANN und ANDREI PYSHKIN. *Breaking 104 Bit WEP in Less Than 60 Seconds.* In: KIM, SEHUN, MOTI YUNG und HYUNG-WOO LEE (Herausgeber): *WISA*, Band 4867 der Reihe *Lecture Notes in Computer Science*, Seiten 188–202. Springer, 2007. URL: http://dblp.uni-trier.de/db/conf/wisa/wisa2007.html#TewsWP07.

[Uni06] UNISYS GMBH DEUTSCHLAND. *Unisys-Verbraucherstudie: Biometrie erhält überwältigenden Zuspruch für Identitätsnachweis*, April 2006. URL: http://www.unisys.de/about__unisys/presse/06042701.htm.

[Way08] WAYMAN, JAMES L. *Biometrics in Identity Management Systems*. IEEE Security & Privacy, 6(2):30 ff., März 2008.

[WD01] WESTERLUND, ASSAR und JOHAN DANIELSSON. *Heimdal and Windows 2000 Kerberos - How to Get Them to Play Together.* In: COLE, CLEM (Herausgeber): *USENIX Annual Technical Conference, FREENIX Track*, Seiten 267–272. USENIX, 2001. URL: http://dblp.uni-trier.de/db/conf/usenix/usenix2001f.html#WesterlundD01.

[Win05] WINDLEY, PHILLIP J. *Digital Identity*. O'Reilly Media, Inc., 2005.

[WLF+05] WANG, XIAOYUN, XUEJIA LAI, DENGGUO FENG, HUI CHEN und XIUYUAN YU. *Cryptanalysis of the hash functions MD4 and RIPEMD.* In: CRAMER, RONALD (Herausgeber): *Advances in cryptology – EUROCRYPT*, Band 3494 der Reihe *Lecture Notes in Computer Science*, Seiten 1–18. Springer, 2005.

[WP07] WENDT, KATHARINA und SEBASTIAN PREUSSE. *Untersuchungen zum Einsatz von Single Sign-On Systemen zur Authentifizierung mittels Smartcards in Windows-Computerpools der Martin-Luther-Universität.* Diplomarbeit, Martin-Luther-Universität Halle-Wittenberg, Januar 2007.

[WSE04] WALDMANN, ULRICH, DIRK SCHEUERMANN und CLAUDIA ECKERT. *Protected transmission of biometric user authentication data for oncard-matching.* In: HADDAD, HISHAM, ANDREA OMICINI, ROGER L. WAINWRIGHT und LORIE M. LIEBROCK (Herausgeber): *SAC*, Seiten 425–430. ACM, 2004. URL: http://dblp.uni-trier.de/db/conf/sac/sac2004.html#WaldmannSE04.

[X509] INTERNATIONAL TELECOMMUNICATION UNION. *The Directory — Authentication framework*. ITU-T Recommendation X.509, November 1993.

[X9.62] AMERICAN NATIONAL STANDARDS INSTITUTE. *X9.62 Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)*. ANSI 2005, 2005.

[XSS97] *X/Open Single Sign-on Preliminary Specification*, Juni 1997. URL: `http://www.opengroup.org/onlinepubs/008329799/`.

# Anhang A

# Aufbau eines X.509-Zertifikats

Ermittlung der Zertifikatsdaten auf einer im PEM-Format kodierten Zertifikatsdatei:

```
openssl x509 -in <cert_in_PEM-Format> -noout -text
```

```
Data:                                       /* Daten-Block des Zertifikats */

  Version: 3 (0x2)                          /* Zertifikat X.509 Version 3 */

  Serial_Number: 24 (0x18)                  /* Seriennummer des ausgestellten */
                                            /* Zertifikats => 24 */

  Signature_Algorithm: sha1WithRSAEncryption /* verwendeter Signaturalgorithmus */

  Issuer: C=DE, ST=GERMANY, L=Halle, O=MLU, OU=UZI, CN=ca.uzi.uni-halle.de/
      emailAddress=wefel@uzi.uni-halle.de    /* eindeutiger Bezeichner (Subjekt) */
                                            /* des Ausstellers */
  Validity:
      Not_Before: May 19 11:01:40 2008 GMT   /* Gültigkeitszeitraum */
      Not_After_: May 19 11:01:40 2010 GMT

  Subject: C=DE, O=MLU, OU=people, CN=Sandro Wefel/emailAddress=sandro.wefel@informatik.uni-
      halle.de/x500UniqueIdentifier=wefel   /* eindeutiger Bezeichner (Subjekt) */
                                            /* des Inhabers */

  Subject_Public_Key_Info:                  /* Block mit öffentlichem Schlüssel */
      Public_Key_Algorithm: rsaEncryption   /* Typ: RSA */
      RSA_Public_Key: (2048 bit)            /* Modulus-Größe: 2048 Bit */
          Modulus (2048 bit):               /* Modulus-Wert: ... */
              00:d3:b2:28:35:40:23:6d:f8:55:b2:01:0c:9a:77:
              a0:a9:96:f8:c6:20:d2:e6:0a:71:a6:f2:a9:b5:7d:
              d5:77:76:2e:cc:1b:5e:d7:e0:34:61:74:27:8c:3a:
              cf:27:4a:44:e0:22:b1:c0:1a:d0:35:81:33:b2:2f:
              24:8f:3a:92:19:70:ef:c3:d3:30:2e:af:35:f6:d8:
              e1:8f:e9:f4:ca:33:2f:50:17:25:6b:80:2e:9d:a8:
              38:2e:08:0b:14:87:68:0d:8c:95:ab:93:bc:c7:56:
              a7:81:ae:4a:62:a8:00:3d:d3:11:05:2e:aa:ec:46:
              a1:9e:e6:62:7d:d0:10:2f:6f:ba:68:97:ad:bd:f9:
              58:cc:0d:bb:e6:ba:2e:cc:17:d8:d2:f8:a2:a6:a7:
              23:0c:7f:83:4b:9a:b4:e2:89:96:46:91:09:4e:da:
              d8:e9:d2:db:e7:d5:32:e7:ee:58:44:9a:67:ff:54:
              e3:07:f6:6f:27:bc:c2:74:a1:a0:07:cb:12:c2:3b:
              4e:29:f6:a3:4f:e8:89:59:8d:a1:0c:75:ab:db:66:
              c7:1b:db:76:eb:3b:28:35:9a:22:bb:f3:3a:ad:8f:
              63:95:a6:a6:1f:25:fb:e4:56:c9:08:d3:b3:02:2d:
              19:d1:c8:a7:2f:c3:98:4b:e5:a3:6c:87:4c:a0:38:
              7c:87
          Exponent: 65537 (0x10001)         /* der öffentliche Exponent */
```

```
  X509v3_extensions:                    /* eingebettete X.509 Erweiterungen */

      X509v3_Basic_Constraints:         /* Nutzungsbeschränkung dieses Zertifikats */
          CA:FALSE                        /* darf nicht für Sub-CA verwendet werden */

      Netscape_Cert_Type:               /* Zertifikat-Typ nach Netscape (veraltet) */
          SSL Client, S/MIME, Object Signing  /* Client-Zertifikat für */
                                              /* Email-Verschlüsselung und Signatur */

      X509v3_Key_Usage:                 /* Schlüsselverwendung nach X.509v3 */
          Digital Signature, Non Repudiation, Key Encipherment, Data Encipherment
            /* Digital Signature: erlaubt SSL-Client-,S/MIME- und Objekt-Signaturen
                                  für Anwendungen mit eher kurzfristigem Charakter und
               non repudiation:   (nichtabstreitbar) langfristigem Charakter
               Key Encipherment:  darf zur Verschlüsselung von sensiblen Informationen,
                                  z.B. (symmetrischen) Schlüsseln und
               Data Encipherment: zur Verschlüsselung von Daten verwendet werden */

      Netscape_Comment:                 /* Kommentar */
          User Certificate UZI Uni-Halle https://uzi.uni-halle.de/ca

      X509v3_Subject_Key_Identifier:    /* eindeutige Schlüssel-ID (Hash) */
                                        /* dieses Schlüssels (Inhaber) */
          AD:62:7C:6D:5E:1F:38:BF:47:28:95:57:AA:D3:2C:05:33:5F:04:53
      X509v3_Authority_Key_Identifier:  /* eindeutige Schlüssel-ID (Hash) */
                                        /* des Aussteller-Schlüssels */
          keyid:4C:9C:66:17:AA:FB:36:E1:74:07:41:9F:EC:18:C0:B4:6A:54:1F:43

      X509v3_Subject_Alternative_Name:  /* alternative Namen des Inhabers */
          email:sandro.wefel@informatik.uni-halle.de
      X509v3_Issuer_Alternative_Name:   /* ~ des Ausstellers */
          <EMPTY>

      Netscape_CA_Revocation_Url:       /* Netscape-Variante zum Abruf der CRL */
          http://uzi.uni-halle.de/ca/ca-crl.pem
      Netscape_Base_Url:                /* Basis-URL der CA, z.B. für Zertifikat-Abruf */
          http://uzi.uni-halle.de/ca
      Netscape_Revocation_Url:          /* URL zur Anfrage, ob Zertifikat zur übergebenen
                                           Seriennummer zurückgezogen wurde, vergleichbar
                                           mit OCSP ohne Datumsprüfung */
          ca-crl.php                    /* => http://uzi.uni-halle.de/ca/ca-crl.php */
      /* die Netscape-Revocation-Variante wird nur noch selten verwendet. An ihre Stelle
         trat die CRL Distribution Points Erweiterung, die aber für dieses Zertifikat
         noch nicht angewendet wurde */

/* der Signatur Block:
   - enthält die Signatur über den gesamten Daten-Block
   - der Datenblock wird von der CA aus den Daten des Request (= Großteil der Inhalte)
     und den zusätzlichen CA-Informationen (z.B. Gültigkeitszeitraum) erzeugt
   - Signatur- und Daten-Block bilden zusammen das Zertifikat */
Signature_Algorithm: sha1WithRSAEncryption
    30:39:01:0e:47:30:dc:13:aa:c2:89:5e:d3:2a:4e:92:96:6c:
    f9:24:08:05:d0:dc:98:c5:bb:51:be:99:96:7e:a2:5c:0d:1a:
    ed:10:41:c5:c8:15:d6:dd:23:ad:98:db:7a:74:07:b1:2e:dd:
    db:31:74:21:d7:ce:89:a4:d5:45:17:61:cf:31:bc:7e:53:9e:
    03:c0:ed:e7:81:19:39:6c:64:90:09:22:ad:12:ea:f6:7b:d7:
    89:63:3e:67:1a:ca:b7:29:14:41:9f:5b:d7:0e:d3:bb:c8:d1:
    65:9a:e6:6f:15:f8:f8:a6:f9:fa:6a:6c:54:a3:18:a9:98:c6:
    2f:3d:7f:e7:86:8f:06:36:ea:7a:38:e0:e3:28:c8:fa:8c:4e:
    fe:2e:1d:63:c4:1b:f0:ea:9c:bd:27:02:12:f3:e3:27:c4:28:
    1c:b3:6b:49:97:08:c5:85:9c:63:76:18:af:10:8a:d9:0e:43:
    82:cb:5e:6d:53:64:c5:f0:4c:96:20:d2:98:4c:67:85:e1:0d:
    8b:44:8b:ff:6e:ab:ee:60:31:67:24:0c:ee:e6:d2:73:fb:e0:
    b0:5a:53:a9:d2:ae:d5:2c:cc:3c:17:b0:4e:7c:e1:38:a9:52:
    f9:b2:6c:25:35:19:66:53:e3:ca:74:8b:de:37:2e:33:d7:ab:
    25:af:b1:f8
```

# Anhang B

# PKCS#11-Kommunikationsablauf

## B.1 Thunderbird, PKCS#11-Proxy mit Agent und OpenSC

Auflistung der Modifikation im PKCS#11-Kommunikationsablauf durch Einsatz der Proxy-Bibliothek des Agenten am Beispiel der zertifikatsbasierten Authentifizierung im Email-Programm Icedove[1]. Die Ausgabe erfolgte durch Debug-Modus der PKCS#11-ProxyBibliothek. Sie wurde gekürzt, der zeitliche Ablauf und die wesentlichen Änderungen bleiben erkennbar.
Rot dargestellt werden Änderungen durch die Proxy-Bibliothek innerhalb einer Übertragung zum PKCS#11-Provider des Herstellers sowie die zusätzlich verwalteten und ggf. geänderten Informationen, wie z.B. der Sitzungszähler.
Das für diese Aufzeichnung verwendete Siemens-Token enthält zwei Nutzer-Schlüssel und - Zertifikate, sowie die Aussteller-Zertifikate. Es wird in der verwendeten OpenSC-Bibliothek auf zwei PKCS#11-Slots abgebildet, wobei der erste Slot alle für Signatur und Entschlüsselung notwendigen Daten enthält. Die Applikation fragt beide Slots ab, verwendet aber nur den ersten für die Operationen.

**Initialisierung:**

```
*************** OpenSC PKCS#11 Agent *****************
Executable: /usr/lib/icedove/icedove-bin

0: C_GetFunctionList     16:04:15 [12.06.09]
Returned:  0 CKR_OK

1: C_Initialize          16:04:15 [12.06.09]
Returned:  0 CKR_OK

2: C_GetInfo             16:04:17 [12.06.09]
      cryptokiVersion:         2.20
      manufacturerID:         'OpenSC (www.opensc-project.org) '
      flags:                   0
      libraryDescription:     'smart card PKCS#11 API          '
      libraryVersion:          0.0
Returned:  0 CKR_OK

3: C_GetSlotList         16:04:17 [12.06.09]
  [in] tokenPresent = 0x0
  [out] pSlotList: Count is 16
  [out] *pulCount = 0x10
Returned:  0 CKR_OK

4: C_GetSlotList         16:04:17 [12.06.09]
  [in] tokenPresent = 0x0
```

[1] Icedove: Debian-Ableger von Thunderbird

```
  [out] pSlotList:
    Slot 0
    ...
    Slot 15
  [out] *pulCount = 0x10      /* 16 verfügbare Slots */
Returned:  0 CKR_OK

/* Abfragen von Informationen zum Slot 0 und dem dort eingelegten Token */
5: C_GetSlotInfo        16:04:17 [12.06.09]
  [in] slotID = 0x0
  [out] pInfo:
        slotDescription:        'Cherry SmartBoard XX44 00 00    '
                                '                                '
        manufacturerID:         'OpenSC (www.opensc-project.org) '
        hardwareVersion:         0.0
        firmwareVersion:         0.0
        flags:                   7
          CKF_TOKEN_PRESENT
          CKF_REMOVABLE_DEVICE
          CKF_HW_SLOT
Returned:  0 CKR_OK

6: C_GetTokenInfo       16:04:17 [12.06.09]
  [in] slotID = 0x0
  [out] pInfo:
        label:                  'Testcard 2 (PIN)                '
        manufacturerID:         'Siemens AG (C)                  '
        model:                  'PKCS#15         '
        ...
        flags:                   40c
          CKF_LOGIN_REQUIRED
          CKF_USER_PIN_INITIALIZED
          CKF_TOKEN_INITIALIZED
  [out] pInfo modified by pkcs11-agentl:
        label:                  'Testcard 2 (PIN)                '
        manufacturerID:         'Siemens AG (C)                  '
        model:                  'PKCS#15         '
        ...
        flags:                   50c
          CKF_LOGIN_REQUIRED
          CKF_USER_PIN_INITIALIZED
          CKF_PROTECTED_AUTHENTICATION_PATH   ==============>  zusätzliches Flag
          CKF_TOKEN_INITIALIZED
Returned:  0 CKR_OK

7: C_GetMechanismList   16:04:17 [12.06.09]
  [in] slotID = 0x0
  [out] pMechanismList[12]: Count is 12
Returned:  0 CKR_OK

8: C_GetMechanismList   16:04:17 [12.06.09]
  [in] slotID = 0x0
  [out] pMechanismList[12]: CKM_SHA_1
    ...
Returned:  0 CKR_OK

9: C_OpenSession        16:04:17 [12.06.09]   /* Start der 1. Session */
  [in] slotID = 0x0                           /* zu Slot 0 */
  [in] flags = 0x4
    pApplication=0xb26902d0
    Notify=0xb233aab0
  [out] *phSession = 0x1
    Nr. of sessions: 1    ======> Session Counter initialisieren
Returned:  0 CKR_OK
```

```
10: C_FindObjectsInit   16:04:17 [12.06.09]   /* Suche Klasse CKO_NETSCAPE... */
  [in] hSession = 0x1
  [in] pTemplate[1]:
      CKA_CLASS             CKO_NETSCAPE_BUILTIN_ROOT_LIST
Returned:  0 CKR_OK

11: C_FindObjects       16:04:17 [12.06.09]   /* Suche durchführen */
  [in] hSession = 0x1
  [in] ulMaxObjectCount = 0x1
  [out] ulObjectCount = 0x0                   /* kein Objekt der Klasse gefunden */
Returned:  0 CKR_OK

12: C_FindObjectsFinal  16:04:17 [12.06.09]   /* Suche beenden */
  [in] hSession = 0x1
Returned:  0 CKR_OK

13: C_GetSlotInfo       16:04:17 [12.06.09]   /* Informationen über Slot 1 */
  [in] slotID = 0x1
  [out] pInfo:
        slotDescription:        'Cherry SmartBoard XX44 00 00    '
                                '                                '
        manufacturerID:         'OpenSC (www.opensc-project.org) '
        hardwareVersion:         0.0
        firmwareVersion:         0.0
        flags:                   7
          CKF_TOKEN_PRESENT
          CKF_REMOVABLE_DEVICE
          CKF_HW_SLOT
Returned:  0 CKR_OK

/* ... diverse wiederholte oder weitere Abfragen, diesmal für Slot 1 */
14: C_GetTokenInfo      16:04:17 [12.06.09]
  ======> (analog Schritt 6 Flag CKF_PROTECTED_AUTHENTICATION_PATH eingesetzt)
15: C_GetMechanismList  16:04:17 [12.06.09]
16: C_GetMechanismList  16:04:17 [12.06.09]

17: C_OpenSession       16:04:17 [12.06.09]   /* Start einer weiteren Session */
  [in] slotID = 0x1                           /* diesmal zu Slot 1 */
  [in] flags = 0x4
    pApplication=0xb2690570
    Notify=0xb233aab0
  [out] *phSession = 0x2
    Nr. of sessions: 2    ======> Session Counter incrementieren
Returned:  0 CKR_OK

18: C_FindObjectsInit   16:04:17 [12.06.09]   /* Suche Klasse CKO_NETSCAPE... */
19: C_FindObjects       16:04:17 [12.06.09]
20: C_FindObjectsFinal  16:04:17 [12.06.09]

/* Wiederholung Schritt 5 ff. für Slot 2 bis Slot 15
   Hier ist allerdings kein Token eingelegt, es wird keine Session geöffnet */
21: C_GetSlotInfo       16:04:17 [12.06.09]
  [in] slotID = 0x2
  [out] pInfo:
        ...
        flags:                  6             /* im Vergleich zu Schritt 5 bzw. 13 */
          CKF_REMOVABLE_DEVICE                /* kein Token eingelegt */
          CKF_HW_SLOT
Returned:  0 CKR_OK
...
48: C_FindObjectsInit   16:04:17 [12.06.09]   /* hier handelt es sich offenbar um */
  [in] hSession = 0x0                         /* einen Programmfehler in Thunderbird */
  [in] pTemplate[1]:                          /* da eine ungültige Session genutzt wird */
      CKA_CLASS  CKO_NETSCAPE_BUILTIN_ROOT_LIST
Returned:  179 CKR_SESSION_HANDLE_INVALID
```

```
49: C_GetSlotInfo       16:04:17 [12.06.09]   /* analog Schritt 5 (Slot 0) */

/* Erfragen von Information zur Session 1 (wird für Slot 0 genutzt). Hierbei wird die
   Rückgabe derart modifiziert, dass ein bereits erfolgtes Login vorgetäuscht wird */
50: C_GetSessionInfo    16:04:17 [12.06.09]
  [in] hSession = 0x1
  [out] pInfo:
        slotID:                  0
        state:                  '           CKS_RO_PUBLIC_SESSION' =====> war Public-Session
        flags:                   4
          CKF_SERIAL_SESSION
        ulDeviceError:           0
  [out] pInfo after state-modification:
        slotID:                  0
        state:                  '           CKS_RO_USER_FUNCTIONS' =====> ist User-Session
        flags:                   4
          CKF_SERIAL_SESSION
        ulDeviceError:           0
    Nr. of sessions: 2
Returned:  0 CKR_OK


51: C_GetSlotInfo       16:04:17 [12.06.09]   /* analog Schritt 49 für Slot 1 */
52: C_GetSessionInfo    16:04:17 [12.06.09]   /* analog Schritt 50 für Session 2 (Slot 1) */
  ======> Modifikation analog Schritt 50: CKS_RO_PUBLIC_SESSION => CKS_RO_USER_FUNCTIONS

/* erneute Abfrage der anderen Slots */
53: C_GetSlotInfo       16:04:17 [12.06.09]
...
66: C_GetSlotInfo       16:04:17 [12.06.09]

/* WaitForSlotEvent -  Warten auf ein Ereignis für einen beliebigen Slot
   Es kann sich um die Entnahme oder das Entfernen eines Tokens handeln.
   Ist das Flag CKF_DONT_BLOCK nicht gesetzt, blockiert die Funktion bis
   ein Ereignis eintritt.
*/
67: C_WaitForSlotEvent  16:04:17 [12.06.09]
  [in] pSlot = 0xb0e61298
        flags:                   0
    WaitForSlotEvent: modified flags. FireFox/Thunderbird/NSS patch
        flags:                   1
          CKF_DONT_BLOCK     ======> spezielles Flag eingefügt
Returned:  8 CKR_NO_EVENT
! Für SSO unter einigen der getesteten Applikation (Thunderbird/Firefox) wird eine
! zwischenzeitliche Tokenentnahme von der Applikation zwar erkannt wird, aber bei
! Wiedereinsetzen des Tokens werden die Zertifikate nicht reinitialisiert.
! Die bisher vom Token gelesenen und von der Applikation genutzten Nutzer-Zertifikate
! und Schlüssel werden bei einer signalisierten Tokenentnahme verworfen.
! Ohne Neustart der Applikation wäre keine erneute Authentifizierung mit dem
! Token möglich.
! Durch Einfügen des Flags CKF_DONT_BLOCK bei der Anfrage einer der genannten Applikation
! in die Übertragung wird dies verhindert. Die Provider-Bibliothek kehrt in diesem Fall
! mit der Status CKR_NO_EVENT sofort zurück, woraufhin die Applikation eine Tokenentnahme
! nicht sofort erkennt.
! Ohne dieses Flag würde C_WaitForSlotEvent erst beendet, wenn ein Ereignis eintritt.
! Einige der getesteten Provider unterstützen diese Funktion nicht und kehren sofort mit
! einem Fehlerstatus zurück. In diese Falle würde die Applikation auf eine
! Polling-Abfrage zur Erkennung der Tokenentnahme umschalten, was die Systembelastung
! unnötig erhöht. Dies zu vermeiden ist der zweite Grund für das Einfügen des Flags.
! Ein Token kann entnommen werden, ohne von der Applikation sofort bemerkt zu werden.
! Insofern in der Zwischenzeit keine Authentifizierung/Entschlüsselung und damit kein
! Zugriff auf die Tokeninhalte erforderlich ist, werden die Zertifikate nicht verworfen
! und stehen bei Wiederverbindung zum Token in Verbindung mit dem Auto-Login-Mechanismus
! sofort zur Verfügung.
```

```
/* Der Applikation sind die Slots und darin enthaltene Token, sowie die Fähigkeit
 * zum Warten auf Slot-Events bekannt. Zu jedem Slot wurde eine Session geöffnet.
 * Jetzt werden die nutzbaren Zertifikate des Tokens ermittelt und gegebenenfalls
 * enthaltene Sperrlisten (CRLs).
 */

68: C_GetSlotInfo       16:04:25 [12.06.09]     /* Status Slot 0 prüfen */
69: C_GetSessionInfo    16:04:25 [12.06.09]     /* Status Session 1 prüfen */
70: C_GetSessionInfo    16:04:25 [12.06.09]
  ======> Modifikation analog Schritt 50: CKS_RO_PUBLIC_SESSION => CKS_RO_USER_FUNCTIONS

71: C_FindObjectsInit   16:04:25 [12.06.09]     /* Suche nach Zertifikaten starten */
  [in] hSession = 0x1
  [in] pTemplate[2]:
      CKA_TOKEN             True
      CKA_CLASS             CKO_CERTIFICATE
Returned:  0 CKR_OK

/* Session 1 (Slot 0) Zertifikatssuche durchführen */
72: C_FindObjects       16:04:25 [12.06.09]
  [in] hSession = 0x1
  [in] ulMaxObjectCount = 0xa
  [out] ulObjectCount = 0x4                     /* 4 Zertifikate gefunden */
    Object 2 Matches
    Object 4 Matches
    Object 7 Matches
    Object 9 Matches
Returned:  0 CKR_OK

73: C_FindObjectsFinal  16:04:25 [12.06.09]

/* Infomationen über TOKEN- und LABEL-Attribute erfragen */
74: C_GetAttributeValue 16:04:25 [12.06.09]     /* für Objekt 2 */
  [in] hSession = 0x1
  [in] hObject = 0x2
  [in] pTemplate[2]:
      CKA_TOKEN             requested with 0 buffer     /* erfragt nur die Grösse */
      CKA_LABEL             requested with 0 buffer
  [out] pTemplate[2]:
      CKA_TOKEN             has size 1          /* Token- oder Session-Objekt? => 1 Byte */
      CKA_LABEL             has size 29         /* Label? => 29 Byte  */
Returned:  0 CKR_OK

75: C_GetAttributeValue 16:04:25 [12.06.09]     /* Inhalt der Attribute */
  [in] hSession = 0x1
  [in] hObject = 0x2
  [in] pTemplate[2]:
      CKA_TOKEN             requested with 1 buffer     /* erfragt den Inhalt */
      CKA_LABEL             requested with 29 buffer
  [out] pTemplate[2]:
      CKA_TOKEN             True                /* Token-Objekt */
      CKA_LABEL             [size : 0x1D (29)]  /* das Label  */
      77656665 6C40696E 666F726D 6174696B 2E756E69 2D68616C 6C652E64 65
      w e f e  l @ i n  f o r m  a t i k  . u n i  - h a l  l e . d  e
Returned:  0 CKR_OK                             /* es ist das 1. Nutzer-Zertifikat */

76: C_GetAttributeValue 16:04:25 [12.06.09]     /* analog für Objekt 4, Grösse */
77: C_GetAttributeValue 16:04:25 [12.06.09]     /* Inhalt? */
  ...
  [out] pTemplate[2]:
      CKA_TOKEN             True
      CKA_LABEL             [size : 0x16 (22)]
      77656665 6C40757A 692E756E 692D6861 6C6C652E 6465
      w e f e  l @ u z  i . u n  i - h a  l l e .  d e
Returned:  0 CKR_OK                             /* es ist das 1. CA-Zertifikat */
```

```
78: C_GetAttributeValue 16:04:25 [12.06.09]     /* für Objekt 7 */
79: C_GetAttributeValue 16:04:25 [12.06.09]
  ...
  [out] pTemplate[2]:
      CKA_TOKEN             True
      CKA_LABEL             [size : 0xF (15)]
      77656665 6C40756E 69786F73 2E6465         /* es ist das 2. Nutzer-Zertifikat */
      w e f e  l @ u n  i x o s  . d e
Returned:  0 CKR_OK

80: C_GetAttributeValue 16:04:25 [12.06.09]     /* für Objekt 9 */
81: C_GetAttributeValue 16:04:25 [12.06.09]
  ...
  [out] pTemplate[2]:
      CKA_TOKEN             True
      CKA_LABEL             [size : 0xE (14)]
      696E666F 40756E69 786F732E 6465           /* es ist das 2. CA-Zertifikat */
      i n f o  @ u n i  x o s .  d e
Returned:  0 CKR_OK

/* weitere Informationen zum Objekt 2 (1. Nutzer-Zertifikat) ermitteln */
82: C_GetAttributeValue 16:04:25 [12.06.09]     /* Grösse */
  [in] hSession = 0x1
  [in] hObject = 0x2
  [in] pTemplate[10]:
      CKA_CLASS             requested with 0 buffer
      CKA_TOKEN             requested with 0 buffer
      CKA_LABEL             requested with 0 buffer
      CKA_CERTIFICATE_TYPE  requested with 0 buffer
      CKA_ID                requested with 0 buffer
      CKA_VALUE             requested with 0 buffer
      CKA_ISSUER            requested with 0 buffer
      CKA_SERIAL_NUMBER     requested with 0 buffer
      CKA_SUBJECT           requested with 0 buffer
      CKA_NETSCAPE_EMAIL(Netsc)                     requested with 0 buffer
  [out] pTemplate[10]:
      CKA_CLASS             has size 4
      CKA_TOKEN             has size 1
      CKA_LABEL             has size 29
      CKA_CERTIFICATE_TYPE  has size 4
      CKA_ID                has size 16
      CKA_VALUE             has size 1320
      CKA_ISSUER            has size 154
      CKA_SERIAL_NUMBER     has size 1
      CKA_SUBJECT           has size 146
      CKA_NETSCAPE_EMAIL(Netsc) has size -1   /* nicht vorhanden */
Returned:  18 CKR_ATTRIBUTE_TYPE_INVALID      /* mind. ein Attribut ist nicht enthalten */

83: C_GetAttributeValue 16:04:25 [12.06.09]   /* trotzdem Inhalt abrufen */
  ...
  [out] pTemplate[10]:
      CKA_CLASS             CKO_CERTIFICATE
      CKA_TOKEN             True
      CKA_LABEL             [size : 0x1D (29)]
      77656665 6C40696E 666F726D 6174696B 2E756E69 2D68616C 6C652E64 65
      w e f e  l @ i n  f o r m  a t i k  . u n i  - h a l  l e . d  e
      CKA_CERTIFICATE_TYPE  CKC_X_509             /* X.509-Zertifikat */
      CKA_ID                [size : 0x10 (16)]    /* Zertifikats-ID */
      95788AA8 2952884C 2F2FD663 344A9862
      CKA_VALUE             [size : 0x528 (1320)] /* Zertifikat (binär) */
      30820524 3082040C A0030201 02020118 300D0609 2A864886 F70D0101 05050030
      ...
      CKA_ISSUER            [size : 0x93 (147)]   /* Subject Aussteller */
      30819031 0B300906 03550406 13024445 3110300E 06035504 08130747 45524D41
      ...
      DN: C=DE, ST=GERMANY, L=Halle, O=MLU, OU=UZI, CN=ca.uzi.uni-halle.de/
```

```
          emailAddress=wefel@uzi.uni-halle.de
      CKA_SERIAL_NUMBER     [size : 0x1 (1)]      /* Seriennummer  */
      18
      CKA_SUBJECT           [size : 0x8B (139)]   /* Subject Inhaber */
      30818831 0B300906 03550406 13024445 310C300A 06035504 0A13034D 4C55310F
      ...
      DN: C=DE, O=MLU, OU=people, CN=Sandro Wefel/emailAddress=sandro.wefel@informatik.uni-
          halle.de/x500UniqueIdentifier=wefel
      CKA_NETSCAPE_EMAIL(Netsc)   has size -1
Returned:  18 CKR_ATTRIBUTE_TYPE_INVALID

84: C_GetAttributeValue 16:04:25 [12.06.09]   /* erneute Anfrage nach CKA_NETSCAPE_EMAIL */
  [in] hSession = 0x1
  [in] hObject = 0x2
  [in] pTemplate[1]:
      CKA_NETSCAPE_EMAIL(Netsc) requested with 0 buffer
  [out] pTemplate[1]:
      CKA_NETSCAPE_EMAIL(Netsc)   has size -1
Returned:  18 CKR_ATTRIBUTE_TYPE_INVALID      /* weiterhin nicht enthalten */

85: C_GetAttributeValue 16:04:25 [12.06.09]   /* analog Schritt 84 */

/* Wiederholung der Schritte 82 bis 85 fuer Objekt 4 (1. CA-Zertifikat)  */
86: C_GetAttributeValue 16:04:25 [12.06.09]
87: C_GetAttributeValue 16:04:25 [12.06.09]
  ...
  [out] pTemplate[10]:
      CKA_CLASS             CKO_CERTIFICATE
      CKA_TOKEN             True
      CKA_LABEL             [size : 0x16 (22)]
      w e f e  l @ u z  i . u n  i - h a  l l e .  d e
      CKA_CERTIFICATE_TYPE  CKC_X_509
      CKA_ID                [size : 0x10 (16)]
      58C3B6EE 15E8D658 194B9A0F 72E52128
      CKA_VALUE             [size : 0x492 (1170)]
      ...
      CKA_ISSUER            [size : 0x93 (147)]
      ...
      DN: C=DE, ST=GERMANY, L=Halle, O=MLU, OU=UZI, CN=ca.uzi.uni-halle.de/
          emailAddress=wefel@uzi.uni-halle.de
      CKA_SERIAL_NUMBER     [size : 0x1 (1)]
      00                                          /* CA-Zertifikat */
      CKA_SUBJECT           [size : 0x93 (147)]
      ...                                         /* Aussteller gleich Inhaber */
      DN: C=DE, ST=GERMANY, L=Halle, O=MLU, OU=UZI, CN=ca.uzi.uni-halle.de/
          emailAddress=wefel@uzi.uni-halle.de
      CKA_NETSCAPE_EMAIL(Netsc)   has size -1
Returned:  18 CKR_ATTRIBUTE_TYPE_INVALID

88: C_GetAttributeValue 16:04:25 [12.06.09]
89: C_GetAttributeValue 16:04:25 [12.06.09]

/* Wiederholung der Schritte 82 bis 85 fuer Objekt 7 (2. Nutzer-Zertifikat)  */
90: C_GetAttributeValue 16:04:25 [12.06.09]
91: C_GetAttributeValue 16:04:25 [12.06.09]
  ...
  [out] pTemplate[10]:
      CKA_CLASS             CKO_CERTIFICATE
      CKA_TOKEN             True
      CKA_LABEL             [size : 0xF (15)]
      77656665 6C40756E 69786F73 2E6465
      w e f e  l @ u n  i x o s  . d e
      CKA_CERTIFICATE_TYPE  CKC_X_509
      CKA_ID                [size : 0x10 (16)]
      7FEC9E61 D1615DAE 00AECF38 C9EDEE16
      CKA_VALUE             [size : 0x4D0 (1232)]
```

```
      ...
      CKA_ISSUER            [size : 0x8E (142)]
      ...
      DN: C=DE, ST=GERMANY, L=Halle, O=UnixOS GbR, OU=Webcenter, CN=unixos.de/
          emailAddress=info@unixos.de
      CKA_SERIAL_NUMBER     [size : 0x1 (1)]
      15
      CKA_SUBJECT           [size : 0x6B (107)]
      ...
      DN: C=DE, O=UnixOS GbR, CN=Sandro Wefel/emailAddress=wefel@unixos.de/
          x500UniqueIdentifier=wefel
      CKA_NETSCAPE_EMAIL(Netsc)   has size -1
Returned:  18 CKR_ATTRIBUTE_TYPE_INVALID

92: C_GetAttributeValue 16:04:25 [12.06.09]
93: C_GetAttributeValue 16:04:25 [12.06.09]

/* Wiederholung der Schritte 82 bis 85 für Objekt 9 (2. CA-Zertifikat)  */
94: C_GetAttributeValue 16:04:25 [12.06.09]
95: C_GetAttributeValue 16:04:25 [12.06.09]
  [out] pTemplate[10]:
      CKA_CLASS             CKO_CERTIFICATE
      CKA_TOKEN             True
      CKA_LABEL             [size : 0xE (14)]
      696E666F 40756E69 786F732E 6465
      i n f o  @ u n i  x o s .  d e
      CKA_CERTIFICATE_TYPE  CKC_X_509
      CKA_ID                [size : 0x10 (16)]
      DD64F6A4 D353E8EE 457E9E05 617239FA
      CKA_VALUE             [size : 0x483 (1155)]
      ...
      CKA_ISSUER            [size : 0x8E (142)]
      ...
      DN: C=DE, ST=GERMANY, L=Halle, O=UnixOS GbR, OU=Webcenter, CN=unixos.de/
          emailAddress=info@unixos.de
      CKA_SERIAL_NUMBER     [size : 0x1 (1)]
      00
      CKA_SUBJECT           [size : 0x8E (142)]
      ...
      DN: C=DE, ST=GERMANY, L=Halle, O=UnixOS GbR, OU=Webcenter, CN=unixos.de/
          emailAddress=info@unixos.de
      CKA_NETSCAPE_EMAIL(Netsc)   has size -1
Returned:  18 CKR_ATTRIBUTE_TYPE_INVALID

96: C_GetAttributeValue 16:04:25 [12.06.09]
97: C_GetAttributeValue 16:04:25 [12.06.09]

98: C_GetSlotInfo       16:04:25 [12.06.09] /* Status Slot 1 prüfen, analog Schritt 68 */
99: C_GetSessionInfo    16:04:25 [12.06.09] /* Status Session 2 prüfen (Slot 1), */
100: C_GetSessionInfo   16:04:25 [12.06.09] /* analog Schritt 69+70 */
  ======> Modifikation analog Schritt 50: CKS_RO_PUBLIC_SESSION => CKS_RO_USER_FUNCTIONS

101: C_FindObjectsInit  16:04:25 [12.06.09] /* Suche nach Zertifikaten in Session 2 */
  [in] hSession = 0x2
  [in] pTemplate[2]:
      CKA_TOKEN             True
      CKA_CLASS             CKO_CERTIFICATE
Returned:  0 CKR_OK

102: C_FindObjects        16:04:25 [12.06.09] /* Suche durchführen */
  [in] hSession = 0x2
  [in] ulMaxObjectCount = 0xa
  [out] ulObjectCount = 0x0                  /* keine Zertifikate gefunden */
Returned:  0 CKR_OK

103: C_FindObjectsFinal   16:04:25 [12.06.09]
```

```
/* Schritt 104 - 117: Prüfung der restlichen 14 Slots */
104: C_GetSlotInfo        16:04:25 [12.06.09]
  ...
117: C_GetSlotInfo        16:04:25 [12.06.09]
  ...
118: C_FindObjectsInit    16:04:25 [12.06.09] /* Session 1: Suche CKO_NETSCAPE_TRUST Obj. */
  [in] hSession = 0x1
  [in] pTemplate[2]:
      CKA_TOKEN             True
      CKA_CLASS             CKO_NETSCAPE_TRUST
Returned:  0 CKR_OK

119: C_FindObjects        16:04:25 [12.06.09]
  [in] hSession = 0x1
  [in] ulMaxObjectCount = 0xa
  [out] ulObjectCount = 0x0                  /* keine Objekte gefunden */
Returned:  0 CKR_OK

120: C_FindObjectsFinal   16:04:25 [12.06.09]
  ...

121: C_FindObjectsInit    16:04:25 [12.06.09] /* Wiederholung für Session 2 */
122: C_FindObjects        16:04:25 [12.06.09]
123: C_FindObjectsFinal   16:04:25 [12.06.09] /* ebenfalls keine Objekte gefunden */
  ...

/* ID und CLASS von Objekt 4 (1. CA Zertifikat) erfragen */
124: C_GetAttributeValue  16:04:25 [12.06.09]   /* Grösse */
125: C_GetAttributeValue  16:04:25 [12.06.09]   /* Inhalt */
  ...
  [out] pTemplate[2]:
      CKA_ID                [size : 0x10 (16)]
      58C3B6EE 15E8D658 194B9A0F 72E52128
      CKA_CLASS             CKO_CERTIFICATE /* es handelt sich um ein Zertifikat */
Returned:  0 CKR_OK

/* Suche nach privaten Objekten in Session 1 starten */
126: C_FindObjectsInit    16:04:25 [12.06.09]
  [in] hSession = 0x1
  [in] pTemplate[2]:
      CKA_ID                [size : 0x10 (16)]  /* zur benannten ID (CA-Zertifikat 1) */
      58C3B6EE 15E8D658 194B9A0F 72E52128
      CKA_CLASS             CKO_PRIVATE_KEY     /* suche privaten Schlüssel */
    C_FindObjectsInit: not logged in and requested CKO_PRIVATE_KEY
    C_FindObjectsInit: do autologin
! Hierbei handelt es sich um eine eingeschobene Funktion, die nicht von Thunderbird
! aufgerufen wird.
! Sie ist notwendig, da der reale Status der Session 1 auf CKS_RO_PUBLIC_SESSION
! steht, der Applikation allerdings der Nutzer-Status CKS_RO_USER_FUNCTIONS vermittelt
! wurde. Die Applikation startet daraufhin eine Suche nach privaten Schlüsseln
! ohne Login. Ohne diese vorherige Modifikation würde eine PIN-Abfrage durch
! die Applikation stattfinden.
! Das Login wird für die erlaubte Applikation mit der PIN aus dem Agenten
! automatisch von der Proxy-Bibliothek durchgeführt.
  127: Login_WithoutPin   16:04:25 [12.06.09]
    [in] hSession = 0x1
    [in] userType = CKU_USER
      Label: Testcard 2 (PIN)
      ManufacturerID Siemens AG (C)
      Model PKCS#15
      SerialNumber
    Returned:  0 CKR_OK
      C_FindObjectsInit: autologin finished
Returned:  0 CKR_OK
```

```
128: C_FindObjects        16:04:25 [12.06.09]
  [in] hSession = 0x1
  [in] ulMaxObjectCount = 0x1
  [out] ulObjectCount = 0x0   /* kein privater Schlüssel zum CA-Zertifikat */
Returned:  0 CKR_OK           /* gefundenes. Somit ist es kein CA-Token. */

129: C_FindObjectsFinal   16:04:25 [12.06.09]

/* Session 2 (Slot 1): Suche Zertifikats-Rückruf-Listen (CRL) zum Subject
  von CA-Zertifikat 1 */
130: C_FindObjectsInit    16:04:25 [12.06.09]
  [in] hSession = 0x2
  [in] pTemplate[2]:
      CKA_CLASS             CKO_NETSCAPE_CRL
      CKA_SUBJECT           [size : 0x93 (147)]
      ...
      DN: C=DE, ST=GERMANY, L=Halle, O=MLU, OU=UZI, CN=ca.uzi.uni-halle.de/
          emailAddress=wefel@uzi.uni-halle.de
Returned:  0 CKR_OK

131: C_FindObjects        16:04:25 [12.06.09]
  [in] hSession = 0x2
  [in] ulMaxObjectCount = 0xa
  [out] ulObjectCount = 0x0                  /* keine CRL gefunden */
Returned:  0 CKR_OK

132: C_FindObjectsFinal   16:04:25 [12.06.09]

/* Session 1 (Slot 0): suche Zertifikats-Rückruf-Listen (CRL) analog Schritt 130-132 */
133: C_FindObjectsInit    16:04:25 [12.06.09]
[in] hSession = 0x1
[in] pTemplate[2]:
    ...
    DN: C=DE, ST=GERMANY, L=Halle, O=MLU, OU=UZI, CN=ca.uzi.uni-halle.de/
        emailAddress=wefel@uzi.uni-halle.de
Returned:  0 CKR_OK

134: C_FindObjects        16:04:25 [12.06.09]
  ...
  [out] ulObjectCount = 0x0                  /* keine CRL gefunden */
Returned:  0 CKR_OK

135: C_FindObjectsFinal

136: C_FindObjectsInit    16:04:25 [12.06.09] /* Session 2: Andere CRLs? */
  [in] hSession = 0x2
  [in] pTemplate[1]:
      CKA_CLASS             CKO_NETSCAPE_CRL
Returned:  0 CKR_OK

137: C_FindObjects        16:04:25 [12.06.09]
  [in] hSession = 0x2
  [in] ulMaxObjectCount = 0xa
  [out] ulObjectCount = 0x0                  /* keine CRL gefunden */
Returned:  0 CKR_OK

138: C_FindObjectsFinal   16:04:25 [12.06.09]

139: C_FindObjectsInit    16:04:25 [12.06.09] /* Wiederholung für Session 1 */
140: C_FindObjects        16:04:25 [12.06.09] /* keine CRL gefunden */
141: C_FindObjectsFinal   16:04:25 [12.06.09]
```

Nach Abschluss der Initialisierung sind Thunderbird die Zertifikate bekannt. Thunderbird kann das Token für Authentifizierung, Signatur und Entschlüsselung nutzen.

Der Email-Abruf vom Server erfordert eine Anmeldung. Auf dem Token muss das passende Zertifikat und der zugehörige private Schlüssel für die clientseitige Authentifizierung enthalten sein. Die folgenden Abschnitte umfassen die PKCS#11-Operationen während der Anmeldung an einem IMAP-Server.

**a) Ermittlung der zur Authentisierung nutzbaren Zertifikate:** Bei Aufbau der SSL-Verbindung mit zertifikatsbasierter Client-Authentisierung erfragt der Server ein Client-Zertifikat. Dazu wird von Thunderbird im Token nach einem Zertifikat mit zugehörigen privaten Schlüssel gesucht.
Bei Aufbau der Verbindung legt der Server die Zertifikate in der Zertifikatskette vor, die dem Client zur Prüfung der Vertrauenswürdigkeit dienen. Hierzu nutzt der Client einen lokalen Zertifikatsspeicher oder einen OCSP-Dienst. Zur vereinfachten Suche nach dem Zertifikat in diesem lokalen Speicher bzw. in der Sperrliste werden Zertifikate über ihre Hashwerte indiziert. Die Berechnung dieses Wertes lagert die Applikation bei Vorhandensein eines Tokens an den PKCS#11-Provider aus, indem eine Digest-Operation des Providers (Schritt 163-165) aufgerufen wird. Schlägt die anschließende Prüfung des Zertifikats fehl, würde der Client nach Schritt 166 die Verbindung zum Server beenden.

```
142: C_GetSlotInfo        16:04:48 [12.06.09] /* Abfrage Status Slot 0 */
143: C_GetSessionInfo     16:04:48 [12.06.09] /* Abfrage Status Session 1 */
144: C_GetSessionInfo     16:04:48 [12.06.09]
  ======> keine Modifikation da im Status CKS_RO_USER_FUNCTIONS (Autologin Schritt 127)

145: C_GetSlotInfo        16:04:48 [12.06.09] /* Abfrage Status Slot 1 */
146: C_GetSessionInfo     16:04:48 [12.06.09]
147: C_GetSessionInfo     16:04:48 [12.06.09] /* Abfrage Status Session 2 */
  ======> Modifikation analog Schritt 50: CKS_RO_PUBLIC_SESSION => CKS_RO_USER_FUNCTIONS

148: C_GetSlotInfo        16:04:48 [12.06.09] /* Wiederholung für Slot 2 bis 15 */
  ...
161: C_GetSlotInfo        16:04:48 [12.06.09]

162: C_OpenSession        16:04:48 [12.06.09] /* weitere Session (3) zu Slot 1 öffnen */
  [in] slotID = 0x1
  [in] flags = 0x4
    pApplication=0xb2690570
    Notify=0xb233aab0
  [out] *phSession = 0x3
    Nr. of sessions: 3    ======> Session Counter inkrementieren
Returned:  0 CKR_OK

163: C_DigestInit         16:04:48 [12.06.09]   /* Session 3: Start der Digest-Operation */
  [in] hSession = 0x3
  pMechanism->type=CKM_SHA256                  /* Algorithmus SHA-256 */
Returned:  0 CKR_OK

164: C_DigestUpdate       16:04:48 [12.06.09]
  [in] hSession = 0x3
  [in] pPart[ulPartLen] [size : 0x4A6 (1190)]
      308204A2 3082038A A0030201 0202011E 300D0609 2A864886 F70D0101 05050030
      ...
      87A94F87 E47F
Returned:  0 CKR_OK

165: C_DigestFinal        16:04:48 [12.06.09]   /* Ergebnis abholen */
  [in] hSession = 0x3
  [out] pDigest[*pulDigestLen] [size : 0x20 (32)]
      DA5DB9D4 51B03EAF 3C747BA6 1F7088F8 B2269670 6226CD13 D8EAF2FC 83482863
Returned:  0 CKR_OK

166: C_CloseSession       16:04:48 [12.06.09]   /* Session 3 schliessen */
  [in] hSession = 0x3
    Nr. of sessions: 2    ======> Session Counter dekrementieren
Returned:  0 CKR_OK
```

```
/* Session 1 (Slot 0) Zertifikate suchen und Zertifikats-Attribute abrufen. */
167: C_FindObjectsInit    16:04:48 [12.06.09]   /* Dies entspricht Schritt 71 bis 81 */
  [in] hSession = 0x1
  [in] pTemplate[2]:
       CKA_TOKEN              True
       CKA_CLASS              CKO_CERTIFICATE
Returned:  0 CKR_OK
...
177: C_GetAttributeValue  16:04:48 [12.06.09]

/* Session 2 (Slot 1) Zertifikate suchen und Zertifikats-Attribute abrufen. */
178: C_FindObjectsInit    16:04:48 [12.06.09]   /* Dies entspricht Schritt 71 bis 81 */
  [in] hSession = 0x2
  [in] pTemplate[2]:
       CKA_TOKEN              True
       CKA_CLASS              CKO_CERTIFICATE
Returned:  0 CKR_OK
179: C_FindObjects        16:04:48 [12.06.09]   /* keine Zertifikate gefunden */
180: C_FindObjectsFinal   16:04:48 [12.06.09]


/* Klasse und ID von Objekt 4 erfragen - das 1. CA-Zertifikat */
181: C_GetAttributeValue  16:04:48 [12.06.09]
  [in] hSession = 0x1
  [in] hObject = 0x4
  [in] pTemplate[2]:
       CKA_ID                 requested with 0 buffer
       CKA_CLASS              requested with 0 buffer
  [out] pTemplate[2]:
       CKA_ID                 has size 16
       CKA_CLASS              has size 4
Returned:  0 CKR_OK

182: C_GetAttributeValue  16:04:48 [12.06.09]
  [in] hSession = 0x1
  [in] hObject = 0x4
  [in] pTemplate[2]:
       CKA_ID                 requested with 16 buffer
       CKA_CLASS              requested with 4 buffer
  [out] pTemplate[2]:
       CKA_ID                 [size : 0x10 (16)]
       58C3B6EE 15E8D658 194B9A0F 72E52128       /* die ID */
       CKA_CLASS              CKO_CERTIFICATE    /* ist Zertifikat */
Returned:  0 CKR_OK

183: C_FindObjectsInit    16:04:48 [12.06.09]   /* suche zugehörigen privaten Schlüssel */
  [in] hSession = 0x1
  [in] pTemplate[2]:
       CKA_ID                 [size : 0x10 (16)]
       58C3B6EE 15E8D658 194B9A0F 72E52128
       CKA_CLASS              CKO_PRIVATE_KEY
Returned:  0 CKR_OK
184: C_FindObjects        16:04:48 [12.06.09]
  [in] hSession = 0x1
  [in] ulMaxObjectCount = 0x1
  [out] ulObjectCount = 0x0                      /* kein Schlüssel gefunden */
Returned:  0 CKR_OK
185: C_FindObjectsFinal   16:04:48 [12.06.09]


/* Klasse und ID von Objekt 7 erfragen - das 2. Nutzer-Zertifikat */
186: C_GetAttributeValue  16:04:48 [12.06.09]
  [in] hSession = 0x1
  [in] hObject = 0x7
  ...
Returned:  0 CKR_OK
```

```
187: C_GetAttributeValue  16:04:48 [12.06.09]
  ...
  [out] pTemplate[2]:
      CKA_ID                [size : 0x10 (16)]
      7FEC9E61 D1615DAE 00AECF38 C9EDEE16
      CKA_CLASS             CKO_CERTIFICATE
Returned:  0 CKR_OK

188: C_FindObjectsInit    16:04:48 [12.06.09]   /* suche zugehörigen privaten Schlüssel */
  [in] hSession = 0x1
  [in] pTemplate[2]:
      CKA_ID                [size : 0x10 (16)]
      7FEC9E61 D1615DAE 00AECF38 C9EDEE16
      CKA_CLASS             CKO_PRIVATE_KEY
Returned:  0 CKR_OK
189: C_FindObjects        16:04:48 [12.06.09]
  [in] hSession = 0x1
  [in] ulMaxObjectCount = 0x1
  [out] ulObjectCount = 0x1                     /* ein Schlüssel gefunden */
    Object 6 Matches                            /* Schlüssel ist Objekt 6 */
Returned:  0 CKR_OK
190: C_FindObjectsFinal   16:04:48 [12.06.09]

/* Schritt 181-185 für Objekt 9 - das 2. CA-Zertifikat */
191: C_GetAttributeValue  16:04:48 [12.06.09]
192: C_GetAttributeValue  16:04:48 [12.06.09]
  ...
193: C_FindObjectsInit    16:04:48 [12.06.09]   /* passender privater Schlüssel? */
194: C_FindObjects        16:04:48 [12.06.09]
195: C_FindObjectsFinal   16:04:48 [12.06.09]   /* kein Schlüssel gefunden */

/* Schritt 181-185 für Objekt 2 - das 1. Nutzer-Zertifikat */
196: C_GetAttributeValue  16:04:48 [12.06.09]
197: C_GetAttributeValue  16:04:48 [12.06.09]
  ...
198: C_FindObjectsInit    16:04:48 [12.06.09]   /* passender privater Schlüssel? */
199: C_FindObjects        16:04:48 [12.06.09]
  [in] hSession = 0x1
  [in] ulMaxObjectCount = 0x1
  [out] ulObjectCount = 0x1                     /* ein Schlüssel gefunden */
    Object 1 Matches                            /* Schlüssel ist Objekt 1 */
Returned:  0 CKR_OK
200: C_FindObjectsFinal   16:04:48 [12.06.09]

/* Wiederholung Schritte 196 - 200 (1. Nutzer-Zertifikat) */
201: C_GetAttributeValue  16:04:48 [12.06.09]
202: C_GetAttributeValue  16:04:48 [12.06.09]
203: C_FindObjectsInit    16:04:48 [12.06.09]
204: C_FindObjects        16:04:48 [12.06.09]
205: C_FindObjectsFinal   16:04:48 [12.06.09]

/* Wiederholung Schritte 186 - 190 (2. Nutzer-Zertifikat) */
206: C_GetAttributeValue  16:04:48 [12.06.09]
207: C_GetAttributeValue  16:04:48 [12.06.09]
208: C_FindObjectsInit    16:04:48 [12.06.09]
209: C_FindObjects        16:04:48 [12.06.09]
210: C_FindObjectsFinal   16:04:48 [12.06.09]

/* Wiederholung Schritte 191 - 195 (2. CA-Zertifikat) */
211: C_GetAttributeValue  16:04:48 [12.06.09]
212: C_GetAttributeValue  16:04:48 [12.06.09]
213: C_FindObjectsInit    16:04:48 [12.06.09]
214: C_FindObjects        16:04:48 [12.06.09]
215: C_FindObjectsFinal   16:04:48 [12.06.09]
```

**b) Authentifizierung mittels einer Signaturoperation:** Die Authentifizierung erfolgt mittels Signatur des Hashwertes, der über alle bisher zwischen Client und Server ausgetauschten Nachrichten während des SSL-Verbindungsaufbaus von beiden Seiten berechnet wird. Da hierbei zufällige Werte von Server und Client einfließen, können die Nachrichten als Challenge, die Signatur mit dem privaten Nutzerschlüssel als Response bezeichnet werden.

```
216: C_GetSlotInfo         16:04:51 [12.06.09]   /* Slot 0: keine Änderung */
217: C_GetSessionInfo      16:04:51 [12.06.09]   /* Session 1: keine Änderung, */
218: C_GetSessionInfo      16:04:51 [12.06.09]   /* Status CKS_RO_USER_FUNCTIONS */
                                                 /* und es gibt 2 aktive Sessionen */

219: C_GetSlotInfo         16:04:51 [12.06.09]   /* Slot 1: keine Änderung */
220: C_GetSessionInfo      16:04:51 [12.06.09]   /* Session 1: keine Änderung, */
221: C_GetSessionInfo      16:04:51 [12.06.09]   /* Status ist CKS_RO_PUBLIC_SESSION */
  ======> Modifikation analog Schritt 50: CKS_RO_PUBLIC_SESSION => CKS_RO_USER_FUNCTIONS

222: C_GetSlotInfo         16:04:51 [12.06.09]   /* Wiederholung für Slot 2 */
...
235: C_GetSlotInfo         16:04:51 [12.06.09]   /* bis Slot 15 */

236: C_OpenSession         16:04:51 [12.06.09]   /* weitere Session (4) zu Slot 1 öffnen */
  [in] slotID = 0x1
  [in] flags = 0x4
    pApplication=0xb2690570
    Notify=0xb233aab0
  [out] *phSession = 0x4
    Nr. of sessions: 3  ======> Session Counter inkrementieren, insg. 3
Returned:  0 CKR_OK

237: C_DigestInit          16:04:51 [12.06.09]   /* erneute Digest-Operation in Session 4 */
238: C_DigestUpdate        16:04:51 [12.06.09]   /* analog Schritt 163 - 165 */
239: C_DigestFinal         16:04:51 [12.06.09]

240: C_CloseSession        16:04:51 [12.06.09]   /* Session 4 schliessen */
  [in] hSession = 0x4
    Nr. of sessions: 2  ======> Session Counter aktualisieren, insg. 2
Returned:  0 CKR_OK

241: C_FindObjectsInit     16:04:51 [12.06.09]   /* Objekt suchen */
  [in] hSession = 0x1
  [in] pTemplate[1]:                              /* gesucht wird das Zertifikat */
      CKA_VALUE             [size : 0x528 (1320)]   /* des 1. Nutzers */
    30820524 3082040C A0030201 02020118 300D0609 2A864886 F70D0101 05050030
    ...
    2E33D7AB 25AFB1F8
Returned:  0 CKR_OK

242: C_FindObjects         16:04:51 [12.06.09]   /* Objekt 2 wird gefunden */
243: C_FindObjectsFinal    16:04:51 [12.06.09]
  ...

244: C_GetAttributeValue   16:04:51 [12.06.09]   /* Abfrage CLASS, ID von Objekt 2 */
  ...                                            /* Grösse der Attribute? */
245: C_GetAttributeValue   16:04:51 [12.06.09]   /* Inhalt */
  ...
  [out] pTemplate[2]:
      CKA_ID                [size : 0x10 (16)]
      95788AA8 2952884C 2F2FD663 344A9862
      CKA_CLASS             CKO_CERTIFICATE     /* es handelt sich um ein Zertifikat */
Returned:  0 CKR_OK

246: C_FindObjectsInit     16:04:51 [12.06.09]   /* zugehöriger privater Schlüssel */
  [in] hSession = 0x1
  [in] pTemplate[2]:
      CKA_ID                [size : 0x10 (16)]
```

```
        95788AA8 2952884C 2F2FD663 344A9862
      CKA_CLASS             CKO_PRIVATE_KEY
Returned:  0 CKR_OK

247: C_FindObjects        16:04:51 [12.06.09]
  [in] hSession = 0x1
  [in] ulMaxObjectCount = 0x1
  [out] ulObjectCount = 0x1
    Object 1 Matches                          /* ist Objekt 1 */
Returned:  0 CKR_OK

248: C_FindObjectsFinal   16:04:51 [12.06.09]
  ...

249: C_GetAttributeValue  16:04:51 [12.06.09]   /* Typ des privaten Schlüssels? */
  [in] hSession = 0x1
  [in] hObject = 0x1
  [in] pTemplate[1]:
      CKA_KEY_TYPE          requested with 4 buffer
  [out] pTemplate[1]:
      CKA_KEY_TYPE          CKK_RSA             /* RSA-Schlüssel */
Returned:  0 CKR_OK

250: C_GetAttributeValue  16:04:51 [12.06.09]   /* ist er auf Token oder termporär? */
  [in] hSession = 0x1
  [in] hObject = 0x1
  [in] pTemplate[1]:
      CKA_TOKEN             requested with 1 buffer
  [out] pTemplate[1]:
      CKA_TOKEN             True                /* auf Token ! */
Returned:  0 CKR_OK

251: C_GetAttributeValue  16:04:51 [12.06.09]   /* ist es ein geschützter Schlüssel */
  [in] hSession = 0x1
  [in] hObject = 0x1
  [in] pTemplate[1]:
      CKA_PRIVATE           requested with 1 buffer
  [out] pTemplate[1]:
      CKA_PRIVATE           True                /* ja, nicht auslesbar! */
Returned:  0 CKR_OK

252: C_GetAttributeValue  16:04:51 [12.06.09]   /* Modulus-Grösse? */
  [in] hSession = 0x1
  [in] hObject = 0x1
  [in] pTemplate[1]:
      CKA_MODULUS           requested with 0 buffer
  [out] pTemplate[1]:
      CKA_MODULUS           has size 256        /* 256 Byte = 2048 Bit */
Returned:  0 CKR_OK

253: C_GetAttributeValue  16:04:51 [12.06.09]   /* Modulus? */
  [in] hSession = 0x1
  [in] hObject = 0x1
  [in] pTemplate[1]:
      CKA_MODULUS           requested with 256 buffer
  [out] pTemplate[1]:
      CKA_MODULUS           [size : 0x100 (256)]
      D3B22835 40236DF8 55B2010C 9A77A0A9 96F8C620 D2E60A71 A6F2A9B5 7DD57776
      ...
Returned:  0 CKR_OK

254: C_GetAttributeValue  16:04:51 [12.06.09]   /* Wiederholung Schritt 251 */
  ...

255: C_OpenSession        16:04:51 [12.06.09]   /* weitere Session (5) zu Slot 1 öffnen */
  [in] slotID = 0x0
```

```
  [in] flags = 0x4
    pApplication=0xb26902d0
    Notify=0xb233aab0
  [out] *phSession = 0x5
    Nr. of sessions: 3    ======> Session Counter: 3
Returned:  0 CKR_OK

/* Initialisierung einer Signatur-Operation in Session 5 */
256: C_SignInit           16:04:51 [12.06.09]
  [in] hSession = 0x5
    pMechanism->type=CKM_RSA_PKCS              /* Typ RSA mit PKCS#1v15-Padding */
  [in] hKey = 0x1                              /* Nutzung von Objekt 1, dem privaten */
Returned:  0 CKR_OK                            /* Schlüssel zum 1. Nutzer-Zertifikat */

257: C_Sign               16:04:51 [12.06.09]   /* Ausführung */
  [in] hSession = 0x5
  [in] pData[ulDataLen] [size : 0x24 (36)]     /* zu signierende Daten */
      B08DB454 29CA7295 98C64B89 B317ED11 89547669 93EB44F7 87C314E2 3C45882C
      56C0BF20
  [out] pSignature[*pulSignatureLen] [size : 0x100 (256)]
      770E9E76 9312C6E4 8A8E7AAA E5373721 46E7CA98 5B171DC5 43C08468 5DE0E5FF
      ...                                      /* erhaltene Signatur */
Returned:  0 CKR_OK

258: C_CloseSession       16:04:52 [12.06.09]   /* Session 5 schliessen */
  [in] hSession = 0x5
    Nr. of sessions: 2    ======> Session Counter aktualisieren, insg. wieder 2
Returned:  0 CKR_OK
```

Thunderbird und der damit verbundene Nutzer hat sich gegenüber dem IMAP-Server authentifiziert und kann die Daten des Posteingangs abrufen.

**Abruf von Email:** Die folgenden Schritte beschreiben die Operationen bei Abruf einer Email. Insofern der Email-Inhalt nicht verschlüsselt ist, wird lediglich auf Änderung des Sitzungsstatus (`user` ⇒ `public`) oder des Slotstatus von Slot 0 geprüft und damit erkannt, ob das Token entnommen wurde.

```
259: C_GetSlotInfo        16:05:25 [12.06.09] /* Slot 0 */
260: C_GetSessionInfo     16:05:25 [12.06.09] /* Session 1 */
261: C_GetSessionInfo     16:05:25 [12.06.09] /* Session 1 */
...

367: C_GetSlotInfo        16:06:49 [12.06.09] /* Slot 0 */
368: C_GetSessionInfo     16:06:49 [12.06.09] /* Session 1 */
369: C_GetSessionInfo     16:06:49 [12.06.09] /* Session 1 */

370: C_GetSlotInfo        16:07:12 [12.06.09] /* Slot 0 */
371: C_GetSessionInfo     16:07:12 [12.06.09] /* Session 1 */
372: C_GetSessionInfo     16:07:12 [12.06.09] /* Session 1 */
```

**Reauthentifizierung, Signatur:** Eine erneute Anmeldung, die für den Zugriff auf einen weiteren IMAP-Unterordner erforderlich ist, läuft nach einem analogen Schema ab.

```
373: C_GetSlotInfo        16:07:12 [12.06.09] /* Slot 1 */
374: C_GetSessionInfo     16:07:12 [12.06.09] /* Session 2 */
375: C_GetSessionInfo     16:07:12 [12.06.09] /* Session 2 */
  ======> Modifikation analog Schritt 50: CKS_RO_PUBLIC_SESSION => CKS_RO_USER_FUNCTIONS

376: C_GetSlotInfo        16:07:12 [12.06.09] /* Slot 2 */
...
389: C_GetSlotInfo        16:07:12 [12.06.09] /* bis 15 */

390: C_OpenSession        16:07:12 [12.06.09] /* weitere Session (6) zu Slot 1 öffnen */
  [in] slotID = 0x1
  [in] flags = 0x4
    pApplication=0xb2690570
    Notify=0xb233aab0
  [out] *phSession = 0x6
    Nr. of sessions: 3  ======> Session Counter aktualisieren, insg. 3
Returned:  0 CKR_OK

391: C_DigestInit         16:07:12 [12.06.09] /* Digest-Operation in Session 6 */
392: C_DigestUpdate       16:07:12 [12.06.09] /* analog Schritt 163-165 */
393: C_DigestFinal        16:07:12 [12.06.09]

394: C_CloseSession       16:07:12 [12.06.09] /* Session 6 schliessen */
  [in] hSession = 0x6
    Nr. of sessions: 2  ======> Session Counter aktualisieren, insg. 3
Returned:  0 CKR_OK

/* Attribute vin Objekt 2 (Nutzer-Zertifikat 1), zugehörigen privaten Schlüssel, etc. ? */
395: C_GetAttributeValue  16:07:12 [12.06.09] /* analog Schritt 244-254 */
396: C_GetAttributeValue  16:07:12 [12.06.09]
397: C_FindObjectsInit    16:07:12 [12.06.09] /* privater Schlüssel? */
398: C_FindObjects        16:07:12 [12.06.09]
399: C_FindObjectsFinal   16:07:12 [12.06.09]
400: C_GetAttributeValue  16:07:12 [12.06.09] /* Schlüssel-Typ? */
401: C_GetAttributeValue  16:07:12 [12.06.09] /* Token-Objekt? */
402: C_GetAttributeValue  16:07:12 [12.06.09] /* privat? */
403: C_GetAttributeValue  16:07:12 [12.06.09] /* Modulus-Grösse */
404: C_GetAttributeValue  16:07:12 [12.06.09] /* Modulus */
405: C_GetAttributeValue  16:07:12 [12.06.09] /* privat? */

406: C_OpenSession        16:07:12 [12.06.09]  /* weitere Session (7) zu Slot 1 öffnen */
  [in] slotID = 0x0
  [in] flags = 0x4
    pApplication=0xb26902d0
    Notify=0xb233aab0
  [out] *phSession = 0x7
    Nr. of sessions: 3  ======> Session Counter aktualisieren, insg. 3
Returned:  0 CKR_OK

407: C_SignInit           16:07:12 [12.06.09] /* Signatur analog Schritt 256-257 */
408: C_Sign               16:07:12 [12.06.09]

409: C_CloseSession       16:07:13 [12.06.09] /* Session 7 schliessen */
  [in] hSession = 0x7
    Nr. of sessions: 2  ======> Session Counter aktualisieren, insg. 2
Returned:  0 CKR_OK
```

Diese Schritte werden werden für jede Authentifizierung genutzt. Sie werden auch ausgeführt, wenn eine Email mit dem 1. Nutzer-Zertifikat elektronisch signiert wird.

**Applikationsende, Tokenentnahme:** Bei Beendigung schließt Thunderbird alle Sessions. Der Zähler wird in diesem Falle zurückgesetzt.

**Entschlüsselung:** Die folgende eingekürzte Aufzeichnung aus einer anderen Applikations-Sitzung zeigt die Entschlüsselung einer mittels S/MIME verschlüsselten Nachricht mit Hilfe des Tokens. Die Nachricht wurde in einer Email im PKCS#7-Format eingebettet.

```
275: C_GetSessionInfo     15:34:06 [07.08.09]
276: C_GetSessionInfo     15:34:06 [07.08.09] /* Session 1 */

277: C_GetSlotInfo        15:34:06 [07.08.09] /* Slot 1 */
278: C_GetSessionInfo     15:34:06 [07.08.09] /* Session 2 */
279: C_GetSessionInfo     15:34:06 [07.08.09]

280: C_GetSlotInfo        15:34:06 [07.08.09] /* Slot 2 bis */
293: C_GetSlotInfo        15:34:06 [07.08.09] /* Slot 15 */

294: C_FindObjectsInit    15:34:06 [07.08.09] /* Session 1: Zertifikate suchen */
295: C_FindObjects        15:34:06 [07.08.09] /* 4 Zertifikate gefunden (analog Schritt */
296: C_FindObjectsFinal   15:34:06 [07.08.09] /* 71 ff. aus der ersten Aufzeichnung) */

297: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 2: TOKEN + LABEL */
298: C_GetAttributeValue  15:34:06 [07.08.09]

299: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 4: TOKEN + LABEL */
300: C_GetAttributeValue  15:34:06 [07.08.09]

301: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 7: TOKEN + LABEL */
302: C_GetAttributeValue  15:34:06 [07.08.09]

303: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 9: TOKEN + LABEL */
304: C_GetAttributeValue  15:34:06 [07.08.09]

305: C_FindObjectsInit    15:34:06 [07.08.09] /* Session 2: Zertifikate suchen */
306: C_FindObjects        15:34:06 [07.08.09] /* keine Zertifikate gefunden */
307: C_FindObjectsFinal   15:34:06 [07.08.09]

308: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 4: ID + CLASS */
309: C_GetAttributeValue  15:34:06 [07.08.09] /* => ID des 1. CA-Zertifikats */
310: C_FindObjectsInit    15:34:06 [07.08.09] /* privater Schlüssel zur ID */
311: C_FindObjects        15:34:06 [07.08.09] /* 0 Schlüssel gefunden */
312: C_FindObjectsFinal   15:34:06 [07.08.09]

313: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 2: ID + CLASS */
314: C_GetAttributeValue  15:34:06 [07.08.09] /* => ID des 1. Nutzer-Zertifikats */
315: C_FindObjectsInit    15:34:06 [07.08.09] /* privater Schlüssel zur ID */
316: C_FindObjects        15:34:06 [07.08.09] /* 1 Schlüssel gefunden (Objekt 1) */
317: C_FindObjectsFinal   15:34:06 [07.08.09]

318: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 7: ID + CLASS */
319: C_GetAttributeValue  15:34:06 [07.08.09] /* => ID des 2. Nutzer-Zertifikats */
320: C_FindObjectsInit    15:34:06 [07.08.09] /* privater Schlüssel zur ID */
321: C_FindObjects        15:34:06 [07.08.09] /* 1 Schlüssel gefunden (Objekt 6) */
322: C_FindObjectsFinal   15:34:06 [07.08.09]

323: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 7, erneut ID + CLASS */
324: C_GetAttributeValue  15:34:06 [07.08.09]
325: C_FindObjectsInit    15:34:06 [07.08.09] /* erneut privater Schlüssel zu Obj. 7 */
326: C_FindObjects        15:34:06 [07.08.09] /* => Objekt 6 */
327: C_FindObjectsFinal   15:34:06 [07.08.09]

328: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 9: ID + CLASS */
329: C_GetAttributeValue  15:34:06 [07.08.09] /* => ID des 2. CA-Zertifikats */
330: C_FindObjectsInit    15:34:06 [07.08.09] /* privater Schlüssel zur ID */
331: C_FindObjects        15:34:06 [07.08.09] /* 0 Schlüssel gefunden */
332: C_FindObjectsFinal   15:34:06 [07.08.09]
```

```
333: C_FindObjectsInit    15:34:06 [07.08.09] /* Session 2: Zertifikat anhand */
  [in] hSession = 0x2                         /* Aussteller und Seriennummer suchen */
  [in] pTemplate[4]:                          /* dieser Aussteller hat das */
      CKA_TOKEN             True              /* 1. Nutzer-Zertifikat signiert */
      CKA_CLASS             CKO_CERTIFICATE
      CKA_ISSUER            [size : 0x93 (147)]
        30819031 0B300906 03550406 13024445 3110300E 06035504 08130747 45524D41
        ...
        DN: C=DE, ST=GERMANY, L=Halle, O=MLU, OU=UZI, CN=ca.uzi.uni-halle.de/
            emailAddress=wefel@uzi.uni-halle.de
      CKA_SERIAL_NUMBER     [size : 0x3 (3)]
        020118                                /* Format 0x0201-18 */
Returned:  0 CKR_OK
334: C_FindObjects        15:34:06 [07.08.09] /* kein Objekt in Sitzung 2 gefunden */
335: C_FindObjectsFinal   15:34:06 [07.08.09]

336: C_FindObjectsInit    15:34:06 [07.08.09] /* Session 2: Zertifikat erneut anhand */
        ...
      CKA_SERIAL_NUMBER     [size : 0x1 (1)]
        18                                    /* diesmal Format 0x18 */
Returned:  0 CKR_OK
337: C_FindObjects        15:34:06 [07.08.09] /* trotzdem kein Objekt in Sitzung 2 */
338: C_FindObjectsFinal   15:34:06 [07.08.09] /* gefunden */

339: C_FindObjectsInit    15:34:06 [07.08.09] /* Session 2: Zertifikat erneut suchen */
340: C_FindObjects        15:34:06 [07.08.09] /* mit anderem Format der Seriennummer */
341: C_FindObjectsFinal   15:34:06 [07.08.09] /* trotzdem nicht in Sitzung 2 gefunden */

342: C_FindObjectsInit    15:34:06 [07.08.09] /* Wiederholung Schritt 339-341 */
343: C_FindObjects        15:34:06 [07.08.09] /* kein passendes Objekt gefunden */
344: C_FindObjectsFinal   15:34:06 [07.08.09]

345: C_FindObjectsInit    15:34:06 [07.08.09] /* Zertifikat suchen, diesmal in Sitzung 1 */
  [in] hSession = 0x1
  [in] pTemplate[4]:
      ...
      CKA_ISSUER            [size : 0x93 (147)]
        30819031 0B300906 03550406 13024445 3110300E 06035504 08130747 45524D41
        ...
        DN: C=DE, ST=GERMANY, L=Halle, O=MLU, OU=UZI, CN=ca.uzi.uni-halle.de/
            emailAddress=wefel@uzi.uni-halle.de
      CKA_SERIAL_NUMBER     [size : 0x3 (3)]
        020118                                /* Format 0x0201-18 */
Returned:  0 CKR_OK
346: C_FindObjects        15:34:06 [07.08.09] /* nicht gefunden */
347: C_FindObjectsFinal   15:34:06 [07.08.09]

348: C_FindObjectsInit    15:34:06 [07.08.09] /* Sitzung 1, Wiederholung mit */
  [in] hSession = 0x1                         /* geandertem Format der Seriennummer */
  [in] pTemplate[4]:
      ...
      CKA_ISSUER            [size : 0x93 (147)]
        30819031 0B300906 03550406 13024445 3110300E 06035504 08130747 45524D41
        ...
        DN: C=DE, ST=GERMANY, L=Halle, O=MLU, OU=UZI, CN=ca.uzi.uni-halle.de/
            emailAddress=wefel@uzi.uni-halle.de
      CKA_SERIAL_NUMBER     [size : 0x1 (1)]
        18                                    /* diesmal Format 0x18 */
Returned:  0 CKR_OK
349: C_FindObjects        15:34:06 [07.08.09] /* gefunden => Objekt 2 */
  [in] hSession = 0x1
  [in] ulMaxObjectCount = 0x1
  [out] ulObjectCount = 0x1
    Object 2 Matches
Returned:  0 CKR_OK
350: C_FindObjectsFinal   15:34:06 [07.08.09]
```

```
351: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 2: TOKEN-Objekt + LABEL ? */
352: C_GetAttributeValue  15:34:06 [07.08.09]
    ...
  [out] pTemplate[2]:
      CKA_TOKEN             True
      CKA_LABEL             [size : 0x1D (29)]
      77656665 6C40696E 666F726D 6174696B 2E756E69 2D68616C 6C652E64 65
      w e f e   l @ i n   f o r m   a t i k   . u n i   - h a l   l e . d   e
Returned:  0 CKR_OK

353: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 2: ID + CLASS */
354: C_GetAttributeValue  15:34:06 [07.08.09] /*
355: C_FindObjectsInit    15:34:06 [07.08.09] /* privater Schlüssel zur ID */
356: C_FindObjects        15:34:06 [07.08.09] /* 1 Schlüssel gefunden (Objekt 1) */
357: C_FindObjectsFinal   15:34:06 [07.08.09]

358: C_GetAttributeValue  15:34:06 [07.08.09] /* Wiederholung Schritt 353-357 */
359: C_GetAttributeValue  15:34:06 [07.08.09]
360: C_FindObjectsInit    15:34:06 [07.08.09]
361: C_FindObjects        15:34:06 [07.08.09]
362: C_FindObjectsFinal   15:34:06 [07.08.09]

363: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 1: KEY_TYPE? => RSA */
364: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 1: auf TOKEN? => ja */
365: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 1: PRIVATE? => ja */
366: C_GetAttributeValue  15:34:06 [07.08.09] /* Objekt 1: PRIVATE? => ja */

367: C_GetMechanismInfo   15:34:06 [07.08.09] /* Informationen zum Token */
  [in] slotID = 0x0                           /* in Slot 0 */
    CKM_RSA_PKCS                              /* über den Mechanismus RSA */
  [out] pInfo:                                /* mit PKCS#1v15 Padding ?*/
    CKM_RSA_PKCS                              /* => das Token unterstützt */
      : min:512 max:2048                      /* RSA_PKCS mit Schlüsselleange zwischen*/
      : flags:0x42A01 ( Hardware Decrypt Sign Verify Unwrap )
Returned:  0 CKR_OK                           /* 512-2048Bit für angegebene Operationen */
                                              /* u.a. Entschlüsselung */

368: C_OpenSession        15:34:06 [07.08.09] /* Start Session */
  [in] slotID = 0x0
  [in] flags = 0x4
    pApplication=0xaa97e08
    Notify=0xb25deab0
  [out] *phSession = 0x6                      /* Nr. 6 */
    Nr. of sessions: 3   ======> Session Counter inkrementieren
Returned:  0 CKR_OK

/* Initialisierung der Entschlüsselungsoperation in Session 6 */
369: C_DecryptInit        15:34:06 [07.08.09]
  [in] hSession = 0x6
    pMechanism->type=CKM_RSA_PKCS
  [in] hKey = 0x1
Returned:  0 CKR_OK

370: C_Decrypt            15:34:06 [07.08.09] /* Entschlüsselung eines */
  [in] hSession = 0x6                         /* 256 Byte - Wertes */
  [in] pEncryptedData[ulEncryptedDataLen] [size : 0x100 (256)]
      560799C2 71854045 D03BF668 282B5346 FCEE8F9D DB922865 AB3E562E 9671E23F
      ...
  [out] pData[*pulDataLen] [size : 0x18 (24)] /* das Ergebnis ist nach Entschlüsselung */
      98CE19E3 ...                            /* und Entfernen des Paddings 24 Byte */
Returned:  0 CKR_OK

371: C_CloseSession       15:34:07 [07.08.09]
  [in] hSession = 0x6
    Nr. of sessions: 2  ======> Session Counter dekrementieren
Returned:  0 CKR_OK
```

Der Kern der Entschlüsselung beginnt im Schritt 333. Die empfangene Email enthält einen verschlüsselten Block, welcher in der PKCS#7-Syntax formatiert wurde. Entsprechend der PKCS#7-Spezifikation ([RFC2315], Abschnitt 10.2 RecipientInfo) werden für jeden Empfänger, für den die Nachricht verschlüsselt wurde, Aussteller und Seriennummer des Zertifikats im Header der Nachricht eingebettet. In diesem Falle handelt es sich nur um einen Empfänger. Auf dem Token wird ein passendes (Nutzer-)Zertifikat mit diesen Daten gesucht und im Schritt 349 gefunden. Zu dem Zertifikat ist der private Schlüssel erforderlich (Schritt 352-366). Falls der Schlüssel als Token-Objekt verfügbar ist, kann die Nachricht entschlüsselt werden, insofern der PKCS#11-Provider den Entschlüsselungsmechanismus unterstützt. Dies wird im Schritt 367 geprüft.
Anschließend erfolgt die Entschlüsselung der Nachricht (Schritt 386-371) mit den ersten 24 Byte (192 Bit). Dieser Wert enthält den symmetrischen Schlüssel, mit dem in Folge der Rest der Nachricht im Rechner ohne Mithilfe des Tokens entschlüsselt werden kann.

# Anhang C

# Testsysteme

## C.1 USB-Token/Smartcards

- Aladdin eToken Pro 32k, USB-Token, basiert auf CardOS 4.2B
- Aladdin eToken Pro 72k, USB-Token, basiert auf einem Java-Card-System
- Axalto Cryptoflex 32k
- Axalto Cryptoflex e-Gate
- Siemens Card mit CardOS 4.3B Karte und Infineon Prozessor SLE 66C322P [Inf02a]
- Siemens Card mit CardOS 4.3B Karte und Infineon Prozessor SLE 66C642P [Inf02b]

## C.2 Hardware

Sämtliche Messungen wurden an drei Testsystemen unterschiedlicher Ausstattung durchgeführt. Die Testsysteme liefen jeweils unter einem Linux-Betriebssystem mit folgender Hard- und Software:

**Arbeitsplatz-PC**

- *Prozessor*: AMD Athlon 64x2 Dual Core Processor 5000+
- *RAM*: 2 GB DDR2-800 (PC2-6400)
- *Kernel*: Debian Linux Kernel 2.6.26-1-686
- *Cardreader 1*: USB Smartcard-Tastatur Cherry G83-6744 (basiert auf Omnikey-Hardware und wird vom Omnikey-IFD-Handler unterstützt)
- *Cardreader 2*: Dual-USB-Terminal Omnikey CardMan 5321, ein externes USB-Terminal für kontaktbehaftete und kontaktlose Karten nach ISO/IEC 7816 und 14443
- *Cardreader 3*: USB 2.0 Schnittstelle für den Anschluss von USB-Token

**Notebook**

- *Modell*: HP/Compaq nw9440
- *Prozessor*: Intel Core-Duo T2600
- *RAM*: 2 GB DDR2-667 (PC2-5300)

- *Kernel*: Linux Kernel 2.6.28
- *Cardreader 1*: PCMCIA Terminal Omnikey 4040
- *Cardreader 2*: Dual-USB-Reader Omnikey CardMan 5321
- *Cardreader 3*: USB 2.0 Schnittstelle für den Anschluss von USB-Token

**Netbook - ultraportables Notebook**

- *Modell*: MSI-Wind U100
- *Prozessor*: Intel Atom 270
- *RAM*: 2 GB DDR2-667 (PC2-5300)
- *Kernel*: Ubuntu Linux Netbook Kernel 2.6.28-12
- *Cardreader 1*: Dual-USB-Reader Omnikey CardMan 5321
- *Cardreader 2*: USB 2.0 Schnittstelle für den Anschluss von USB-Token

## C.3 Software / Middleware

- *Middleware*: pcsc-lite (V1.5.3)
- *IFD-Handler*:
  - für USB-Module, z.B. Aladdin eToken: pcsc-libccid (V1.3.10)
  - für Smartcards über Terminal: pcsc-omnikey (V2.6.0, Release: 10/15/07)
- *(Provider-)PKCS#11-Module*
  - Aladdin eToken Pro 32k (basierend auf CardOS 4.2B) und eToken Pro 72k (Java-Card): Aladdin PKI-Client (V5.00.28)
  - Axalto Cryptoflex 32k, Axalto Cryptoflex e-Gate, Siemens CardOS 4.3B: OpenSC PKCS#11-Library (V0.11.8)
  - zusätzlich für Siemens CardOS 4.3B: Siemens HiPath SIcurity Card API V3.1B (Build 3111)

# Anhang D

# Konfiguration

Ausgewählte Beispiele zur Konfiguration von Server- und Client-Applikationen. Aufgelistet sind die Konfigurationsoptionen zur Aktivierung der zertifikatsbasierten Authentifizierung.

## D.1 Server

### D.1.1 Apache HTTP-Server

Aktivierung der TLS/SSL-Funktion:

```
1 # serverseitige Zertifikate und Schlüssel
  SSLCertificateFile     /etc/ssl/certs/server.crt
3 SSLCertificateKeyFile /etc/ssl/private/server.key
  SSLCACertificatePath  /etc/ssl/certs/ca # CA-Zertifikate
5 SSLOptions            +StdEnvVars +ExportCertData
```

Die im Zertifikat enthaltenen Informationen werden in Umgebungsvariablen abgelegt (Zeile 5).
Authentifizierung für URL `https://<server>/doauth` mit Fallback-Lösung:

```
  <Location "/doauth">
7     SSLVerifyClient optional  # Client-Zertifikat optional
      # Zertifikatskette über max. 3 Stufen prüfen
9     SSLVerifyDepth 3

11    # in Abhängigkeit der Zertifikatsprüfung weiterleiten
      RewriteEngine On
13    # bei nicht erfolgreicher Prüfung => https://.../dopasswordauth
      RewriteCond %{SSL:SSL_CLIENT_VERIFY} !=SUCCESS
15    RewriteRule .* /dopasswordauth [L]
  </Location>
```

Zeile 7 aktiviert die zertifikatsbasierte Authentifizierung für die betreffende Adresse. Das Client-Zertifikat wird gegen die CA-Zertifikate im Verzeichnis `/etc/ssl/certs/ca` (Zeile 4) geprüft. Dabei wird ausgehend vom Client-Zertifikat die Kette (analog Abbildung 2.9) über maximal 3 Schritte (Zeile 9) in Richtung Wurzel verfolgt. Schlägt die Prüfung fehl, wird der Browser zu einer anderen Seite umgeleitet (Zeile 12 bis 15), die als Alternative auf Passwortprüfung zurückgreifen könnte.

### D.1.2 Dovecot IMAP/POP3-Server

Aktivierung der TLS/SSL-Funktion, Aktivierung der zertifikatsbasierten Client-Authentifizierung, Bestimmung des Logins aus dem Zertifikats-*Subject*:

```
ssl_cert_file = /etc/ssl/certs/dovecot.pem    # Servercert
ssl_key_file  = /etc/ssl/private/dovecot.pem  # Serverkey

# ssl_ca_file enthält Zertifikate der akzeptierten CAs
# verbunden mit einer aktuellen CRL
ssl_ca_file   = /etc/dovecot/dovecot-validssl.pem

# Wenn ein Login (username) aus dem Zertifikat des Nutzers bestimmt
# wird, ist das Feld x500UniqueIdentifier aus dem Subjekt zu nehmen
ssl_cert_username_field = x500UniqueIdentifier

auth default {
  # erlaubte Mechanismen: anonyme Anmeldung
  mechanisms = anonymous
  # Autorisierung über Dovocot-PAM-Modul
  passdb pam {
    args = dovecot
  }
  # Mailverzeichnis, numerische UID und Gruppen-ID (GID)
  # wird aus LDAP-Eintrag bezogen. Falls kein LDAP-Eintrag vorhanden ist,
  # wird die Verbindung unterbrochen. (Teil der Autorisierung)
  # LDAP-Konfiguration über separate Datei:
  userdb ldap {
    args = /etc/dovecot/dovecot-ldap.conf
  }
  # Fordere ein gültiges Client-Zertifikat an. Wird kein gültiges
  # Zertifikat übermittelt, wird der Verbindung unterbrochen
  # (ebenfalls Teil der Autorisierung)
  ssl_require_client_cert = yes
  # Nutzername wird aus Zertifikat bestimmt
  ssl_username_from_cert = yes
}
```

Autorisierung über Dovecot-PAM-Modul `/etc/pam.d/dovcecot`:

```
#%PAM-1.0
auth    required        pam_permit.so
account required        pam_permit.so
```

Die PAM-Konfiguration schränkt den Zugriff nicht ein. Hat der Client sich mit einem gültigen Zertifikat angemeldet und existiert für den Nutzer ein LDAP-Eintrag, erhält der Nutzer uneingeschränkten Zugriff auf seine im LDAP-Server angegebenen Postfächer. Eine Nutzerfilterung kann über die LDAP-Konfiguration erfolgen.

### D.1.3 Postfix SMTP-Server

Weiterleitung der Emails für Client-MTAs aus eigenem Netz oder bei akzeptiertem Client-Zertifikat:

```
### TLS/SSL Parameter
tls_random_source = dev:/dev/urandom   # Quelle für Zufallswerte
# Client-Zertifikat erfragen oder fordern:
#smtpd_tls_ask_ccert = yes             # Zertifikat erfragen
#smtpd_tls_security_level = may
smtpd_tls_req_ccert = yes              # Zertifikat fordern
smtpd_tls_security_level = encrypt
smtpd_tls_auth_only = yes
smtpd_starttls_timeout = 300s
# Schlüssel und Zertifikate des Postfix-SMTP-Servers
smtpd_tls_cert_file = /etc/ssl/certs/postfix.pem
smtpd_tls_key_file =  /etc/ssl/private/postfix.pem
# Zertifikate der von Postix akzeptierten CAs
smtpd_tls_CAfile = /etc/postfix/postfix-validssl.pem

### Autorisierung
# kein Zugriff durch anonyme Nutzer (kein OpenRelay)
smtpd_sasl_security_options = noanonymous
# Zugriff für lokale Netze (mynetworks) und Nutzer mit
# Zertifikats-Authentifizierung erlauben
smtpd_recipient_restrictions = permit_mynetworks,
                               permit_tls_all_clientcerts,
                               reject_unauth_destination
# Zugriff ggf. auf bestimmte Client-Zertifikate beschränken:
#relay_clientcerts = hash:/etc/postfix/relay_clientcerts
```

### D.1.4 OpenSSH SSH-Server

Authentifizierung mit gültigem Client-Zertifikat, Prüfung diverser CRL-Quellen inkl. LDAP, Autorisierung über LDAP-Server:

```
# Prüfung X.509v3-Erweiterung,
# Zertifikat muss als SSL-Client-Zertifikat einsetzbar sein:
AllowedCertPurpose sslclient
# akzeptierte CA-Zertifikate
CACertificateFile /etc/ssh/certs/ca-bundle.pem
# CRLs
CARevocationFile /etc/ssh/certs/ca-bundle.crl
CARevocationPath /etc/ssh/certs/crl
# Abfrage des LDAP-Servers nach CA, CRL und Nutzern:
CAldapVersion 3
CAldapURL "ldap://ldap-server.informatik.uni-halle.de"
CAldap...
# erlaubte Mechanismen
X509KeyAlgorithm x509v3-sign-rsa,rsa-md5
X509KeyAlgorithm x509v3-sign-rsa,rsa-sha1
X509KeyAlgorithm x509v3-sign-dss,dss-asn1
X509KeyAlgorithm x509v3-sign-dss,dss-raw
```

## D.2 Client

### D.2.1 PAM-PKCS#11

Nutzeranmeldung mit PKCS#11-Token und Bestimmung des Logins aus dem Zertifikats-*Subject* des Nutzers:

```
pam_pkcs11 {
  # ein Passwort, welches bereits an den PAM-Stack übergeben wurde,
  # wird nicht als PIN genutzt
  use_first_pass = false;
  try_first_pass = false;
  # genutzter PKCS#11-Provider
  use_pkcs11_module = opensc;

  # Konfiguration des Providers
  pkcs11_module opensc {
    module = /usr/lib/opensc-pkcs11.so;
    description = "OpenSC PKCS#11 module";
    # genutzter PKCS11-Slot, "none" nimmt den ersten besetzten Slot
    slot_description = "none";
    # akzeptierte CA-Zertifikate
    ca_dir = /etc/pam_pkcs11/cacerts;
    # CRLs
    crl_dir = /etc/pam_pkcs11/crls;
    # Policy:
    # Prüfung der CA, Prüfung des privaten Schlüssels über Signatur
    cert_policy = ca, signature;
  }

  # Mapper (Zuordnung Zertifikat <-> Nutzer)
  # diverse Einträge sind möglich, z.B. Auflösung über LDAP-Server oder
  # eine lokale Datenbank, Bestimmung anhand der Email-Adresse uvm.
  # Der UID-Mapper nutzt das im Zertifikats-Subject enthaltende Feld
  # x500UniqueIdentifier:
  use_mappers = uid;
}
```

In der jeweiligen PAM-Konfiguration kann die Token-gestützte zertifikatsbasierte Authentifizierung durch Einfügen des PAM-PKCS#11-Moduls aktiviert werden. Beispiel `/etc/pam.d/gdm` für die Nutzer-Anmeldung über ein Grafik-Terminal:

```
#%PAM-1.0
auth     requisite     pam_nologin.so
# Token-Authentifizierung
auth     sufficient    pam_pkcs11.so
# Fallback Passwort
auth     required      pam_unix.so try_first_pass nullok_secure
# Start des Agenten und Hinterlegen der PIN
session  sufficient    pam_pkcs11.so
session  required      pam_unix.so
account  required      pam_unix.so
```

### D.2.2 OpenSSL mit Engine PKCS#11

Einbindung der Engine-PKCS#11 aus OpenSC in OpenSSL `/etc/ssl/openssl.cnf`:

```
openssl_conf = openssl_def
[openssl_def]
engines = engine_section

[engine_section]
pkcs11 = pkcs11_engine

[pkcs11_engine]
engine_id = pkcs11
dynamic_path = /usr/lib/engines/engine_pkcs11.so
MODULE_PATH  = /usr/lib/opensc-pkcs11.so
init = 0
```

Applikationen können durch Aktivierung der Engine bei Nutzung von OpenSSL auf PKCS#11-Token zugreifen. Beispiel Aufruf von Kommandozeile:

```
openssl -config /etc/ssl/openssl.cnf -engine pkcs11 ...
```

### D.2.3 WPA-Supplicant mit OpenSSL

Der WPA-Supplicant nutzt OpenSSL. Durch Angabe folgender Parameter in der Konfiguration können PKCS#11-Token eingebunden werden. In der Konfiguration ist der zu verwendende Schlüssel anzugeben:

```
# OpenSSL Engine support
# Einbindung der PKCS11-Engine
pkcs11_engine_path = /usr/lib/engines/engine_pkcs11.so
# Benutzung der Proxy-Bibliothek (keine PIN-Angabe erforderlich)
pkcs11_module_path = /usr/lib/pkcs11-agentl.so
# Einstellung der Parameter für einen 802.1X gesicherten Zugang
network = {
  ssid = "Infrastructure" # Bezeichner des Netzwerks
  key_mgmt = WPA-EAP      # Aktivierung EAP
  proto = WPA2            # für WPA2 WLAN-Zugang
  eap = TLS               # EAP-Protokoll: TLS
  # akzeptierte CA-Zertifikate
  ca_cert = "/etc/wpa_supplicant/ca-bundle.pem"
  identity = "wefel"
  #
  # das passende Zertifikat zum Schlüssel (muss momentan einmalig
  # ausgelesen und auf der Festplatte abgelegt werden)
  client_cert = "/etc/wpa_supplicant/cert_cardsiemens1.pem"

  # Aktivierung der PKCS11-Engine mit angegebener PKCS11-Schlüssel-ID
  engine = 1
  engine_id = "pkcs11"
  key_id = "af2c65e7e37e13c1705e398b43fca8a9"
}
```