Mobile Security with Smartcards

Vom Fachbereich Informatik
der Technischen Universität Darmstadt
genehmigte

Dissertation

zur Erlangung des akademischen Grades
eines Doktor-Ingenieurs (Dr.-Ing.)

von Diplom-Informatiker

Roger Kilian-Kehr

aus Marburg/Lahn

Referent:	Prof. A. Buchmann, PhD, TU Darmstadt
Korreferent:	Prof. Dr. F. Mattern, ETH Zürich
Tag der Einreichung:	08. April 2002
Tag der mündlichen Prüfung:	06. Mai 2002

Darmstädter Dissertationen D17

Summary

Mobility in conjunction with communication facilities in the form of mobile telephony seems to be one of the major technology trends observed throughout the last decade. Many experts and analysts expect that the arrival of mobile services such as *mobile commerce, location-based services, multimedia messaging,* and *mobile gaming* in the third generation of mobile networks will be the next step in this success story. However, protecting service providers from fraud and mobile users from new threats such as *identity theft* or other attacks on privacy and security matters is equally challenging.

Historically, cryptography has been used to protect information in the digital world from eavesdropping or tampering. In future *person-to-person* and *person-to-service* interaction scenarios cryptography will be of at least equal importance. However, the situation today is not *people-centric* but more *application-centric*, i.e. for each application new security measures are defined and implemented. As an example one may just consider that almost any access control in the Internet is managed through simple account/password schemes different for each application. But passwords are known to be a generally weak security measure in many practical settings (cf. [MT79]). From the user perspective the account/password approach additionally leads to numerous login accounts an individual has to manage – something which is inconvenient and as a consequence often error-prone.

Cryptographic measures can be applied but shifting towards such mechanisms especially in mobile settings is often hard to implement since people cannot easily carry around their personal cryptographic keys, let alone memorize them or input them when needed. Therefore, we believe that some kind of *personal security assistant* or *device* is needed that safely keeps a user's security-sensitive data and enforces the user's security-related interests. Otherwise, people will be forced to use traditional weak protection mechanisms that are applicable without strong cryptographic measures – a situation we do not think is desirable in the digital age of tomorrow.

Smartcards are devices that could be used to solve at least some of the problems mentioned. They are *tamper-resistant*, can safely *store* information, are able to perform *unobserved (cryptographic) operations,* and can be conveniently carried around. As such they seem to be ideal candidates for personal security modules. However, it is yet unclear *how* smartcards can be empowered to actually play the role of true personal and ubiquitous security modules. Furthermore, the smartcard alone is not sufficient to act as a security module since it lacks reasonable user interfaces such as a display and input facilities. Thus, suitable terminals are needed that allow users to communicate with their smartcards, i.e. personal security modules are comprised of suitable terminals and personalized smartcards that work together in order to fulfill the users' needs.

Henceforth, this thesis will contribute approaches, architectures, protocols, and systems how smartcards can be put in place to become true security modules for people in the digital age. The most visible contributions of this thesis are as follows:

- The *JiniCard* framework for the integration of off-the-shelf smartcards into local environments. It enables smartcards that have traditionally played the role of passive servers to become truely active entities after they are inserted into suitably configured card terminals. Users could carry around their smartcard, insert them into available readers and make immediate use of their security services. The approach is centered around the idea to dynamically instantiate "software substitutes" for resource-limited devices such as smartcards.
- The so-called *Personal Card Assistant* approach solving the problem of smartcard usage in a "hostile" environment. It is comprised of an off-the-shelf personalized user terminal – such as a PDA – that cooperates with a personal smartcard. The personal terminal is used instead of terminals considered to be public, i.e. it acts as a "trust amplifier" for its user. The advantage is that mobile users communicate with their smartcard through their own mobile terminal which they possibly consider much more trustworthy than other unknown components. The personal terminal and the smartcard are linked together using cryptographic measures such that no device is usable without the other.
- The *WebSIM* system that integrates into the Internet the SIM smartcards found in all GSM mobile phones. In this approach people now can use mobile phones as "wireless smartcard readers" which are reachable from the Internet by means of a small HTTP Web server implemented in the SIM. This approach allows among others to perform security-critical operations such as authentication to be initiated from a remote context, e.g. from an Internet shop. Hence, smartcards become Internet nodes that encapsulate security services a mobile user offers to peers.
- The *SIMspeak* platform allowing for the execution of mobile code within a smartcard. This approach was motivated by the need for end-to-end secure communication between a service provider and its customer and the ability to easily create electronic signatures on small devices. It allows a service provider to "rent" persistent storage on a user's personal smartcard, e.g. to store cryptographic keys used to send end-to-end encrypted mobile code from the provider to the user's smartcard. The smartcard then becomes the most active component in a personal security module and uses available terminals to communicate with its user. This approach essentially shifts as much security-critical components and computations as possible from less trustworthy components into the secure context of the smartcard. This approach leads to new trust models for smartcard issuers which can be particularly well applied in the context of electronic signature creation in mobile scenarios.

These results can be used independently from each other but equally well composed into more general security solutions. As such they can be considered as building blocks enabling the composition of suitable personal security modules meeting the personal security demands of the future.

Summing up, this thesis provides solutions to the question how smartcards can become true personal security modules. It does this by proposing concrete architectures and protocols all of which have been prototypically implemented to yield meaningful proofs-of-concepts.

Zusammenfassung

Der ungeahnte Erfolg des Mobilfunks im letzten Jahrzehnt hat zu vielen Prognosen von Experten und Analysten geführt, wie Menschen in der Zukunft miteinander und mit ihrer Umwelt kommunizieren werden. Wichtig wird nach wie vor das Bedürfnis der Nutzer nach Sprachkommunikation sein, jedoch hat der Erfolg des *Short Messaging Service* (SMS) gezeigt, dass auch andere Dienste durchaus überraschende Nutzungsdimensionen aufweisen können. Für die dritte Generation der Mobilfunknetze mit entsprechenden Basisdiensten zur Datenkommunikation wird daher erwartet, dass ein Großteil der mobilen Kommunikation über Datendienste in den Bereichen *M-Commerce*, *lokations-basierte Dienste*, *Multi-Media Messaging* und *mobiles Spielen* stattfindet.

Sicherheit in der digitalen Welt von heute und morgen

Notwendigerweise wird die Nutzung solcher und anderer Dienste entsprechende Maßnahmen zur Zugangskontrolle, Authentifikation und auch der Abrechnung erfordern, die es Dienstanbietern erst ermöglichen, Zugang zu Diensten zu kontrollieren und genügend Einnahmen für einen erfolgreichen Betrieb zu erwirtschaften. Üblicherweise erfordert dies die Lösung einer Reihe von grundlegenden Problemen wie z.B.

— **Authentifikation** als Basis zur Identifizierung von Kunden und zur Missbrauchsprävention,

— **Nicht-Abstreitbarkeit** zur rechtlich bindenden Vertragsgestaltung und für Einwilligungserklärungen und

— **Verwaltung personenbezogener Daten** zum Schutz der Nutzer vor diversen Gefahren des Datenmissbrauchs.

Diese Liste kann in vielerlei Hinsicht erweitert werden und ist hier nur als exemplarisch anzusehen. Technische Maßnahmen zur Umsetzung dieser Ziele sind Teil einer Gesamtlösung und müssen durch entsprechende Geschäftsprozesse und -verträge ergänzt werden.

Traditionell werden insbesondere Maßnahmen wie Authentifikation, daran anschließende Zugriffskontrolle und insbesondere Nicht-Abstreitbarkeit mit Hilfe kryptografischer Algorithmen und Protokolle implementiert. Dies bedeutet in der Praxis, dass darauf aufbauende Lösungen und Systeme die Möglichkeit haben müssen, kryptografische Berechnungen durchzuführen und dazugehörige Schlüssel zu verwalten.

Chipkarten als Teil von universellen persönlichen Sicherheitsmodulen

Chipkarten sind prinzipiell in der Lage genau diese Anforderungen zu erfüllen. Sie sind resistent gegen Manipulation, erlauben die sichere Hinterlegung von Daten, wie z.B. kryptografische Schlüssel, und verbergen die Ausführung von Applikationen und Algorithmen auf der Karte. Diese Eigenschaften machen Chipkarten potenziell zu persönlichen Sicherheitsmodulen welche ihre Eigentümer in ihrem täglichen Ablauf bei sicherheitsrelevanten Aktionen unterstützen könnten.

Zur Zeit werden Chipkarten allerdings weniger als persönliche Sicherheitsmodule verwendet, sondern im Allgemeinen immer anwendungsspezifisch eingesetzt. Am Beispiel GSM dient die SIM im engeren Sinne als Sicherheitsmodul des Betreibers und nicht des Nutzers. In einer zukünftigen mobilen Welt werden die jetzigen Sicherheitslösungen in den anwendungszentrierten Ansätzen sehr wahrscheinlich allein durch die Anzahl der Dienste und Anwendungen nicht geeignet skalieren. Ein von vielen Diensten genutztes persönliches Sicherheitsmodul, welches universell einsetzbar ist, könnte ein Ausweg aus diesem Dilemma sein. Dies zieht notwendigerweise eine Reihe von generellen Fragen nach sich:

- Wie muss ein persönliches Sicherheitsmodul für den mobilen Gebrauch auf der Basis einer Chipkarte aussehen?
- Wie können "kleine" Geräte wie Chipkarten mit begrenzten Ressourcen in Bezug auf Rechenleistung und zur Verfügung stehender Kommunikationsbandbreite einfach in Umgebungen integriert werden?
- Welche Sicherheitsanforderungen müssen von den Endgeräten erfüllt werden, damit Nutzer diese zur Kommunikation mit der persönlichen Chipkarte verwenden können?
- Welche Sicherheitsniveaus können mit den vorgeschlagenen Lösungen erreicht werden?

Weiterhin können die hier aufgeworfenen Fragen nicht unabhängig voneinander beantwortet werden und erfordern grundsätzlich einen umfassenderen Ansatz für den Problembereich der Mobilität und Sicherheit mit Chipkarten. Diese Arbeit liefert daher Ergebnisse sowohl auf der Architekturebene genauso wie Vorschläge für neue Interaktionsprotokolle und generell neue Aufgaben für Chipkarten. Die einzelnen Ergebnisse können als Basis und Bausteine eines Rahmenwerks angesehen werden, welches die Bedürfnisse vieler Nutzer nach sicherer Interaktion mit Partnern in mobilen Szenarien befriedigt.

Ergebnisse dieser Arbeit

Die vorliegende Arbeit beginnt mit einer Analyse zukünftiger mobiler Szenarien in denen mobile Nutzer Dienste "spontan" verwenden. In Zukunft werden Dienste in solchen Szenarien oftmals über lokale Kommunikationsmedien wie Kurzstreckenfunknetze oder Infrarotkommunikation angeboten werden. Es wird herausgearbeitet, dass die Spontaneität bei solchen Konstellationen üblicherweise darin besteht, dass sich die Kommunikationspartner möglicherweise a priori nicht kennen und das gegenseitige Kennenlernen erst bewerkstelligen müssen.

Als Beispiel für spontane Kommunikationsbeziehungen möge GSM im Falle des Roamings in fremden Gastnetzen angesehen werden. Hierbei tauschen Mobiltelefon, SIM und Gastnetz Informationen untereinander aus, die es dem Gastnetz erlauben, das Heimatnetz des Teilnehmers zu kontaktieren, eine Authentifikation durchzuführen und den Zugang zu kontrollieren. Die Bequemlichkeit für den Benutzer resultiert aus international festgelegten Standards, die über Jahre hinweg definiert und gepflegt wurden.

Die Spontaneität der Interaktionen zwischen mobilen Nutzern, ihren Geräten und der Umwelt ist eines der Grundprobleme einer neuen Ära in der Informationstechnik die mit den Begriffen *Pervasive* und *Ubiquitous Computing* umschrieben wird.

Pervasive Computing, Ubiquitous Computing und Spontane Vernetzung

Die zukünftigen Szenarien der Dienstnutzung beschreibt IBM's offizielle Definition des allgemein verwendeten Begriffs *Pervasive Computing* mit "*Convenient access, through a new class of app-*

liances, to relevant information with the ability to easily take action on it when and where you need to.". Der noch weiter gefasste Begriff des *Ubiquitous Computing* geprägt durch Mark Weiser schließt darüber hinaus auch die "unsichtbare" Integration von Computern in Alltagsgegenstände und Umgebungen mit ein, die den Menschen in seinem täglichen Lebensablauf mehr oder weniger unbemerkt unterstützen.

Wesentlicher Bestandteil solcher Visionen ist die Annahme, dass Nutzer eigene mobile Geräte als Werkzeug zur Kommunikation mit Diensten und anderen Geräten verwenden werden und in der Lage sein müssen *spontan* Kommunikationsbeziehungen einzugehen. Technologien zur *spontanen Vernetzung*, also zur Integration von Diensten und Geräten in Netzwerkumgebungen zum Zwecke einer sofortigen Dienstverfügbarkeit ohne manuellen Eingriff, sind daher als elementarer Baustein von Pervasive oder Ubiquitous Computing anzusehen.

Verschiedenste Technologien sind für diesen Zweck vorgeschlagen worden und exemplarisch werden davon das *Service Location Protocol* (SLP) und *Jini* untersucht und gegenübergestellt. Beide Technologien treffen keine grundlegenden Annahmen über die ihnen zugrunde liegenden Kommunikationsmedien außer eines IP-basierten Netzwerks, über das Dienstanbieter Dienste anbieten und Klienten diese ausfindig machen. Dieses schließt jedoch eine große Klasse von Geräten, die über andere Kommunikationsmechanismen verfügen, von vorneherein aus und SLP und Jini sind damit nicht für beliebige Anwendungsszenarien geeignet.

Ein Rahmenwerk zur Integration kleiner Geräte in lokale Infrastrukturen

Wesentliches Ergebnis der Gegenüberstellung der Technologien zur spontanen Vernetzung ist jedoch die Tatsache, dass sie für die Integration "besonders kleiner" Geräte im lokale Umgebungen nicht geeignet sind. Wir entwickeln daraus die generelle Prognose, dass eine geeignete Integrationsunterstützung für diese Art von Geräten durch deren umgebende Infrastruktur immer nötig sein wird. Auch wenn die aktuelle Generation von Geräten durch technologischen Fortschritt dem Bedarf an Unterstützung durch die umgebende Infrastruktur entwächst, wird es neue Geräte geben, die wiederum Unterstützung benötigen.

Aus dieser durchaus diskussionsfähigen Annahme heraus wird nachfolgend ein Rahmenwerk bestehend aus einer Architektur und geeigneten Interaktionsprotokollen zur Integration von besonders kleinen Geräten in eine lokale Infrastruktur vorgeschlagen. Dem Rahmenwerk liegt die Idee zugrunde, dass man bei der Integration auf den untersten Ebenen einer Kommunikationshierarchie in der Regel immer Informationen wie z.B. eindeutige Seriennummern, etc. für Kommunikationspartner zugänglich sind. Mit Hilfe dieser Informationen kann die Umgebung über geeignete Protokolle gerätespezifischen mobilen Code (sog. *Device Proxy*) laden, der die bis dato unbekannten Fähigkeiten des Gerätes ermittelt. Dieser Device Proxy ist dann in der Lage, als Stellvertreter des Gerätes zu agieren und weitere Schritte zur Integration des Gerätes in eine lokale Infrastruktur vorzunehmen.

Generell ermöglicht dieser Ansatz, kleine Geräte mit Hilfe von mobilem Code als "Software-Stellvertreter" in lokale Umgebungen auf flexible Art und Weise zu integrieren und die im Gerät verfügbare Funktionalität über externe Komponenten zu komplettieren. Der entwickelte Ansatz dient später als Basis für die Integration von Chipkarten in lokale Umgebungen.

Das Spannungsfeld zwischen Chipkarte und Terminal

Im weiteren Verlauf der Arbeit wird analysiert, welche spezifischen Probleme die Integration von Chipkarten in Umgebungen mit sich bringt. Zentral ist dabei insbesondere die Frage, welche Kommunikationswege benutzt werden, um mit der persönlichen Chipkarte zu kommunizieren, da diese nicht über eigene Ein- und Ausgabemöglichkeiten wie Anzeigeeinheit oder Tastatur verfügt. Wie

später noch dargelegt wird, lässt sich aus den Forschungsergebnissen der letzten Jahre grundsätzlich ersehen, dass sich Chipkarten in einer komplett “feindlichen” Umgebung nur sehr schwer schützen lassen, falls kein sicherer Kommunikationskanal zum Benutzer vorhanden ist. Ein vollständiges persönliches Sicherheitsmodul umfasst aber immer sowohl die personalisierte Chipkarte als auch ein geeignetes Terminal, mit dessen Hilfe der Benutzer mit der Karte kommuniziert.

Daher wird in der vorliegenden Arbeit eine Klassifikation des Spannungsfelds von Chipkarte und Terminal vorgenommen, die auf den folgenden von uns identifizierten Kriterien beruht:

— **Vertrauenswürdigkeit des Terminals:** Dieses Kriterium definiert, als wie vertrauenswürdig das Terminal aus Sicht des Nutzers eingeschätzt werden kann. Ein öffentliches Terminal wird zum Beispiel mit hoher Wahrscheinlichkeit als weniger vertrauenswürdig eingeschätzt als ein dem Nutzer gehörendes Endgerät, welches er oder sie prinzipiell ständig unter Kontrolle hat.

— **Personalisierung des Terminals:** Hierunter wird verstanden, dass neben der Chipkarte auch das Terminal personalisiert ist. Üblicherweise impliziert ein personalisiertes Terminal auch eine höhere Vertrauenswürdigkeit, so dass beide Kriterien in der Regel stark korrelieren und aus Gründen der Einfachheit oft auch zusammen betrachtet werden können.

— **Mobilität des Terminals:** Die vom Nutzer verwendeten Terminals können mobil sein und von diesem “herumgetragen” werden oder es werden lokal zur Verfügung stehende Terminals verwendet, abhängig von der jeweiligen Umgebung, die der Nutzer vorfindet.

— **Kommunikation zwischen Terminal und Chipkarte:** Dieses Kriterium definiert, wie Karte und Terminal miteinander gekoppelt sind. Mögliche Varianten sind unter anderem eine direkte physische Kopplung oder eine Kommunikation über ein Netzwerk. Letzteres kann wiederum einen “öffentlichen” Charakter haben, in dem die Kommunikation zwischen Karte und Terminal geeignet geschützt werden muss, oder das Netzwerk kann als “privat” und damit vertrauenswürdig eingestuft werden, was möglicherweise keine besondere Sicherung der Kommunikation erfordert.

— **Anwendungskontrolle:** Dieses Kriterium legt fest, wie die Verteilung der Realisierung einer Anwendung auf Chipkarte und Terminal erfolgt. Grundsätzlich ergibt sich dabei ein Spektrum von Anwendungen, die nahezu ausschließlich im Terminal ablaufen, bis zu Applikationen, die bis auf Nutzerein- und -ausgaben vollständig in der Karte ablaufen.

Auf der Basis der vorgestellten Kriterien ergeben sich vier Problembereiche, die im Rahmen dieser Arbeit untersucht wurden und für die Lösungen erarbeitet wurden. Für alle Problembereiche werden in dieser Arbeit Lösungen vorgestellt, die in prototypischer Form umgesetzt und implementiert wurden. Diese Lösungen ergeben miteinander kombinierbare Grundbausteine für die Entwicklung von persönlichen Sicherheitsmodulen. Die vier Ansätze werden nachfolgend kurz vorgestellt.

A1: Ein Rahmenwerk zur Spontanen Vernetzung von Chipkarten

Dieser Ansatz beruht auf der Problemstellung, dass ein Nutzer nicht über ein eigenes persönliches Terminal verfügt, sondern gezwungen ist, seine personalisierte Chipkarte über lokal vorgefundene Terminals zu nutzen.

Unsere *JiniCard*-Lösung besteht aus dem bereits vorgestellten Rahmenwerk zu Integration kleiner Geräte in lokale Umgebungen. Hierzu werden nach dem Einführen einer Chipkarte in ein geeignetes Kartenterminal spezifische Informationen der Karte abgefragt, die als Schlüssel für die Lokalisierung des zu der Karte passenden mobilen Stellvertreter-Objekts oder Proxy dienen. Der entsprechende Karten-Proxy übernimmt dann die weitere Kommunikation mit der Karte und kann

insbesondere auf der Karte befindliche Dienste ermitteln und für diese ebenfalls Dienst-Proxys instanzieren. Der Nutzer kann dann das lokale Terminal verwenden, um mit der Karte zu kommunizieren und sicherheitsrelevante Entscheidungen vornehmen.

Zusammenfassend kann der Ansatz als eine spezifische externe Ergänzung der auf der Karte befindlichen Dienste verstanden werden. Dazu werden entsprechende Komponenten in Form von mobilen Objekten aus dem Netz dynamisch geladen und der Umgebung komplettierend hinzugefügt. Dies ermöglicht es Chipkarten, sich als aktive Komponenten mit Hilfe ihrer externen “Stellvertreter” in Umgebungen transparent und spontan zu integrieren und ihre Dienste für potentielle Klienten anzubieten.

A2: Ein personalisiertes und vertrauenswürdiges mobiles Terminal

Ansatz A2 beruht auf dem Problem, wie eine Chipkarte in einer für den Nutzer nicht vertrauenswürdigen Umgebung trotzdem sicher eingesetzt werden kann.

Die erarbeitete Lösung namens *Personal Card Assistant* (PCA) schlägt vor, ein dem Benutzer gehörendes mobiles Terminal, wie beispielsweise einen persönlichen digitalen Assistenten (PDA), hinzuzuziehen, der als vertrauenswürdiges Instrument zur Kommunikation mit der Karte Verwendung findet.

Hierbei gibt es grundsätzlich die Variante der direkten physischen Kopplung von Terminal und Karte oder einer Kommunikation über einen potentiell nicht vertrauenswürdigen Kommunikationskanal. Eine sichere Kommunikation über einen unsicheren Kanal wird hierbei über die Personalisierung des Terminals mit Hilfe kryptografischer Verfahren vorgenommen.

Konkret bedeutet dies, dass Terminal und Chipkarte im Besitz des öffentlichen Schlüssels des jeweils anderen sind. Dieses Prinzip der Paarbildung ermöglicht es, einen authentisierten und verschlüsselten Kommunikationskanal untereinander aufzubauen, der auch in einer unsicheren Umgebung eine sichere Nutzung der Chipkarte erlaubt. Diese Anordnung wird noch verbessert durch die Tatsache, dass die Karte keinerlei Operationen durchführt, die nicht über einen vom Terminal aufgebauten Kommunikationskanal angefordert wurden. Umgekehrt ist das Terminal so ausgerichtet, dass es alleine nicht in der Lage ist, sicherheitsrelevante Entscheidungen ohne die Karte vorzunehmen. Dies ergibt ein höheres Sicherheitsniveau, da nur beide Geräte zusammen eine funktionsfähige Einheit bilden.

Dieses Prinzip der Nutzung eines vertrauenswürdigeren persönlichen Gerätes ist am Beispiel der Erstellung von elektronischen Signaturen mit konkreten kryptografischen Protokollen umgesetzt und implementiert worden und ist auf weitere Anwendungen beliebig ausdehnbar.

A3 Eine mobiles Terminal welches über einen drahtlosen mobilen Kommunikationskanal verfügt

In vielen Situationen des täglichen Lebens, beispielsweise bei der Bestellung von Waren per Telefon, ist eine geeignete Einbindung eines Sicherheitsmoduls in den eigentlichen Kommunikationsablauf nötig. Im Beispiel der Telefonbestellung ist diese aber schon aus technischen Gründen nicht gegeben, der direkte Einsatz eines Sicherheitsmoduls “beim Telefonieren” also nicht ohne weiteres möglich. Trotzdem ist es wünschenswert, auch hier ein Sicherheitsmodul flexibel und einfach mit einbinden zu können und Verträge und Bezahlvorgänge mit kryptografischen Mechanismen einfach und kostengünstig abzusichern.

Der *WebSIM* genannte Lösungsansatz besteht darin, das Terminal als einen “drahtlosen Chipkartenleser” zu betreiben. Als idealer Untersuchungsgegenstand wurden GSM Mobiltelefone gewählt, die in der Regel über eine universelle Erreichbarkeit zu jeder Zeit und an nahezu jedem Ort verfügen.

Des Weiteren bieten GSM Mobiltelefone über das sog. *SIM Application Toolkit* den SIM Chipkarten eine standardisierte Schnittstelle an, mit deren Hilfe eine SIM Interaktionen mit dem Nutzer durchführen kann. Die drahtlose Schnittstelle von GSM erlaubt es nun, Sicherheitsdienstleistungen der Chipkarte, wie z.B. Authentifikation "per Handy" anzubieten.

Konkret wurde in dieser Arbeit eine Möglichkeit entwickelt und implementiert, welche die SIM vom Internet aus per HTTP ansprechbar macht. Dies bedeutet, dass die Chipkarte als normaler Internet-Knoten erreichbar ist und deren Dienste auch vom Internet aus angesprochen werden können. Durch die weltweite Verbreitung von über 500 Millionen GSM SIMs hat die erarbeitete Lösung potenziell eine hohe praktische Relevanz, auch wenn die neue Funktionalität der WebSIM erst mit der Herausgabe neuer Karten erweitert werden kann.

A4: Eine personalisierte Chipkarte, die über eine Ausführungsplattform für mobilen Code verfügt

Dieser Ansatz ergibt sich aus dem generellen Bedürfnis für Dienstanbieter eine Ende-zu-Ende sichere Kommunikationsmöglichkeit mit der Chipkarte des Nutzers zu schaffen, die im Ansatz A3 aus verschiedenen Gründen nicht gegeben ist.

In der konzipierten *SIMspeak*-Lösung verfügt die Chipkarte über einen Interpreter, der Programme in einer speziell für diesen Anwendungsbereich entworfenen Programmiersprache ausführen kann. Dienstanbieter senden Programme in dieser Sprache zur Chipkarte, deren Interpreter diese dann zur Ausführung bringt. Dabei profitieren die Programme von der sicheren Ausführungsumgebung in der Chipkarte und können auch auf "gemieteten" persistenten Speicher in der Karte zurückgreifen. Grundprobleme dieses Ansatzes wie *kontrollierte Terminierbarkeit* von Anwendungen, *Typsicherheit* auf der Basis geeigneter *Code-Verifikation* sind untersucht und Lösungen erarbeitet worden. Insbesondere wird aufgezeigt, dass für den intendierten Anwendungsbereich auch das Problem von "böswilligem Code" kontrollierbar ist.

Dieser Ansatz ermöglicht es, komplette Anwendungen mit Nutzerinteraktion in einer Chipkarte ablaufen zu lassen und lediglich die Nutzerein- und -ausgaben über ein Terminal vornehmen zu lassen. Alle sicherheitskritischen Abläufe werden von der Karte ausgeführt und kontrolliert. Dieser Ansatz steht im Gegensatz zur bisherigen Praxis, Chipkarten nur als letztes Glied in einer Anwendungshierarchie zu nutzen und rückt stattdessen Chipkarten als sichere persönliche Module in den Mittelpunkt des Geschehens.

Ermöglicht durch den Ansatz der Plattform für mobilen Code wurden mehrere Protokolle im Umfeld der Erstellung von elektronischen Signaturen entwickelt, die einerseits insbesondere für die Nutzung in mobilen Szenarien geeignet sind und andererseits eine grundsätzliche Verbesserung der Sicherheit bei der Erstellung von Signaturen darstellen.

Fazit

Die vorliegende Arbeit entwickelt auf der Basis eigener Einschätzungen und Annahmen, wie mobile Nutzer in der Zukunft mit Diensten kommunizieren, die generelle These, dass auf den Nutzer personalisierte Chipkarten ideale Komponenten eines persönlichen Sicherheitsmoduls in Verbindung mit einem geeigneten Terminal sind. Aus dieser These wird ein Entwurfsraum aufgezeigt, der die besonderen Merkmale der Chipkarte und der Terminals strukturiert darlegt und Vergleiche unterschiedlicher Ansätze zulässt.

Auf der Basis dieses Designraums werden spezielle Kernprobleme identifiziert, für die jeweils konkrete Lösungen entwickelt, vorgestellt, als Nachweis der Umsetzbarkeit implementiert und be-

wertet werden. Diese Lösungen können als Standardlösungen und -bausteine eines Rahmenwerks herangezogen werden, mit dessen Hilfe der Entwurf persönlicher Sicherheitsmodule unterstützt werden kann.

Die verschiedenen Bausteine können darüber hinaus in verschiedenster Weise miteinander kombiniert werden, um für besondere Anwendungszwecke neue Lösungen zusammenzustellen. Weiterhin lässt sich über den aufgespannten Designraum nachweisen, dass die vorgestellten Lösungen in Bereiche der Chipkartennutzung vordringen, die von "klassischen" Anwendungen nicht abgedeckt werden und auch in dieser Hinsicht als neu betrachtet werden können.

Insgesamt werden alle wesentlichen Aspekte der Integration und des Einsatzes von Chipkarten als persönliche Sicherheitsmodule vorgestellt. Aus dieser Arbeit ergeben sich daher grundlegend neue Erkenntnisse und Richtlinien, wie zukünftige persönliche Sicherheitsmodule gestaltet werden sollten.

Acknowledgements

Performing a doctoral thesis while at the same time the Internet booms seems to be sort of a purely idealistic effort since the seduction of big money in these times was definitively given. However, being embedded in the academic world of a university and at the same time being a member of a research department of a large telecommunication enterprise had the definitive advantage of "taking best of both worlds" and it allowed me to act as a "bridge" between those two institutions and communicating ideas from the academic world into the industry and *vice versa*. This, after all, was one of the best experiences I've made throughout the course of my work and many people were part of it.

First of all I'd like to thank my supervisors Alex Buchmann and Friedemann Mattern who gave me the academic freedom vitally necessary to experiment with new ideas and backing me whenever needed. Very special thanks go to Joachim Posegga who made it possible for me being hooked at T-Nova, and who has been the best discussion partner about smartcards, telcos, and the GSM world I can imagine.

The guy who I've most closely worked together in the course of this thesis is Andreas Zeidler with whom I discussed *ubiquitous computing* all the way from top-down to bottom-up. Maybe the most essential result of this was that "UbiComp" does not (yet) have a real business case, which is a real pity.

Next is Harald Vogt, who taught me the basics of smartcards and formal methods and who left us towards Zurich just because of chocolate and cheese, and cooking food with radio frequency equipment. Then there is Michael Rohs, best diploma thesis worker I've ever had and who really did extraordinarily well with the JiniCard framework except that he left us also for cooking food with radio frequency.

Since individually listing more colleagues that influenced my work would span too many paragraphs, I'd like to jointly thank my colleagues at the *Databases and Distributed Systems Group* (DVS) and the PhD program *Enabling Technologies for Electronic Commerce*; furthermore the people at the *Information Technology Transfer Office* (ITO), my colleagues at the *Information Security Department* (ES2) at T-Nova, and the members of the EURESCOM Project P1005. Then I'd like to thank *Schlumberger* who delivered some GSM SIMs for the WebSIM, the guys at *Radiomobil, S.A.* in Prague who gave us two GSM IMSI/Ki pairs needed to make the WebSIM fly, and Jürgen Dethloff for inventing the smartcard.

Special thanks to my proof readers starting with Felix Gärtner and Markus Schumacher who also helped me to shape the most central parts of this thesis, and furthermore Thomas Ziegert and Andreas Zeidler.

Many, many thanks to my parents who supported me throughout the last three decades and who always believed in me.

Lastly, but actually most importantly, I'd like to thank the most wonderful person in this world, my wife Moni. She gave be the mental backing necessary to finish this work and she often motivated me in the evenings to once more open my laptop to write down some more pages for my dissertation.

In retrospective this personal undertaking can be characterized as a continuum ranging from pure fun to many moments when I was considering to just cancel all this. Keeping motivation steadily up is maybe the most important single factor that eventually leads to a written down piece of paper documenting the work done throughout the last couple of years.

Source of this motivation has been more than others my social environment: my family, my friends, and my colleagues. Thank you all. . .

Contents

Summary **i**

Zusammenfassung **iii**

List of Figures **xv**

List of Tables **xvi**

List of Acronyms **xvii**

1 Introduction **1**
1.1 Motivation 1
1.2 Contributions of this Thesis 4
1.3 Organization of this Thesis 5

2 Connected Devices, Services, and their Users **7**
2.1 Introduction 7
2.2 On Ubiquitous and Pervasive Computing 8
2.3 Issues in Spontaneous Networking 9
2.4 A Brief Overview of Spontaneous Networking Technologies 11
2.4.1 Fundamental Concepts 12
2.4.2 Case Studies 12
2.4.3 Summary 17
2.5 On the Necessity of Infrastructural Support for Small Devices 17
2.6 Summary 19

3 Smartcards as Personal and Ubiquitous Security Modules **21**
3.1 Introduction 21
3.2 Smartcard Technology Overview 22
3.2.1 A Brief History of Smartcards 22
3.2.2 Micro Processor Cards – Essence and Applications 23
3.2.3 Standard Issuance Models for Smartcards 27
3.3 Smartcards in Hostile Environments 28
3.4 Design Dimensions of Smartcards and Terminals 29
3.4.1 Mobility and Personalization of Terminal 31
3.4.2 Placement and Communication between Terminal and Smartcard 32
3.4.3 Application Control 33
3.5 Design Options of Personal Security Modules 33

3.6 Summary . . . 37

4 Integration of Smartcards into Networked Environments 39
4.1 Introduction . . . 39
4.2 Mobile Code as an Enabling Technology for Spontaneous Networking . . . 41
4.2.1 The Memory Wallet Example . . . 41
4.2.2 A Bootstrapping Framework for Small Devices . . . 42
4.3 Middleware for Smartcards . . . 46
4.3.1 Smartcard Terminal . . . 47
4.3.2 Characterization of Middleware Approaches . . . 48
4.3.3 Design Choices for Smartcard Middleware . . . 49
4.3.4 Middleware as an Execution Platform for Mobile Code . . . 50
4.4 The JiniCard Framework . . . 51
4.4.1 The JiniCard Terminal Layer . . . 53
4.4.2 Smartcard Exploration Layer . . . 55
4.4.3 Smartcard Services Exploration Layer . . . 57
4.5 Related Work . . . 59
4.6 Security Aspects . . . 62
4.6.1 Smartcard Communication Assumptions . . . 62
4.6.2 End-to-End Security Approaches . . . 63
4.7 Summary . . . 65

5 Personal Security Modules based on Mobile Personalized Terminals 67
5.1 Introduction . . . 67
5.2 Motivation: Electronic Signature Creation . . . 68
5.3 The Personal Card Assistant . . . 69
5.3.1 PCA Overview . . . 69
5.3.2 The Underlying Cryptographic Protocol . . . 71
5.3.3 Informal Threat Analysis . . . 73
5.3.4 Related Work on the Trustworthiness of Terminals . . . 73
5.4 Design Issues for Personal Security Modules . . . 76
5.4.1 Personalization Enables Strong Cryptographic Binding . . . 76
5.4.2 Replay Prevention . . . 77
5.4.3 Mutual Discovery . . . 77
5.5 Summary . . . 77

6 The WebSIM or How to implement a Web Server in a GSM SIM? 79
6.1 Introduction . . . 79
6.2 GSM Subscriber Identity Module . . . 80
6.3 SIM Application Toolkit . . . 81
6.4 The SIM as a Security Module in the Internet . . . 83
6.4.1 Security Status Quo in the Internet . . . 83
6.5 WebSIM – A Web Server in a GSM SIM . . . 84
6.5.1 Communication Protocols . . . 84
6.5.2 Architecture . . . 85
6.5.3 Command Set . . . 86
6.5.4 Implementation . . . 88

6.6 Security Analysis . . . 88
6.7 Applications . . . 90
6.7.1 WebSIM-based Authentication in the Internet . . . 90
6.7.2 Using WebSIMs as I/O Channels . . . 92
6.8 Related Work . . . 92
6.9 Summary . . . 94

7 Open and Secure Service Platforms for Smartcards 95
7.1 Introduction . . . 95
7.2 Principles and Concepts of Trusted Platforms . . . 97
7.2.1 Hardware Protection based on Tamper-Resistant Hardware . . . 98
7.2.2 Software Protection . . . 98
7.3 Architecture . . . 100
7.3.1 Components Overview . . . 100
7.3.2 Role and Trust Model . . . 102
7.4 Programming Model and Language . . . 102
7.4.1 A Domain-specific Approach . . . 103
7.4.2 Virtual Machine Model and Instruction Set . . . 104
7.4.3 Programming with a Platform . . . 105
7.5 Mobile Code Security . . . 107
7.5.1 Host Security . . . 107
7.5.2 Code Security or the Malicious Host Problem . . . 111
7.6 Communication Protocols . . . 112
7.6.1 Verification and the End-to-End Argument . . . 112
7.6.2 Communication Protocols . . . 113
7.7 Non-Repudiation and Electronic Signatures . . . 117
7.7.1 Motivation . . . 117
7.7.2 Electronic Signatures with On-Card Hash Computation . . . 118
7.7.3 Electronic Signatures Assisted by a Trusted Third Party . . . 119
7.7.4 Electronic Signatures with Recipient Addressing . . . 119
7.7.5 Electronic Signatures with Samples . . . 120
7.7.6 Electronic Signatures on Interactions . . . 121
7.8 Application Scenarios . . . 123
7.9 Related Work . . . 124
7.10 Summary . . . 125

8 Conclusion 127
8.1 Results . . . 127
8.2 Future Work . . . 128

A Platform Specification 130
A.1 Prototypical Implementation of the Smartcard Platform . . . 130
A.2 Language Primitives . . . 130
A.3 Grammar . . . 132

Bibliography 133

Curriculum Vitae 150

List of Figures

1.1 Dependency graph of the thesis' chapters 6

2.1 Service Location Protocol overview 13
2.2 Jini protocols overview 15
2.3 Mobile devices design space 18

3.1 Components of a smartcard 23
3.2 Structure of an application protocol data unit (APDU) 26
3.3 Design space of personal security modules 34

4.1 Overview of the framework for the integration of small devices 43
4.2 Architecture of a generic smartcard terminal 47
4.3 Components of the JiniCard framework 52
4.4 Download and instantiation of card explorers for unknown smartcards 56
4.5 Download and instantiation of card services 58

5.1 The PCA in the context of signing documents 71
5.2 Cryptographic View of Information Flow 72

6.1 GSM authentication overview 80
6.2 SIM application toolkit architecture overview 82
6.3 WebSIM communication stack 85
6.4 WebSIM architecture overview 86
6.5 Examples of WebSIM user interaction 87

7.1 Overview of smartcard platform components 101
7.2 Mobile auction client script 106
7.3 Sample platform control flow 110
7.4 Mobile auction client with interaction signatures 122

A.1 BNF grammar and instruction set of the platform language 132

List of Tables

3.1 Design options of terminal- and smartcard-based personal security modules 31

4.1 Comparison of remote and local device explorers 44

6.1 WebSIM command set supported in the URL interface 87

7.1 Compiler and verifier placement 113

A.1 Smartcard platform primitives 131

List of Acronyms

3GPP	Third Generation Partnership Project
AUC	GSM Authentication Center
APDU	Application Protocol Data Unit
ATM	Automated Teller Machine
ATR	Answer to Reset
B2B	Business-to-Business
B2E	Business-to-Employee
BCV	Byte-Code Verification
CAD	Card Acceptance Device
CBC	Cipher Block Chaining
CHV	Card Holder Verification
CORBA	Common Object Request Broker Architecture
DES	Data Encryption Standard
DNS	Domain Name Service
DTD	Document Type Definition
EAL	(Common Criteria) Evaluation Assurance Level
ECB	Electronic Code Book
ETSI	European Telecommunication Standards Institute
GPRS	General Packet Radio Service
GSM	Global System for Mobile Communication
IANA	International Assigned Numbers Authority
IETF	Internet Engineering Task Force
IMSI	International Mobile Subscriber Identity
IrDA	Infrared Data Association
JC	Java Card
JCRE	Java Card Runtime Environment
JCVM	Java Card Virtual Machine
JVM	Java Virtual Machine
MIME	Multipurpose Internet Mail Extensions
MS	GSM Mobile Station
OTA	Over the Air
OCF	OpenCard Framework

PDA	Personal Digital Assistant
PLMN	Public Land Mobile Network
PSE	Personal Security Environment
RF	Radio Frequency
RFC	(Internet) Request for Comments
RMI	(Java) Remote Method Invocation
RPC	Remote Procedure Call
SAT	SIM Application Toolkit
SIM	GSM Subscriber Identity Module
SLP	Service Location Protocol
SMS	GSM Short Message Service
SMSC	GSM Short Message Service Center
SCA	Signature Creation Application
SSCD	Secure Signature Creation Device
TCB	Trusted Computing Base
TOE	Target of Evaluation
UMTS	Universal Mobile Telecommunication System
URL	Uniform Resource Locator
USB	Universal Serial Bus
USAT	UMTS SIM Application Toolkit
USIM	UMTS SIM
VM	Virtual Machine
WAP	Wireless Application Protocol
WIM	WAP Identity Module
XML	Extensible Markup Language

Chapter 1
Introduction

Humans are incapable of securely storing high-quality cryptographic keys, and they have unacceptable speed and accuracy when performing cryptographic operations. (They are also large, expensive to maintain, difficult to manage, and they pollute the environment. It is astonishing that these devices continue to be manufactured and deployed. But they are sufficiently pervasive that we must design our protocols around their limitations.)

Kaufman, Perlman, and Speciner [KPS95]

1.1 Motivation

The tremendous success of mobile telephony in the last decade has inspired many people to think of how users will communicate in the future with their social and physical environment. Much of the inter-personal communication demand seems to be met by mobile telephony and related peer-to-peer messaging services. For the third generation of mobile communication networks and their built-in support for data services it is expected that users will additionally interact with and make use of mobile services ranging from *mobile commerce*, *location-based information services*, and *multi-media messaging* to *mobile gaming*.

Securing Cyberspace

In the business world of tomorrow, mobile service usage will typically require some kind of *access control* and *billing mechanism* that enables service providers to control access to services and gain sufficient revenue to successfully run their services. This requires to solve a number of security-related problems of which the most relevant ones are:

— **Authentication and authorization:** How do users authenticate themselves before using a service and how is access control implemented?

— **Non-repudiation:** How can users create legally-binding evidence of their participation in a contract?

— **Privacy management:** How can users specify their privacy preferences for the interaction with a service provider?

Usually, appropriate technical measures are only one part of a solution domain but have to be accompanied by legal directives, contracts, and inter-business rules of conduct.

In the "real world", there are standard solutions which all have their counterparts in the domain of digital computers:

— **Authentication:** Traditionally, three different approaches are used for authentication purposes [Amo94, Chap. 18]:

  - **Possession or something possessed:** Something the person has, e.g. an identification card or a ticket. In the digital world, however, pure possession of a digital item is not sufficient, since digital data can be easily copied.
  - **Knowledge or something known:** Something the person knows, e.g. user name, password, PIN number, etc. This is not different in the digital world as long as the person does disclose this knowledge only to trusted components, e.g. trustworthy user terminals.
  - **Characteristic or something embodied:** A physical characteristic of the person, e.g. fingerprint, retina, speech and handwriting recognition, etc. Again the trustworthiness of the device performing the "scanning process" is crucial in the digital world.

  The security level obtained by these approaches depends on the likelihood that an attacker is able to gain access to the assets that are used to perform the authentication with. Thus, if an attacker manages to "steal" an item used to authenticate a user based on the possession of that item, the authentication process is undermined. Similarly, if an attacker gains knowledge of what a person knows or what a person is, attacks on authentication systems based on this knowledge or characteristic become possible.

— **Authorization:** Authorization is often implemented using authentication to prove the client's identity in combination with some access control decision about who is allowed to perform which operation. Another option which is in place in most of today's non-digital world is to directly use some kind of "ticket" – possessed by the owner – that directly represents the permission to grant some kind of operation.

  In the digital world, however, such digital tickets are potentially subject to theft and illegal multiplication. This requires that upon usage of a ticket its holder additionally has to prove correct ownership, often using cryptographic measures.

— **Non-repudiation:** Non-repudiation in the real world is usually enforced by the use of handwritten signatures. The digital counterparts are usually based on cryptographic one-way functions and public key cryptography, and thus are vitally dependent on cryptographic measures.

  General problems in the context of non-repudiation are the necessity of suitable *time stamps*, the *settlement* in case of *conflicts*, the provision of a trustworthy infrastructure based on public key cryptography, etc.

Hence, in contrast to the real world, implementing security measures in the digital world requires to a significant extent the capability to perform cryptographic computations. However, in practice cryptography is still not widely used in the field of mobile services, especially not in the consumer market. Hence, although the fundamental solutions are well-known the practical embedding into these settings has yet to come. We think that the reason for this *status quo* is the lack of solutions for the following set of general requirements:

- The pervasive availability of mobile devices that users can use to perform cryptographic operations and which easily integrate into a user's communication environment.

- The possibility for a user to safely carry around security-critical data such as cryptographic keys. This is necessary since according to [Sch00, Chap. 9] users can hardly memorize cryptographic keys of a length needed for a secure cryptographic system.
- The general reachability of a user to perform security-critical decisions anywhere and anytime.

This thesis proposes solutions to all of the above requirements based on *smartcards* as the core components of a *personal security module*.

Towards Smartcards as Universal Personal Security Modules

Smartcards are tamper-resistant hardware modules that are able to securely store secret cryptographic keys and perform unobservable execution of cryptographic algorithms. They can assist users in all of the above mentioned activities. Hence, they are ideally suited to serve as personal security modules especially in mobile scenarios.

Today smartcards are mostly used in an *application-specific* manner, i.e. for a particular application there exists a particular smartcard acting as the application's security module. In future mobile service scenarios, however, this approach does not scale for the many services people might use. The reason is simply that potentially for each of these applications a smartcard needs to be issued to a user. This dilemma might lead to the general requirement that security modules are needed that are universally usable in many different application scenarios.

Consider, for example, the scenario of a mobile user who has a daily subscription for a digital newspaper that can be downloaded as soon as the user enters the public transport system, e.g. a bus. The user's information appliance automatically integrates into the bus' network by means of *spontaneous networking* and searches for the subscription download server. Both establish a cryptographically protected channel, the server challenges the appliance and after the user's security module could successfully reply with the correct response, download of the newspaper is started. For means of simplicity the user's subscription is not checked on-line but an *authorization certificate* is used instead.

This example illustrates the role a user's personal security module could play in future scenarios. The module not only safely keeps the authorization certificate but also computes the response of the server's challenge. Generally, this raises several questions:

- How must a personal security environment based on smartcards look like in mobile scenarios?
- How can devices like smartcards with limited resources in terms of computational power and communication bandwidth be efficiently integrated into such environments?
- What security characteristics have to be met by the appliances and terminals used to interact with the personal smartcard?
- What level of security can be achieved in any of the proposed solutions?

The overall scope of the questions raised above is not focused on a single problem domain. Therefore, a more holistic view of mobility and security is need. We adopt such a view in this thesis and contribute novel solutions at the architectural level of mobile security and to the design of interaction protocols in this area. All solutions can be characterized as extending the functionality of smartcards in a flexible, but still secure way.

1.2 Contributions of this Thesis

The starting point of this thesis is the vision of future mobile scenarios in which users are likely to use services "spontaneously", i.e. in an *ad hoc* manner. Location-dependent services, i.e. services that are only available at a particular site will play a central role in such scenarios and service usage will occur to a significant extent through mobile devices and appliances the users carry around.

Before service interaction actually takes place, however, enough information and communication parameters must be obtained by a device and its user to allow for elementary communication with an environment and its services. Subsequently, services can be located and relevant communication parameters can be negotiated. In mobile scenarios such *a priory* knowledge is usually not available since the peers do not know each other in advance. *Spontaneous networking* is a mechanism to make this state transition as seamless as possible for a user by providing infrastructural support that enables a device to explore the characteristics and services of an environment more or less automatically. After services have been located, users need appropriate *security modules* to enable them to control and enforce the security properties they desire in the communication with these services.

In this thesis we argue that a convenient way to implement such a security module is to use a smartcard and a suitable terminal needed to access and control the smartcard. Such security modules can be subject to a number of categorizations leading to a design space for these devices. We always assume that the smartcard is *personalized*, i.e. that it contains information that is directly bound to its owner.[1] A typical example is a private key in a public key cryptography scheme that is bound to the user's identity. The terminal itself can also be personalized which usually implies that it is somehow "owned" by a person. However, it is left open, whether it is personalized to the smartcard or its owner, or both. Furthermore, it can be *mobile*, i.e. it is carried around by its user, or *non-mobile*, i.e. located at particular sites and potentially used by many different users independently. Finally, the card and the terminal can be *co-located*, i.e. they are physically attached to each other, or *remotely* connected.

Based on this categorization we have identified four different fundamental problems that have been solved using suitable approaches:

A1: A spontaneous networking framework for smartcards

This approach solves the problem of "spontaneously" integrating smartcards into a networked environment. It consists of a framework that allows a smartcard to spontaneously integrate itself into a local environment after insertion into a suitably equipped card terminal. It builds upon the idea of using mobile code as an enabling technology to complement the card-resident resources with off-card resources in a dynamic and flexible way.

A2: A personalized and trustworthy mobile terminal

This approach solves the problem how smartcards can be used in a so-called "hostile" environment. Such environments are characterized by the lack of confidence the user has into the available terminals where the personal smartcard is to be used. The solution takes advantage of a personal terminal the user carries around such as a *Personal Digital Assistant* (PDA) which is trusted by its owner. The PDA turns into a powerful front-end for the smartcard and both, PDA and smartcard, are *cryptographically bound* together such that none of the two devices can be used without the other.

[1] In the rest of this thesis we will use the terms *personalized* and *personal* in a similar fashion since we consider anything that is personalized to be personal to that user.

A3: A mobile terminal with a wireless mobile communication link

The third approach presents a solution to the problem how personal security modules can be used virtually anywhere and anytime. Our *WebSIM* system builds upon the idea of a wireless smartcard reader in the form of a mobile phone. We have made smartcards reachable from the Internet by means of an architecture that allows a the GSM SIM smartcard in the mobile phone to appear as a Web server in the Internet. Other Internet nodes can connect to the SIM using the HTTP protocol which is transparently forwarded the mobile phone the SIM is attached to. Hence, the SIM as a personal security module of a user is accessible from the Internet anytime and anywhere the user is located.

A4: A personalized smartcard that allows for the execution of mobile code supplied by a service provider

The fourth approach solves the problem how the security context of a tamper-resistant smartcard can be used to build value-added services on top of. The *SIMspeak* solution consists of a card-resident runtime execution platform for mobile code, a program verifier, and suitable mechanisms that allow a service provider to “rent” secure storage in a smartcard. This platform can be used by service providers to send mobile code into a customer’s smartcard (possibly over a wireless link). This opens up new application domains and leads to improvements in the way how non-repudiation in the form of electronic signatures is achieved.

Summing up, this thesis shows how to architect and build solutions for the personal security module approach. It contributes to the overall understanding on the role smartcards can play as personal security modules in future mobile communication and service usage scenarios and, therefore, represents a significant step towards meeting the security demands of the information society of the future.

1.3 Organization of this Thesis

Figure 1.1 on the following page shows the overall structure of this thesis which is organized as follows:

- Chapter 2 presents the basic motivation underlying this thesis by giving a vision of future mobile scenarios and describes the setting in which subsequent solutions are located. In this vision local and remote services will be used through suitable terminals. It is particularly focused on mobile devices and discusses the aspects of “spontaneously” integrating such devices into local environments.

- Chapter 3 discusses which security problems exist in the presented settings and which solutions are possible. In particular it considers and analyzes the design space of smartcards and terminals as components of a personal security module. This leads to a categorization of the different conceptual possibilities from which concrete approaches in the subsequent chapters are derived.

The following chapters each discuss the approaches A1 through A4 presented in the previous section:

- Chapter 4 represents approach A1 by describing a framework for the integration of smartcards into local environments. The solution essentially defines a framework that enables a smartcard

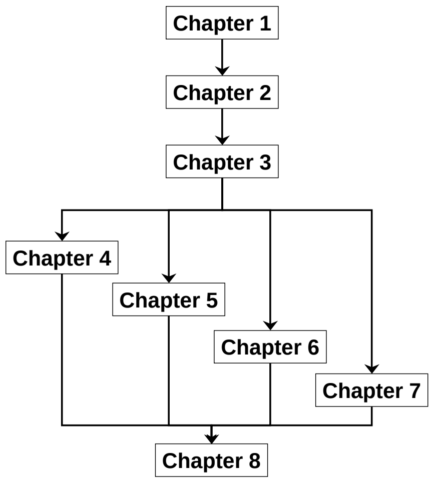

Figure 1.1: Dependency graph of the thesis' chapters

to offer security-specific services in a network. These services can then be accessed from any terminal attached to the network.

- Chapter 5 represents approach A2 in which a personalized and trustworthy terminal and a smartcard together form a pair of devices that assist a user in performing security-critical tasks. Both are linked together using cryptographic measures to prohibit the use of one device without the presence of the other. To demonstrate the general usefulness we have applied this approach to the problem of electronic signature creation in hostile environments.
- Chapter 6 represents approach A3 that comprises a personalized smartcard and a wireless terminal. A GSM mobile phone was used as the mobile terminal and a GSM SIM smartcard was chosen to implement the personalized security module. Due to the *mostly-on* aspects of GSM mobile phones and SIMs and their pervasive use, both can play an important role as security modules in future mobile service scenarios.
- Chapter 7 represents approach A4 focusing on the idea of a mobile code platform inside a smartcard. Again we have chosen the combination of a GSM mobile phone and a SIM as our target of evaluation since it seems to be optimally suited for this approach. The architecture and relevant protocols are described and how this approach leads to several improvements in the process of electronic signature creation such as *on-card hash computation*, *recipient addressing*, *third-party assisted signatures*, *samples*, and *signatures on interactions*, especially in mobile environments.
- Chapter 8 draws final conclusions and discusses future work.

Related work is discussed in each of these chapters individually.

Chapter 2
Connected Devices, Services, and their Users

. . . a billion people interacting with million e-businesses
with a trillion intelligent devices interconnected. . .

Lou Gerstner, IBM Chairman and CEO

2.1 Introduction

According to Durlacher Research [Dur01] the mobile telecommunication sector has invested roughly 120.000.000.000 Euro in the years 2000/01 for the *Universal Mobile Telecommunication Services*[1] (UMTS) licences in the Western European countries. Obviously, the telecommunication operators are interested in turning this major investment into a success story. Key factor of UMTS is its design to enable significantly improved data services and information delivery in the third generation telecommunication networks. These features are considered to be key distinguishing factors between the second generation systems, e.g. GSM, and the third generation systems (3G).

UMTS could indeed serve as one of the basic communication technologies of a new paradigm of computing – often called *ubiquitous* or *pervasive computing* – in which numerous devices, information appliances, and personal gadgets assist (mobile) users in their everyday life. This is likely to be accompanied by new short-distance communication technologies such as Bluetooth [Blu01] and Wireless LAN [IEEE802.11].

In the citation that precedes this chapter Lou Gerstner expresses his understanding of the notion of *pervasive computing*. Another possible definition is given by IBM saying that pervasive computing is about

> "[. . .] *convenient access, through a new class of appliances, to relevant information with the ability to easily take action on it when and where you need to.*" [IBM01]

This chapter presents a vision of how computing will evolve in the future, what technological challenges arise from this vision, and what basic technologies are present today with which parts of this vision can be tackled.

The rest of this chapter is organized as follows: In Sect. 2.2 we describe the pervasive and ubiquitous computing paradigms that play a central role in our vision of future computing environments.

[1] UMTS is currently being specified in the *third generation partnership project* (3GPP) at [3GPP01a].

A central technology is needed to turn this paradigm from a vision into reality: *Spontaneous networking*. It essentially deals with the integration of people, devices, and services into networked environments in a largely autonomous manner. In Sect. 2.3 we describe the fundamental problems of spontaneous networking and give a definition of *spontaneity* and spontaneous networking which can be essentially understood as the integration of services and devices into networked environments without manual intervention.

An overview of current technologies that enable spontaneous networking is given in Sect. 2.4. Two representatives, namely the *Service Location Protocol* and *Jini* have been selected among the currently available approaches to study the common and distinguishing concepts of these technologies.

The comparison of spontaneous networking technologies leads to the observation that small devices are currently not well supported by off-the-shelf technologies. Hence, the design space of such devices is investigated in Sect. 2.5 leading to the observation that today and in the future there will be always devices around that need special assistance from their surrounding infrastructure.

The chapter closes with a summary in Sect. 2.6.

2.2 On Ubiquitous and Pervasive Computing

The original idea of *ubiquitous computing* was created by Mark Weiser in his visionary paper "*The Computer for the 21st Century*" [Wei91] commencing with the statement that

> "*The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*"

According to Weiser, ubiquitous computing is driven by a new major trend in computing, characterized by the fact that users own, control, and use many different devices and appliances throughout their daily life. This is in contrast to the mainframe era in which many people shared a single computer, and the PC era in which a person exclusively uses one PC at a time.

Technological progress especially in the micro-electronics sector is considered to be one of the major driving forces behind the possibility of implementing ubiquitous computing [Mat01]. Examples of such technologies are advances in the resolution and size of micro- and macro-displays, short-range communication such as radio frequency tags (or smart labels) and the Bluetooth communication standard, new global communication technologies such as UMTS, sensor technologies, smartcards, power-based networks, e.g. body area networks, or smart paper paving the way for new forms of computing.

Whereas the currently popular human-machine interaction paradigm is a mere one-to-one relation, ubiquitous computing especially in the shape of *calm* or *disappearing* technology is about hiding computers into the users' environment, making them as invisible as possible, performing their task often calmly and mostly unnoticed by the users [WB96]. Weiser's vision is likely to require a complex infrastructure to surround users and it seems that his vision might require another decade to become reality.

In contrast to Weiser's vision, the industrial term *pervasive computing* does not directly address the idea of calm computing but follows a more pragmatic approach in which users and their access to information through suitable appliances are of primary concern. Revisiting the tremendous success of mobile telephony in Europe in the last decade this might be a first indication that we are indeed at the beginning of the pervasive computing era and mobile phones seem to be the initial enablers.

Personal digital assistants (PDAs) are also becoming more wide-spread and future information appliances seem to follow an evolutionary approach by merging the features of mobile phones and PDAs into one device.

Regardless of whether the future will be termed pervasive or ubiquitous computing, it is unclear how an information appliance will look like in three, five, or ten years from now, but people are likely to continue to carry around some kind of "smart" gadget through which they communicate with their physical and social environment. Thus, mobility is one concept that is of particular importance in shaping the future of computer science.

2.3 Issues in Spontaneous Networking

Besides the mobility of users and their devices and the spirit of "computers everywhere", pervasiveness also considers the idea that many interactions between users and their environment are location-dependent. From a beverage vending machine in a public place, a cashier in a supermarket, and a telematics application in the automotive sector to an access control barrier in a company building, users are about to interact with other mobile or non-mobile devices to use or provide services.

Here, the term *service* should be considered as a very general concept covering all kinds of "resources" a peer provides which is roughly captured by the following definition:

> *A* service *is a* concept *with arbitrary implementation which offers an* interface *that is addressable and accessible from peer nodes.*

Hence, it is not important whether the service is implemented as a CORBA object or a Web application, but that it is addressable and accessible from clients. Furthermore, we assume that a service can be described by means of a suitable *service description*, i.e. a formal representation that can be used by a peer to learn about a service's capabilities.

This raises new interesting questions, e.g. how can communication links between entities be "spontaneously" established? A good example of spontaneous integration is GSM roaming in a foreign country when a mobile phone discovers the available *public land mobile networks* (PLMN) and books into one of the accessible ones. Furthermore, it is of particular interest how such communication links can be kept static while being mobile. Again GSM is a nice example how communication can continue although the mobile phone subsequently establishes new radio communication links with so-called *base transceiver stations* (BTS) while moving. As such GSM is a tremendously complex technical system that was specified and compiled into standards over a period of several years.

However, the time-frame for development and deployment of today's information and telecommunication products often does not allow for such long standardization processes, and heterogeneous environments and different technologies to solve the spontaneous networking problem are likely to coexist in the future.

On Spontaneity

Since in ubiquitous and pervasive systems spontaneous interaction among devices, services, and users is one of the fundamental operations we will now study the causes and effects of spontaneity in such systems in more detail. Although there might be a vague common understanding of what *spontaneous networking* – or *spontaneity* for short – means in social terms, a more technical definition is not widely accepted. A possible definition has been given in [PC99] as

"[...] *spontaneous networking is* [...] *the integration of services and devices into network environments with the objective of an instantaneous service availability without any manual intervention.*"

Put differently, the integration should be automatic and the necessary information required to interact with other components and services in the environment is obtained through appropriate measures. Thus, "spontaneous" is how this process of resolving the information gap appears to users.

This above definition tries to cover cases from GSM roaming and infrared beaming to plugging an Ethernet cable from a laptop into a local area network. Spontaneous networking deals among others with the following set of questions:

- How can devices physically interact with other nodes in a telecommunication or simple peer-to-peer network?
- How can necessary communication parameters be probed or exchanged?
- How does advertising of services work and how can potential clients locate services and vice versa?
- How can such devices be given access to services of peers?

Answers to these questions will given throughout the course of this thesis.

Layers of Spontaneous Interaction

The above questions must be answered at least at the following communication layers:

— **Physical and link layer (OSI layers 1 and 2):** Different technologies are available for interconnecting devices. Basically any technology currently found needs to offer some support for spontaneity such that new communication partners are able to communicate with each other. For mobile devices at least the following technologies may be of interest in the mid-term future: Infrared, e.g. IrDA [IrD01], Bluetooth [Blu01], DECT, Ethernet, UMTS, Wireless LAN, etc.

— **Transport protocol(s) (OSI layers 3, 4, and 5):** Different transport protocols are necessary to communicate among devices, ranging from TCP/IP to lower-level protocols based on the underlying link technology (e.g. the IrDA session protocols IrOBEX, IrLAN, etc.). It is important to notice that the transport protocol should ensure that the device is somehow addressable from the communication layer (e.g. by assigning an IP address). On top of the core transport protocol other protocols such as SSL/TLS [RFC2246], HTTP, etc. could be used resulting in a complete communication stack that links the communication partners.

— **Infrastructure and service discovery (OSI layers 7 and above):** This topic essentially addresses issues concerning the kind of middleware used for service description and discovery. In a more general context it must be able for a pervasive device to explore its "environment" and "find" services residing on peer nodes it can use. Examples of such service-trading middleware are the Service Location Protocol [RFC2165], Jini [Wal99; Sun99a], and Universal Plug and Play [UPn99b]. Protocol gateways might be necessary to map between these different kinds of technologies. Beyond pure service-trading, the underlying communication middleware such as RPC, CORBA, DCOM, or Java RMI adds another level of complexity for solving the integration problem.

— **Service interaction (beyond OSI model):** After clients and services are connected to each other there might be an additional phase needed to negotiate protocol versions and other parameters.

Summing up, spontaneous interaction usually happens at different layers in a communication hierarchy. Hence, before the user of a pervasive device is able to use a service numerous different spontaneous interactions take place at different levels of communication.

Spontaneity Caused by Mobility

The above investigation of spontaneous interaction allows us to more concretely define what the term *mobility* means in conjunction with spontaneity. Hereafter, we define mobility as follows:

> *Mobility is the movement of an entity in the physical environment which leads to* changes *in the digital representation of the entity's world.*

For all communication technologies generally considered to be *short-range* this means that an existing link is often lost after the range of the communication system is left. Typically, being away more than ten meters from an Infrared receiver implies that communication is not possible anymore. Therefore, a mobile device being confronted with a new physical network, e.g. a new Infrared link, has to bootstrap the "spontaneous" communication stack from the bottom up to the higher levels.

Hence, mobility in conjunction with a short-range communication link implies spontaneity at the very lowest level in the communication stack. The overall process can be considered as a transition from a state with minimal *a priori* knowledge about the environment into a state where sufficient information is available. The initial state itself is caused by the user's mobility. The amount of an entity's *a priori* knowledge in the initial state generally includes information about the built-in communication technologies and protocols supported by that entity. Furthermore, we assume that the communication partners need to exchange information to learn of each others' capabilities, communication parameters, and environment information to reach a state of knowing each other's profile.

Spontaneity caused by Unknown Peers

In contrast to the form of spontaneity caused by mobility, we are also faced with the problem of spontaneity caused by unknown peers at higher levels in the communication stack. Revisiting the definition of spontaneity which defines the instantaneous use of services as one fundamental principle, spontaneity also occurs when a Web browser contacts a Web server and tells it the browser's supported document or MIME types [RFC2045]. Another example is the handshake protocol run during SSL/TLS session setup [RFC2246]. Many Internet protocols have some kind of negotiation phase in which the peers exchange necessary information and parameters for further communication. However, this is not caused by mobility and interaction at a lower level, but interaction at a higher level in the communication stack.

2.4 A Brief Overview of Spontaneous Networking Technologies

In the previous section it was identified that the infrastructure and service discovery problem is central to the notion of spontaneous networking since it is where the actual service advertisement and lookup takes place. In the sequel a brief overview of current technologies for service description and discovery is given with a particular focus on the SLP [RFC2165] and Jini [Sun99a; Sun99b] technologies.

2.4.1 Fundamental Concepts

Technologies that support spontaneous networking are centered around the following fundamental tasks:

- **Service advertisement:** A component that wishes to offer a service to potential clients needs a well-defined procedure to "register" the available services and make them accessible for clients.
- **Service lookup:** A client that is in search of a suitable service needs a well-defined procedure to locate potential service providers.

For the implementation of the advertisement and lookup a common framework is needed to describe and query services. A so-called *service description* is used to create a machine-processable description of a service that is made available to potential clients.

The typical work-flow in a particular spontaneous networking technology usually follows some generic pattern:

1. **Service description creation:** This is often done as part of the configuration of a service or directly supplied by the service itself.
2. **Service advertisement:** This describes the process of making the service description available to potential clients. This may include the transfer of the service description to a central authority such as a service directory, etc. It often requires appropriate *one-to-many* communication protocols such as *broadcast* or *multicast* to "find" such central authorities.
3. **Client lookup:** Clients in search of a particular service assemble some kind of *service query* that is sent to the particular component(s) in charge of matching service descriptions with corresponding queries. Similar to service advertisement this also might require the use of appropriate multicast protocols to find the peers maintaining service description information.
4. **Mutual binding:** After clients have successfully found appropriate services both have to connect to each other to enter the service usage phase.

The above tasks are to some extent independent from the underlying communication link except for the possibly needed multicasting features which are usually highly dependent on the underlying technology. More precisely, such systems usually require a fully functional communication stack that is properly configured to perform their task. Any issues related to the configuration of this stack are therefore beyond the scope of these technologies.

2.4.2 Case Studies

Based on the overview of the tasks a spontaneous networking technology has to support we investigate two representatives of the available systems in more detail, namely, the *Service Location Protocol* and *Jini*.

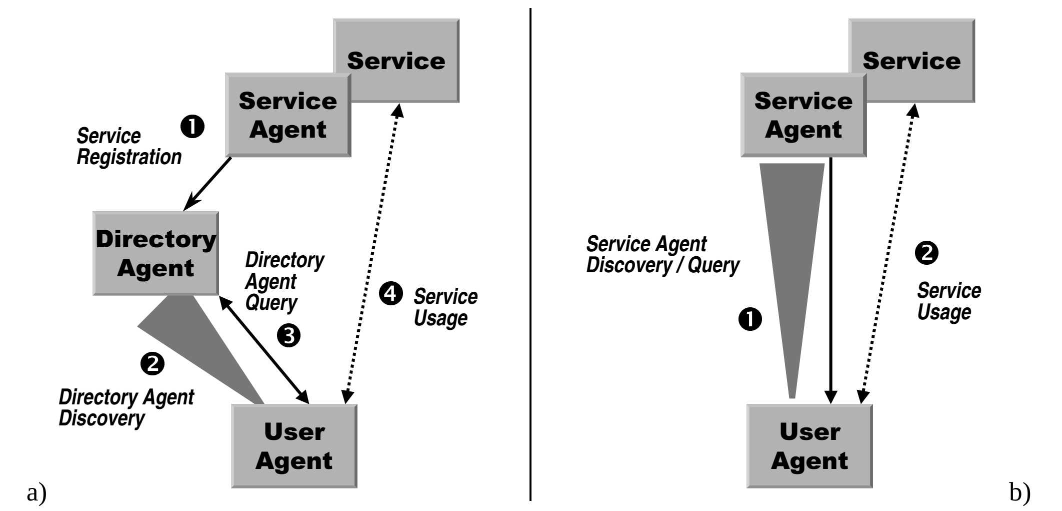

Figure 2.1: Service Location Protocol overview: a) with directory agent, b) without directory agent

2.4.2.1 SLP

The Service Location Protocol (SLP) [RFC2165] is a framework for spontaneous networking on top of IP developed in an Internet Engineering Task Force (IETF) effort. SLP builds around the following components:

— **Service and service agent:** A service agent is a component that offers services in a network. It may implement the service itself or just offer it on behalf of the actual service.

— **Directory agent:** A directory agent is responsible for collecting all service descriptions from service agents and offers a central repository for user agents searching for services.

— **User agent:** A user agent is a component that tries to find a particular service, i.e. it is a client.

User and service agents locate a directory agent using DHCP options [RFC2131] or IP multicast messages. Service agents register their service descriptions with the directory agent which is able to execute queries from clients against the registered services in the directory (see Fig. 2.1a).

However, SLP can also work without the presence of a central directory agent. This requires that service agents are themselves able to answer the queries of user agents (see Fig. 2.1b). A user agent then needs to wait for a certain period of time to collect all answers from service agents until it has a clear picture where the services it desires are located.

Service *descriptions* in SLP consist of a *time-to-live* field denoting the validity period of the service description, a *service URL* defining the type and location of the service, and a set of string-based key/value-pairs describing the service's *attributes*. Service agents have to refresh their registration with the directory agent before the time-to-live runs out. Part of the service URL is the service's type denoted by a string and the host, port, and path information as known from Web addresses. SLP suggests to register service names with the IANA organization [IAN01] to standardize popular services.

Service *requests* sent by querying user agents may contain the desired service name and a Boolean expression over the available key/value-pairs. This allows clients to send more complex filters to the directory agent or service agent for evaluation.

Other features of SLP are so-called *scopes* which allow for grouping of services and mapping of scopes to directory agents.[2] Furthermore, SLP allows to electronically sign service descriptions using so-called *authentication blocks* to provide for basic security in the registration process.

Summary

SLP was one of the first technologies addressing the problem of spontaneous networking. It has been designed in the traditional Internet spirit to make things as small as possible and as a consequence being easy to implement. As a result, besides a working TCP/UDP/IP stack SLP requires just basic computational resources and may be easily implemented on small devices.

2.4.2.2 Jini

Jini™ [Sun99a; Wal99; Edw99] is a Java-based framework developed by Sun Microsystems™ and is publicly available since January, 1999. Jini is built around the following components:

— **Service:** A Jini service is a network resource hosted on an arbitrary network node. Typically a service is implemented as a Java object.

— **Service proxy:** The proxy is a so-called *serialized* Java object [Sun98] representing the service. Jini uses Java's mobile code facilities to transfer such a proxy object from the service to the client as part of the service description.

— **Lookup service:** The lookup service [Sun99c] is the central directory where service providers advertise their services. Services get in contact with the lookup service using IP multicast-based *discovery protocols* [Sun99b] (see Fig. 2.2a). After a service provider has located a lookup server it "uploads" the service proxies together with further service description information in the format of serialized Java objects to the lookup server (see Fig. 2.2b). Technically, this is achieved by first downloading a lookup service proxy object from the lookup server which in turn is used to advertise the service (so-called *join*).

— **Client:** A client is an entity that uses services to perform a certain task. In Jini a client is required to run a Java Virtual Machine (JVM) to be able to download and execute the service proxy (see Fig. 2.2c). In contrast to the service, Jini is vitally dependent on the clients' support for executing the service proxy within a JVM.

Jini uses serialized Java objects to implement a service description. In particular the information sent from the service provider to the lookup server – the so-called *service item* – is a set of serialized Java objects including the service proxy which describe the service and its validity period. The lookup server is able to use Java's type system to match clients' queries – called *service templates* – with registered service types.

Besides the server-side registration of a service and the client-side queries the lookup service also supports so-called *distributed events* that enable interested parties to subscribe with the lookup service for the occurrence of particular events, such as, the appearance or disappearance of specific services.

[2] In SLP, scopes can be essentially considered as name spaces for services managed by directory agents.

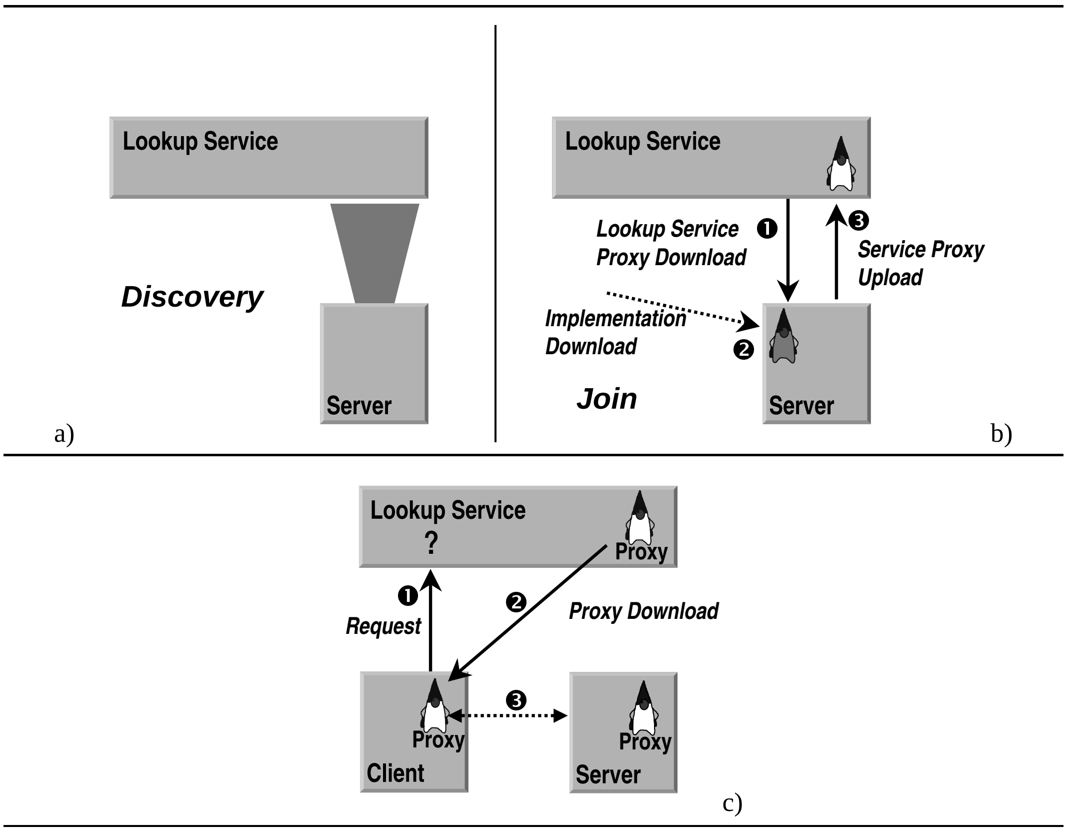

Figure 2.2: Jini protocols overview: a) *discovery*, b) *join*, and c) client-side *proxy download*

Lookup services can also be associated with *service groups* which allow to separate the responsibility of lookup services to manage certain groups only. However, this feature in not a real group concept, but rather a simple optimization scheme for managing work load.

Summary

All components in a Jini scenario require a working TCP/IP stack. Clients and servers locate the lookup server using IP multicast packets that are sent to a well-known IP address and port number. This information is shared by all participants in advance to be able to find each other, further communication is done through TCP.

As mentioned previously, all clients need a working Java VM and require a minimum Java API to be able to successfully execute the service proxies downloaded from the lookup service. Jini services require a JVM if they register through the lookup service's proxy – which depends on the particular lookup service implementation. Jini is a well-designed framework that tremendously simplifies the implementation of spontaneous networking facilities. Furthermore, the concept of proxies allows to run arbitrary protocols between the proxy and its service. This can heavily simplify the implementation of a client for a particular service, since the client needs to know only a high-level "Java interface" to talk to. Otherwise a low-level communication protocol would be needed if it would directly talk to the service itself. This is actually the most interesting concept that has been introduced by Jini.

However, Jini requires a significantly large infrastructure for both – clients and servers – to successfully serve as a spontaneous networking technology and it always requires the presence of a central registry – the lookup service. More precisely, it cannot be easily used in environments with limited devices that are not able to run this infrastructure.

2.4.2.3 Related Technologies

Besides the two presented technologies numerous other systems and frameworks have been proposed. In the following paragraphs we present some of the more interesting ones.

The *Home Audio Video Interface* (HAVi) [Con98] is an initiative targeting the spontaneous integration of audio and video devices. It builds upon the IEEE 1394 (FireWire) technology and offers protocols and facilities for service registration and binding.

Salutation [Sal97] has focused on the integration of typical office devices such as printers and fax devices. It builds upon Sun's ONC RPC [RFC1057] protocol and defines entire device hierarchies including their interfaces.

Universal Plug and Play (UPnP) [UPn99b; UPn99a] has been developed by Microsoft™ and Hewlett-Packard™ and beyond a pure framework for spontaneous networking it also defines a model for devices and the services offered. The design of UPnP is largely influenced by SLP although the protocols are largely based on HTTP for communication and XML for service description and communication between clients and servers.

The *Service Discovery Service* (SDS) [CZH^{+}99] was developed as part of Berkeley's Iceberg project [Ice99]. SDS servers are central authorities used to collect the service announcements sent by services. Service announcements consist of XML documents describing services according to a certain XML *Document Type Definition* (DTD). Clients query SDS services using so-called *templates* that are matched against the known service descriptions using a high-performance XML matching engine. In contrast to most other systems, SDS is designed to support essential security mechanisms such as encryption, integrity protection, and authentication.

The *Intensional Naming System* [ASBL99] is a service announcement framework combined with an optimized routing scheme for service queries. Intermediate nodes in a network aggregate service announcements and store "routes" to neighbour nodes closer to a service. This is possible since the description of services is chosen in a way that eases the aggregation of service descriptions for building compact routes.

DEAPspace [Nid99] is a technology for providing spontaneous networking facilities for *ad-hoc networks*. The design of DEAPspace was heavily influenced by the demand for a fast service discovery mechanism used in conjunction with mobile devices. Its outstanding feature is its collaborative architecture in which all components together cache the announcements of services and serve clients' queries. This leads to a generally very fast responsiveness especially useful in mobile scenarios.

Bluetooth/SDP [Blu01, Part E] is the service discovery part of the Bluetooth™ protocol suite. It exploits features of the underlying radio link and is optimized towards small bandwidths and fast responsiveness. It is completely decentralized and offers only limited facilities for service description and matching, however, making it easily implementable on devices with limited computational resources.

UDDI [UDDI01] stands for Universal Description, Discovery and Integration. The UDDI Project is an industry initiative that is working to enable businesses to quickly, easily, and dynamically find and transact with one another. UDDI enables a business to a) describe its business and its services, d) discover other businesses that offer desired services, and c) integrate with these other businesses. In contrast to most of the previously discussed approaches, UDDI is intended for locating services in a wide-area environment like the Internet. As such, it does not include any multicast-based discovery mechanisms but is based on a traditional client/server interaction model.

2.4.3 Summary

Several technologies have been proposed for approaching the spontaneous networking problem. Their most relevant distinguishing parameters are:

— **Design:** As shown there exist purely centralized systems like Jini, others operate purely decentralized, and systems such as SLP offer hybrid schemes of operation.

— **Service description richness:** Each of the systems uses different techniques for representing services. Some of the systems allow for rather complex matching algorithms comparing requests with advertisements (e.g. SDS), others allow only for very limited service descriptions (e.g. Bluetooth/SDP and DEAPspace).

— **Performance:** Some of the systems (e.g. DEAPspace, Bluetooth/SDP) are better suited for mobile environments where responsiveness might be a critical issue.

— **Scalability:** Systems such as SDS and SLP are likely to scale better for large installations, whereas others might be only suited for small installations in the home/office domain.

— **Security:** Besides the authentication features of SLP, only SDS offers built-in security for all steps in the component interaction.

— **Resource consumption:** Several components, e.g. the Jini lookup service or the SDS XML engine are rather complex systems requiring reasonable resources in the network. In contrast, the peer-to-peer systems such as DEAPspace and Bluetooth/SDP have been optimized for resource-limited systems sacrificing the abstraction layer introduced by the former systems.

Summing up, different application domains might have different requirements for a spontaneous networking technology. There is often a certain trade-off between the distinguishing factors listed above, making general statements about the usefulness of these technologies very difficult. Smartcards, for example, are not supported by any of the previously presented technologies.

We now sketch how such an infrastructure must look like in order to support very small and resource-limited devices to properly integrate into an environment.

2.5 On the Necessity of Infrastructural Support for Small Devices

The technologies for spontaneous networking presented in the previous section require a minimum amount of computational resources not offered by many mobile devices. Figure 2.3 on the next page shows the design space of such devices concerning their capabilities according to three different dimensions.

- The first dimension denotes the amount of *computational* or *processing power* a device has. Here we subsume the three main items *CPU performance*, *memory size*, and *size of persistent storage*, into one singular dimension, since usually all three of them grow at similar rates as devices get larger.

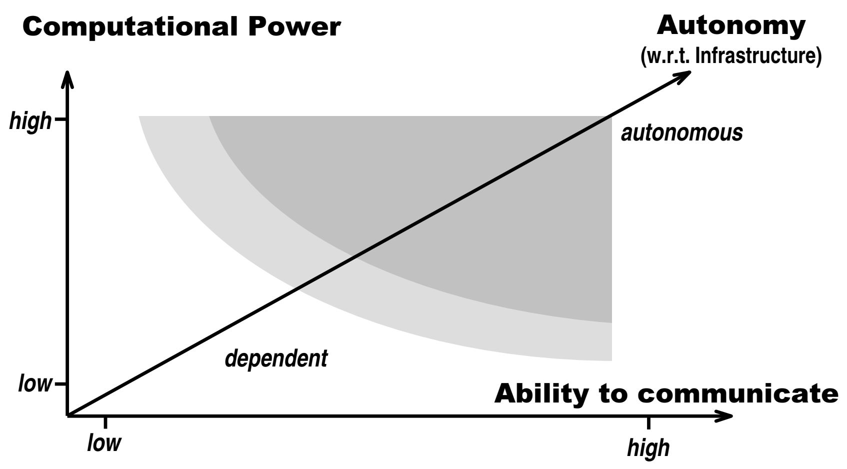

Figure 2.3: Mobile devices design space

- Along the second dimension we consider the *ability to communicate* as another relevant factor. It includes *bandwidth*, the underlying communication technology, and discrete issues, such as *on-* vs. *off-line* operation and support for *uni-* vs. *bi-directional* communication.
- The third dimension is *autonomy*, an indication of how dependent a device is on support of its surrounding infrastructure. Usually, autonomy depends on the first two dimensions and is therefore not truly orthogonal. The shaded regions are meant to indicate that the more powerful a device is along the first two dimensions, the more the device is likely to be autonomous.

As an illustration of our design space certain device categories can be classified. This classification gives a rough impression of the character of these devices and the kind of support they need from the infrastructure for proper operation.

— **Sensors:** Sensor devices are equipped with only minimal computing power and memory size. Their communication facilities are wire-based or wire-less, e.g. Infrared. These devices are pure information sources, cannot be controlled or managed from outside, and thus have a uni-directional link. Additionally, they may send their information in short time intervals in contrast to continuous transmission.

Typical examples of such devices are the Infrared-emitters in the Active Badge location system [WHFG92] or sensors in medical health-care systems such as the temperature sensing device described in [SA99].

To make use of these devices, appropriate receivers must be installed at every location they are to be used at. Information processing occurs outside the devices somewhere in the infrastructure and/or at the application level.

— **Actuators:** In contrast to simple sensors, these devices are able to receive commands that can in turn control the device, or perform actions with the actuators. They implement a protocol that can be used to adjust parameters such as rate of sensing, power management, etc. Typical actuators are flow control devices, e.g. valves, that are opened or closed by the actuator. In our design space these devices are probably located in the light grey shaded area of Fig. 2.3. Many embedded systems probably fall into this category.

Since some of these devices may not be constantly on-line, e.g. for reasons of battery consumption or aspects of mobility, interaction with these devices will mostly happen via a proxy object for that device running somewhere on a network node. This proxy knows about the on-line times of the device and exchanges data in both directions. The device proxy can be constantly on-line and accepts commands for parameter changes. The proxy waits until the next time the device is on-line and then transmits the control information to the device. It is up to the proxy to implement a synchronous or asynchronous API for its clients.

— **Information-processing devices:** For devices of this category we can assume that they offer enough computational power to perform most of the tasks including the integration into an infrastructure on their own, though they still might lack enough memory or bandwidth to perform resource-intensive tasks. The dark shaded area in Fig. 2.3 might be the place to find devices of this category.

  State-of-the-art handhelds and PDAs are likely to fall into this category.

Although the categorization performed is rather coarse-grained we consider it a valuable instrument to roughly sketch the *autonomy* and *dependability* of a device from its surrounding infrastructure.

Prediction on the Integration of Small Devices

Usually, devices belonging to the sensors and actuators category require some kind of support from their infrastructure to be spontaneously integratable into environments. More generally, one can argue that this is just a temporary problem that vanishes automatically when these devices are equipped with more computational resources, an expectation that is met with most of today's electronic artifacts.

However, as devices grow the desire increases to integrate other small devices that have previously not been considered as integratable. Hence, our fundamental prediction can be stated as follows:

> *There will always be small devices with limited resources in terms of computational power, memory, persistent storage, and energy consumption, that require special assistance from their surrounding infrastructure.*

The consequence of this statement is that the problem of integrating such small devices will always be present and that generic solutions must be found.

2.6 Summary

In this chapter we have created a vision of how users will interact with devices and services in the future. The most important keywords describing this vision are pervasive and ubiquitous computing.

We have introduced the concepts of spontaneous networking which is a special sub-problem of the pervasive and ubiquitous computing paradigm. The Service Location Protocol and Jini have been chosen as primary evaluation targets to study common and distinguishing factors of spontaneous networking technologies, such as, *centralized* or *decentralized design*, the *service description richness*, *performance*, *scalability*, *security*, and *resource consumption*. Additionally, other technologies for spontaneous networking have been briefly presented.

Based on the observation that the support for very small devices in the considered technologies was not given, a characterization of the design space of small devices according to the dimensions of the *computational* and *processing power*, the *ability to communicate*, and the somewhat correlating dimension of *autonomy* of a device w.r.t. its surrounding infrastructure was undertaken. This design space allows to roughly identify three different device categories which are *sensors*, *actuators*, and *information-processing devices*.

Since many devices fall into device categories that require a significant support from their surrounding environment to be spontaneously integratable, a general prediction was formulated saying that there always will be a demand to integrate such small devices into local environments. This issue will be further discussed in Chapter 4 considering the integration of smartcards into local environments.

The small devices' design space has been presented in [KVZ99] together with a possible research agenda towards solutions for the integration of small devices into local environments.

Chapter 3
Smartcards as Personal and Ubiquitous Security Modules

Boss (calling Dogbert's Tech support): "My keyboard is broken.
It only types asterisks for passwords."
Dogbert: "Try changing your password to five asterisks!"
Boss (to himself): "I hope I can remember it."

Dilbert comic, by Scott Adams, 2001 [1]

3.1 Introduction

Although security practices are often comparable to the one illustrated in Scott Adams' comic, the plot that precedes this chapter is definitely something that under no circumstances should be considered as the standard means of security in pervasive computing environments. More concretely the comic demonstrates several typical security problems:

- It illustrates the use of passwords that are considered to be a generally *weak security mechanism* (cf. [And01, Chap. 3]).
- It demonstrates the harmful aspects of *social engineering* in a security-sensitive domain.
- It shows an example of *human (mis-)behaviour* w.r.t. security matters.

In Chapter 1 we have briefly motivated the need for reasonable protection in mobile service usage scenarios of the future. Traditionally, many of the security measures in digital worlds are based on cryptography. Thus, the computation of complex mathematical algorithms in combination with secret and sometimes public keys will be fundamental concepts in the daily life of tomorrow. Although many people think that most of the practically relevant problems have been solved, making things practically usable and applicable, i.e. building an "interface to real life", is of at least equal importance.

The central theme of this thesis is to consider this aspect of computer security and as a consequence we contribute with proposals and results at the level of components, architectures, and protocols to face this problem. Hence, this chapter can be considered as the methodological core of this thesis since the remaining chapters further elaborate ideas and approaches that are introduced here.

[1] Cited with kind permission from Mrs. Heike Schmeling, kipkakomiks GmbH, *http://www.kipkakomiks.de.*

This chapter is organized as follows: Section 3.2 gives an overview of the current state-of-the-art in smartcard technology and considers typical application and trust models.

Section 3.3 considers an important issue which is the usage of smartcards in so-called “hostile” environments by revisiting related work published throughout the last decade. Essentially, the results suggest that currently there is still no practical method known, how users can safely interact with smartcards in such environments.

In Sect. 3.4 five different dimensions of the design space of personal security modules are identified. We consider these to be (a) the *mobility* and (b) the *personalization* of the terminal, (c) the *co-located* or *remote positioning* of the smartcard as another dimension, (d) the *protection of the communication link* between the smartcard and the terminal, and finally (e) the *degree of application control* exercised by smartcard and terminal as the last dimension.

Based on the design space opened up, four design options affecting the design of personal security modules are identified in Sect. 3.5. In particular these are the solutions concerning

(1) the spontaneous networking capabilities of smartcards,

(2) the use of trustworthy terminals in hostile environments,

(3) the idea of a wireless smartcard terminal, and

(4) the concept of a mobile code platform for smartcards.

For each of these approaches we identify the part of the design space to which the particular solution is applicable and compare it to more traditional smartcard usage scenarios. The comparison shows that this thesis considers fundamentally new usage scenarios leading to truely personal security module for smartcards.

Section 3.6 finally summarizes the current chapter and gives a brief overview of the remaining parts of this thesis that provide an in-depth study of the problems, approaches, and solutions for each of the four design options identified in this chapter.

3.2 Smartcard Technology Overview

Smartcards are the outcome of thorough research and engineering during the last three decades. Basically, they can be perceived as small electronic “safes” for digital information implemented as a tiny computer in the shape of a small hardware module.

This section gives a brief overview of the state of the art in current smartcard technology concentrating on the relevant aspects necessary for the remaining parts of this thesis. Further information on these technological artifacts can be found in [RE97; Gut01].

3.2.1 A Brief History of Smartcards

The first plastic card for cash-less payments was issued by Diners Club™ in 1950 and was accepted by major hotels and restaurants at that time only. Later on Visa™ and MasterCard™ also issued cards for payment services and credit cards became the components of a widely accepted payment scheme. The security of these cards was based on the quality of the printing of logos, customers name, and credit card number.

The next generation of identification cards introduced the magnetic stripe to electronically encode more customer’s data in a machine-readable form. Since magnetic stripes can be easily read and overwritten they could not be used to store security-sensitive data and improved maintenance

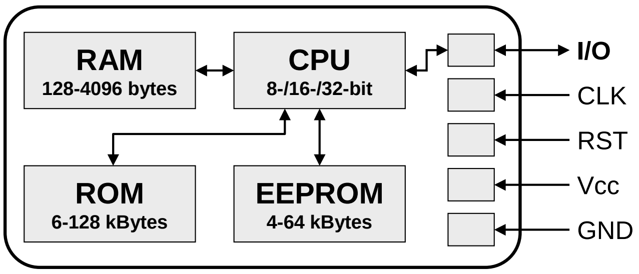

Figure 3.1: Components of a smartcard

and ease of use only. Therefore, most security-critical systems built around magnetic stripe cards use the information on the card for creating an on-line connection to a backend system. Since on-line connections are costly, secure off-line solutions were needed.

In the 1960ies the technological advances of micro electronics allowed for the integration of memory and logic circuits on small silicon discs of a few square millimeters of size. The idea to implant such an integrated circuit into an identification card was described in 1968 by the German inventors Jürgen Dethloff and Helmut Gröttrup and applied for a patent [DG69] in Germany and later in the U.S. [Det78]. A similar patent was applied for in Japan in 1970 by Kunitaka Arimura. Later Roland Moreno applied for several patents in France starting from 1974 and the development and deployment of integrated circuits of the necessary size for reasonable prices was possible.

The first major deployment of cards was undertaken in 1984 in France and later in Germany for a new type of application – public phone cards. In the following years several million deployed cards were in use. Today, an estimated number of 600 million smartcards have been issued in the GSM world for securely authenticating mobile users.

3.2.2 Micro Processor Cards – Essence and Applications

In this section we give a brief overview of current micro processor smartcards.

3.2.2.1 Card Architecture

Smartcards are basically tiny computers. Figure 3.1 illustrates the components a smartcard comprises. In particular these are:

— **CPU**: The most important difference between a memory card and a smartcard is that the latter has a programmable central processing unit. CPUs currently range from 8-bit micro controllers with a few MHz of clock frequency to 32-bit RISC processors working at much higher rates.

— **ROM**: ROM is non-volatile persistent memory that is usually cheap to implement and contains the card operating system. Typical sizes range from 6 kB to 128 kB. Some modern cards use **FlashROM** as a replacement of ROM for development purposes. The contents of FlashROM memory can be updated by special loading protocols, e.g. to test new versions of a card operating system. Memory sizes of FlashROM are usually comparable to ROM.

— **RAM**: Since RAM is rather expensive to manufacture and consumes a rather large portion of silicon, typical sizes of modern smartcards range from 128 bytes to 4 kB. Access to RAM is usually a magnitude faster than access to ROM or EEPROM. Therefore, RAM is used for the runtime control and data stack. Since RAM is volatile storage it cannot be used for storing persistent data.

— **EEPROM**: In contrast to ROM, EEPROM is persistent, but its content can be changed during operation. It is mostly used for persistent but alterable data and dynamically installed applications. Typical sizes range from 4 kB to 64 kB. Instead of EEPROM there have been recently prototyped smartcards based on other memory technologies offering up to 1 MB of space for applications and data. Hence, it is likely that new memory capacities are available in a mid-term.

— **I/O-Block**: The I/O-Block offers serial communication with the outside world (I/O) and contains other lines for external clock (CLK), card reset (RST), power supply (V_{CC}), and ground (GND).[2]

The previous technical specifications cover standard smartcards which come in the form factor of a credit card. Other smartcard-like devices such as the iButton™ [Dal00] offer about 128 kB of battery-powered RAM instead of EEPROM and a digital clock. The iButton comes in a variety of new form factors such as buttons and rings which are necessary to meet the additional space requirement due to the size of the battery. Another recently emerging type of smartcard-like device are Universal Serial Bus (USB) tokens that are equipped with a USB interface to be attached to a PC. They have different external links but most of them are otherwise comparable to smartcards.

3.2.2.2 Smartcard Essentials

Micro processor cards were first used in the French bank card starting in 1984. The possibility to store keys secretly in the card and perform cryptographic algorithms lead to off-line payment schemes with high-level security. Smartcards basically offer a combination of

- a *secure* and *tamper-resistant* storage for, e.g., cryptographic keys,
- the implementation of *unobservable* algorithms (in particular cryptographic algorithms), and
- *mobility* since smartcards can be easily carried around

that make smartcards an ideal device for off-line as well as on-line systems. Put in other words, a smartcard allows to store a secret and perform computations with that secret without revealing it. For good reasons the term *tamper-proof* has been avoided since several successful attacks on smartcards such as *differential power analysis* have been reported in the past, e.g. [AK96; Koc96; AK97]. Design principles to avoid such attacks are, for example, described in [KK99]. An overview of the notions of tamper-proofness and tamper-resistance is given in [Sch00, Chap. 14].

Current smartcards implement a number of cryptographic algorithms (cf. [Sch96; MOV97]) such as DES (ECB and CBC modes) and Triple-DES (3DES) that can be easily implemented for the 8-bit micro controllers. Due to performance reasons RSA is usually available through an additional cryptographic coprocessor. *Elliptic curve Cryptography* is also said to be supported in future smartcards since it is computationally less expensive than other public-key cryptographic algorithms.

[2] The exact positions of these contacts on a plastic card are specified in ISO 7816-2 [ISO88].

Pseudo random number generators are available for computing cryptographic challenges, padding data, etc. Furthermore, many modern cards also implement a message digest algorithm such as MD5 [RFC1321] or SHA-1 [FIPS95].

3.2.2.3 Smartcard Applications

Since secure and persistent storage on one hand, and crypto algorithms on the other hand are the key benefits of smartcards, they are mostly used in security-critical application domains such as, but not restricted to

- identity cards (personal information),
- access control (buildings, rooms, computers, mobile networks, services, etc.),
- secure data storage (medical information, cryptographic keys, etc.),
- electronic signatures, and
- electronic wallet and banking cards (monetary values, loyalty systems, etc.).

Today, smartcards have become small, but pervasive computers used all over the world. Furthermore, the past has shown that the more smartcards are capable of, the more application domains have emerged.

3.2.2.4 Card Operating System

Smartcard operating systems are responsible for the following tasks:

- Data transmission to and from smartcard,
- control of the execution of commands,
- file management, and
- management and execution of cryptographic algorithms.

Basically, smartcards operate as a server in a traditional client/server system as follows:

1. **Request**: A request containing a command to be executed is received by the I/O manager via the serial interface. Error correction due to transmission failures are usually directly handled by the I/O manager.
2. **Processing**: The card interprets and subsequently executes the received command. State transitions may occur during computation. A messaging manager is usually responsible for appropriate de- and encoding of messages. A command interpreter decodes the commands and triggers appropriate actions to perform the interpretation. The return code manager takes the result of the interpreter's computation and generates a corresponding return code.
3. **Response**: After the card has processed the command the return code and computed data are returned to the outside client via the I/O manager.

Smartcard computations only occur synchronously after an appropriate request has been issued to a card. Hence, smartcards are *reactive* devices that are not able to *proactively* initiate external activities on their own.

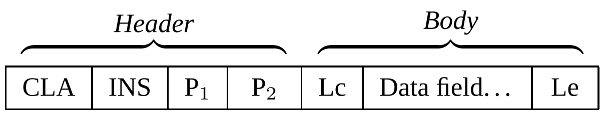

Figure 3.2: Structure of an application protocol data unit (APDU)

3.2.2.5 Smartcard Communication

We briefly introduce the most basic communication principles of ISO-compliant smartcards.

Transmission Protocols T=0 and T=1

Communication to and from the card to the terminal a smartcard is attached to occurs via the serial interface. The electronic signals and transmission protocols (cf. OSI physical layer 1) of smartcards are standardized in ISO 7816-3 [ISO89]. It specifies basic electronic characteristics of integrated circuit cards and power needed for data transfer. Furthermore, it specifies the structure of the answer-to-reset (ATR) and describes the data transmission protocol T=0 (cf. OSI data link layer 2). Amendment 1 [ISO92] of ISO 7816-3 describes the frequently used block transmission protocol T=1.

Application Protocol Data Unit (APDU)

The entire data exchange between smartcard and terminal takes place using so called *application protocol data units* (APDU) as specified in ISO 7816-4 [ISO94]. The structure of an APDU is shown in Fig. 3.2. They consist of a so-called *class byte* (CLA), the *instruction byte* (INS), two *parameter bytes* (P_1, P_2), a *length field* (Lc) of the data sent to the card, the *data field*, and a *length field* (Le) of the data sent back from the card.

The data field of an APDU is often structured according to the *abstract syntax notation* (ASN.1) as defined in [ISO90a] and [ISO90b]. This encoding is based on a triple (*tag*, *length*, *value*) or TLV with *tag* specifying the type of data, *length* the length of the data, and *value* the actual data. With different kinds of "container" tags, arbitrary nested data structures can be described.

Answer to Reset (ATR)

After booting the power supply, the clock and the reset signal, the smartcard sends out an *answer to reset* (ATR) at the I/O pin. This data string, up to 33 bytes long, is always sent and contains various data relevant to the transmission and to the card. An interesting portion of the ATR are the *historical characters* or *historicals*. The historicals are not prescribed by any standard but mostly they contain information about the smartcard, its operating system, version number, etc. Besides the ATR, ISO 7816-4 [ISO94] defines an *ATR file* with the file ID '2F01', that may contain further information about the ATR.

The ATR and the ATR file play an important role for the integration of smartcards into local networks which is the main theme of Chapter 4.

3.2.3 Standard Issuance Models for Smartcards

Most of the systems that make use of smartcards can be categorized as "single-application systems". This means that a dedicated application such as banking, electronic cash (cf. [Gen99]), electronic signature creation, public transportation systems, etc. use a dedicated smartcard that is issued for the sole purpose of implementing this application.

Issuance Players

The players involved in such a system are the following:

— **Card manufacturer:** The manufacturer[3] usually implements the card operating system and performs the packaging of a smartcard which usually includes

- embedding the core integrated circuit into plastic;
- loading the operating system into the ROM;
- loading the EEPROM with appropriate data, e.g. file system, etc.;
- optionally perform some personalization in cooperation with the card issuer such as key installation and management.

The manufacturer is usually not further involved in the actual operation of the application itself.

— **Card issuer:** The issuer is the operator of the system and the application the smartcard is a component of. Typical examples are banks issuing banking and electronic cash cards, mobile operators issuing GSM SIM cards, etc. They usually

- personalize the cards – sometimes in cooperation with the manufacturer,
- issue the cards to the subscribers and clients,
- operate the system and application, e.g. run the mobile network which includes tasks like billing in case of the mobile operator.

The issuer usually has some kind of contract with the card holder that establishes a legal relationship between both.

— **Card holder:** The card holder uses the card obtained from the issuer to participate in the application or system, e.g. inserts the SIM card into the mobile phone to make phone calls with.

Summing up, from the perspective of the card holder *the smartcard essentially is the application since a card is only used for one particular application*. For each application a client obtains a new card form the issuer running the application. The consequences for the holders are well-known: Everybody carries around a significant number of cards and everybody wonders why the applications in use cannot be integrated into only a small number of multi-application cards.

The reasons for this are manifold and (non-)experts in this area give several possible subjective explanations:[4]

[3] Some of the best known names here are Gemplus™, Schlumberger™, Giesecke & Devrient™, Oberthur™, De la Rue™, Orga™, Setec™, and Sagem™.

[4] Further information on the smartcard business sector concerning some of these issues can be found in [Ovum99].

- Multi-application cards are expensive, and issuance can only be done if a critical amount of applications is found to be loaded onto such a card.
- The application and trust management with multi-application cards is not easy to solve, especially the legal issues seem not to be easily solvable.
- Many companies are not interested in being "integrated" into a multi-application card due to branding issues; they are interested that their customers carry a card with the companies' logo.[5]

The trust models in systems using smartcards are basically such that the card holder trusts the issuer to enforce a policy that is somehow part of the mutual contract between them. As an example a user trusts her mobile operator to correctly perform the billing and not charge the user for phone calls that were not made. Since there is no difference between the issuer and the application the situation is quite simple.

In multi-application environments though, the situation becomes different. Since only one entity is able to actually issue the card there must be relationships between an issuer and the other service providers running applications in the card holder's smartcard. We will come back to this problem in Chapter 7.

3.3 Smartcards in Hostile Environments

As has been previously noted smartcards are currently used in conjunction with a particular application only. This further implies that the card's surrounding infrastructure is also reasonably trustworthy. Consider, for example, the use of a smartcard in an ATM. From the bank's perspective the ATM is considered to be reasonably trustworthy and as long as the card is used in the context of an ATM this is not a problem for the bank. Furthermore, the bank usually implements measures that do not allow to make use of their smartcards in other contexts than the ATM.

If smartcards, however, are not bound to a particular application but instead are used at different places in different situations the smartcard's environment must be investigated under security considerations. Several research groups have studied this problem of using smartcards in untrusted environments during the last decade.

Abadi *et al.* [ABKL93] consider the lack of a clock in a smartcard as crucial shortcoming for the creation of electronic signatures. They enumerate that "*... the smart-card can no longer generate secure timestamps for its certificates, and it cannot check that other certificates have not expired.*" Furthermore, they argue that "*... a variety of replay attacks become possible unless that card can obtain the time somehow.*"[6] They present different protocols for authentication and delegation based on ideal smartcards containing a display and a keyboard. Additionally, they present the idea of securely entering the PIN if the smartcard is in control of a secure display by the card showing a one-time password that the users "modifies" into the PIN by entering a series of '+' and 'nextdigit' operations via the (possibly untrusted) keyboard.

[5] Other application areas such as e-government and health-care seem not to suffer from these problems. The European Commission, for example, has started the *eEurope* initiative [Zob01; CEC00] aiming at the "*harmonization of smartcard-based infrastructures across sectors by building a consensus for compatibility*", and "*stimulating inter-sector cooperation to encourage interoperability*". The Commission's initiative seems to pave the way for other attempts driven by governments, e.g. Austria's *Bürgerkarte* [Ott01].

[6] They do not further consider that the lack of a clock also implies that it is not possible to perform useful differential timing measurements needed to detect artificial slow-down in the delivery of messages from a timestamping service.

Yee and Tygar [Yee94; YT95] more precisely sketch the general problem of the private communication path between a secure coprocessor – such as a smartcard – and the user without, however, offering new solutions.

Gobioff *et al.* [GSTY96] further investigate the direct communication problem due to the smartcard's lack of user I/O and propose schemes to substitute secure input (to enter a shared secret into a smartcard) with secure output (to communicate shared secrets from the card to the user) and vice versa. However, their schemes are not applicable for practical use since they require the user to perform manual computations to communicate with the card using a shared secret.

Stabell-Kulø [SK00] considers the problem of smartcards in hostile environments with a special focus on electronic signature creation. He states that the basic problem of smartcards is that there is no secure channel between the smartcard and the user since all messages pass through some (potentially malicious) machinery. Basically, data integrity is the core of the problem that either requires authenticated channels or secret information. The approach considered comprises an additional trusted third party supporting the user and his smartcard that signs a statement that it has verified a signature it received from the user's card. Furthermore, it sends a transformation of the signed text (some kind of message authentication code) to the user who can then manually verify that the card actually signed the data intended. Subsequently the user discloses a secret that is necessary to unwrap the signature and make it available for the recipient. Similar to Gobioff *et al.* his solution requires the user to perform manual computations and is applicable for small texts only with a small number of symbols in the document's alphabet.

Summing up, it seems that until now there does not exist a practical solution to this problem. Therefore, the trustworthiness of the critical path between the user terminal and the smartcard is of vital importance to any security mechanism based on smartcards. Otherwise, exactly this link becomes one of the most vulnerable parts of the whole system that might be the first target for a potential attacker.

3.4 Design Dimensions of Smartcards and Terminals

Apart from the hostile environment problem, smartcards seem to be ideally suited to become personal security modules of the future. They can be used as safe containers for security-sensitive cryptographic key material and can be carried around for the users' convenience.

Trustworthiness of Smartcards

The most fundamental assumptions of the idea of a personal security environment is the trustworthiness of the smartcard. This trustworthiness is based on several issues:

— **Tamper-resistant design:** The tamper-resistance of a smartcard must be guaranteed by the card's design. This includes the non-observability of the memory cells, on-chip bus systems, CPU, etc. Furthermore, the implemented algorithms should be unobservable and the smartcard should be in possession of an unpredictable random number generator implementation.

— **Card production process:** The card production process must meet the tamper-resistance design goals and avoid the introduction of any kind of trap-doors.

— **Card-resident applications:** Besides the hardware and the card's operating system all card-resident applications must be subjected to a careful software engineering process. This process sometimes even requires the (formal) verification of the applications to prove certain security properties.

— **Card issuance:** The card issuance process must guarantee that no manipulation of a card can occur until the card is delivered to the end user. This includes also a proper and secured personalization process – possibly at a point of sale – that must guarantee that only certified applications and data are brought into the card.

— **User diligence:** The user is responsible to appropriately care for the device, e.g. avoiding access by other people, or passing authorization information such as PIN numbers.

The above requirements have to be met by all smartcards that are to be used within the context of a personal security module. However, these requirements should be considered beyond the scope of this thesis since they are mostly enforced today.

Smartcard Personalization

The smartcard acts as a personal security module if one or more of the following criteria are met:

- It contains one or more private keys in a public key crypto-system that are bound to the user's identity.
- It contains one or more shared secret keys in a symmetric crypto-system.
- It contains public key certificates of peers (persons, hosts, services, etc.) that are considered to be trusted.
- It contains other secret information not intended for public use such as electronic tickets, e.g. in the form of authorization certificates or capabilities.

One typical personalization of a smartcard is that the card is a tamper-proof container for its owner's secret key in a public key crypto-system.

Although ideally suited for personalization, it has been observed that smartcards alone are not sufficient from a security perspective since they lack reasonable user interfaces. There is some research and development going on to equip smartcards with small displays, clocks, and keypads but for many application domains this will not be sufficient for at least the following reasons:

- The amount of information to display will not fit on the small displays provided by such "super"-smartcards.
- The display and keyboard mounted most likely will be less tamper-resistant than the integrated circuit itself and thus be subject to attacks.
- Some smartcards such as *plug-in cards* like GSM SIMs are embedded into another device and not directly exposed to a user. Hence, there is no opportunity for equipping such cards with some input and output facilities.

Hence, for practical use, suitable terminals must be used in order to properly "control" and make use of smartcards. A personal security module therefore comprises a *personalized smartcard* and a *terminal*. Furthermore, the overall security of a system consisting of these two components will always have to take the security and trustworthiness of the terminal into account.

In our approach *personalization* is defined as follows:

> *Personalization is a process during which information is brought into a device. This information makes a statement about a device's peer or is actually property of the peer.*

	Dimension	*Domain*
(a)	Mobility of terminal	{*mobile*, *non-mobile*}
(b)	Personalization of terminal	{*personalized*, *non-personalized*}
(c)	Placement terminal/card	{*co-located*, *remote*}
(d)	Communication protection	{*unprotected*, *protected*}
(e)	Application control	*terminal*. . . *card*

Table 3.1: Design options of terminal- and smartcard-based personal security modules

Basically, it depends on the information brought into the device that decides about the relation between the device and its corresponding peer. Think of a user's credit card number that is safely kept in a smartcard and only given to other trustworthy parties such as shops. In this particular example the user is the smartcard's peer.

Since we are interested in how personal and ubiquitous security modules should be designed we separately consider the issues as listed in Tab. 3.1.

Our methodological approach considers these issues as fundamental dimensions spanning the design space of personal security modules based on smartcards. Each of these "axes" is now discussed throughout the following sections.

3.4.1 Mobility and Personalization of Terminal

In contrast to smartcards, terminals are not considered to be tamper-resistant. Whereas the smartcards available today have demonstrated that it is indeed possible to protect a small integrated circuit to a reasonable degree, the protection of a "larger" device with input and output facilities is still beyond today's engineering practice for the mass market. Thus the trustworthiness of the terminal as part of a personal security module is critical for the overall security of such a system and assumed to be given for any terminal used to access a personalized smartcard.

Furthermore, depending on the concrete capabilities, the price of off-the-shelf smartcards with cryptographic support is somewhat around a few Euro to a few tens of Euro. Terminals, however, are still devices that are at least one order of magnitude more expensive than smartcards. Thus, a personal security module's price mostly will depend on the cost of the terminal and not the cost of the smartcard. Our working hypothesis is as follows:

> *The relationship of roughly one order of magnitude between the price of smartcards and terminals will not change significantly in the mid-term future.*

As a consequence, we believe for practical and economic reasons that any approach and proposal for a personal security module must take into account that off-the-shelf devices such as information appliances like personal digital assistants, mobile phones, etc. or public terminals such as Web browsers will be used as terminals.

Basically, we can distinguish between *mobile* and *non-mobile terminals*, both of which will be considered as front-ends to personalized smartcards in the course of this thesis.

Mobile Terminals

People get more and more used to carry around mobile appliances such as mobile phones and PDAs. Thus, it can be safely assumed that many mobile users could potentially use such a device as a terminal for their security module – the smartcard. A further assumption with mobile terminals is that they are somehow owned or even personalized by the user and that they are to some degree trustworthy devices. This requires appropriate user diligence and physical control over the device to avoid tampering.

Estimating the trustworthiness of mobile terminals is generally a hard problem. However, there exist studies investigating the security features of current operating systems for PDAs. For example, Eckert [Eck01] analyses the security features of the PalmOS™, Windows CE™, and EPOC R5™ operating systems and concludes that authentication and access control implemented in these targets of evaluation are poorly implemented or even non-existing.

Mobile phones on the other hand have been until recently based on closed proprietary operating systems. These systems often did not offer any direct access to applications and data available in the device. However, this is also about to change since newer mobile phones offer application platforms based on the Java environment. Thus, it is likely that mobile phones will soon reach a development step that makes them as vulnerable as PDAs or computers are today to different sorts of attacks such as viruses and manipulation.

Non-Mobile Terminals

Another option is to use non-mobile terminals for accessing the card. Users do not carry their own terminal around, but rather use the terminals available at a particular site where they are about to perform a security-critical decision. This is similar to the “hostile environment” problem previously discussed in Sect. 3.3. However, there might be situations in which the user has some confidence in the terminal depending on the importance of the decision to be made. For example, it might well be the case that a user performs a high-volume stock transaction from a terminal at her home but not from a publicly available terminal in a shopping mall.

Depending on the security characteristics of the terminal and the multi-user aspects, non-mobile terminals are probably less often personalized than mobile terminals.

3.4.2 Placement and Communication between Terminal and Smartcard

Communication between the terminal and the smartcard can be subject to protection based on two different approaches:

— **Physical co-location:** In this case the terminal and card are directly attached to each other without any intermediate party. Typical examples are a GSM SIM inserted into a mobile phone or a smartcard inserted into an ATM.

— **Remote location:** In this case the terminal and card communicate over a potentially insecure communication channel. This type of operation is usually not found today in combination with a user terminal. However, a nice example of communicating with a remote smartcard are the mobile operators’ *over-the-air* services that often use cryptographic measures to yield an end-to-end secure communication channel between the operators’ network elements – though not counting as a user terminal – and their SIMs.

In the former case the physical protection of the link must be guaranteed, whereas in the latter case appropriate communication protection, e.g. based on cryptography, should be used.

We present approaches for both options in the course of this thesis.

3.4.3 Application Control

Since the security module of interest comprises two components – terminal and card – both of which are tiny computers, each of them could be the device "running" and "driving" the application. Distinguishing these cases may seem strange since both can be considered as devices acting concurrently and exchanging messages with each other. The standard usage pattern, however, is that the application is hosted in the terminal or even a further component and that the smartcard performs some computation in response to a particular request.

Hence, the card can be considered as being more "active" if the actual messages exchanged result in a communication link that is end-to-end secured between an external party and the smartcard. If the smartcard then uses a particular terminal as its input and output device only, the card can be considered to "conduct" the interaction. Thus, the relationship between the terminal and the smartcard can be characterized depending on the actual placement of functionality, i.e. *card-resident* or *terminal-resident*. For example, the creation of an electronic signature is divided into several sub-tasks:

(a) fetching the document to sign,

(b) displaying the document,

(c) user interaction to accept to sign the document,

(d) computing the hash of the document,

(e) electronically signing the document's hash, and

(f) returning the electronic signature.

In this example tasks (b) and (c) have to be performed by the terminal since it is the only component suited for I/O, and task (e) has for legal issues to be done by the smartcard. Tasks (a), (d), and (f) can be arbitrarily shifted between the terminal and the smartcard. Thus, if these steps are done in the terminal the card appears to be controlled by the terminal, whereas in the other case the terminal is controlled by the card.

Most of the smartcard applications in use today follow the paradigm of a more "passive" card and a more "active" terminal. One of the contributions of this thesis is to investigate how smartcards can be made more active and what the security implications of such an approach are.

3.5 Design Options of Personal Security Modules

This section discusses the different design options that have been identified as central to the problem of architecting personal security modules for mobile users.

In the previous section we have characterized the relevant dimensions of a design space for personal security modules based on the distinction between a smartcard and its terminal(s).

These dimensions have guided our analysis as to which design options are needed to architect personal security modules for mobile users. Essentially, each design option provides a solution to

Smart-card (c)	(d)	Terminal (a) mobile		non-mobile	
	(b)	personalized	non-personalized	personalized	non-personalized
co-located	protected	A2V1	A3V1	T	T
	unprotected	A3V2	A3	T	T
remote	protected	A2	A2V2	A1V1	A1
	unprotected	X	X	X	X

Figure 3.3: Design space of personal security modules according to Tab. 3.1 on page 31: (a) mobility and (b) personalization of terminal, (c) placement of terminal and smartcard, and (d) communication protection between terminal and smartcard.

a general problem in the design of a personal security module. Each of these solutions which have already been briefly introduced in Chapter 1 is now discussed individually based on

- a problem description,
- a solution description,
- an indication to which sub-space of the overall design space the solution is applicable to, and
- which particular design area the prototype matches.

We refer to the overall design space depicted in Fig. 3.3 to identify the area in the design space each solution and its particular prototype can be found.

Design Space Considerations

We do not consider the part of the design space that is characterized as remote and unprotected (see lower row in Fig. 3.3 denoted with ‘X’). The reason is that using smartcards remotely from a terminal without any protection mechanisms is considered unsafe. It is impossible to achieve any benefits in terms of security and therefore this case is not further investigated.

The upper right corner of co-located and non-mobile terminals denoted with ‘T’ covers the more “traditional” type of smartcard usage. Examples of such usage scenarios are a bank’s ATM, a home banking scenario with a PC and an attached smartcard reader, a Eurocheque or cash card terminal at a particular point of sale.

A1: A Spontaneous Networking Framework for Smartcards

Problem: The problem underlying this approach is motivated by the use case that a mobile user is only in possession of a smartcard without any terminal. To be able to communicate with the personal card the user has to make use of terminals available at a particular site. However, the user is faced with a general usage dilemma due to the fact that either the local terminal knows how to talk to the smartcard on the level of smartcard APDUs, or the card is able to offer its services at a reasonably high level to potential applications.

The former approach is in our opinion not very interesting, however, the latter is faced with a particular problem: Since smartcards are very limited devices in terms of computational power and communication bandwidth they need sufficient support from their infrastructure to allow for their spontaneous integration.

Solution: The resulting *JiniCard* framework presented in Chapter 4 comprises an architecture and suitable protocols to bootstrap the smartcard after insertion into a specially equipped so-called *card terminal*.

The solution builds upon the idea of using mobile code as an enabling technology to complement the card-resident resources with off-card resources in a dynamic way. Upon insertion of the smartcard the surrounding infrastructure enables the smartcard to bootstrap mobile objects that represent the card and its services. Thus, the framework essentially defines a spontaneous networking technology ideally suited for smartcards.

Application: Approach A1 is essentially a design option applicable to the mobile and non-mobile terminal variants of the design space. It solves the problem of a remote card providing services on a network. Hence, the user accesses a local and non-mobile terminal through which communication with the smartcard occurs. The terminal and the smartcard do not need to be physically attached to each other. This fact is indicated by cell A1 and its variant A1V1 considering the special case of a personalized terminal.

JiniCard can also be used in a co-located constellation of the user's terminal and the smartcard. However, this region is already covered by the "traditional" type of smartcard usage at non-mobile and co-located terminals such as a bank's ATM, a home banking scenario with a PC and an attached smartcard reader, a Eurocheque or cash card terminal at a particular point of sale. Hence, the figure illustrates that the design options of a personal security module somehow "leave" the traditional area of smartcard usage.

A2: A Personalized and Trustworthy Mobile Terminal

Problem: The general problem solved by this approach is how smartcards can be used in hostile environments where the user does not trust any of the terminals that are locally available at a particular site.

Solution: The *Personal Card Assistant* (PCA) presented in Chapter 5 demonstrates how PDAs can be used as trustworthy mobile terminals to access the security services of a smartcard. Hence, the basic idea is that the users carry and use their own personal trustworthy terminals instead of the locally available ones.

Furthermore, terminal and smartcard are cryptographically linked together using suitable protocols such that no component can be used without the other to perform a security-sensitive operation. Put differently, the PDA is the smartcard's peer and the card is personalized with the PDA's public key and *vice versa*.

Application: The trustworthiness aspect of the PCA is not directly presentable in the design space depicted in Fig. 3.3 on page 34. Besides this shortcoming, the card may be co-located (see A2V1) but the presented solution is designed for the more general case if the smartcard is inserted into a public card acceptance device and the PDA performs remote control of the card. Thus, approach A2 actually solves the communication problem in the presence of a personalized and highly trustworthy terminal.

The related variant A2V2 considers the unpaired version of A2 that does not cryptographically bind the terminal and the smartcard together, i.e. A2 without pairing.

A3: A Mobile Terminal with a Wireless Mobile Communication Link

Problem: The third approach is a solution to the problem how personal security modules can be used virtually anywhere and anytime. Some interaction between the user and a service provider, for example via phone or when interacting with an Internet shop, might not allow for a direct communication link to the security module.

Solution: The *WebSIM* solution (Chapter 6) builds upon the idea of a wireless smartcard reader in the shape of a mobile phone. A GSM phone, for example, can be considered as mostly online and it already contains a smartcard – the *Subscriber Identity Module* (SIM). More concretely, we have made smartcards reachable from the Internet by means of a suitable architecture and the implementation of a tiny Web server in a SIM. This allows a SIM to appear as an ordinary Web server on the Internet. Other Internet nodes can connect to the WebSIM using the HTTP protocol which is transparently forwarded to the SIM of a mobile user. It can among others act as a security server that brings the security infrastructure of the GSM world into the Internet. Hence, the personal security module of a user is accessible from the Internet anytime and anywhere the user is currently located.

Application: The approach is applicable to mobile terminals with a wireless link that do not need to be personalized. Furthermore the solution is particularly useful with co-located smartcards, i.e. smartcard and terminal are directly attached to each other.

Variants of approach A3 can also be identified in Fig. 3.3 on page 34. Approach A3V1 is a WebSIM that has a co-located and additionally protected communication link between terminal and smartcard. Variant A3V2 additionally uses a personalized terminal – which today's mobile phones are already to a certain degree.

A4: A Personalized Smartcard that allows for the Execution of Mobile Code supplied by a Service Provider

Problem: Approach A4 steps outside the boundaries drawn for the design space of Fig. 3.3 on page 34. Essentially, it is orthogonal to the other approaches and can be applied to all of them individually. Therefore, it is not shown in the corresponding figure. A4 is the result of an attempt to correct two drawbacks in the WebSIM design. In particular these are the missing features for end-to-end security, transactional behaviour on the mobile device, and better support for non-repudiation on mobile devices. Approaches A1–A3 are based on the concept of a smartcard that executes code on behalf of its user. However, value-added services can additionally benefit from a smartcard's security context if the smartcard contains an execution platform for mobile code originating from a service provider.

Solution: The *SIMspeak* platform (Chapter 7) demonstrates a platform for the execution of mobile code within a smartcard with a particular focus on GSM SIMs. The approach comprises a runtime execution platform for mobile code, a program verifier, and suitable mechanisms that allow a service provider to “rent” secure storage in a smartcard. It introduces a new trust model since both the user *and* the service provider must have sufficient confidence into correct implementation of the card’s execution platform and runtime environment.

Application: This platform elaborates the idea of shifting as much functionality as possible into a smartcard by providing a platform that allows a card to “conduct” the applications. Basically, this opens up new application domains and leads to improvements in the way how non-repudiation in the form of electronic signatures is achieved. In particular these are

- card-controlled document presentation and hash computation,
- recipient-based signature computation,
- trusted third party assisted electronic signatures,
- signatures with samples, and
- electronic signatures on user interactions.

The relevant architecture and involved protocols of this approach are presented. In particular, this chapter presents some new approaches for the creation of electronic signatures based on this paradigm.

Its most promising application domain is the same as for approach A3 and our particular prototypical implementation is actually built upon the WebSIM approach.

3.6 Summary

In this section we have described how smartcards can be used as the central component of a personal security module consisting of a user terminal and a personalized smartcard. We have begun with a brief overview of the history and current state-of-the-art of smartcard technology covering technical issues as well as deployment and trust models for smartcards.

After identifying the role of terminals in a personal security module we have revisited the pertinent literature about the problem of using smartcards in so-called “hostile environments” leading to the observation that currently there is still no practical method known for users it interact safely with smartcards under such circumstances.

Subsequently, we have identified five different dimensions that span the design space of personal security modules based on a smartcard and a terminal. In particular we consider the mobility and the personalization of the terminal as two dimensions, the co-located or remote positioning of the smartcard as another dimension, the protection of the communication link between the smartcard and the terminal, and finally the degree of application control exercised by smartcard and terminal as the remaining dimensions. On top of these dimensions other factors such as wireless communication capabilities are of further interest.

Based on the design space opened up in this analysis we have identified four design options that affect the design of personal security modules. Each of the problems concerning the spontaneous networking capabilities of smartcards, the use of trustworthy terminals in hostile environments, the idea of a wireless smartcard terminal, and the concept of a mobile code platform for smartcards has been tackled by a particular approach. Each of the approaches leads to a solution of the problem that is applicable in a certain sub-space of the overall design space.

The design space allows to compare the different approaches with each other and the more traditional usage scenarios of smartcards. It shows that this thesis considers fundamentally new approaches towards a truely personal security module for smartcards.

The design options identified in this chapter will be subject to a more detailed discussion throughout the remaining parts of this thesis.

Chapter 4
Integration of Smartcards into Networked Environments

The historical bytes tell the outside world how to use the card. . .

ISO/IEC 7816-4, Section 8

4.1 Introduction

As already described in Sect. 3.2 smartcards are tiny devices that communicate with their environment through a device called a *card reader* or *reader* for short, sometimes also referred to as a *card accepting device* (CAD). Card readers are usually either connected to a PC or workstation or part of a special device such as an *automated teller machine* (ATM).

Smartcard Integration Issues

Since a smartcard needs appropriate terminals to communicate with its user, a number of functional requirements on the integration of smartcards exist:

- The smartcard must be able to *communicate with* and possibly *control* other components that are part of the security module. There must exist mechanisms for the smartcard to control a display and keyboard. Furthermore, the card must be able to *communicate* with other components outside the security module.
- Although smartcards are servers in the traditional sense, mechanisms are needed to enable smartcards to *explore* their environment and act upon the components and services discovered, i.e. being *proactive*.
- The card must be able to describe its card-resident *services* and both *export* and *announce* them to the environment through its communication links.

Comparing the previously listed requirements with the actual capabilities of smartcards deployed in the market, today's smartcards can be considered to be rather inflexible devices. We see three main practical reasons for the relative inflexibility of smartcards.

- Smartcard interaction is standardized on a *per-application* basis. Institutional standards exist for (simple) applications in a variety of domains, e.g. banking, transportation, and mobile telephony (cf. [MP92; GSM11.11]). Standards for more sophisticated applications, such as digital signatures, have just recently begun to show up [PKCS#11]. But standards guarantee interoperability only to a limited degree, as marketing needs require certain variability and proprietary extensions.

- The second reason is that smartcards have very limited resources in terms of memory size, computing power, and communication bandwidth. This limits their range of applicability and makes them extremely dependent on their environments. Without an appropriate card reader and a software package, it is impossible for a user to access a smartcard's functionality.
- Furthermore, such smartcard environments are often highly proprietary, thus further restricting interoperability. Applications in such environments usually work only with specifically configured smartcards and refuse to interact with third party cards.[1]

This analysis mirrors the market structure: Usually, smartcards are distributed in large quantities and applications are implemented by tight interaction between card manufacturers and card issuers. Hence, these partnerships tend to be very strong and long-lasting, which opposes open architectures.

Smartcards in Networked Environments

In networked systems, though, devices and applications are working together on different levels. On the network level, protocols are used to facilitate the exchange of data, e.g. TCP/IP. On top of the networking level, protocols such as HTTP allow for peer-to-peer communication upon which services can be implemented. On the service level a server provides services to other clients and/or users.

Integrating smartcards on the service level requires the description of smartcard services, their announcement in a service-trading environment, and the establishment of links between clients and services. These are generic tasks that are usually facilitated by suitable middleware systems such as the ones presented in Sect. 2.4. Such systems provide frameworks for the description and standardization of services. Lower-level details are hidden in such descriptions, thus standardization can focus on the service descriptions themselves without referring to technical details. Service management is carried out by standard application level services, while communication is performed over standard protocols. This makes access to smartcard services transparent w.r.t. their location and the card's communication features.

Furthermore, smartcards are devices that are temporarily accessible. Their availability usually corresponds to the physical presence of their users. This requires transparent and quick integration of smartcards into the local environment. Additionally, applications must be designed to handle abrupt disconnections smoothly. Therefore, appropriate middleware that offers means to address these requirements is needed.

A Spontaneous Networking Framework for Smartcards

This chapter presents solutions to the above mentioned problems based on a framework for the integration of smartcards into local environments. The framework can be considered as spontaneous networking middleware for smartcards that allows to dynamically integrate smartcards into local environments. Upon insertion into a suitably equipped *card terminal* the card-resident portions of a smartcard's services are complemented by dynamically loaded mobile code running on a network

[1] A prominent example are payment transactions with the German "Geldkarte" (cash card, cf. [Gen99]) that are only possible at particular terminals.

node. It enables smartcards that have traditionally played the role of pure servers to overcome their passive role and enables them to proactively explore their environment. Special attention has been payed to the integration of "legacy" smartcards which is also often a problem in deploying new technologies in the smartcard sector. Thus, it can be considered as a spontaneous networking framework for smartcards representing approach A1 as introduced in Sect. 3.5.

Organization of this Chapter

The rest of this chapter is organized as follows: Section 4.2 presents a generic bootstrapping framework for the integration of small devices into a local infrastructure. This framework is directly inspired by the statement made in Sect. 2.5 that there will be always small devices around that need special assistance from their surrounding infrastructure.

Section 4.3 performs a requirements analysis of how a suitable middleware for smartcards could look like in principle and which advantages and disadvantages each of these general approaches presents. It essentially leads to the observation that the spontaneous networking framework introduced in Sect. 4.2 is a good candidate to build a particular solution on top of.

The *JiniCard* framework presented in Sect. 4.4 describes the developed framework in detail. It essentially consists of two layers: The *smartcard exploration layer* responsible for exploring the smartcard and the *smartcard services exploration layer* that deals with the exploration and instantiation of the services residing on a smartcard. Essentially, it achieves the spontaneous integration of smartcards as envisioned in the introduction of this chapter.

Related work concerning other approaches aiming at a more convenient integration of smartcards into networked environments is presented in Sect. 4.5. Security aspects of the JiniCard framework are considered in Sect. 4.6 where some possible solutions are presented.

The chapter ends with a summary in Sect. 4.7.

4.2 Mobile Code as an Enabling Technology for Spontaneous Networking

In Chapter 2 it was identified that small devices need special assistance from their environment to properly take part in a spontaneous networking scenario. Smartcards as devices belonging to a category that need such special assistance are of our primary concern. However, before we concentrate on smartcards, we take a more generic viewpoint on the integration of small devices and provide a rather generic solution which in turn is then later applied to smartcards in Sect. 4.3.

4.2.1 The Memory Wallet Example

As a running example we consider a fictitious device equipped with a wireless communication link based on transponder technology [Fin99] at reasonable bandwidth that offers persistent storage (EEPROM) for data exchange. This *memory wallet* constantly tries to spontaneously connect to its surrounding environment to offer its service as soon as it is within the magneto-electrical field of a corresponding reader. The device offers the following operations:

— list():	Lists the names of all items stored on the device.
— store(*name*, *item*):	Store an item in the wallet's memory.
— erase(*name*):	Erases the named item from the memory.

— getRand(): Fetch the current random value from the wallet.

Since the device is of limited computational resources it offers only basic security features, i.e. in our case it is only able to compute a cryptographic hash of a set of data. Furthermore it owns a random number generator that can be used to generate fresh random numbers which can be read with the getRand command. Random values are used to prevent replay attacks and guarantee the "freshness" of the operations performed on the device. Finally, it is in possession of a secret user-definable password. Together these can be used to provide *integrity* and *authenticity* of the operations performed on the wallet in the following manner:

- Before an operation can be performed a new fresh random has to be obtained from the wallet using getRand.
- The operations list, store, and erase have to be electronically signed using the cryptographic hash $s = h(c, d, n, p)$, with c being the command, d the command data, n the fresh random number previously obtained, p the shared secret password between the user and the wallet, and h a suitable one-way hash function.
- Upon receipt of a command, the wallet can verify the signature using the secret p and the current random value n and grant or reject access. After a command has been finished, a new random number is computed.

This offers simple though sufficient protection in most cases making the wallet a quite useful device if it existed.

Obviously, the wallet needs sufficient communication bandwidth for exchanging data, but in terms of computational resources such as RAM and CPU speed it can be considered as a relatively small device. Thus, it does not need a working IP stack for operation. However, the applications that make use of the wallet might be located somewhere in a desktop environment where a user simply wants to "mount" the wallet allowing to manipulate the wallet's content.

4.2.2 A Bootstrapping Framework for Small Devices

We now describe how our framework for the integration of small devices into a local infrastructure could look like in general by referring to the memory wallet example presented previously. The framework considers and distinguishes between

(1) device detection,

(2) detection event compilation,

(3) device explorer lookup,

(4) device exploration,

(5) service proxy lookup,

(6) service proxy instantiation, and

(7) service integration.

An overview of the involved components and their interactions is given in Fig. 4.1 on the following page. In the sequel we describe the different parts of the framework.

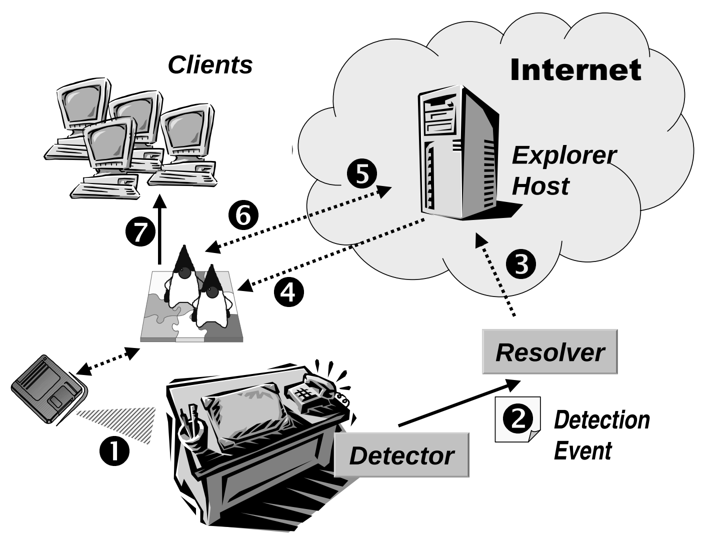

Figure 4.1: Overview of the framework for the integration of small devices

4.2.2.1 The Initial Detection Event

After a small device such as the wallet has entered an environment it must be “detected”. In the wallet example the radio frequency (RF) reader is the only device in the infrastructure that is able to notice the *detected device,* i.e. the wallet, after it is within the reader’s range. Depending on the lower-level communication protocols, certain information about the device can be gathered. Very often a device is able to disclose information such as manufacturer, type identification, version number, serial number, date of production, etc.

Upon detection the detector compiles a so-called *initial detection event* which should consist of at least the following information:

- The detected device information as described above.
- Information about the detector itself: manufacturer, model, etc.
- A link identifier that can be used to address the detected device relative to the detector which might be needed for further communication with the detected device.

The detector sends this event to a so-called *resolver* which is responsible for finding a *device-specific explorer*.

4.2.2.2 Device Exploration

At this stage the type of the detected device is known and communication at the link layer is possible. The next step is to somehow communicate with the device at a higher, more device-specific level. The general thesis is that the missing knowledge about the protocols supported by the device can be obtained by some device-specific interrogation. However, since the environment does not know more about the device than is available from the initial detection event this interrogation must either

Remote device explorer	*Local device explorer*
• Requires active network component to perform exploration (could be potential bottleneck, if may devices are explored).	• Only requires passive components to perform explorer delivery (e.g. Web server).
• Requires remote exploration protocol standardization.	• Requires standardization of mobile code platform(s) for explorer execution.
• Discloses privacy information, i.e. location and details of device usage.	• Only local communication needed, i.e. less disclosure of usage information.

Table 4.1: Comparison of remote and local device explorers

be delegated to another more knowledgeable component or the environment must be appropriately extended to include this knowledge:

— **Delegation:** This approach requires the resolver to find a suitable resource in the net that is capable of communicating with the device. Such a *remote device explorer* is searched for using the information from the initial event and the resolver could play the role of a *relay* that provides for a direct link between the remote explorer and the device until the explorer provides further information about the device.

— **Environment extension:** Another option is to find a so-called *device explorer* in the form of mobile code that can be downloaded into the local environment. Upon instantiation of such a device explorer "agent" it can locally talk to the device and perform the actual exploration.

In the wallet example it might well be the case that the information provided by the wallet does include the type of device but not its version number. Thus, the explorer's task is to figure out the wallet's version number and subsequently the protocol it supports. This information might be accessible through a command only known by the explorer, but which is not known by the detector. Thus, the explorer is responsible for obtaining as much information from the device as possible by exploiting the device-specific knowledge it already possesses. Obviously, each of the two approaches has different advantages and disadvantages as briefly summarized in Table 4.1.

4.2.2.3 Explorer Lookup

The most crucial problem with the explorer approach is how to find a suitable explorer? In either case – the remote explorer or the mobile code explorer approach – suitable network resources must be found to answer a resolver's query appropriately. The following two approaches could be generally thought of:

— **Search engine:** The most obvious way of implementing the search for the correct network resource is the use of a suitable search engine. In contrast to the Web search engines used today a more machine-friendly version should be used. In particular this means that all queries are based on a standardized query form, e.g. in the form of a well-known URL such as

```
https://query.iana.org/q?serno=0123456789abcdef
                              &manufacturer=Wallet Corp
                              &version=1.22
```

In this example the search engine is hosted by some international authority such as the International Assigned Numbers Authority (IANA) [IAN01]. All device manufacturers have to register their device information that is part of an initial event and supply a corresponding explorer that is returned as the result of the query submitted to IANA.[2]

— **Domain Name Service (DNS) extension:** Another option is to use the pervasively available Domain Name Service (DNS) [RFC1034]. Among others, DNS provides for a distributed database that maps Internet hostnames to IP numbers. However, this can be easily extented to provide a mapping from serial numbers of devices to hosts. For example, most Ethernet network cards have a built-in 48 bit MAC address that is unique for each device. Manufacturers have their own "address space" of numbers which can be easily exploited to provide a DNS mapping as follows:

- A a new top-level domain at ARPA such as '`.dev-serno.arpa`' is defined.
- The next level domain indicates the type of serial number, e.g. an IEEE 802 Ethernet card number as '`.ieee802.dev-serno.arpa`'.
- The full MAC address is then put in front of the serial number domain. For example, if the Ethernet MAC address is `00.04.76.40.F0.84`, then it could be represented as the DNS name '`84.F0.40.76.04.00.ieee802.dev-serno.arpa`', i.e. using reverse order to comply with DNS standards.
- If a resolver queries DNS for this name the Ethernet card manufacturer could be responsible for serving address space of, e.g. '`.76.04.00.ieee802.dev-serno.arpa`'. Thus the implementation of the DNS server for the above domain could use the additional data to perform a precise lookup of the "best" explorer for the given device and return an IP number of the hosts that hosts this explorer.

Obviously, the DNS approach might not be sufficient to find the best explorer since it makes a decision only on the serial number and not on additional information available in the initial event. Likely, the DNS approach can be further extended to provide for a more precise search, but it might be even more useful for running a hybrid approach in conjuction with the search engine. The DNS extension could be used to find more specific search engines for a detected device based on its serial number which is likely to scale much better than a central search engine. Furthermore, it should be obvious that the information queried must be treated as confidential and the answers obtained must be authentic.[3]

Summing up, from a purely technical point of view there are easy approaches how to set up a central or distributed directory for mappings from the information available in the initial event to some Internet resource. However, it is clear that the provision of a global infrastructure for this kind of mapping tasks requires enormous effort and involves many different players ranging from vendors to standardization organizations. Hence, a simple, practical, and inexpensive solution is yet still to be found.

[2] Obviously, a XML-based encoding of the data could provide even better results, if the initial event data are highly structured.

[3] In particular this could mean that a secure version of DNS [RFC2535] is used for the DNS infrastructure.

4.2.2.4 Service Integration

After the explorer has – remotely or locally – explored the device, more information is available that should allow for the integration of the device into the local infrastructure. In contrast to the remote device exploration a “remote device usage”, i.e. the usage of a device “through” another remote authority is prohibitive for reasons of responsiveness, scalability, and privacy. Thus, the only real option is to use the mobile code approach to somehow “install” a local driver for the device into the infrastructure that complements the device-resident portion of the service implementation.

Technically, this means that the device explorer comes up with more information about the device of which the most important ones are a list of the services that reside on the device and the corresponding mobile code objects – *service proxies* – that represent each service. In case of the wallet there would be only one service object that implements the wallet’s interface at the service level.

Since each service proxy is implemented in a particular piece of mobile code there is a need to standardize a platform in which this proxy can live. Basically, any kind of mobile code platform that offers enough flexibility can be used for that purpose. A very promising candidate for this purpose might be Java [GJS96] since in its Java 2 Microedition variant it offers different so-called *configurations* and *profiles* that define building blocks for different device categories that can be easily used to describe the properties of a particular platform available in the infrastructure. Furthermore, Java allows for fine-grained security policies monitoring the execution of mobile code that can be used to protect the infrastructure from malicious service proxies.

Based on the platform the service proxies can now use any technology for spontaneous networking to advertise and offer the services that live on the device. Thus, the wallet proxy could use SLP or Jini or one of the other technologies presented in Sect. 2.4 to advertise the wallet service to potential clients. Obviously, the device explorer should be given information about the spontaneous networking technologies supported by the local environment to come up with a reasonable service proxy. Hence, the service proxy integrates with the surrounding infrastructure and service usage can begin.

Summing up, the bootstrapping framework presented in the previous sections gives a more structured view of the overall integration problem. It identifies components, structures, and interactions that together represent a solution to the underlying problem domain at a reasonable level of abstraction. As such it forms the basis for solving the integration problem for another kind of small devices: Smartcards.

4.3 Middleware for Smartcards

Until recently, smartcards and their applications were tightly coupled, resulting in the card being useful for one application only. But the idea of viewing a smartcard as a mere platform is widespread. Increasingly, applications show up that try to use smartcards already in the field for new applications, e.g. electronic ticketing combined with cash cards [Blu00a]. This is possible if the specifications allow for third party access which, however, is usually hampered by strict regulations of the card issuers.

The most through step in this direction is the emergence of execution environments for smartcards, such as Java Card [Sun00a; Sun00b; Che00; HNSS99], and Windows for Smartcards [Mic00]. This increases extensibility and flexibility and opens up the smartcard market to independent application providers. As smartcards are becoming increasingly open, their role as a software platform

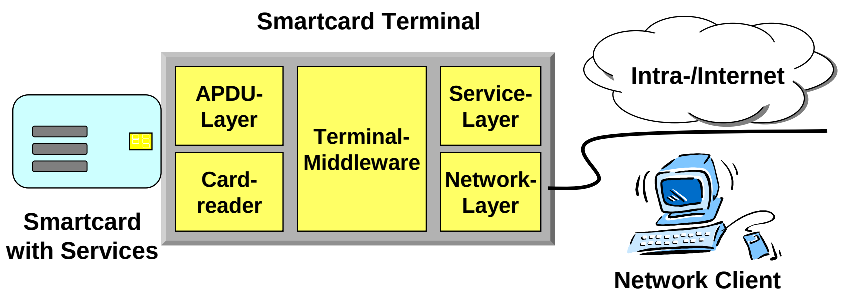

Figure 4.2: Architecture of a generic smartcard terminal

gains importance and the smartcard paradigm changes, since the separation of cards and applications becomes possible. A card issuer is then able to "buy" independent smartcard applications that are loaded onto the issuer's cards. A market for smartcard applications becomes potentially possible since the shift towards the platform paradigm is accompanied by a simplification of software development. This is achieved by bringing high-level, standard programming languages to smartcards, e.g. Java [GJS96] or Visual Basic [Mic01], opening up smartcard programming to a new class of developers. Similarly, access to smart cards from applications is unified by architectures like the OpenCard Framework [OCF99] and PC/SC [PCS00], that integrate the card reader infrastructure into operating systems and programming languages. However, integrating smartcards into networks by means of suitable *middleware* is a challenge that has not yet been discussed in the past.

The goal is to design middleware architectures and systems that facilitate smartcard integration into service federations as much as possible. In the sequel we discuss general design issues of middleware systems leading to our proposed architecture for a smartcard middleware. We start by defining the concept of a *smartcard terminal*, a component that offers network connectivity for smartcards. We continue by comparing different design paradigms for the middleware implemented in such a terminal using the requirements listed in Section 4.1. In Sect. 4.2 a generic framework for the integration of small devices into local environments based on the mobile code paradigm was presented. Based on the results of this framework this section discusses a particular instance of this generic framework which integrates smartcards into local networked environments.

4.3.1 Smartcard Terminal

Services implemented in the smartcard must be able to offer their interfaces to the network the smartcard terminal is attached to. Figure 4.2 illustrates the role of the terminal in a smartcard middleware architecture. A smartcard terminal could be partitioned into the following components:

— **Card reader and APDU layer:** The card reader component provides access to the smartcard based on standardized protocols such as ISO 7816 [ISO89; ISO94]. Essentially, it handles communication between the smartcard and the terminal by exchanging APDUs.

— **Network layer:** This layer provides basic terminal connectivity to the network. In case of IP this layer would implement an IP stack.

— **Service layer:** This layer presents smartcard services to the network in any suitable form. A number of technologies such as CORBA [OMG00], Java/Jini [Sun99a; Wal99], or DCOM [Mic96] might be used to make the smartcard services accessible from arbitrary network clients. The actual technology chosen for representing smartcard services should be independent of the scope of the concrete middleware component. Hence, the component should be able to support any of those technologies.

— **Terminal middleware:** The terminal middleware has to perform a number of tasks which are comparable to the bootstrapping framework for small devices introduced in Sect. 4.2.2:

- It explores the services and applications on a smartcard as it gets inserted into the card reader.
- Based on the service information found it informs the service layer about the interface of the smartcard applications exported to potential clients.
- It acts as a gateway for incoming requests from network clients that access the smartcard services via the service layer and forward requests to the APDU layer back and forth.

As such the terminal middleware represents the "glue" between the externally offered network services of the card and the communication layer connected with the smartcard.

The card terminal seems to be an ideal candidate for implementing at least some of the middleware needed.

4.3.2 Characterization of Middleware Approaches

Middleware can be designed in various ways that can be characterized according to the kind of "agreement" between the involved components. In the sequel we briefly compare pure *protocol-based* approaches with more *platform-based* approaches. This differentiation is useful for defining criteria under which different approaches can be valuated.

Protocol Standardization

Protocols define the structure of communication according to various rules the communication partners must follow in order to successfully communicate with each other. Protocol standardization means that a standardization body is formed to bring companies and organizations together that are interested in the goal of standardizing the communication between such components. Typical examples are the protocols specified in ISO 7816 [ISO89; ISO94].

Unfortunately, standards in the domain of smartcards have shown to become mature only after a considerable amount of time, often several years, since the development and deployment is a rather complicated process compared to a software update in the desktop computer world. Compared to the growth of the Internet and the protocols developed and used there, the smartcard world basically has not evolved since the ISO protocols from the early nineties.[4]

Coming back to our problem domain this would mean to standardize the communication protocols between the smartcard and all its surrounding components. Furthermore, all communication between the environment and the smartcard itself must be standardized. Since software updates for smartcards are notoriously problematic and security-sensitive it is not easy to adapt already deployed cards to new communication facilities and environments.

[4] Consider for example that ISO 7816-4 was standardized in 1994, just before the Internet took off.

Platform Standardization

In contrast to pure protocol standardization, platforms are intended to serve as runtime environments that can be used by applications to perform computations. A typical platform is for example the Java virtual machine and the runtime environment comprises the available packages, classes, and native method implementations (cf. [GJS96]).

Bringing the platform idea to our problem domain, the basic idea is to move any higher-level protocol engines that would deal with the communication between the card and its surrounding components from the card into the outer platform. Within the platform mobile code is executed that implements the necessary adapters and proxies between the smartcard and the various other components. The clear advantage is that all "technology"-dependent pieces such as communication protocol engines are implemented as mobile code which is much easier to adapt to new middleware technologies (e.g. Jini). Hence, changes in the middleware are likely not to affect the operating system and applications within the smartcard but can be reflected in the mobile code running in the platform.

4.3.3 Design Choices for Smartcard Middleware

Various implementation strategies can be envisioned for the smartcard terminal as outlined above. We describe some possible approaches and compare their strengths and weaknesses w.r.t. the criteria

- simplicity,
- flexibility, and
- standardization effort.

These criteria are discussed from the perspective of service and application developers on the one hand and middleware implementors on the other hand.

Middleware as an APDU Gateway

This approach can be described as a simple gateway for APDU-requests to the smartcard. Clients send packets to the service-layer of the smartcard terminal containing APDUs that are routed via the APDU-layer to the card reader. Hence, there is no real abstraction above APDUs, and the middleware would be responsible only for multiplexing communication between arbitrary clients and smartcard services.

The interfaces at the service layer would therefore offer methods such as *sendAPDU*, *enterMutex* and *leaveMutex* (needed for locking access to the card for a certain period of time), etc. From the perspective of the middleware implementor this is a rather simple middleware, easy to implement and flexible, since it offloads all the complexity on the service developer. Services operate at the same level of abstraction as before, but with the intricacies of distributed application programming such as partial failures.

Middleware as Request Broker

With this approach the middleware first explores the services available on the card. This requires an enormous standardization effort since apart from detecting the correct type of card, there must be a standardized way to perform this exploration. This could be achieved by the definition of new class and instruction bytes in the line of ISO 7816 that return descriptions of the services available on the card. Usually, service descriptions consist of interface descriptions, additional information and annotation blocks, and addressing information, e.g. application identifiers, needed to address the service from a smartcard client. This information could come in a variety of formats ranging from binary encoded descriptions to IDL- or XML-based documents.

The middleware could implement a generic server that is capable of processing incoming requests from clients and transforming them into appropriate sequences of APDUs. As an example one could imagine a CORBA IDL description [OMG00, Chap. 3] that describes a smartcard service that can be used to automatically generate server skeleton code, bind a CORBA object with an *object request broker* running in the smartcard terminal and register the object with a CORBA naming service. In addition to a pure interface description the mapping of method invocations to sequences of APDUs sent to a smartcard needs to be defined.

The request broker middleware operates at a much higher level of abstraction than the APDU gateway. For clients, the smartcard services appear as objects or services in a distributed system such as CORBA, Java/RMI, etc. Service implementors only need to provide an interface definition and appropriate APDU-mappings to integrate legacy applications into the sketched middleware. However, a major drawback from the perspective of the middleware implementor is that numerous standardization steps have to be taken first: exploration of service descriptions, format of descriptions, mapping to distributed object system of choice, service publication, to name a few. This approach though being promising in general suffers from the amount of standardization steps necessary for real-world deployment.

Summing up, both the APDU gateway and the request broker approaches are instances of the protocol standardization approach introduced in Section 4.3.2 which has already been identified as rather inflexible and we concentrate further on the platform paradigm.

4.3.4 Middleware as an Execution Platform for Mobile Code

The middleware architecture presented in this subsection tries to circumvent most of the drawbacks of the previous approaches by completely reconsidering the underlying middleware paradigm. The middleware is not only “glue” code such as the APDU gateway and broker between components but a platform for the execution of dynamically downloaded mobile code. This can be illustrated with the following scenario:

- The smartcard gets inserted into the terminal and the *answer-to-reset* (ATR) identification string is read.
- The ATR is used to fetch a component that acts as a card manager from a well-known set of Web sites hosting such proxies. These proxies are implemented in a mobile code programming language such as Java. The smartcard terminal provides an execution platform such as a Java virtual machine (JVM). The service proxy consist of an appropriate Java archive (JAR) file that is downloaded to the terminal and executed in its JVM. In the basic scenario this card manager itself could now register as a service representing the card to the network environment.
- In a more advanced scenario the card manager explores the contents of the smartcard in search for smartcard services. This is possible if we assume the implementation of the card manager knows about the particular kind of card that triggered its activation. Hence, it knows how to actually explore the card and find its available services. Each service found may consist of a URL pointing to a service manager that in turn can be fetched and instantiated in the execution platform and offer its particular service to the environment.

This approach essentially defines

- an execution platform for mobile code,
- a well-defined process to fetch a card manager from the network, and
- some API or protocol for the manager to access the smartcard and the network.

Compared to broker-based middleware much less standardization is needed, though the overall flexibility has even increased, since the card and service manager are active components that not only act as services but can also proactively be clients to other services. The most significant drawback with this approach is the fact that the complexity is mostly shifted to the implementors of card and service managers and the proper definition of an execution platform.

4.4 The JiniCard Framework

We have found the idea of using an execution platform for the integration of smartcards into networked environments sufficiently appealing to investigate and prototype such a system. This section presents the result of the chosen approach, the *JiniCard* framework.[5]

As previously outlined, smartcards are temporary devices. Consequently, the availability of the services that they offer is short-term and volatile in nature. Smartcards, and hence their services, can appear and disappear without prior notice, i.e. spontaneously. Smartcards are physically portable and can easily be carried into unknown environments[6]. Yet smart cards are utterly dependent on their environment to be useful, as they generally lack any input or output facilities for their users. On all dimensions, smartcards rank at the lower end, which means that they are very dependent on proper support from their environments' infrastructure. These usage characteristics call for a seamless integration into different environments that do not require any setup or configuration. Service discovery and integration must take place spontaneously.

The architecture is named *JiniCard* to emphasize the fact that it makes card services available as Jini services, independent of the type of smartcard used. It was a key design objective to support a wide variety of smartcards by imposing only a minimal set of requirements on the smartcard's side. Basically, the only requirement is that the card adheres to the ISO 7816 standard, i.e. that it communicates by exchanging APDUs, as the vast majority of smartcards does.

Impromptu Service Integration

One of the main issues that we encountered was how to deal with smartcards that are completely unknown to an environment, given the extremely limited amount of information that can be extracted from a card of which one might only know that it adheres to ISO 7816. A related issue was how to dynamically instantiate card services that are not yet present in the environment at the time of card insertion.

Smartcard users are not interested in physical smart cards themselves, but in the services they provide. Therefore, the main goal was to make these services available without much effort on the user's side. Ideally, card services should become part of the infrastructure as soon as the card that carries them is inserted into a card terminal. This should be possible even if there is no *a priori*

[5] In the sequel the actual Java interface specifications have been omitted for reasons of simplicity. For the interested reader more detailed versions are available in [Roh00; KRV00b].

[6] Examples are public and semi-public places like offices, meeting-rooms, banks, post offices, and shops, in which smartcards act as user agents.

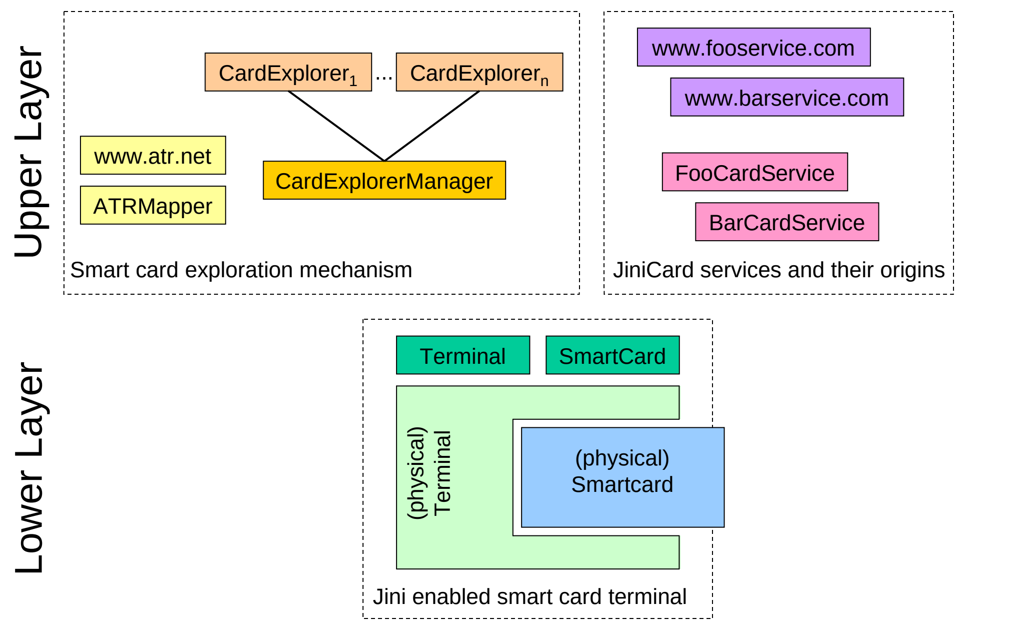

Figure 4.3: Components of the JiniCard framework

knowledge of the services that are contained on a particular smart card. Another desirable feature, especially if one takes on a more net-centric perspective, is to have these services available not only locally, but as part of a local or wide-area network. Therefore, the goal can be described as making instances of smartcard services immediately available in a network environment, as a result of inserting a card into a card reader.

The Card Terminal as a Network Component

We think that the design of current card readers and their device drivers is unsatisfactory to meet these goals. They are usually not self-contained, but attached to a general-purpose PC to function. We propose to view a card terminal as a self-contained entity that provides access to smartcards from a whole network infrastructure. The ultimate vision is to build the JiniCard terminal as a physical device that contains a Java VM, can be plugged into a network, and does not need any additional hardware. To make the card terminal available as a network-wide resource, we decided to model it as a Jini service. This has the following benefits:

- The terminal is modeled as a Java interface which means that low level technical details of the implementation of the terminal are abstracted from and are no longer important.
- The terminal is seamlessly integrated into an infrastructure and can be used by any client, without any knowledge of the concrete underlying terminal technology.
- The client may be located anywhere in the environment.

The JiniCard framework consists of three categories of components that can conceptually be divided into two layers. The *lower layer* provides the abstraction of a card terminal as a Jini service and serves as a common base for the other components of the framework. The *upper layer* consists of a mechanism to explore smartcards to identify services that are contained on them. The actual *card services* can also be seen as part of this layer. Card services get instantiated as the result of an exploration process. Figure 4.3 on the preceding page gives a simplified layout of the architecture.

4.4.1 The JiniCard Terminal Layer

Card services are meant to be downloaded into many different settings. This requires a well-defined environment, consisting of well-known interfaces, into which these services can be embedded. One way to provide this foundation is by modeling a card terminal as a network component that provides a standard means of remote access to a smartcard.

Accessing Smartcards Remotely

The purpose of the lower layer of the JiniCard framework is to provide a uniform and simple way to access smartcards remotely. With regard to uniform access, motivations similar to those that led to the development of the Opencard Framework (OCF) [OCF99; OCF00] apply here. OCF is a Java-based framework that provides a uniform application interface for building smartcard applications. A major difference to OCF is that the JiniCard terminal is designed to be used remotely and is not restricted to be used by a single Java VM. This means that remote mutual exclusion of access to a smartcard has to be considered.

We have modeled the JiniCard terminal as an ordinary Jini service with the Java interface Terminal (see 4.3 on the page before). It becomes part of the local Jini federation by finding lookup services and performing the service registration process also known as discovery and join [Sun99b].

The SmartCard interface[7] provides a uniform and easy to use abstraction for all kinds of smartcards, but it does not change the basic principles of interaction with a smartcard. The APDU as the low level protocol unit is visible in the interface. A step in the protocol still consists in the exchange of a pair of APDUs – a command APDU followed by a response APDU. This makes the interface very flexible and does not constrain its applicability to certain kinds of smartcards.

Mutually Exclusive Access to a Smartcard

Multiple clients of a single JiniCard terminal can hold a reference to the current smartcard simultaneously. Interactions with a smartcard often require the atomic exchange of multiple APDU pairs, e.g. to navigate through a file system hierarchy. During this process state transitions may occur in the card. This means that APDUs are not independent of one another, but depend on earlier APDUs. It is not possible to provide transparent scheduling of access to a smartcard, because it is unknown what state was established by one card client, and how to reestablish that state, after another client has been using the card in between. This fact, and the fact that multiple clients can hold references to the same smartcard, requires some kind of mutual exclusion mechanism that is exposed in the interface. This is achieved through the methods beginMutex and endMutex available in the SmartCard interface. They provide mutual exclusion between distributed clients of a smartcard. A potential drawback is that a client can effectively block a smartcard if it does not relinquish control of the smartcard once it has acquired exclusive access to it.

[7] Subsequently, we will denote Java class names with the following typographic face: JavaClassNameExample.

Possible reactions to this problem are (1) to ignore it, (2) to use a fixed maximum amount of time that a client is allowed to access a smartcard, (3) to let the client specify in advance (on calling beginMutex) how long it needs the card, and (4) to use a fixed maximum inactivity time after which the card is revoked from the client. However, none of these approaches is without disadvantages as it represents a general distributed resource allocation problem.

A client of the smartcard interface should access a smartcard exclusively only during a single atomic sequence of APDU pairs. Exclusive access should be held as shortly as possible, to give other clients a chance to obtain access to the card. The actual means to talk to the card is to send command APDUs and to receive response APDUs. JiniCard is fully transparent in this respect. A service implementor can be sure that JiniCard will not change the content of the exchange of APDU messages. This has the advantage that JiniCard works with all ISO/IEC 7816 compliant cards that rely on exchanging APDUs to communicate.

Answer-to-Reset

Immediately after reset, smartcards issue a short sequence of bytes, called the answer-to-reset (ATR) as introduced in Section 3.2. It contains information about low level communication protocol parameters. Furthermore, it contains up to fifteen so called *historical characters* that are used in different ways by different vendors. ISO 7816-3 [ISO89] basically states that

> "[...] *the historical characters designate general information, for example, the card manufacturer, the chip inserted in the card, the masked ROM in the chip, the state of the life of the card.*",

and furthermore

> "*The specification of the historical characters falls outside the scope of this part of ISO/IEC 7816.*",

and finally in ISO 7816-4, Sect. 8 [ISO94]

> "*The historical bytes tell the outside world how to use the card.*"

Although these quotes seem to be contradictory at first, in practice the ATRs reported by off-the-shelf smartcards reveal sufficient information about the manufacturer, the card type, and operating system such that the ATR can be used as a key to obtain further information about the card.

In terms of implementation, the ATRs of a card are obtained by invoking the getATRs method. It returns an array of ATRs to reflect the fact that some smartcards have several different ATRs. By consecutively resetting a card, it is possible to cycle through the set of ATRs of such cards.

Service Trading on Top of Jini

A JiniCard terminal service together with the SmartCard it manages provides an effective abstraction of the underlying card reader technology. It makes the card terminal and an inserted smartcard part of the network infrastructure. By modeling the terminal and smartcard as Java interfaces they become easy to use. Clients just need to know the Terminal and SmartCard interfaces and how to look up a card terminal in a Jini environment. Details related to remote communication are hidden by Jini and RMI. Details concerning the interaction with the physical terminal are hidden by JiniCard. Mutual exclusion allows multiple applications at different locations to act as clients of a single smartcard in an ordered manner. Keeping the exchange of APDUs as the basic means of communication retains the flexibility that is needed to use a wide variety of different smartcards.

As such, the lower layer of JiniCard is an instance of the APDU-gateway middleware described in Section 4.3.3 and provides an API for the upper layer of the JiniCard framework to access the smartcard.

4.4.2 Smartcard Exploration Layer

The components described previously provide a uniform way to access smartcards as network components. But they are not sufficient to achieve our goal to integrate effortlessly the services that a smartcard offers into a networked environment. To reach this goal, we propose an exploration mechanism to identify the services that are contained on a smartcard and to make them available in the environment. Our approach to achieve the goal of card service integration includes the dynamic download of exploration components as well as card-external parts of card services. As such the smartcard exploration represents an instance of the generic bootstrapping framework for small devices as introduced in Sect. 4.2.2.

As our target environment we have again chosen Jini that serves as a platform that represents all system entities as services. Similarly, we represent all applications contained on a smartcard as Jini services. This places services that are offered by smartcards on an equal footing with other Jini services. In the following sections, we describe the steps that the card exploration mechanism takes.

Smartcard Insertion

The exploration process is triggered by the insertion of a smartcard into a JiniCard terminal. This causes the terminal to distribute a remote event (cf. initial detection event in Sect. 4.2.2) to all listeners (cf. resolvers) that previously registered for such events (see Step 1 in Fig. 4.4 on the following page). The event contains the ATRs of the card to allow listeners to decide early on, if they are interested in the event and wish to respond to it. The set of ATRs is the only information that can be obtained from a card if there is no a priori knowledge about it.

Card Exploration with the Card Explorer Manager

The component that controls the card exploration process is known as the CardExplorerManager. This component is registered at the card terminal as an event listener. The card explorer manager manages a set of CardExplorers. Card explorers carry out the actual work of exploring a certain kind of smartcards to identify the services contained on them. Card explorers are dynamically loaded into the Java virtual machine of the card explorer manager, if an unknown kind of smartcard is encountered. As explained before, the only information that is available after a card is inserted are its ATRs. The card explorer manager passes this information together with a reference to the smartcard to its card explorers and asks them to explore the card (Step 2). The result of this exploration process is an instance of class ExplorationResult (Step 3), that contains a set of ServiceInfo objects or an indication that the card explorer could not handle the card. A ServiceInfo object describes a single service and provides enough information to engage in the service instantiation process.

The Role of Manifest Files

What happens if the card explorer manager did not find a card explorer in the set of known card explorers that could handle the card? In this case the card explorer manager contacts a special, well-known Web server. For the following assume that this server is named *www.atr.net*[8] and hosts card

[8] *www.atr.net* is just used for illustrative purposes here, so don't worry if it doesn't actually contain card explorers.

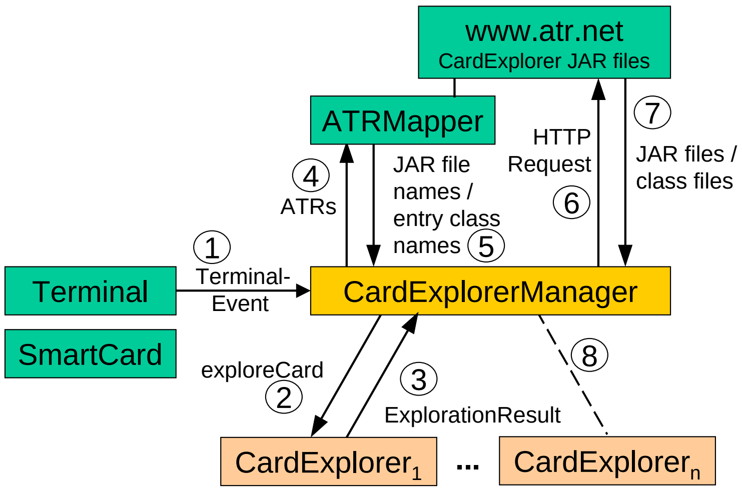

Figure 4.4: Download and instantiation of card explorers for unknown smartcards

explorers for many types of smartcards. These card explorers are made available as Java archive (JAR) files. A single JAR file aggregates multiple Java class files and other relevant files. An important part of a JAR file is its *manifest file* [Sun96] that contains information about the archived files. An example manifest file could look as follows:

```
Manifest-Version: 1.0
Main-Class: jinicard.javacardexplorer.JavaCardExplorer
Created-By: 1.2.2 (Sun Microsystems Inc.)
SmartCard-ATR: O78RAMAQMf5EU01AU1QgQ0FGRSAxLjFDwQ==
```

JAR files for card explorers contain two special entries in their metainf/manifest.mf files. The first special entry is the `Main-Class` attribute that was introduced with the Java 2 platform. It allows to designate the class that serves as the entry point into the card explorer. It refers to a class that implements the CardExplorer interface. The example manifest file refers to a card explorer that is able to explore Java Cards. The second special entry is named `SmartCard-ATR`. Its value is a set of Base 64-encoded ATRs.[9] This set determines the set of cards that the explorer is willing to handle. The example shows the Base 64-encoded ATR of a Java Card.[10] This mechanism can be extended by using regular expressions to gain more flexibility.

ATR Mapper

A component called ATRMapper (see Fig. 4.4) inspects all card explorer JAR files that are stored on *www.atr.net*, in order to establish a mapping from a set of ATRs to a set of names of card explorer JAR files. A card explorer manager that was not able to find a suitable card explorer for a particular

[9] Base 64 encoding is specified in [RFC2045] and allows to encode 8 bit data with 6 bit symbols. The ATRs have to be Base 64 encoded, because the manifest file specification does not allow for arbitrary 8 bit entries.
[10] Identifying the type of card used in the implementation phase can be left as a task to the interested reader.

card locally, contacts the ATR mapper available on *www.atr.net* (see Step 4) The result is (hopefully) the name of a suitable card explorer (see Step 5) that the manager can then use for download (see Steps 6 and 7) and instantiation (see Step 8) by using a custom class loader. This newly instantiated card explorer is then in charge of exploring the card in question. Alternatively, the ATR mapper could return the actual implementation directly instead of a URL.

Service Information Objects

The result of a successful smartcard exploration process is an object of type ExplorationResult that contains a set of ServiceInfo objects – one for each service. An ExplorationResult object is what is handed back from a card explorer to the card explorer manager, to enable it to instantiate card services as the final step. A ServiceInfo object contains Jini related information, such as the *service identifier*, *codebase information* and *entry point information*. The service identifier is used to uniquely identify the card service as a Jini service instance. The groups array specifies names of service categories that the service belongs to. *Name* and *comment* are user editable descriptions of a service. The *locators* attribute explicitly specifies lookup services that the service has to connect to once it gets initiated. The Jini specification prescribes that these service attributes (service ID, groups, attributes, and lookup locators) are stored persistently. Once a Jini service gets a service identifier assigned to it, it should remember that identifier and use it in all future interactions with lookup services and other Jini services. To comply with the Jini specification Jini related information is stored on the smartcard whenever possible.

4.4.3 Smartcard Services Exploration Layer

To enable the card explorer manager to retrieve the actual card service code, the codebase and entry point information are essential. The *service URL* refers to a site that contains the code of the card service described (named *www.service.com* in Fig. 4.5 on the following page). The *service class name* denotes a class that implements the interface CardService or one of its subclasses (e.g. AdministrableCardService). With this information the card explorer manager is able to dynamically download the card service code and create an instance by using a custom class loader.

To be useful, a card service must have access to its card-resident counterpart and therefore to the physical smartcard. This is achieved by using the SmartCard interface that the JiniCard terminal provides. The JiniCard framework passes the corresponding object to the card service by calling the setCard method (see Step 5) with a remote reference to the smartcard object.

The getAttributeSets method returns Jini attribute sets that are immutable and that do not depend on the specific service instance. Vendor information, for example, fits into this category of attribute sets. The getProxy method returns the proxy object that will (in serialized form) be uploaded (Step 6) to the Jini lookup service (abbreviated as LUS in Fig. 4.5 on the next page), where it can be downloaded by clients. No restrictions are imposed on the proxy object other than that it is serializable.

Implementing a Card Explorer

If a card service is to be written for a smartcard type of which a card explorer does not yet exist, then the developer has to provide an implementation of the CardExplorer interface. This interface has just a single method, named exploreCard, that takes a SmartCard object as an argument. The card explorer must find a way to explore the set of cards that it is wishes to handle. This can be done by using an on-card directory, that is particularly useful, if multi-application Java Cards are used. Another way

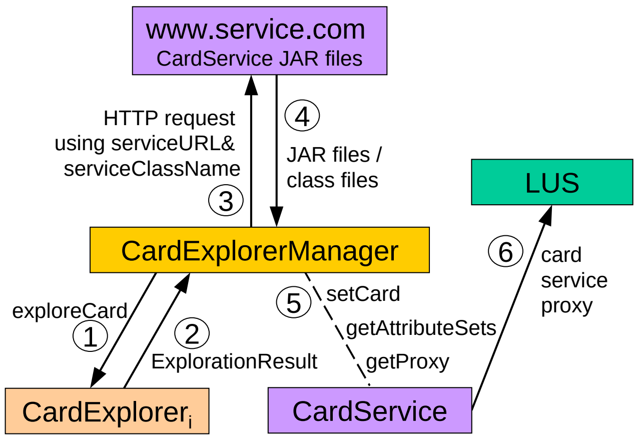

Figure 4.5: Download and instantiation of card services

to explore a card may be to simply probe the card by using some selection APDUs and by examining if the card generates the expected responses.

As described above, the result of the exploration process is a set of ServiceInfo objects that provide information about a service and also describe how to instantiate them. There are two different possibilities to instantiate card services: One is to provide a URL from which the service implementation can be downloaded (called *serviceURL*), the other is to provide a reference to the card service that the card explorer is able to instantiate by itself. The method getService is intended to get a reference to a card service that was instantiated this way. The JiniCard framework first tests if getService returns a valid (i.e. non-null) reference. If it does not, the ServiceInfo object must give a service URL to download the code from. The first approach might be useful if the set of services for a given card is fixed. This allows to store the service implementation together with card explorer implementation. Also, if the card-external code of a smartcard application is stored on the card itself, instead of being stored on a Web server, this might be advantageous as has been successfully demonstrated in [EUR01].

The reason for making the card-external part of a smartcard application available on a Web server, instead of storing it on the card itself, is the limited amount of memory that is available on current smartcards. The card-external part of a card application in fact may be orders of magnitude larger than what current smartcards are able to provide. It may, for example, contain a graphical user interface that often needs large amounts of code.

To install a card explorer, all class files that are related to it have to be stored in a JAR file. Its metainf/manifest.mf file has to contain the ATRs that are to be handled by the card explorer as well as the name of the implementation's entry class. Finally, the JAR file has to be uploaded to a well-known Web server, like *www.atr.net*, where it can be inspected by an ATR-mapper.

Implementing a Card Service

To implement a card service that the JiniCard framework can handle, the following steps must be taken: First, the interface CardService (or its sub-interface AdministrableCardService) has to be implemented. Apart from implementing the interface methods, this means implementing the actual service methods. The service uses the SmartCard interface to talk to the card. At runtime, an object implementing this interface will be provided through the setCard method. It is important to emphasize that the JiniCard framework does not define the way in which the card-external part of an application talks to its card-resident counterpart. Both parts have to agree upon a proprietary protocol, i.e. a set of APDUs and their meaning. The developer is free to define this private protocol, using APDUs. The developer is also free to design the card-resident part of the application in any way that he or she deems appropriate. This flexibility allows for the integration of cards that provide a fixed APDU protocol such as GSM SIM smartcards that use a standardized APDU protocol that is defined in [GSM11.11]. In that case the card-resident part, and therefore the APDU protocol was fixed, and our task was to write a card-external part that integrates a service for GSM cards into the JiniCard framework.

The service related class files have to be packaged as a JAR file and have to be made accessible to an HTTP server. If such a JAR file is small enough, it may also be stored on the card. In any case, the card explorer that explores the card must be able to examine the service information and to find a way to acquire access to the service code.

If a card service implementation is installed on a multi application card, then its existence has to be announced. This can be done by storing service information in some kind of on-card directory. Card explorers examine this directory to learn about services that are available from the card.

Summary

The JiniCard framework described in detail in the previous sections, though reasonably complex, has been prototypically implemented. Our prototype provides a solution to the problem of integrating smartcards into networked environments. It does this by providing a platform onto which the off-card counterparts of card-resident services can be dynamically downloaded and instantiated. Together, the card-resident and off-card components tightly co-operate using the platform's resources.

The framework can be considered as very flexible since it allows for the integration of all types of smartcards that comply with the named ISO standards. It facilitates smartcard integration and enables smartcards to become true active network nodes. This has been demonstrated by the rapid implementation of smartcard services on top of the JiniCard framework and has shown that the abstractions represented in the framework are useful.

4.5 Related Work

Recently, the integration of smartcards into networked environments has become more and more attractive to both academia and industry. In the sequel we briefly discuss the most important projects relevant to this problem domain and draw some comparisons to the JiniCard approach.

OpenCard Framework

Closely related to our approach of smartcard integration is the OpenCard Framework [OCF00; HH98; OCF99]. Originally, OCF was designed to run within a single Java VM which would block card readers to other applications. OCF uses an "application driven" paradigm. An application, that runs in the same Java VM as OCF itself, asks for a particular card service and waits until a card implementing that card service arrives. The card remains passive and does not get a chance to announce its capabilities and available services. To achieve this goal, a proactive paradigm is needed, in which the card is asked for its services that are then made available to the environment.

OCF also has no support for remote smartcard access. The proxy concept is used to hide the protocol to the service implementation on the smartcard. Similar to our approach is the use of so-called *CardServiceFactory* objects that produce Java objects through which card resident services can be accessed.

Although OCF defines interfaces and classes for application and card management, they are realized only rudimentary. In particular, the mapping from service descriptions to service instances is not defined. OCF is a statically configured framework, where all available services must be registered in a configuration file. However, this does not meet the requirement of spontaneous integration that we identified as an important issue in smartcard middleware.

CITI TCP/IP Stack

Webcard [RH99; RH00] is an implementation of a stripped-down Web server on a Java smartcard. The implementation consists of a card-resident Java applet that implements a subset of the TCP/IP stack that is necessary to implement a HTTP Web server. Several assumptions such as one TCP connection at a time, the HTTP request must fit into a single packet, etc., have been made for the particular implementation. Furthermore, only a subset of the HTTP/1.0 protocol has been implemented. The host PC routes incoming packets to the card's IP address through a serial interface to the card reader and finally to the card. Packets in the reverse direction are fetched from the PC from the card and further dropped onto the network.

CITI's Webcard server is to our knowledge the first attempt to integrate smartcards at the IP level. Although the first implementation does not implement a complete TCP/IP stack it demonstrates the general feasibility of this approach. With future smartcards with more computational resources it is possible to implement more complete networking stacks allowing for transparent integration of smartcards into the Internet. However, it does not provide a spontaneous networking framework for the integration of smartcards but instead focuses on the integration on the IP layer only.

The Internet Draft *"IP and ARP over ISO 7816"* [GM01a] that has its origins in the Webcard project mentioned previously describes the transport of IP datagrams over the asynchronous, half-duplex link layer protocols found on ISO 7816 compliant integrated circuit cards. This memo proposes a standard for communicating with cards using Internet protocols, thus connecting smartcards directly to the Internet and thereby lowering the barrier of integrating smartcards into Internet applications.

The ISO 7816 link layer protocols are half-duplex with the terminal always initiating the communication. In order to enable IP packets to flow from the ICC to the terminal, the draft specifies that the terminal may regularly poll the card sending it appropriate polling requests. This enables a card to also initiate communication. Another draft specification [GM01b] further concentrates on transport of TCP and UDP packets on this network layer with particular attention to header compression.

Currently, the only implementation said to be almost compatible with the first draft is CITI's UDP and TCP/IP stack.

ETSI Smartcard Platform Proposal

Lamotte [Lam01] has proposed a communication protocol between a terminal and a smartcard based on the Point-to-Point Protocol (PPP) [RFC1661]. The proposal considers a long-term effort to bring IP capabilities to smartcards. The approach uses standard Internet framing protocols to provide a layer upon which IP packets can be exchanged. At the time of this writing, however, no actual implementation is available, yet, and the paper should be viewed as a stimulus for providing strategic directions on how such integration can be achieved with a particular focus on the long-term development and deployment issues in the smartcard sector.

Bull Internet Card

Internet Card [Uri00; UST00; Uri01a] is an approach developed at the CP8 smartcard research labs of Bull. Their architecture comprises host-resident *agents* that are responsible for routing between the host's IP stack and card-resident agents. Beyond IP and UDP they are also capable of managing TCP sessions between a network node and the smartcard. Communication between the host and card agents is achieved by means of an asynchronous protocol called *SmartTP* comparable to a simplified version of TCP. The agents transfer the TCP or UDP packet payload, e.g. HTTP stream enclosed in SmartTP packets between host and card. On top of this system a card-resident Web server and a so-called *trusted proxy* have been implemented.

At present the architecture and protocols have been submitted to the Java Card Forum [JCF01] as a proposal for smartcard integration into networks [Uri01b].

Gemplus DMI

In contrast to the IP-based integration of smartcards, [VV98] describes an approach aiming at an RPC-like model of communication with Java smartcard applets. The basic idea is to apply the programming model of distributed applications to smartcards. This is achieved by an interface description language being a subset of the Java Card language from which stub and skeleton code are generated, similar to the CORBA programming model. Whereas in CORBA the IIOP "wire" protocol between object has been standardized, in this approach a protocol called *Direct Method Invocation* (DMI) has been introduced that can be viewed as a generic applet invocation protocol running on top of standard APDUs. The advantages of this approach that has been incorporated into Gemplus' development environments are similar to those usually listed for distributed programming, namely the higher level of abstraction of the communication model between clients and servers.

The DMI approach does not offer further abstractions such as service description, etc., targets simplified smartcard application development only.

CORBA Integration done at Darmstadt University of Technology

Similar to the DMI approach presented previously, [FMM99] describes a prototypical implementation of a CORBA-based architecture for the integration of smartcards. Here, different ORB models are discussed that separate the functionality between the card terminal and the card. The prototype targets GSM SIMs and uses a mobile phone simulator that can be used by the card-resident application to perform user interactions.

Gemplus Jini Surrogate Project

Gemplus is currently implementing a smartcard integration based on the *Surrogate* community project at Jini.org [Sun01b]. The Surrogate project aims to extend the Jini technology with an architecture that allows services with limited resources to participate in a Jini federation. Hence, the Surrogate project shares similar ideas with our approach supporting the thesis that there is demand for the integration of small devices into local environments.

Surrogate is closely related with Jini since it offers devices with limited resources a platform into which those devices can “inject” mobile code. This idea is comparable to the approach in the JiniCard framework where the card terminal offers an execution platform for the mobile code that can be used by the smartcard to offer its services to other clients in a network. Prior to the launch of the Surrogate project an active discussion about possible architectures and approaches for the integration of small devices took place on Sun’s *Jini-Users* mailing list.

Similar to the *“IP Interconnect Specification”* [Sun01a] Gemplus intends to provide an interconnect specification [Sur01] defining the integration smartcards into the Surrogate framework. At the time of this writing, no specification or implementation is available to the public. However, the authors have announced to release a version in the future for public review which would enable a more thorough comparison with the JiniCard approach.

Summary

The integration of smartcards into networked environments is a hot topic these days and quite different approaches have been proposed and some of them implemented. They can be roughly categorized according to the level of integration provided. Approaches such as CITI’s Webcard and Bull’s SmartTP focus on the integration at the IP layer or one level above. Lamotte’s proposal goes down even further by proposing PPP as a communication protocol much at the link layer. In contrast, other approaches such as the Gemplus’ Jini Surrogate approach and JiniCard focus on solutions at a much higher level of abstraction.

Due to the different nature of these approaches comparisons are largely depending on the application context such solutions are used for. Anyhow, it seems that the integration of smartcards has finally gained attention not only in the academic world but also in industry.

4.6 Security Aspects

The JiniCard framework allows smartcards to offer services in a network by means of a proactive exploration mechanism initiated by the card terminal. Clients access these card services through Jini service proxies that use the basic interface methods of the card terminal to communicate with the card-resident portion of the service. The most obvious problem with such an approach is the secure access from remote clients to the card, and the problem of the *card holder verification* procedure (CHV).

4.6.1 Smartcard Communication Assumptions

The CHV problem does not only arise in the context of JiniCard but equally well applies to the other integration approaches if remote access to the smartcard is considered. In traditional architectures the underlying assumptions of smartcard usage are that

a) communication between the card terminal and the card is trustworthy,

b) communication between the application and the terminal is trustworthy, and

c) communication between the input and output devices of the application is trustworthy.

Most often card readers are attached to terminals with which the card holder performs CHV to unlock the card for security-sensitive operations. This can be the PIN typed into an ATM, or the PIN entered into a GSM handset to activate the network authentication procedure.

Hence, the underlying assumption with this approach is that the communication link between the pinpad or keyboard and the smartcard is secure and cannot be eavesdropped or tampered with. For most of the practical application scenarios smartcards are used in, this assumption sounds quite reasonable. In scenarios based on remote usage of smartcards though, the link between the client and the smartcard must potentially be considered as untrusted and insecure, and special precautions have to be taken, i.e. the hostile environment problem has to be solved.

4.6.2 End-to-End Security Approaches

This section discusses some of the approaches that implement various degrees of end-to-end security with smartcards.

Secure Messaging

The ultimate solution to this problem would be to completely encrypt all data exchanged between the client and the smartcard, i.e. achieve end-to-end security. This would imply that the traditional ISO 7816 interface based on APDUs cannot be used any longer since it assumes the exchange of unencrypted APDUs. However, *secure messaging* as specified in [ISO94, Sect. 5.6] can be used

> "[. . .] *to protect [part of] the messages to and from a card by ensuring two basic security functions: data authentication and data confidentiality. Secure messaging is achieved by applying one or more security mechanisms. Each security mechanism involves an algorithm, a key, an argument, and often, initial data.*"

Basically, secure messaging offers a standardized framework for the encryption of the payload parts of APDUs. Unfortunately, for this to work the problem of key distribution must be solved. This basically means that the smartcard's remote client must be configured with the appropriate secret key(s) first before communication can take place. In a real distributed setting this would mean that the user has to somehow type in the shared key used for the session which is rather awkward and not suitable for practical use.

SSL/TLS Communication

A better solution would be to use protocols similar to the Internet's *de facto* standards SSL and TLS [RFC2246] that use public-key encryption to agree on a shared session key between a client and a server used for confidentiality and integrity of communication. Furthermore, the protocol allows for mutual authentication that might be necessary to not leak confidential information such as the card's CHV through a man-in-the-middle attack.

In [Tab00] we have performed an analysis and evaluation of the feasibility of a server-side implementation of the TLS protocol. The scenario comprises a TLS-enabled server hosted on a smartcard and an anonymous client, i.e. without TLS client authentication which is the standard case in the Internet today. Our analysis has led to the following essential observations:

- Several steps of the protocol are optional and can be omitted but the number of messages in the protocol after handshake cannot be reduced.
- Only the client has to perform computationally expensive operations during the handshake. On the server side, only the master secret has to be calculated but needs a random number generator to send its *ServerHello message*. Most smartcards with cryptographic features are today equipped with such random number generators.

- The largest amount of data exchanged is the server's certificate that is roughly about 1–2 kB in size.
- The server portion of the protocol is computationally less expensive than the client side, since parsing and signature verification of the certificate is necessary on the client side only.
- Session recovery could be used to reduce the handshake if repeated connections to the card occur.
- Current smartcards with cryptographic coprocessors might be able to offer a few TLS cipher suites matching most of the potential clients' cipher suites.
- Support for 32 bit arithmetics on the card is important for achieving reasonable performance.

Although we have not actually tried an implementation ourselves, the theoretical analysis reveals that an implementation would be possible with current smartcards, though not without a fair amount of effort.

CITI Secure Internet Smartcard

Instead of using Internet standard protocols, the implementation described in [IFH00] uses the SPEKE protocol [Jab96] for establishing a session key for channel encryption and at the same time authenticating the card and its remote user with a shared secret. SPEKE is a protocol achieving the same objectives as the EKE protocol [BM92] but is better suited for a smartcard implementation. SPEKE is based on the Diffie-Hellman protocol [DH76] and additionally achieves authenticity, thus avoiding man-in-the-middle attacks.

The authors describe how their implementation has been integrated into both SSH [YKS+01] and Kerberos [NT94] client applications. The SSH client uses the smartcard to electronically sign a challenge presented by the SSH server. The Kerberos client uses the remote card to unseal a DES-encrypted *ticket granting ticket*. They report a number of timing measurements that demonstrate that a secure channel using SPEKE can be established in the order of several seconds of which most of the time is spent executing the cryptographic operations in the smartcard.

Since research and formal underpinning of authenticated key exchange protocols has been undertaken (cf. [BPR00]), we think that remote smartcard usage is possible with suitable transparency from the user's perspective using the presented approach.

Compared to the TLS approach the advantage of these kind of protocols is that proving authenticity of the peer is done through the peer's knowledge of the password. In contrast, the drawback of using TLS is that a user at a terminal has to achieve and verify the authenticity of the remote TLS server in the card based on the server's certificate which is usually not the most user-friendly option.

Swisscom CASTING Protocols

Swisscom™ has applied for a patent [Swi00] for the SECTUS protocols developed in the CASTING research project. The basic problem is the remote usage of a smartcard from a PC over an untrusted wireless link, e.g. Infrared [IrD01] or Bluetooth [Blu01; HNI+98]. The basic approach taken is to involve the user into the protocols let him securely transmit secrets from the card to the PC or vice versa. In this scenario the card is inserted into a card reader equipped with input and output facilities. In a prototypical implementation described in [RV01] a mobile phone was used as the wireless card reader and demonstrated the integration of a RSA-enabled GSM SIM smartcard over an Infrared link as the security module for client-side authentication of a TLS-protected [RFC2246] HTTP session.

Essentially, the user acts as a trusted third party between two devices and has to read a secret from a display and enter it into another device. Obviously, this might be inconvenient and error-prone but if used correctly provides a reasonable solution to the remote smartcard problem.

4.7 Summary

In Sect. 2.5 we formulated the statement that there will always be a demand to integrate very small devices into local environments. In this chapter we have presented a generic framework that provides a solution to the question how small devices can be integrated into such environments. It is based on the mobile code paradigm and distinguishes between device detection, detection event compilation, device explorer lookup, device exploration, service proxy lookup, service proxy instantiation, and service integration. Furthermore, it makes proposals how the building blocks and protocols could look like and how the problem of finding suitable device explorers and service proxies could be solved. Since the overall problem depends to a reasonable degree on the underlying link layer technologies and device characteristics, a framework was chosen that allows for its instantiation into more concrete domain- and device-specific versions as needed.

Smartcards highly depend on their environments to provide useful services. Given these characteristics we have identified four key areas, that need to be taken into account by middleware for the spontaneous integration of smartcards. These are

- spontaneous integration into (networked) environments,
- transparent usage of card services,
- remote access to card services, and
- security that is effectively controllable and observable by the card's owner.

In this chapter a middleware for the integration of smartcards into networked systems has been presented. Our middleware comprises an execution platform for mobile code in a card terminal and a well-defined process of how appropriate mobile code is transferred to the terminal as smartcards are inserted into the terminal's card reader. We think that our approach outperforms other approaches w.r.t. flexibility and effort of standardization, which we generally consider a crucial point in proposing new middleware. The approach is easily extensible by uploading new card explorers to a well known Web server and by providing card service implementations. It also handles mutual exclusion of multiple clients that try to use a card concurrently. The independence of applets on Java cards seems to make a relatively transparent scheduling approach possible. The Jini network infrastructure has been used both as the trading platform for services offered by smartcards and as a means to implement the JiniCard framework as a set of cooperating network services. We have found Jini to be particularly well-suited for this purpose since it builds upon mobile code, which nicely fits into the paradigm of our proposed middleware.

Accessing smartcards remotely poses new security issues. In particular, the assumption that the communication between clients and card services is confidential no longer holds. Communication between clients somewhere in a network and smartcard services can potentially be eavesdropped and even tampered with. Approaches such as the SPEKE protocol to solve this problem have been presented that make the remote usage of smartcards possible as envisioned in this chapter. However, these security mechanisms have not been implemented in JiniCard since they are highly card- and application-specific and therefore belong into the card explorer and service proxy components.

In general we can conclude that the presented approach is well-suited to allow for the integration of smartcards into networked environments. Furthermore, there exist means to make the cards remotely and securely accessible within (partially) hostile environments from a reasonably trustworthy user terminal.

The basic approach of mapping device and object identifiers to executable proxy objects in the manner presented here was originally presented in [ADH$^+$99]. There it was applied to the domain of the management of networked objects and the notion of “nannies” was introduced, i.e. mobile objects that represent and care for real-world devices.

The approach taken by JiniCard presented in this chapter was initially described in [KPV00] and has been further explored and fully implemented in the diploma thesis of M. Rohs [Roh00]. The analysis of the card-resident server-side implementation of TLS was undertaken in the diploma thesis of H. Taborda [Tab00].

An overview of this work focusing on middleware for smartcards was published in [KRV00b]. An abridged version discussing some of the security aspects considered in Section 4.6.2 was presented in [KRV00a].

Chapter 5
Personal Security Modules based on Mobile Personalized Terminals

There are no systems that remain trustworthy when exposed to normal consumer Internet use and software acquisition. There are research projects trying to create so-called "trusted computing bases," but none has succeeded. Some systems are shipped today with the label "trusted," but none could protect a consumer's data and software in such an environment.

J. K. Winn and C. Ellison [WE99]

5.1 Introduction

The fundamental problem underlying this chapter is raised by the question how smartcards can be used in so-called "hostile environments". In Sect. 3.3 pertinent literature discussing various aspects of this problem was presented, however, currently it seems that secure communication between a user and a personalized smartcards in hostile environments is only possible through reasonably trustworthy terminals and appropriate protocols.

This chapter discusses another building block of personal security modules by proposing the use of a *mobile* and *trustworthy terminal* to access a user's personalized smartcard in a hostile environment. More precisely it builds upon the general assumption that a user is likely to trust a personal device more than a public terminal that is not under his or her direct control. Thus, a user takes advantage of his or her personal device to perform security critical decisions in a potentially hostile environment.

Furthermore, the terminal and the smartcard are *coupled* in a way that prevents usage of one of the devices without the other. Hence, the smartcard performs only those actions that the terminal has been previously accepted and the terminal is not capable of performing security-critical actions without the smartcard.

The rest of this chapter is organized as follows: Section 5.2 begins with a common example that exemplifies the general problem domain: The creation of electronic signatures with smartcards and suitable terminals. It outlines the legal frameworks underlying this application and discusses security issues in the signature creation process.

In Sect. 5.3 we present the *Personal Card Assistant* (PCA), which is a personalized security module based on an off-the-shelf PDA and a smartcard. The PDA acts as a mobile terminal used to communicate with the smartcard. Since it is assumed to be under the user's control, it might be much more trustworthy from the user's perspective than other non-mobile devices. We describe the general setting of the PCA, the involved communication protocols, and the underlying cryptographic protection. Furthermore, related work on the trustworthiness of terminals is discussed focusing on approaches that also consider PDAs as trustworthy terminals, trustworthy operating systems, and trustworthy document presentation.

Some technical opinions on the approach presented are discussed in Sect. 5.4 focusing on the *pairing* of the terminal and smartcard, i.e. the cryptographic binding of both devices.

The chapter ends with a summary in Sect. 5.5.

5.2 Motivation: Electronic Signature Creation

Cryptography can provide security services based on well-founded mathematics. A key problem with applying cryptography to real-world problems is, however, the interface to real life. In this chapter we first investigate an application area where this problem is very evident, i.e. the presentation of a document that is to be electronically signed.

According to the directive of the European Parliament [EUP99, Article 2.2] an advanced electronic signature means an electronic signature that meets the following requirements:

a) it is uniquely linked to the signatory;
b) it is capable of identifying the signatory;
c) it is created using means that the signatory can maintain under his sole control, and
d) it is linked to the data to which it relates in such a manner that any subsequent change of the data is detectable.

Based on this definition a working group of the *European Electronic Signature Standardization Initiative* (EESSI) specifies so-called *secure signature creation devices* (SSCD) [CEN01a] as a *protection profile* (PP) according to the *Evaluation Criteria for IT Security* [ISO99a; ISO99b; ISO99c].[1] It provides standards for the high security requirements found in electronic commerce and e-government scenarios.

Some countries have already embedded electronic signatures into legal frameworks, the most prominent example being the German electronic signature law “Signaturgesetz” [SigG97; SigV97]. This law requires (among other things) the following evaluation criteria levels for a system used for dealing with electronic signatures:

- the secure signature creation device (usually a smartcard) must meet the criteria of the *Evaluation Assurance Level* 4 (EAL 4).
- the other components, e.g. for presenting a document (document viewer) must meet EAL 2.

Both requirements form the basis for electronic signatures that are *legally binding* under this law.

When considering this legal framework from a technological perspective, it is evident that one of the weakest components is in practice a document viewer running on a PC with a standard operating system like Windows: Even if evaluated at EAL 2, the PC Software offers little protection against manipulation by malicious software such as viruses or Trojan horses.

[1] Also known as *Common Criteria* (CC).

In contrast to this, a smartcard is a (comparably) tamper-proof device that offers cryptographic and other functions that can be accessed over a simple I/O interface. For performing critical functions, it is required that the legitimate user is authorized against the card by entering a PIN code (often referred to as card holder verification, CHV). As has been discussed earlier, a smartcard has no interface to interact directly with a human being, all communication is done via a card reader using a keyboard and screen that is either built into the reader or is attached to a computer (cf. 3.3).

In the protection profile [CEN01a, Annex, Sect. 3.1] which also refers to the *signature creation application* (SCA) proposal [CEN01b], for example, some general assumptions are listed of which one states that

> "*The signatory uses only a trustworthy SCA* [signature creation application, ed.]. *The SCA generates and sends the DTBS-representation* [data to be signed, ed.] *of data the signatory wishes to sign in a form appropriate for signing by the TOE* [target of evaluation, ed.]."

This is in particular problematic if the platform used for viewing such a document is not "under control" of the electronically signing party, but belongs to the other party that wants someone to sign a document: It is fairly trivial to manipulate such a system, so a person signing a contract or a money order in an unknown, untrusted environment cannot be sure what her smartcard actually signs. This could turn out to be a major obstacle against the wide-spread use of electronic signatures in practice.

This problem is, in principle, easy to solve: Raise the security level and require a closed, trustworthy system for applying electronic signatures. Unfortunately, this solution is extremely hard to put into practice, both because it is expensive and since dedicated hardware, that would be required, simply does not fit into today's computing world.

Summing up, applications based on smartcards rely upon the trustworthiness of the environment the card is working in. However, this trustworthiness is not given in many settings smartcards are used in.

5.3 The Personal Card Assistant

In this section we present the *Personal Card Assistant* (PCA), a scenario that brings together PDAs and smartcards. The underlying idea is that a PDA acts as a personal device for controlling a smartcard attached to it using an asymmetric key pair. We describe how such an approach can be used for creating electronic signatures: in particular, we can circumvent the problems involved with untrusted document viewers in this context.[2]

5.3.1 PCA Overview

We propose a pragmatic approach that reduces the risks of using electronic signatures by integrating a customer's PDA into the creation of electronic signatures: The PDA is used as a document viewer and it controls the smartcard by unlocking the card's signing function using cryptographic means. We refer to this approach as the *personal card assistant*. A PCA does not increase security *per se*, since a PDA can be attacked similarly to a PC. However, as we assume that a PDA *belongs to* and

[2] Hence, the main goal is to have a **WYSIWYG** (what you see is what you get) system and not a **WYSIAWYG** (What You See Is Almost What You Get) or a **WYSIMOLWYG** (What You See Is More or Less What You Get) style of presentation that, from a security perspective, should be avoided under all circumstances.

therefore *is under the control of* a person who wishes to apply an electronic signature, such a device will *in practice* be more trustworthy for that person than, for instance, a vendor's PC, and thus is better suited for use in hostile environments than a smartcard alone.

Therefore, the approach can be regarded as *pragmatically more secure*, making users of electronic signatures feel more comfortable with the technology. The notion "trust amplifier" for such a PCA covers this quite precisely.

The PCA aims at improving this situation: It consists of a secure core component, the smartcard, and a conventional, personal computing device, the PDA. Both can either be tightly coupled by integrating the card into the PDA, or communication between the PDA and the smartcard is over a possibly untrusted network. This leads to the general design option whether the card is co-located with the terminal or remotely connected (cf. Sect. 3.4).

We consider the latter, more difficult case, in which the coupling is achieved by the fact that each component knows the public key of the other one. Key exchange takes place in a secure environment, e.g. when the smartcard is personalized or purchased. In the PCA scenario, the role of the smartcard is to provide both secure storage and a trusted platform for cryptographic computations, and the PDA provides a user interface, computing power, and additional storage. The sharing of public keys enables both to establish a secure communication channel even if they are physically separated.

An application will typically run on the PDA, making use of its I/O capabilities, and access the smartcard for cryptographic functions. Thus in terms of our framework the PDA is considered to be the "active" component whereas the card is rather "passive". But it is also possible to run the application on the smartcard and use the PDA simply as a supplementary I/O device – an approach further discussed in Chapters 6 and 7.

A PDA is open to attacks similar to those applicable to a PC. However, it is likely that the PDA owner accepts a much more restrictive security policy on her PDA than on her workstation, e.g. concerning the download and execution of unknown software. It is also realistic to set a separate PDA aside for performing critical transactions such as electronic signatures.

From a pragmatic viewpoint, one may accept the PCA as a "trust amplifier" due to its nature of being directly associated with a person. To its owner, it is much more trustworthy than an unknown terminal, controlled by strangers, located in an untrusted environment.

The PCA for Electronic Signatures

To better motivate the PCA a scenario is presented where the use of the PCA can enhance the process of creating an electronic signature. The example describes a setting where a document created by one party, e.g. a contract offered by a vendor, is to be signed by a second party, the customer. This approach involves the following components:

- A PC or workstation that is used to create a document to be signed. This could be a vendor's terminal.
- A smartcard reader, either connected to this PC or being a separate device.
- A PDA that belongs to the person who wants to sign a document.
- A smartcard for signing a document by encrypting a hash value.

This requires that both the PDA and the smartcard have the public key of the other one stored, i.e. they together constitute a PCA. We assume that components can communicate over arbitrary communication channels; as an example one can imagine using the PDA's infrared interface.

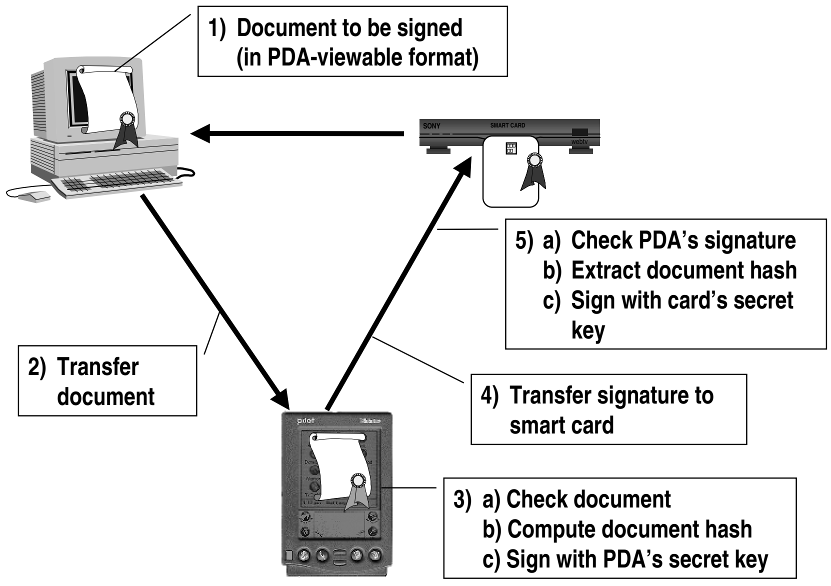
1) Document to be signed (in PDA-viewable format)
5) a) Check PDA's signature
b) Extract document hash
c) Sign with card's secret key
2) Transfer document
4) Transfer signature to smart card
3) a) Check document
b) Compute document hash
c) Sign with PDA's secret key

Figure 5.1: The PCA in the context of signing documents

A Bird's Eye View of the Scenario

Figure 5.1 outlines the interworking of the components of this approach:

- A document to be signed is created on the PC, and this document is stored in a format that can be displayed on the PDA.
- The document is transferred to the PDA.
- The user checks the document on the PDA and approves it by signing the document's hash with the PDA's secret key.
- The document hash is transferred to the smartcard, that extracts the document's hash value again and creates the final signature.

This procedure differs from the standard approach to using electronic signatures in two important points: First, the document is "routed" over the PDA for being checked by the signing person; second, it assumes that the smartcard of the signing person and the PDA form a pair, tied together by their public keys. In particular, the card will not sign any data unless these data were "approved" by the PDA's secret key. We shall elaborate the concrete procedure for this subsequently.

5.3.2 The Underlying Cryptographic Protocol

Hereafter, the identifiers E_{Crd} and D_{Crd} are used for denoting the smartcard's public and private keys respectively, and similarly E_{PDA} and D_{PDA} for the PDA. The application of a key K to a message M, e.g. encrypting the message, will be denoted by $K(M)$.

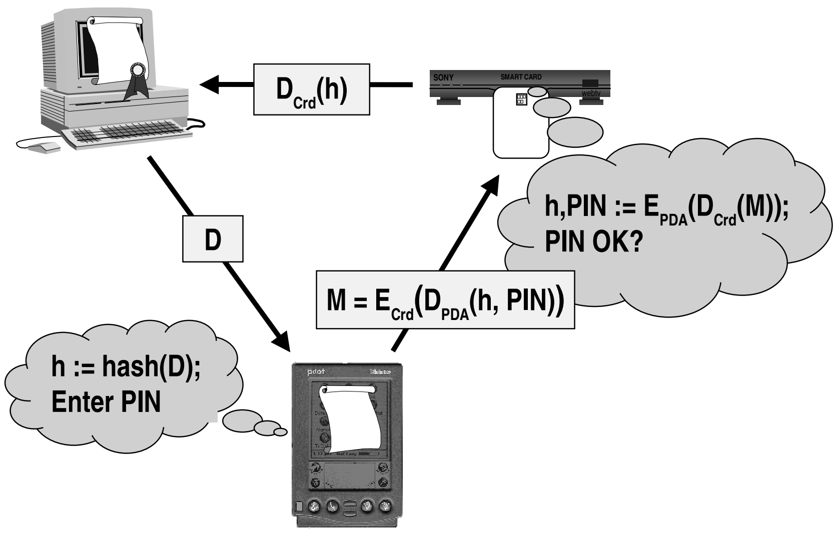
$D_{Crd}(h)$
D
$M = E_{Crd}(D_{PDA}(h, PIN))$
h,PIN := $E_{PDA}(D_{Crd}(M))$;
PIN OK?
h := hash(D);
Enter PIN
SONY
SMART CARD

Figure 5.2: Cryptographic View of Information Flow

Figure 5.2 visualizes the communication between the PCA components:

— **PC → PDA:** The PC sends a document D to the PDA.

— **PDA:** The PDA displays the document D and computes $h = \text{hash}(D)$. If the user wishes to sign the document, she approves it. We propose to implement this by having the user enter the card's PIN, which has the side-effect that the PDA is used as a PIN pad.[3]

— **PDA → Smartcard:** The PDA sends the message $M = E_{\text{Crd}}\big(D_{\text{PDA}}(h, \text{PIN})\big)$ to the card. Thus, the PDA signs the document hash h and the PIN with its private key and encrypts the resulting data with the card's public key. Note, that the contents of M can only be reconstructed with the secret key D_{Crd} matching E_{Crd}.

— **Card:** The card deciphers the message using $(h, \text{PIN}) = E_{\text{PDA}}\big(D_{\text{Crd}}(M)\big)$, i.e. the card extracts the PIN and the hash h from the message M using its own private and the PDA's public key. The procedure aborts if verification of the PIN fails.

— **Card → PC:** The card sends $D_{\text{Crd}}(h)$ to the PC, which is the document hash signed with the card's secret key. This constitutes the final signature.

By signing the data sent to the card, the PDA assures the authenticity of the data. This is necessary since the smartcard will only sign a hash value that originates from the PDA. By the PDA's signature, separate steps for authentication and key exchange are avoided.

[3] The scenario and the subsequent protocol can be easily modified to allow a user to enter the PIN using a (secure) PIN pad attached to the card reader. In this case, the user's approval shall be implemented by other appropriate means, like pressing an "OK"-button.

Entering the PIN ensures that the signing process is authorized by the owner of the PCA. This addresses the issue that PDA's are not very well protected against unauthorized use. To protect the PIN from attackers intercepting the message to the card, the message is encrypted with the card's public key.

5.3.3 Informal Threat Analysis

Under the assumption that the PDA and the card of the scenario described in Section 5.3.2 are trustworthy, the protocol can only be attacked by manipulating data sent between the components:

— **PC $\rightarrow$ PDA:** An attacker does not gain anything from manipulating D since the document will be checked by the signer. Neither does replaying this message, or preventing it from arriving offer any advantage to attackers.

— **PDA $\rightarrow$ Card:** Under the assumption of secure cryptographic algorithms and sufficient key lengths, the contents of the message M is not reconstructable[4].

  Since the range of the PIN is restricted, there is a slight chance that a forged message gets signed by the card, even if D_{PDA} is not known. However, the signature produced will surely not be valid for any document, as the hash value reconstructed by the card will be totally random and not correspond to any meaningful document.

  A replay of this message to the card would create only duplicates of the signature computed by the card, which is acceptable. Blocking the communication between the PDA and the card prevents only the generation of signatures.

— **Card $\rightarrow$ PC:** Since only the completely generated signature is transmitted, there is no meaningful attack left. The signature can be easily verified by interested parties.

Note that the assumption about the PDA's trustworthiness made above is not necessarily justified: A PDA is usually not a secure system and is, in principle, as easy to manipulate as a PC if an attacker can temporarily control the device. However, in practice it is certainly more difficult to attack such a mobile device than a PC.

5.3.4 Related Work on the Trustworthiness of Terminals

Besides the communication path between the user and the smartcard also the trustworthiness of the terminal itself is crucial in a signature scenario. Already Gasser [Gas88] has discussed the design of secure computer and operating systems and proposed how a *trusted computing base* (TCB) can be integrated in such systems. He has, for example, proposed to use a LED to inform the user about an interaction that originates from the TCB. This LED is not controllable from non-trusted components to avoid fake dialog attacks.

Trustworthy Terminals based on PDAs

The following systems use dedicated hardware such as PDAs acting as "trustworthy" end-user terminals.

[4] Note that encrypting the PIN alone without the hash h or suitable padding schemes changes this situation: Since usually only $10^{|\mathrm{PIN}|}$ values for PIN exist, a brute force attack by enumerating possible PINs would be possible. The attached hash value h – though known – prevents such attacks, since the contents of the encrypted message becomes too long for being enumerated.

Balfanz and Felten [BF99] present a PKCS #11 [PKCS#11][5] compatible library implementation on a PC that delegates cryptographic operations on a PDA. Their work is based on the observation that normal personal computers should not be trusted to perform cryptographic operations at all. Similar to the PCA approach is that the PDA is used to enter the PIN code which controls access to the cryptographic keys – both therefore never leave the PDA. However, as they admit their solution is not secure because the document is displayed and hashed on the still insecure PC – which has been solved with the PCA. The PDA acts only as a smartcard with a secure keyboard but does not provide for more security at the application level, in this case the electronic signature process. Furthermore, in comparison to a smartcard a PDA cannot be considered as being tamper-resistant.

Daswani and Boneh [DB99] describe a personal security environment built on a Palm PDA. It can be used as an electronic wallet that is capable of producing electronic signatures as part of their wallet protocol. They discuss in depth different implementations and performance measurements of cryptographic algorithms such as RSA and ECC on the Palm PDA, however, without taking a smartcard into consideration.

Freudenthal *et al.* [FHW00] have implemented a system on a Palm PDA that enables them to sign documents that are accessible on the PDA through its *Memopad* application. They have essentially implemented a PKCS #11 library that is used to control a smartcard inserted in a reader attached to the PDA. Thus they rely on the PDA as the trusted component for document presentation and hash computation. The major drawback in comparison to the PCA, however, is that there is no direct coupling between the PDA and the card. Hence, their approach is only suitable for usage in co-located scenarios of terminal and smartcard (cf. Sect. 3.4).

Blumert [Blu00b] gives a good overview of the problems around electronic signature creation in the context of German signature laws. Furthermore, he has implemented an electronic signature system including a document viewer on a Palm PDA. His work is comparable to Freudenthal *et al.* and the PCA approach without, however, integrating a smartcard into the signature application, although he proposes this in his future work section.

The ESPRIT Project CAFE [BBC$^+$94] have proposed an off-line digital payment scheme using two kinds of devices: so-called *wallets* and *guardians*. Wallets are devices that contain a screen and some sort of keyboard and thus are comparable to PDAs. Guardians are smartcards that store financial information. Payment information is communicated to the guardian through the wallet which asks the user to confirm that this information is correct. Since the wallet is under the user's control it prevents *fake terminal attacks* where the payment terminal shows one amount on the screen but deducts a completely different amount from the user's card. The PCA shares basic ideas with the CAFE wallets and guardians, however, it aims at a more general binding between both kinds of devices by means of a cryptographic coupling in which smartcards can be used remotely from the PDA.

Personal Trusted Computing Bases

Secure operating systems have already been investigated back in the sixties with the MULTICS operating system [CV65]; numerous other research efforts have been undertaken meanwhile. With the advent of PDAs, trusted computing bases for personal use have been investigated in several research projects during the last decade of which we briefly discuss some of the more recent activities.

Eckert [Eck00; Eck01; BE01] observes that PDAs might offer great opportunities to become trusted personal security modules to enhance the security of distributed and mobile computing. However, the devices itself must be reasonably secure if they are used to perform security-critical

[5] Also known as the *cryptoki* or cryptographic token interface standard.

transactions. In the sequel she performs a risk analysis of current PDA operating systems[6] focusing on operating system security and availability of cryptographic functionality. She concludes that "*due to the lack of appropriate authentication and access control measures mobile devices are to a great extend exposed to unauthorized access and information disclosure.*"

Pfitzmann *et. al.* [PPSW95] discuss *portable end-user devices* (POBs) and security modules and define a number of requirements to be made for such devices. They observe and conclude that "[...] *No mobile user devices currently available are tamper-resistant in all their parts, so it is reasonable to supplement less secure devices available from many manufacturers with security modules. Providers can concentrate on developing protocols for security applications that can run in any device, from mobile phones to personal digital assistants.*" In [PPSW96] they further distinguish between different types of trust in a mobile user device:

— **Personal agent trust:** The trust of mobile users that their devices act according to their wishes.

— **Captured agent trust:** The trust that the mobile device protects its user even if it is lost or stolen.

— **Undercover agent trust:** The trust of third parties relying on the mobile device to protect them from the user.

In particular they consider a combination of a tamper-resistant device such as a smartcard in combination with a user terminal, e.g. a PDA. In particular they focus on the design, production, shipment, and personalization of mobile user devices, each of which should be reasonably trustworthy to obtain an end-system that is also trustworthy. Our work is therefore motivated by similar considerations as brought up by Pfitzmann *et al.* and further emphasizes the role that personal terminals could play in future mobile scenarios.

Stüble *et al.* [Stü00; PRS$^+$01] are currently implementing a secure operating system called PERSEUS based on a micro-kernel architecture. Their observation is that *commercial off-the-shelf* (COTS) operating systems for mobile devices will probably not be made sufficiently secure in the near future for various reasons, mainly commercial aspects. Therefore, they propose to run such COTS systems on top of a secure trusted operating system kernel. The kernel provides core facilities, such as, trusted communication channels between a security module, e.g. a smartcard, and the display and keyboard. These can be used to perform security-critical operations, such as, digital signature creation. They have implemented parts of their system on top of the FIASCO μ-kernel [Hoh98b]. Their goal is to come up with a kernel sufficiently compact that an evaluation according to the Common Criteria [ISO99a] is feasible.

Trustworthy Document Presentation

Scheibelhofer [Sch01] has addressed the separation of a document's *content* and its *presentation* that is to be signed. He discusses the problem of presentation of a document on different devices using different markup languages. His approach is built on top of XML [W3C00] for encoding the data to be signed according to XML schema descriptions [W3C01] that could be signed by a legal authority. Furthermore, this authority could also sign style sheet transformation descriptions such as XSLT [W3C99] for explicitly defining the presentation of the content on different presentation engines such as HTML or WML [WAP99a] browsers. He also introduces the concept of *transformation filters* performing transformations during presentation, e.g. a terminal could read a document to a deaf user.

[6] PalmOS 3.x, EPOC R5 and Windows CE.

Although the approach is very generic, we consider his work as a first step towards a framework for the trustworthy adaptation of content to be signed on different devices and media channels.

5.4 Design Issues for Personal Security Modules

Throughout the previous sections we have described research efforts focusing on the design and implementation of personal security modules. Since the problem of a trusted communication path between a tamper-resistant trustworthy device and its user cannot be considered as solved in practice, any attempt has to consider the trustworthiness of the terminal. The PCA approach presented takes this into account and tries to shift critical computations from non-trustworthy components towards more trustworthy ones.

5.4.1 Personalization Enables Strong Cryptographic Binding

One of the most significant contribution of the PCA is the cryptographic binding introduced between a reasonably trustworthy terminal – the PDA – and the core security module – the smartcard. In the PCA approach, both know each other's public key and the protocol is considered to be strong w.r.t. the cryptographic security implemented. However, using public key cryptography requires

- the knowledge of a rather "large" amount of cryptographic key material – up to several thousand bits (two *private keys* plus two *public keys*),
- reasonably sized storage to safely keep this key material, and
- the ability of smartcard and personalized terminal to find and make use of each other.

The advantages of this effort are

- shifting of critical computations from less trustworthy towards more trustworthy components,
- tight coupling between personalized terminal and security module, i.e. both are not usable without each other, and
- more flexibility since potentially, several different terminals can be used depending on the concrete application and users' needs.

Compared to non-personalized and co-located terminals a coupling between terminal and smartcard is much harder to achieve. For example, the SPEKE protocol used in CITI's Secure Internet Smartcard (cf. [IFH00], Sect. 4.6.2) uses a shared secret between the user and the smartcard to achieve both – authentication and symmetric session key agreement – using a password. The advantage is that a non-personalized terminal can be used for that purpose, but with the disadvantage that the password must contain sufficient bits to avoid brute-force attacks and that it might be eavesdropped at the terminal.

Hence, in general some kind of cryptographic binding as presented in our approach seems to be advantageous over password-based approaches but comes at a certain cost in terms of implementation. The actual protocols between the terminal and the smartcard, however, seem to get simpler.

5.4.2 Replay Prevention

Another problem with smartcards that has been already mentioned is the lack of clocks in current off-the-shelf smartcards. Although more advanced security modules such as the JavaRing™ [Dal00] and some newer USB tokens have a built-in clock, this has the general drawback of an increased form factor introduced by the needed battery that is more vulnerable to physical attacks.

As a consequence, if a smartcard is used as a security module within a binding relationship, the card is generally vulnerable to replay attacks, if no additional measures are taken. A straightforward approach is to introduce a shared unguessable counter between both devices that can be easily implemented using a reasonably good random number implementation shared by both devices. This makes guessing the "next" value in a sequence rather problematic for an intruder.

Thus, prior to usage of the smartcard from a terminal, both devices have to be suitably *paired*. Pairing as known from Bluetooth [Blu01] considers the process of establishing a mutual state in two devices that can be used to perform mutual authentication and possibly allows for shared secret generation. Pairing in the form of *imprinting* has been successively studied by Anderson and Stajano in [SA99] where different aspects of the life-time of such paired relationships have been studied. For security reasons, pairing has to take place in a secured context, i.e. where unobserved exchange of a secret can take place.

If a security module is used in conjunction with several terminals simultaneously, pairwise counters can be used for each pair of terminal and smartcard. Furthermore, if the message exchange between the terminal and the smartcard is not reliable enough suitable "sliding window" schemes on the counter sequence can be used resulting in a certain degree of *resistance* against lost messages.

5.4.3 Mutual Discovery

Besides the choice and implementation of the communication protocols between the smartcard and its terminal(s), the mutual service discovery is important if flexible use is considered. Thus, terminal and smartcard can be subject to spontaneous networking as introduced in the chapters presented previously. For example, in our working implementation of the PCA we used Jini as the spontaneous networking middleware. More precisely, the PDA offers a SignatureService to electronically sign documents that are sent through the service's interface from clients. In turn its implementation uses the CardSignatureService implemented by the smartcard to perform the final and legally binding encryption of the document hash.

Here, we can easily see how the idea of a smartcard offering services in the spirit of the JiniCard framework nicely fits with the personalized terminal approach as introduced with the PCA. The advantage is that decoupling smartcard and terminals generally offers more flexibility and thus might allow for a more differentiated use of a smartcard as a security module in different usage contexts.

5.5 Summary

Electronic signature creation is an ideal application to demonstrate the general problems of smartcards in hostile environments. We have presented the PCA approach substituting a "trustworthy" PDA for an "untrusted" terminal in order to create electronic signatures with a smartcard. The card and the PDA are tightly bound together through a pair of cryptographic keys. This is the main difference to the work presented by Freudenthal *et al.* [FHW00] who have not further coupled the PDA and the signature card.

The PCA was implemented on a Palm PDA using a standard Java smartcard to implement the signature card. The card was inserted into a card reader at a vendor's terminal simulated by a PC preparing the document to be signed. Communication from the vendor's terminal to the PDA was done using the IrDA IrOBEX protocols [IrD01] used for object exchange. Jini [Sun99a] was used as spontaneous networking technology for dynamically discovering services in such an environment. The card reader and the PDA's signature service were implemented as Jini services registering with a vendor's lookup service. We have used the RSA algorithm for the public key encryption and our measurements on a Palm™ III PDA showed that RSA encryption of a 512 bit block (64 bytes) on the PDA requires roughly 12 seconds. These numbers are comparable to the measurements made in [DB99].

Although the work presented in this chapter is explicitly targeted towards the use case of electronically signing a document it is by no means restricted to this application domain. Which data are transferred between the terminal and the smartcard is actually application-specific. Hence, the cryptographic protocols that provide some kind of *secure envelope* for data can be easily extended for other applications without requiring completely new solutions.

The PCA could be a first step towards practically usable personal security modules and the flexibility it provides is likely to be needed in future scenarios. In combination with the JiniCard framework it might be one example of a more flexible system for the integration of a set of user terminals implementing different degrees of trustworthiness that are under control of a central trustworthy component – a smartcard.

The PCA was initially described in [KPV99b] and later published in [KPV99a].

Chapter 6
The WebSIM or How to implement a Web Server in a GSM SIM?

A mobile phone is a disposable, wireless smartcard reader which can be used to make phone calls with. . .

Bertrand Du Castel, Java Card Workshop, Cannes, September 2000

6.1 Introduction

GSM [MP92] is currently the largest mobile telephony system with roughly 500 Mio. subscribers run by more than 200 mobile operators in the world.[1] Each GSM mobile phone is required to contain a smartcard – the so-called GSM *Subscriber Identity Module* (SIM) [GSM11.11] whose most important task is the authentication of the mobile operators' subscribers.

From a different viewpoint that has been nicely pointed out by du Castel in the citation that precedes this chapter, the mobile phone essentially can be seen as a wireless smartcard reader for the SIM card. Consequently, this chapter describes an approach to integrate the SIM smartcards used in GSM mobile phones into the Internet.

Our *WebSIM* system described in the sequel provides an HTTP-based interface to the Internet that can be used to access services in a customer's SIM. This communication channel can be used in various ways to implement security protocols significantly improving secure electronic and mobile commerce. The WebSIM represents approach A3 which is about a mobile terminal that is in possession of a wireless communication link as identified in Sect. 3.5. It essentially turns the mobile phone into a mobile terminal used to offer and access the security services residing in a user's SIM.

The rest of this chapter is organized as follows: Section 6.2 briefly introduces the SIM and the role it plays in the GSM authentication protocol. In Sect. 6.3 we present the SIM application toolkit which is the core technology layer on top of which the SIM can perform interactions with its user and communicate with the rest of the world.

The current state of security practices found in the Internet today is briefly summarized in Sect. 6.4. It basically leads to the observation that the idea of bringing the GSM security infrastructure into the Internet is a promising approach that should be further investigated. Section 6.5 describes the WebSIM protocols, architecture, and proof-of-concept implementation that is part of

[1] The actual numbers differ quite substantially so we just give a rough number here.

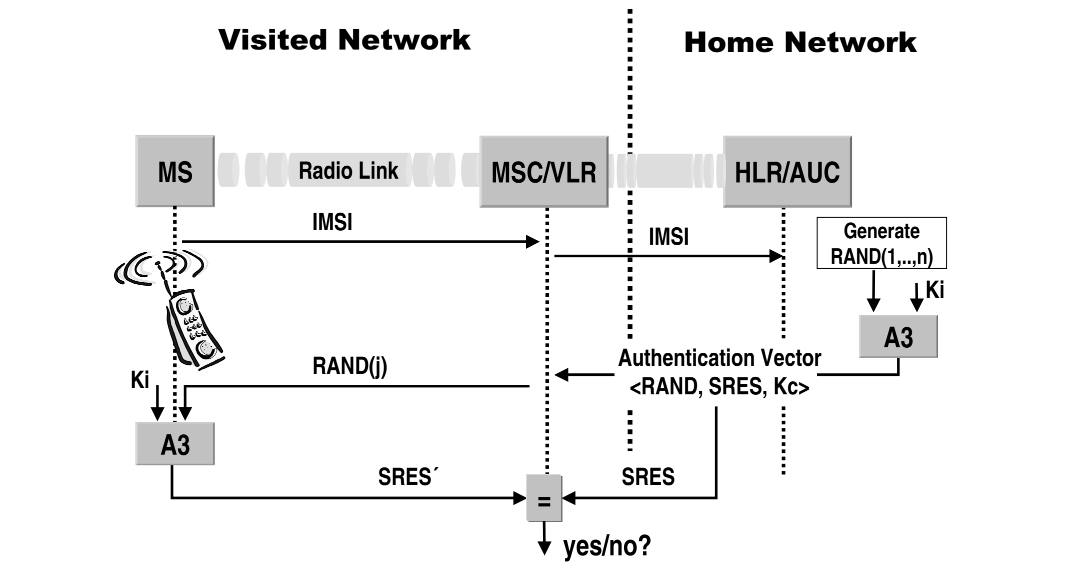

Figure 6.1: GSM authentication overview

this thesis. It demonstrates that such a system can be set up in realistic scenarios and that it really contributes technical solutions to today's Internet security problems.

Security aspects of the WebSIM approach are discussed in Sect. 6.6 leading to the general observation that although the WebSIM provides interesting security features these might not be sufficient for all application domains. The most important drawback is the lack of end-to-end security between a WebSIM client and the SIM, an issue further discussed in Chapter 7.

Section 6.7 describes a number of possible applications that can be built on top of the WebSIM. It illustrates that the WebSIM is a technology layer forming the basis of other security-related services.

Related work is discussed in Sect. 6.8 and a summary in Sect. 6.9 completes this chapter.

6.2 GSM Subscriber Identity Module

A GSM SIM is an operator-trusted security server in GSM, performing computations on behalf of the GSM subscriber. The SIM is issued by a mobile operator to its customers as part of their contract. The main purpose of the SIM is to authenticate a subscriber to the GSM network which is vitally necessary for access control and billing. As a side effect it is used to establish shared secrets between the mobile phone and the GSM network used for encrypting the over-the-air voice traffic. The GSM security architecture is specified in [GSM03.20]. Figure 6.1 gives a simplified overview of the GSM authentication protocol illustrated by the example of roaming users. After the *mobile station* (MS) is switched on, the mobile requires the user to enter the *card holder verification* (CHV, aka. PIN) of the SIM and upon successful verification by the SIM has access to the so-called *International Mobile Subscriber Identity* (IMSI), a globally unique subscriber number stored in a special file on the card. This IMSI is sent in plain text to the visited mobile network and comprises, among others, the identifiers of the *mobile switching center* (MSC) and the *visitor location register* (VLR). Based on the IMSI they infer the identity of the home network of the subscriber to obtain a so-called *authentication vector*. This vector is compiled by the home network by computing so-called *authentication triplets*:

$$AUT = \big(RAND, SRES = A_3(RAND, Ki), Kc = A_8(RAND, Ki)\big).$$

Here, Ki denotes the shared secret key between the customer's SIM and *authentication center* (AUC) of its home network. Furthermore, A_3 and A_8 are proprietary algorithms implemented in the SIM and its AUC only.

A number of such triplets are sent back from the corresponding home network to the visited network which then arbitrarily chooses a triplet and sends the challenge *RAND* via the mobile to the SIM which in turn computes $SRES' = A_3(RAND, Ki)$ using the local implementation of A_3 and the secret key Ki.[2]

The network then grants access based on whether *SRES* equals *SRES'*, or not. Then the network knows to which home network accumulated billing records must be sent. Encryption of the wireless link between the MS and the network is performed through a standardized algorithm A_5 (not shown) using the corresponding key Kc.

Summing up, the protocol the SIM participates in, achieves authentication of a subscriber as the fundamental security-related operation in GSM. The SIM therefore provides a security service to the mobile network and the mobile phone.

6.3 SIM Application Toolkit

The SIM Application Toolkit (SAT) [GSM11.14][3] is an interface implemented by GSM mobile phones offering among others the following services to the GSM SIM:

— DisplayText(*text*): Displays the supplied text on the display of the mobile phone.

— GetInput([*title*],[*type*]): Displays an optional title text and queries the user for input. Several syntactic categories such as digits, hidden input, etc. are supported. The text entered by the user is returned to the SIM.

— SelectItem([*title*],{*item*...}): Displays an optional title and a number of items among which the user can chose. The number of the chosen item is returned to the SIM.

— ProvideLocalInformation(*type*): Return localization and network measurement results depending on the given type selector. In particular it can be used to yield the network *cell identifier* and *location area information* enabling the rough localization of the user's current position.

— SendShortMessage([*title*],*dest*,*payload*): Sends a short message with the given payload to the destination.

Figure 6.2 on the next page gives an overview of the SIM toolkit architecture. The lower part shows the APDU interface to the mobile phone (MS). The middle block represents the SIM API framework that contains mechanisms for applet triggering, e.g. after receiving a so-called *envelope*. Furthermore, it implements mechanisms for applet installation and de-installation.

Central to the toolkit is the so-called *proactive command manager* which is responsible for managing a *proactive session*. Such a session can be initiated by an applet wishing, for example, to

[2] Although the most widely used implementations of A_3, COMP128, has shown some weaknesses (cf. [BGW98]) the algorithm seems to be reasonably resistant against feasible on-line attacks.

[3] The API [GSM02.19] describes how SAT services can be used from on-card applications. A Java Card binding is specified in [GSM03.19].

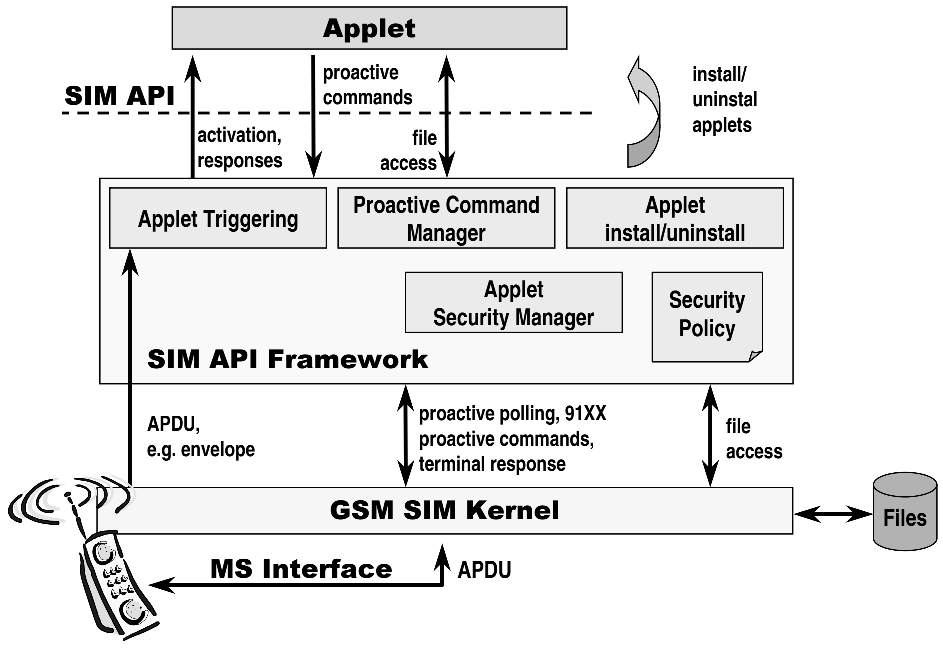

Figure 6.2: SIM application toolkit architecture (SAT) overview

execute the toolkit command SelectItem. The applet invokes the appropriate method in the SIM API that in turn activates the proactive command manager who sends a response APDU to the mobile phone in the form of a status word $(SW_1/SW_2) = (91, len)$. This response code indicates to the MS that the SIM wishes to start a proactive session. The MS then fetches the next command with the given length that contains the proactive command which in our example contains the items the user has to select from. It then decodes the proactive command contained in the response APDU and in case of a SelectItem displays the menu items on the screen. After the user has selected an item, the MS compiles a so-called *terminal response* APDU that contains among other information the number of the item the user has selected. This response is now intercepted by the proactive command manager who in turn resumes the applet and passes the users selection back to the SIM toolkit application.

Besides the commands listed above, the SIM toolkit further supports a number of mechanisms for registering timers that can be used to wake up an applet at regular intervals, registering for certain types of events such as the arrival of a SMS [GSM03.40] or a change in the current network cell. The most important triggering mechanisms are the arrival of a SMS and the selection of the applet in the phones SIM-specific menu.

Summing up, the SIM application toolkit allows to temporarily exchange the role of client (which is now the SIM) and server (which is the MS offering services to the SIM). It can be seen as a platform on top of which card-resident applications can be implemented that have access to an API that allows to perform user interaction and communication.

6.4 The SIM as a Security Module in the Internet

As mentioned previously, the GSM system can be characterized as the largest security infrastructure the world has seen so far. Interestingly enough this infrastructure has been set up for the sole purpose of making phone calls with. The central theme of this chapter is to exploit this existing pervasively distributed infrastructure to provide additional security services.

6.4.1 Security Status Quo in the Internet

A security infrastructure comparable to GSM is still missing in today's Internet. The most successful security infrastructure currently available is the SSL/TLS suite of protocols [RFC2246] in combination with a public key infrastructure (PKI). TLS allows for peer-to-peer authentication of clients and servers by means of certificates and is used to securely transmit a shared secret from the client to the server on top of which a session key for symmetric encryption of the communication session is derived. Off-the-shelf browsers come with an installed base of root certificates of vendors offering PKI services. A Web site wishing to participate in the PKI buys a certificate from one of those vendors which is signed by the vendor's private root key. The client connecting to a TLS server downloads the server's certificate and checks whether it is signed by one of the locally installed root certificates. It subsequently computes a so-called *pre-master secret* consisting of a random string which is then encrypted with the server's public key stored in the server's certificate and transmitted to the server. Both client and server then compute the so-called *master secret* which is then used to derive further session keys. If both simultaneously switch to encrypted communication, the client can be sure the server is the one as claimed in the certificate.

However, in practice only the server authenticates to the client. Authenticating the client to the server would require that the client is in possession of a certificate which is linked to a public key infrastructure. Today such infrastructures only exist for the Web servers but not for ordinary users. The reasons are manifold:

- Today certificates are still expensive for end users.
- There is no commonly accepted standard for certificates resulting in numerous interoperability problems.
- Reasonably secure approaches should rely on smartcards issued by the vendors to end users which contain the certificate and the private key. Such a solution is expensive and requires the integration of card readers into standard operating systems and the necessary plug-ins for Internet browsers – a task that is still burdensome for inexperienced users.

As a consequence, only in certain niches of the general consumer market client certificates are used, whereas in the enterprise domain they are found much more often.

Account Verification Practices

At the moment best practice is that users register a login and a password at every site they want to have access to. Usually, the registration process is different for each site. Some typical approaches are:

- No further verification of the account data. The service provider simply trusts the client that the submitted data are correct.

- Verification of the user's email address by sending a one-time password that must be used to complete the registration process. A variant of this is that the service provider sends the one-time password by postal service.
- Verification of the account data with the help of another authority, e.g. by requesting the user to present his/her identification card to this authority. This is often required for on-line banking and brokerage services, e.g. in Germany the Deutsche Post offers such a service at its postal offices.

Hence, service providers of more security-sensitive services often rely on other parties such as the postal service to implicitly establish confidence in the correctness of the user's account data.

Bringing the GSM Security Infrastructure into the Internet

The large number of deployed and used SIM cards is based on a contract between the subscriber and the mobile operator.[4] Furthermore, a mobile subscriber already has a unique user name which is his or her mobile phone number. This leads us to the central goal of this chapter, namely

> *to bring the security infrastructure in GSM namely the SIM and the contract between the mobile operator and the subscriber into the Internet,*

or put in another way

> *several hundred million GSM subscribers carry powerful smartcards around in wireless card readers, i.e. mobile phones – why not use these cards to secure Internet transactions?*

Hence, if it could be achieved that transactions of an Internet user can be confirmed with the help of a mobile operator, substantial progress towards Internet security could be achieved.

6.5 WebSIM – A Web Server in a GSM SIM

In this section we describe the WebSIM system that is an implementation of a small stripped-down Web server in a GSM SIM that integrates a user's SIM into the Internet.

6.5.1 Communication Protocols

The general idea of the WebSIM is to allow for a communication channel from the Internet to a subscriber's GSM SIM. Thus, one part of the communication must use standard Internet protocols, whereas the second part must deal with the protocols spoken in the GSM world. Currently, the only practically useful communication path to a SIM is the *short message service* (SMS) [GSM03.40]. As noted in Sect. 6.3, SIM application toolkit applications can be triggered upon arrival of short messages. Furthermore the SIM toolkit allows for SAT applications to send short messages to any destination. The SIM is therefore able to receive requests, process them, and return responses based on SMS as the communication protocol. Missing is still a component bridging the Internet world with the GSM world.

[4] For a moment we do not consider the fact that a substantial number of issued cards are so-called *pre-paid* cards where the operator is not always in the possession of the subscriber's personal information.

		Protocols	
ISO/OSI Layer		*Internet*	*WebSIM*
7	Application	HTTP	embedded HTTP request URL
5	Session	SSL/TLS	WebSIM-specific message frame
4	Transport	TCP/UDP	GSM 03.48 security envelope
3	Network	IP	SMS, i.e. GSM 03.40 message

Figure 6.3: WebSIM communication stack

Since SMS is used as the bearer technology in GSM it must be considered that the *maximum transfer unit* (MTU) of SMS is currently 140 bytes. Furthermore, most of the GSM SIM cards we know of use the GSM security mechanisms for the SIM application toolkit as specified in [GSM03.48] for securely transmitting SMS. This mechanism needs at least 27 bytes for encoding security transmission parameters, key addressing, and payload description, leaving at most 113 bytes.[5] Furthermore, depending on the technical realization in the GSM infrastructure, SMS delivery times range from a few seconds to sometimes several tens of seconds. This is mainly because mobile operators do not yet offer enough time-critical services over SMS. Technically, there is no problem to provide a transmission time of two to five seconds. Anyhow, the number of exchanged messages must be minimized in order to have a reasonable and acceptable immediacy and round-trip time.

In Sect. 4.5 we have briefly described CITI's Web server implementation on a Java Card. The implementation included a partial TCP/IP stack that is responsible for some of the TCP session setup and acknowledgments (cf. [RFC0761]). For each request sent to the card-resident Web server several messages have to be exchanged in order to establish a TCP session and subsequently implement packet transport.

This is clearly not acceptable for an implementation on a GSM SIM due to the number of packets needed to establish a single connection without exchanging any user data. The approach taken in the WebSIM is to define a new protocol that is better suited for transmission on top of SMS. The resulting communication stack is depicted in Fig. 6.3. The underlying network layer comparable to IP is SMS. As transport layer we use the security envelope from [GSM03.48]. In the payload of this envelope the WebSIM message frame containing further information such as magic bytes, the session identifier, and sequence number are encoded. The remaining payload is occupied by the URL of the initial HTTP request from the client.

6.5.2 Architecture

The overall architecture of the WebSIM is shown in Fig. 6.4 on the next page. The core component is the WebSIM *proxy* that is basically a Web server connected to the Internet. The TCP/IP communication is handled by the proxy's network stack. HTTP [RFC1945] or HTTPS can be used to connect from the Internet to the proxy. The proxy understands HTTP requests of the form

GET /+*phone*/st/*cmd*=(*parameters*)

[5] Other implementations might leave more, but the MTU is always bound to at most 140 bytes.

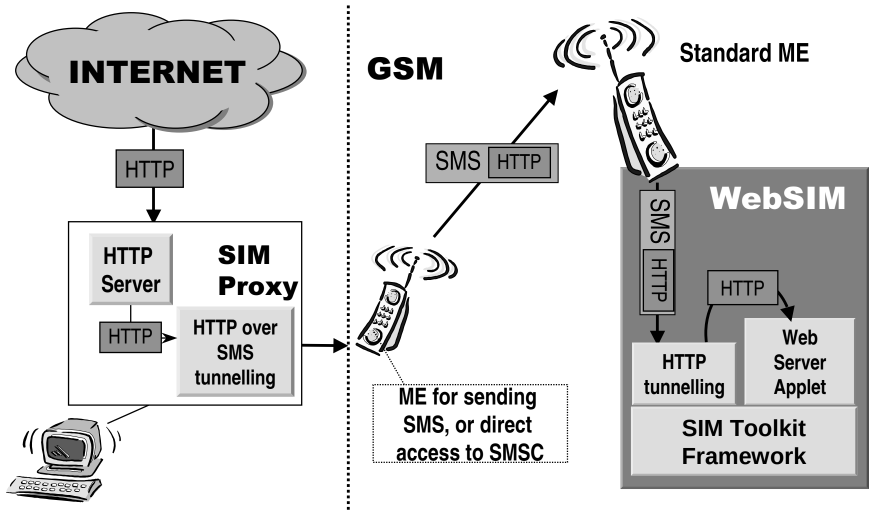

Figure 6.4: WebSIM architecture overview

The URL is prefixed by the phone number of the destination SIM. The proxy only extracts the request URL omitting other HTTP header fields such as `User-Agent`, `Authorization`, etc. which are not used by the WebSIM. The substring "`/st/`" stands for "SIM toolkit" and prefixes the specific commands implemented in the WebSIM. This request is embedded into a short message conforming to [GSM03.40] and sent to the destination SIM.

Upon arrival in the mobile phone, the SMS is immediately delivered to the SIM due to certain type information in the SMS header. The SIM passes the SMS to the Web server Java Card applet and launches it. The applet parses the URL in the request and triggers any commands encoded. Some of the commands invoke SIM toolkit commands starting a toolkit session with the mobile phone. After the session has finished, e.g. the user has typed in a number or selected an item from a menu, the response is sent back as an SMS to the proxy. A separate process running on the proxy checks the incoming messages at regular intervals and decodes newly arriving SMS. Based on the header information in the SMS it is able to correctly forward the response from the SIM to the pending HTTP request and thus it sends the response back to the Internet client.

6.5.3 Command Set

The SIM-resident stripped-down Web server does only support the HTTP `GET` request. It implements the commands listed in Tab. 6.1 on the following page. Most of the commands are based on the available commands in the SIM toolkit (cf. Sect. 6.3). Sample user sessions based on the SelectItem command are shown in Fig. 6.5 on the next page.

The `sign` command has been implemented to allow for the creation of electronic signatures on the SIM. Due to the lack of a SIM capable of running cryptographic algorithms we have used a simple XOR algorithm to just simulate the presence of strong cryptography.

URL	*Description*
`/dt=`(*text to display*)	Invokes the SAT command DisplayText to display the given text on the mobile phone. No response is returned to the client.
`/gi=`(*title*)	Presents an alphanumeric input form to the user based on the SAT command GetInput titled with the given title. The text entered by the user is returned as the HTTP response.
`/gn=`(*title*)	Similar to `/gi`, but accepts only numeric input.
`/gp=`(*title*)	Similar to `/gn`, but display stars '$*$' while typing. Convenient for entering passwords or PINs.
`/sit=`(*title*, $item_1, \ldots, item_n$)	Displays a menu with the given items by invoking the SAT SelectItem command. The user can select one of the items that is then returned back to the Internet client.
`/sign=`(*text*)	Displays text and asks the user for confirmation, whether it should be electronically signed. The signed data is returned to the client.
`/info`	Returns information about the current location of the mobile phone. It uses the SAT ProvideLocalInformation command to query the mobile for the *mobile country code* (MCC), *mobile network code* (MNC), *location area information* (LAI), and *cell identity*.

Table 6.1: WebSIM command set supported in the URL interface

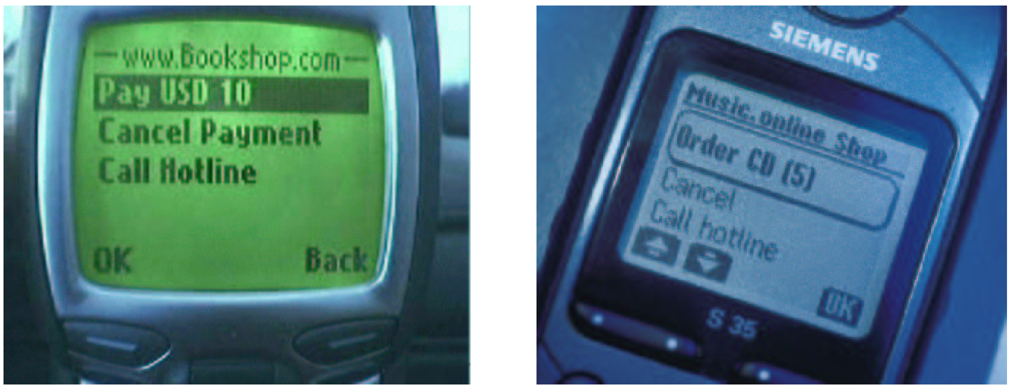

Figure 6.5: Examples of WebSIM user interaction. The pictures show how the SelectItem command is used to perform user interaction in an m-commerce setting.

6.5.4 Implementation

The implementation of the WebSIM comprises the following components:

— **Proxy:** The proxy is a laptop running Linux directly connected to the Internet.[6] It runs an Apache Web server that is configured to launch a Perl CGI script when a URL of the pattern */+phone/…* is requested. The script extracts the HTTP `GET` request and spools the request into a SMS spooling system.

— **SMS spooler:** The spooling system is implemented in Perl and is responsible for encoding and decoding short messages conforming to the [GSM03.40] and [GSM03.48] standards. Attached to the laptop via a serial link is a mobile phone implementing the AT command sets [GSM07.05; GSM07.07] for sending and receiving short messages via a mobile phone.[7] The spooler uses these commands to send the SMS to the destination phone and perform a blocking-wait for the response that is returned to the waiting CGI script upon reception.

— **Mobile phone:** Any mobile phone compliant to Phase 2+ of the GSM standards implementing the SIM toolkit can potentially be used as terminal device.

— **GSM SIM:** The SIM is a Schlumberger Simera™ GSM SIM [SLB00] implementing Java Card 2.0 and the card-side portion of SIM toolkit. It has 32 kB of EEPROM of which roughly 23 kB are free for applets. Triggering of applets occurs via the [GSM03.48] *toolkit application reference* (TAR) fields, i.e. that the SMS contains information about the *application identifier* (AID) of the Web server applet processing the embedded request.

— **Web server applet:** The Web server applet has a card-resident size of roughly 10 kB. No space and performance optimizations have been implemented yet.

The WebSIM system went on-line in February 2000 and was fully operational until December 2001.

6.6 Security Analysis

In the previous sections we described the WebSIM architecture and its implementation based on the core components proxy and SIM-resident Web server. This section focuses on the security properties *integrity*, *confidentiality*, *authenticity*, and *non-repudiation* of the WebSIM approach in general and our particular implementation.

Integrity and Confidentiality

The WebSIM system consists of basically three communication links that handle the integrity and confidentiality of the exchanged data:

1. **Client → Proxy:** HTTP or HTTPS is used as the communication protocol between client and proxy. HTTP which is implemented over a TCP connection does not offer means for integrity and confidentiality. HTTPS which is HTTP over SSL/TLS can guarantee both integrity and confidentiality provided by the underlying TLS layer.

[6] The server is connected to Deutsche Telekom Research Department (DTRD); DNS host name: `websim.dtrd.de`.

[7] Another option would be to directly connect to a mobile operators' *short message service center* (SMSC) which is the central SMS *store-and-forward* node in an operator's infrastructure.

2. **Proxy → ME:** SMS are transferred over the GSM signaling channel which is encrypted by the A_5 encryption algorithm. Although attacks against this algorithm have been published in the past, such attacks seem not to be feasible for on-line exploitation. However, integrity of the signaling channel is not implemented in GSM at the moment, but will be available in future UMTS systems. Furthermore, there exist attacks where the complete encryption between the ME and the *base transceiver station* (BTS) is turned off using a so-called *IMSI catcher* (cf. [Fox97]).[8] Hence, this channel can be considered secure only to a certain degree.

   The general solution to this problem is to protect the SMS with a *security envelope* such as standardized in [GSM03.48] that provides confidentiality and integrity of the message. Furthermore, this standards defines replay counters to protect from replay attacks.

3. **ME → SIM:** The communication between the ME and the SIM is done over the built-in card reader and not encrypted or protected otherwise.

4. **SIM ↔ ME (SAT session):** The communication link between the SIM and the mobile phone during the SIM toolkit session is unprotected, i.e. both devices are physically co-located. This is the reason, why the WebSIM approach does not fall into the remote communication category identified in Sect. 3.5.

Thus, the WebSIM system guarantees under normal circumstances “node-by-node” confidential transmission of data. Integrity is not achieved on the over-the-air link. Furthermore, there exist attacks, although hard to mount, that allow potential eavesdroppers to intercept the over-the-air communication. With additional security measures such as the secure SMS envelopes, however, a reasonably secure system for many applications can be obtained.

Authenticity

Authenticity of the exchanged messages is the subject of the following analysis:

1. **Client → Proxy:** Authenticity of the proxy and the client could potentially be provided again by TLS. Here, server authenticity is of major importance to the client. Client authentication could be necessary for the server in order to implement access control mechanisms, e.g. for billing purposes.

2. **Proxy → SIM:** SMS are exchanged via a short message service center (SMSC) which is the store-and-forward node in a mobile operator’s infrastructure. Today, many service providers have direct connection to an SMSC for sending and receiving SMS. Since some of the systems are allowed to send arbitrary messages to destination SIMs, the *originating address* (OA) of a SMS cannot be used to prove the authenticity of the sender of a message, i.e. the SIM cannot be sure that the proxy sent the request. This problem can equally well be solved by using the security envelope of [GSM03.48].

Summing up, authenticity is not guaranteed on the over-the-air by default. However, security envelopes can be used to overcome this drawback. Nonetheless, end-to-end authenticity cannot be guaranteed with the WebSIM system as it stands.

Non-Repudiation

Non-repudiation on the SIM-side is available with the `sign` command only. No public key signature is available due to the lack of a smartcard capable of public key routines. Client-side non-repudiation is not supported. Further non-repudiation issues are discussed in Chapter 7.

[8] Such systems are available for roughly € 20k.

Summary

The WebSIM as it is, does not provide end-to-end security between the service provider and the SIM. This results from the fact that the proxy has to perform a protocol translation from the Internet to the GSM world back and forth. Hence, the architecture shares some similarity with the WAP 1.2 security model (cf. [WAP98b; WAP99b]) which has the WAP gateway as its protocol translation engine in between.

Although the underlying GSM system has certain weaknesses, a secure channel between the Internet client and the proxy on the one hand, and between the proxy and the SIM on the other hand can be established resulting in a reasonably secure architecture for many application domains.

6.7 Applications

The previous section analyzed the security context of the WebSIM which is the technological basis of the sample applications presented in this section.

6.7.1 WebSIM-based Authentication in the Internet

Internet service or application providers such as online book stores, Web shops, or banks need secure identification of customers, i.e. authentication. Today, on-line orders are usually placed via Web forms or call centers and authentication takes place in various forms, e.g. using password-based authentication schemes.

In the course of this section S is a service provider, P is the WebSIM proxy, W is the WebSIM, M is the mobile operator of P, resp. W, and U is the user.

GSM-based Authentication

Involving the WebSIM into the authentication of Internet users allows for more elegant solutions that can take advantage of secure cryptographic keys, like the subscriber's individual key *Ki* based on the following protocol:

$$\begin{array}{lrl}
\text{a)} & S \rightarrow W: & \mathit{RAND} \\
\text{b)} & W: & \mathsf{SelectItem(compute?,yes,no)} \vDash \mathit{SRES} = f(\mathit{Ki},\mathit{RAND}) \\
\text{c)} & W \rightarrow S \rightarrow M: & \{\mathit{RAND},\mathit{SRES}\} \\
\text{d)} & M: & \mathit{SRES'} = f(\mathit{Ki},\mathit{RAND}) \\
\text{e)} & M \rightarrow S: & \mathit{SRES} = \mathit{SRES'}\ ?
\end{array} \tag{6.1}$$

a) An Internet application requiring a user authentication sends a random challenge *RAND* to a server-side application within the WebSIM.

b) The WebSIM server-side application asks the subscriber via SIM toolkit to authorize the computation of a response $f(\mathit{Ki},\mathit{RAND})$.

c) Then $f(\mathit{Ki},\mathit{RAND})$ is returned to the originator of the request.

d) The ISP passes *RAND* and $f(\mathit{Ki},\mathit{RAND})$ to the card issuer (resp. the party that knows *Ki* and f) who can verify the result.

e) The result of the verification is sent back to the service provider.

This is a classical challenge/response authentication protocol that can be applied to many other scenarios, e.g. home banking and access control. Analogously it can be adapted to provide, for instance, a session key for other purposes. The function f can be based upon encryption algorithms (like Triple-DES or RSA) or hash functions (such as SHA-1 [FIPS95] or MD5 [RFC1321]) and keys other than *Ki*.

The scenario can also be extended to sign transactions (like on-line payments). Here it might be of importance, that the incoming SMS that carries the HTTP-request contains a (reasonably) trustworthy time stamp originating from the short message center (SMSC) that was involved. Furthermore, subscriber-individual IDs like the IMSI are available in the SIM.

Provision of One-Time Passwords

One-time passwords for login procedures or *transaction numbers* (TAN) for bank transactions can be queried over WebSIM requests. We consider on-line shopping as an example:

$$\begin{array}{lrl} \text{a)} & U \rightarrow S: & \textit{Pho} \\ \text{b)} & S \rightarrow U: & \textit{RAND} \\ \text{c)} & S \rightarrow W: & \mathsf{GetNumber} \\ \text{d)} & W \rightarrow S: & \textit{RAND'} \\ \text{e)} & S: & \textit{RAND} = \textit{RAND'}\ ? \end{array} \tag{6.2}$$

a) A user subscribes to a service on the Internet and tells her mobile phone number.

b) The user compiles a shopping list in an Internet shop and orders by submitting the list. The Internet shop's Web server sends a one-time password via a Web page to the customer.

c) Simultaneously, the Web server issues a WebSIM request to the customer's SIM asking to enter the one time password on the mobile phone. The WebSIM issues an appropriate SAT command such as GetNumber to the mobile phone, prompting the user for the one-time password.

d) The user enters the one-time password, that is sent back to the Web server, possibly over an encrypted communication channel.

e) The Web server of the shop checks whether the entered one-time password is correct, and if so, it acknowledges the purchase.

The advantage is that an authentic channel (GSM) is used to verify the identity of the customer. Another, reversed variant of this might be as follows:

$$\begin{array}{lrl} \text{b)} & S \rightarrow U: & \text{send Web form} \\ \text{c)} & S \rightarrow W: & \mathsf{DisplayText}(\textit{RAND}) \\ \text{d)} & U \rightarrow S: & \textit{RAND'} \\ \text{e)} & S: & \textit{RAND} = \textit{RAND'}\ ? \end{array} \tag{6.3}$$

b) After submitting the shopping list, the Web server generates an input form where the user has to enter a one-time password.

c) The server sends the one-time password to the WebSIM, that is displayed on the mobile phone.

d) The user enters the one-time password that is displayed by the WebSIM dialog on the mobile into the Web form and sends it to the server.

e) The Web server checks the one-time password and accepts submission.

Note, that similar one-time password schemes can be implemented by sending a password over a text SMS to a mobile phone user. The difference to a WebSIM-based approach is that text SMS is just a (non-interactive) messaging service that is not protected beyond the standard GSM mechanisms.

6.7.2 Using WebSIMs as I/O Channels

The WebSIM can also be used as a pure I/O channel to a mobile user as follows:

Secure User Interaction

A person holding a mobile phone with a WebSIM is standing in front of an ATM, and calls a telephone number displayed on the ATM. The ATM system knows the Web address of the WebSIM (from CLIP signaling [ETSI00]) and can run several subsequent SAT commands in the WebSIM to authenticate the transaction, choose the amount of cash to be issued, etc. Essentially the GSM phone has become the human interface to the ATM and one can imagine ATMs that do not have complex and expensive human interface hardware but are just a telephone number sign and a cash-dispensing slot.

Analogously, one can implement online payments, access control, ticket vending, etc. Note that cryptographic means to secure the result of the user interaction (the input, or the user's choice) are easily available inside the SIM.

Push-based Internet Applications

The WebSIM allows to implement *push*-based Internet applications that push content and interaction from an Internet node to a mobile user. As an example, consider on-line Internet auctions in which a mobile user participates by regularly checking for newly placed bids. If a *pull*-based model is used, e.g. based on polling the new status using HTTP or the *Wireless Application Protocol* (WAP) [WAP98b], this would not only be annoying for potential users but also slow and expensive. Using the WebSIM one can implement a push-based client that behaves as follows:

a) After registration for a certain item in an auction, a user delegates auction interaction to the WebSIM by providing the mobile phone number to the auction company.

b) Each time a higher bid is placed – other invocation schemes can be easily thought of – the auction sends a request to the WebSIM that informs the mobile user about the currently active highest bid and asks for entering a new, higher bid.

c) The user can then decide to decline or to increase and enter the new bid that is then sent back to the auction house that places the bid.

Together, besides being a personal security module the WebSIM is a communication module with important features such as user interaction, mobility, i.e. anytime and anywhere, and push-enabled.

6.8 Related Work

Recently, numerous systems have been proposed and implemented to offer, e.g. mobile payments and mobile signatures that are considered the enablers of successful m-Commerce. A more general overview of systems in the area of mobile commerce can be found in [Dur99] and [WSZ01]. In this section we present some of the more interesting initiatives that share some similarities with the WebSIM approach although none of them provides a technology layer other applications can be built on top of.

Mobile Operator's OTA Services

Many GSM SIM smartcard manufacturers implement proprietary over-the-air message interfaces in the card's operating system. This interface is usually based on point-to-point short messages that can be used by mobile operators to trigger certain predefined applications on a SIM. Often the messages are protected by an appropriate envelope (cf. [GSM03.48]) and used for remote maintenance of the SIM. Some of the implementations also offer means to trigger certain SAT commands such as SelectItem or GetInput.

The main drawback still is that mobile operators either do not allow for any kind of access from the Internet to a customer's SIM, or they offer this service only in a highly proprietary manner which is in contrast to the Internet "philosophy" presented in this work.

Paybox

Paybox™ [Pay01] is a payment system that is offered in Germany to allow for mobile payments. A service provider, e.g. an Internet shop or a taxi issues a payment request to Paybox which forwards the request to the mobile phone of the user. A voice-box reads the amount of money to be confirmed and the user then enters her Paybox PIN via dial-tone on the mobile's keyboard. This confirms the transaction and Paybox transfers the requested amount to the service provider on behalf of the client.

Currently, Paybox offers only voice-based payment and does not allow for user interaction in the fashion of the WebSIM. Furthermore, it does not benefit from the SIM as a smartcard, e.g. for providing electronically signed documents.

mSign Consortium

The mSign consortium is in the process of standardizing an interface how service providers can obtain electronically signed documents from mobile users to enable secure mobile commerce.[9] The specification [mSi00] says that the consortium

> "[...] *has specified a protocol between a* primary service provider, *e.g. a merchant or a bank, and a* mobile service provider. *The goal is to provide a standardized interface for the primary service provider to request* [electronic, ed.] *signatures from an end-user through a mobile service provider.*"

How the particular electronic signature is obtained from the end user is beyond the scope of the specification and left for implementation by the mobile service provider. A mobile operator could, for example, implement the mSign specification on top of the WebSIM.

mSign distinguishes between three different levels of security of which only the third one has end-to-end security features. The first and second are based on signature creation by the mobile service provider. In both cases the creation of a signature can be considered as *remote-controlled* by the user, hence the user is not in the possession and control of its signing key directly.

The way mSign tries to bring electronic signatures into play is actually constrained by the limitations of the SIMs already deployed in the market. Only recently, SIMs capable of public key encryption are used in the market which would allow for end-to-end digital signatures.

[9] The initiative was lead by Brokat Informationssysteme GmbH which holds a patent [Bro97] covering several aspects of the creation of electronic signatures on mobile phones.

Other Approaches

There exist other applications based on the SIM toolkit offering dedicated solutions for security problems. Examples are products for Single Sign-On applications, dedicated authentication systems for remote login, etc. However, none of the systems and products we know of offers a technology layer comparable to the WebSIM approach.

6.9 Summary

In this chapter a system was presented that integrates the existing GSM security infrastructure deployed in the shape of several hundred million GSM SIM cards into the Internet. The realization basically consists of a component that bridges the gap between the Internet and the GSM world allowing for transparent interaction between an Internet host and a GSM SIM. HTTP was chosen as the connection protocol since it is *de-facto* standard in the Internet today for client/server communication and E-commerce applications. The interface can be used by anybody with reasonable knowledge of Internet technology and hides the complexity of all the lower-level protocols.

To the best of our knowledge the WebSIM is the first attempt to implement a stripped-down Web server in a GSM SIM. Therefore, there is no direct related work except the Webcard server developed at CITI presented in Sect. 4.5. Most likely the reason is that GSM SIMs and the SIM application toolkit are – though an interesting technology – not easily accessible by the academic world, if at all.

Within the overall security framework as presented in Sect. 3.5 the WebSIM shifts more functionality from the terminal to the card, since the SIM uses the mobile phone just for user interaction and communication made available by the SIM application toolkit. The rest of the processing is completely done within the card. The number of application domains this model is applicable to suggests that the idea of a more card-centric view of a security module which uses available terminals for user interaction seems appealing. Open with the WebSIM approach as already mentioned is the problem of end-to-end security and non-repudiation features. These issues are further discussed in the next chapter.

Most of the ideas presented in this chapter are published in [GKP00]. A preliminary version appeared in [GKPV00] and some of the possible application scenarios have been described in [KPSW01]. The system has been in practical use by different research groups of European telecommunication operators participating in the EURESCOM Project P1005 [EUR01] since February 2000.

Chapter 7

Open and Secure Service Platforms for Smartcards

DSLs [Domain Specific Languages, ed.] automatically provide programmers with strong guarantees of correctness, performance, and security which are hard to achieve with general-purpose languages such as C, C++, or Java (think of Yacc and SQL). . .

Thomas Ball, Bell Labs, Lucent Technologies, 1999

7.1 Introduction

In the previous chapter we have shown that integrating GSM SIMs into the Internet can provide a substantial benefit to Internet security both in terms of usability and simplicity. However, one of the major open problems not solved by the WebSIM approach is the lack of

- end-to-end security mechanisms, and
- non-repudiation.

In this chapter we propose an architecture based on an open service platform for smartcards that allows for the execution of mobile code written in a domain-specific language. The approach offers substantial advantages in the way service providers might interact with customers or employees in the foreseeable future. Furthermore, the security-critical process of creating electronic signatures on mobile terminals can be improved by shifting most of the critical parts involved in the signature creation process into a smartcard with the additional benefit of improved usability.

The WebSIM Revisited

The WebSIM as introduced in the previous chapter basically allows for running a synchronous interaction in a client/server fashion between an Internet client and a mobile user.

- The protocol is *synchronous* due to the HTTP communication based on request and response pairs. This is somewhat problematic, if latency on the SMS link is high.
- It allows for invocation of at most one SIM toolkit command at a time. However, simple sequencing of commands could be added rather easily.

- There is no *end-to-end* security between the Internet client and the SIM, since the WebSIM proxy has to recode the request in order to forward it to the SIM and vice versa. This problem is comparable to the end-to-end security breach known from the WAP 1.2 gateway architecture [WAP98a].
- As a consequence, *atomicity* of a sequence of user interactions is not provided.
- The WebSIM does not provide *persistent storage* on the card for potential services, e.g. loyalty points applications, or electronic ticketing.

Hence, the WebSIM is quite useful in a number of application scenarios but has certain drawbacks that should be corrected in order to become a flexible security module for Internet use.

Towards an Open and Secure Smartcard Platform

To overcome the limitations enumerated previously we propose to implement an open and secure platform for the execution of mobile code in a smartcard that functions as follows:

- The smartcard implements an interpreter for mobile code written in a domain-specific language optimally supporting the intended application domain.
- An Internet client sends either synchronous requests or asynchronous messages containing so-called *scripts* written in the domain-specific language the card-resident interpreter understands.
- The card's runtime platform executes the script, handles user interaction, and sends back the responses (synchronous- or asynchronously) to the client.
- The platform implements key management facilities in order to provide end-to-end security between the Internet client and the smartcard.

The platform is *open* in the sense that virtually anybody on the Internet should be able to send such scripts to a mobile user.

The platform is *secure* in the sense that neither the mobile code nor the user is able to harm the platform's integrity. Furthermore, the platform gives certain guarantees to both – code and user – that the scripts are executed as intended and no information leakage or secret storage manipulation can occur by malicious code or an external attacker.

Thus, the platform acts as a *trusted computing base* running in a tamper-resistant device protecting the user from the code and vice versa. As such it implements all three types of trust – *personal agent trust*, *captured agent trust*, and *undercover agent trust* – as defined by [PPSW96] (cf. Sect. 5.3.4).

Hence, this chapter discusses approach A4 as introduced in Sect. 3.5 focusing on mobility, and card-driven control of applications. It provides solutions to the above mentioned problems and gives new insights into the way smartcards can be used as active components in security infrastructures.

The rest of this chapter is organized as follows: Section 7.2 describes the general principles of a trusted computing base and gives a brief introduction into hardware and software protection approaches used to yield such a trusted computing base.

The overall *architecture* of our smartcard platform is presented in Sect. 7.3. It describes the platform's components and the platform's role and trust model, thus introducing the fundamental ideas of our approach.

Section 7.4 describes in more detail the programming model and language that underlies our platform. More concretely, it describes the chosen *domain-specific approach*, the machine model, and gives a taste of programming with such a platform.

Security issues of our platform are raised in Sect. 7.5 where the pertinent literature on *host security* and *code security* is revisited. This allows to characterize our approach in the design space defined by the criteria found in the literature. Furthermore, the core design decisions w.r.t. security issues in our platform are motivated and their solutions are presented.

Section 7.6 concentrates on the communication protocols between a potential client and the platform. Since end-to-end security is a desirable feature, its impacts on the platform, i.e. its interpreter, code, and data verifier are closely examined. Based on the results different cryptographic protocols are presented how end-to-end secure communication can be achieved and how key management is solved in these approaches.

Non-repudiation and the creation of electronic signatures are extensively elaborated in Sect. 7.7. The results are five different protocols how non-repudiation can be implemented with such a platform, what security risks still exist, and which benefits the solutions give over traditional electronic signature creation protocols.

Section 7.8 presents a number of sample application scenarios if the platform approach is realized in combination with a wireless terminal.

Related work is discussed in Sect. 7.9 and the chapter is finished with a summary in Sect. 7.10.

Appendix A presents more technical information on the prototypical implementation undertaken, a brief listing of the platform's language primitives, and the language grammar.

7.2 Principles and Concepts of Trusted Platforms

The *Orange Book* [DoD85, Sect. 6.3] defines a *trusted computing base* as follows:

> "[...] *The heart of a trusted computer system is the Trusted Computing Base (TCB) which contains all of the elements of the system responsible for supporting the security policy and supporting the isolation of objects (code and data) on which the protection is based.*
>
> [...] *Thus, the TCB includes hardware, firmware, and software critical to protection and must be designed and implemented such that system elements excluded from it need not be trusted to maintain protection.*
>
> [...] *The TCB will necessarily include all those portions of the operating system and application software essential to the support of the policy. Note that, as the amount of code in the TCB increases, it becomes harder to be confident that the TCB enforces the reference monitor requirements under all circumstances.*"

A trusted computing base is usually implemented using a mix of the following approaches:

— **Hardware protection:** This can obviously be achieved by physically restricting the access to the device and the use of tamper-resistant hardware.

— **Software protection:** This includes e.g. operating system support and language-based protection.

In the sequel we briefly discuss examples of each of these categories.

7.2.1 Hardware Protection based on Tamper-Resistant Hardware

A trusted computing base that depends on the tamper-resistance of the hardware it is running on has been described extensively in Yee's thesis [Yee94] on *secure coprocessors* and subsequent work published be Yee *et al.* in [YT95]. The basic idea is that the data available in the tamper-resistant security module is not directly accessible but protected by the applications running in the module. Thus, the underlying assumption is that the hardware operates in a relatively untrusted environment and must be appropriately protected. Yee *et al.* give a number of possible application areas for such devices, in particular these are *copy protection for software*, *electronic cash* (monetary values are kept in the module), *electronic contracts* (allowing for advanced reselling mechanisms for electronic goods), and *secure postage* (electronic stamps).

Smith [Smi96] further categorizes different architectures that consist of a secure data storage in the trusted computing base and an off-processor host into five different flavours as follows:

— **Generalized access** which uses the computational ability of the coprocessor to enforce non-trivial access rules;

— **Generalized revelation** which uses the computational ability of the coprocessor to transform the accessed data, depending on the context of the access;

— **Autonomous auditing** which uses the computational ability of the coprocessor to autonomously extract and archive audit data;

— **Trusted execution** application model which uses the computational ability of the coprocessor to carry out execution without manipulation by an adversary; and

— **Hidden execution** application model which uses the computational ability of the coprocessor to carry out execution whose details are hidden from an adversary.

Usually more than one style is applied in a particular system, e.g. the German "Geldkarte" (*cash card*, cf. [Gen99]) uses generalized access and revelation for controlling access and modification of the electronic cash stored in the card. Furthermore trusted and hidden execution are used to avoid the observation and manipulation of the algorithms manipulating the amount of electronic cash available on the card.

Typical examples of tamper-resistant hardware are smartcards. Other systems based on PC plug-in cards have been described in [Yee94].

7.2.2 Software Protection

After discussing hardware protection, we briefly give an overview of secure operating systems and language-based protection of which the latter will be used as a basis for the design of the smartcard platform.

Secure Operating Systems

Many implementations of a trusted computing base build upon the security of the underlying operating system. Systems such as MULTICS [CV65] have been designed according to particular security demands. In contrast to tamper-resistant hardware, however, the assumption is that the environment the system is hosted by is much more trusted, and physical access to the system is usually not given. Hence, the operating systems is often designed to survive attacks from a network or – in the context of a multi-user system – from its users. In the sequel we briefly describe examples of systems aiming at a trusted operating system using different and somehow complementary approaches.

Shapiro *et al.* have implemented a *capability-based* operating system called EROS [SSF99]. In EROS a capability is a pair (*object identifier*, {*authorized operations...*}), i.e. access to each object in the operating system such as *processes*, *nodes*, and *pages* are controlled by such capabilities. Applications execute within so-called *protection domains* that are essentially composed of sets of capabilities to which these applications have access. Although an implementation of EROS on current mobile devices is not feasible, it demonstrates that low-level security in operating systems might be an interesting approach to build trusted operating systems on top of which trusted applications can be implemented.

Stüble *et al.* [Stü00; PRS$^+$01] (cf. Sect. 5.3.4) are currently implementing a secure operating system called PERSEUS based on a micro-kernel architecture. This operating system can be used as a trusted layer on top of which different "untrusted" components, e.g. commercial off-the-shelf operating systems for mobile devices, can be run. Their approach is primarily motivated by the pragmatic observation that new solutions cannot be built without considering market facts.

Besides pure research projects today also a number of commercial systems exist that either provide their own operating system or offer add-ons and modifications to commercial off-the-shelf products. One such system is the PITBULL operating system add-on by *Argus Systems* that is available for LINUX and *Sun Microsystems's* SOLARIS 8. Essentially, PITBULL comprises patches to the COTS operating system that modify kernel data structures, file system layout, and system utilities to implement fine-grained access control and protection mechanisms. It further defines so-called *compartments* that implement new protection domains in the operating system to further control communication between processes and resources.

The Trusted Computing Platform Alliance (TCPA) [TCPA00] is an industry initiative defining a so-called *trusted subsystem* of a computer system. This is achieved by a *trusted platform module* (TPM) based on a secure coprocessor that is involved in the secure booting of the operating system. During boot it checks the integrity of the BIOS, subsequently the operating system's boot loader and finally the operating system kernel. Essentially, it builds up a cryptographically protected chain of trust into the different components involved in the booting phase. The main purpose of this approach is that the TPM can be externally challenged to verify whether the system is in an expected state after booting. Thus, it can be checked, whether the system has been tampered with, e.g. new software has been installed. This is to gain confidence and establish trust into the integrity of the system.

Summing up, there are several approaches to implement trusted systems: tamper-resistant devices and coprocessors, secure operating system design, secure microkernel systems, compartmentalization, and secure booting with the help of additional hardware. Since these approaches are to a large extent orthogonal, combined usage of these concepts could also be envisioned.

Essentially, the use of tamper-resistant hardware is recommended in environments where the system cannot be physically protected from an adversary. Thus, secure operating systems can only work when a potential attacker is subject to all security measures implemented by the system itself. A tamper-resistant device, though, might give advanced protection even in an hostile environment as discussed in Sect. 3.3 as long as the terminal the device is attached to is reasonably trustworthy.

Language-based Protection

Language-based protection enforces security policies at the level of programming language objects in contrast to the resource-centric protection at the operating system level. As defined in [HvE98], language-based protection

"[...] *rests on the safety of a language's type system* [...]

Type safety means that a program can only perform operations on instances of a type that the language deems sensible for that type. [...]

On top of type-safety, the language must also provide some form of access control for the object [...]"

The Java virtual machine, for example, implements language-based protection using its run-time environment in combination with the *byte-code verifier* (BCV) [LY99]. Both guarantee that the language protection semantics such as object encapsulation are preserved and objects cannot circumvent encapsulation to get access to otherwise protected information.

The basic ideas of language-based protection will be brought into the design of the smartcard platform envisioned.

7.3 Architecture

In the previous section we have briefly examined state-of-the-art in trusted platform design and implementation. For our approach we have identified smartcards already as one component in our security device. Essentially, we are interested in the answer to the question

How much in terms of security can be obtained, if as much functionality as possible is shifted from untrusted components into a trustworthy platform available in a tamper-resistant device?

This section describes a system, its architecture, and properties that is built around this idea.

7.3.1 Components Overview

The overall architecture of our system is depicted in Fig. 7.1 on the following page. We identify the following components:

— **Smartcard:** The smartcard is the tamper-resistant device hosting the platform. It is connected to a terminal that provides input and output facilities to communicate with a user via a trustworthy terminal.

— **Platform:** The card-resident platform is an implementation of a virtual machine responsible for the execution of mobile code and a management unit for persistent storage.

- **Security manager:** The security manager is responsible for the correct decoding of any downloaded code. It checks any electronic signatures, decodes the payload according to the information given in the header, and loads the code into the proper compartment.
- **Service compartments:** The service compartment can be considered as a secure container for code and data. Based on the identity of the code sender the code is put into the appropriate compartment or the so-called *public* compartment otherwise. A compartment contains a service-specific *key ring* that is used to store keys owned by the service provider.
- **Verifier:** The verifier is the component that checks whether all security-relevant properties are satisfied by the code. This is to ensure that no security violations occur during execution and that the code can be safely run.

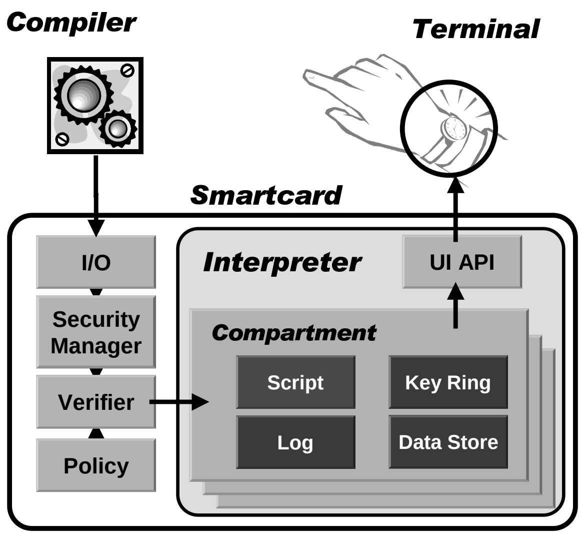

Figure 7.1: Overview of smartcard platform components

- **Interpreter:** After successful verification the code is executed by the interpreter that performs a simple *fetch-decode-execute*-cycle. The interpreter follows a traditional stack- and register-based model and implements an instruction set optimized for the intended application domain.
- **User key ring:** The user key ring contains all relevant secret cryptographic keys and certificates needed for operation.
- **Policy database:** The policy database is owned by the user and contains policies that guide the interpreter at run-time.
- **Audit log:** The audit log is persistent storage that is used to record so-called *traces* generated by the interpreter during execution.

— **Terminal:** The terminal provides trustworthy input and output channels to the user and provides general communication facilities.

— **Compiler:** The compiler is a component outside the security module that generates code for execution by the platform.

Although the general architecture is straightforward, the integration of all the security-relevant components in combination with the components providing for the execution of mobile code into a smartcard that essentially controls the terminal is to our knowledge a novel approach not yet found in the literature.

7.3.2 Role and Trust Model

One of the main issues in an open and secure platform for smartcards is the underlying role and trust model. We can identify at least the following roles:

— **Platform operator/issuer:** After issuance, the platform operator is a trusted third party responsible for the integrity of the platform during operation. Thus the operator guarantees that the platform

- adheres to a particular specification defining its behaviour,
- does not tamper with the code during execution,
- does not leak sensitive data such as cryptographic keys or other confidential data of both, platform user and service provider,
- correctly implements any kind of authentication and access control.

Essentially, the platform provider is responsible for giving guarantees to both – platform user and service provider – that their policies are enforced.

— **Platform user:** The platform user is the owner of the security module and uses it to perform security-critical operations such as electronic signatures or authentication.

— **Service provider:** The provider is the originator of the mobile code sent to the platform for execution.

Essentially, the trust model is as follows:

— **Service provider → Operator:** The provider trusts the operator that the code is correctly executed and cannot be observed during execution. Furthermore, no secret information owned by the provider may leak from the platform, in particular it must be possible to guarantee the confidential transmission of data between the provider and the platform.

— **Platform user → Operator:** Similarly, the user also trusts the operator that the code is correctly executed, no secret information is leaked, and that any operations authorized by the user are performed as desired.

Basically, the platform implements a well-defined intermediate entity between provider and user that both trust.

7.4 Programming Model and Language

The first component of interest in our architecture is the compiler that is responsible for transforming a human-readable input language into a form understandable by the interpreter. The general concepts of compiler construction are well-known (cf. [ASU86]) and numerous compiler construction tools are available today. More interesting, however, is the design of the language itself, an issue that is to be discussed throughout this section.

7.4.1 A Domain-specific Approach

The general idea of the platform is to provide an execution environment for mobile code, resp. script, that is sent from service providers to users. Compared to a general-purpose language the application domain of our language requires at least support for operations such as

- user interaction,
- communication,
- cryptographic operations,
- persistent storage, and
- key management.

Hence, the focus of our language is rather *domain-specific* and concentrates on user interaction and cryptographic operations in contrast to a *general-purpose* programming language.

The very nature of domain-specific programming languages is well expressed in the following quote from Thomas Ball:[1]

> "*Domain-specific languages (DSLs) have had a substantial impact on how software is created, maintained and modified. Prototypical examples of DSLs are YACC, SQL, spreadsheets and HTML. These languages exemplify many of the unique attributes of DSLs:*
>
> 1. *DSLs automatically provide programmers with strong guarantees of correctness, performance, and security which are hard to achieve with general-purpose languages such as C, C++, or Java (think of Yacc and SQL);*
> 2. *DSLs allow non-programmers to program (think of spreadsheets);*
> 3. *DSLs, by providing high-level abstractions tailored to the problem domain, allow programmers with general skills to program in a new domain without having to know platform details (think of HTML and Web services).*"

Using a domain-specific language that optimally supports the intended application domain of running applications in a smartcard seems to be appealing for at least two reasons:

- An efficient interpreter can be implemented more easily for a domain-specific language since any necessary primitives and types can be efficiently encoded into suitable runtime data structures.
- The complexity of the verifier increases with the complexity of the language in which the programs are written. Furthermore, knowledge about the application domain is a necessary precondition to be able to perform a verification process. Hence, a small domain-specific language might be easier to verify than a general purpose language – an issue elaborated further in Section 7.5.

Summing up, a domain-specific approach avoids complexity where possible by concentrating on the necessary functionality. Especially, the implementation of a verifier and an interpreter *on-card* is a challenge that seems to be realizable for a restricted problem domain only.

[1] Taken from the introduction to the Proceedings of the second conference on Domain-Specific Languages (DSL'99), October 3–5, Austin, Texas, ACM Press, page 5.

7.4.2 Virtual Machine Model and Instruction Set

Virtual machines and interpreters are known since the early sixties. The basic concepts are well-understood and our concrete implementation is inspired by two modern interpreted languages – Forth [ISO97] and the Java VM instruction set [LY99].

In this section we just briefly list the minimal set of commands a platform should at least support. A more detailed listing of the commands is given in Appendix A.

Type System

Based on the general functionality needed at least the following data types are mandatory:

- Strings, i.e. concatenated arrays of characters
- Integer numbers
- Booleans
- Cryptographic keys

Hence, no higher data structures are available. Although this seems rather restrictive we have found these primitives to be sufficient for our application domain.

Data Stack

The virtual machine uses a simple stack for computations. Primitive operations essentially operate on the operands available on the stack. Typical operations are `push`, `pop`, `swap`, `dup`, `rot`, etc. as known from Forth. The stack size is fixed and any stack *over*- and *underflow* must be detectable by the verifier prior to execution.

Registers

Registers are used as volatile or persistent storage for intermediate or persistent data. Registers are accessed simply by their address. Primitives operating on registers are `load` and `store`, transferring data from a register onto the stack and vice versa.

Key Store

Keys are sensitive cryptographic data that must be protected from leakage. Therefore, we introduce an elementary type *key*, and operations using keys are specially type-checked to avoid leakage. Keys are organized in a key store that is essentially organized as an array being a container for the keys. Keys can be installed by a service provider to allow for confidentiality and authentication of transmitted scripts. In contrast to registers, the key store is addressed using the primitives `kload` and `kstore`.

Transactions

Atomic updates of persistent data on the card should be possible in general. We follow a straightforward approach for transactions based on logging storage updates from the execution of the primitive `begin` until one of the primitives `commit` or `rollback` is encountered in which case the temporary changes are ultimately committed or discarded. No nested transactions are supported due to the complexity for an on-card implementation and the questionable need of such functionality.

User Interaction

User interaction plays a central role in our personal security module. The card must be able to connect to a suitable terminal as defined by the user in an appropriate policy description and configuration. From a language perspective this is not subject of control, since applications written by a service provider have no knowledge about a user's terminals.

Hence, the platform must offer suitable primitives that allow running scripts to interact with the user through the user's preferred terminal. Although a diversity of user interaction models could be considered, we concentrate on a small set of core user interaction primitives that are necessary to cover most of the possible interactions needed for our application domain and which are implemented by small terminals such as mobile phones. Therefore, we start with the primitives `playtone`, `display`, `input`, and `select` only.

Besides user interaction primitives there exist mechanisms to create a log of user interactions that can be considered as first class objects. This issue is further discussed in Sect. 7.7.6.

Communication Primitives

Besides user interaction primitives, other means for communication are needed. Essentially, facilities for sending messages to other communication partners are needed. Furthermore, is should be possible for scripts to register for the reception of incoming messages. We use the concept of *channels* as a communication link abstraction. Each script has access to a standard channel, that is the link the script was received from. This allows a script to communicate without further knowledge about its environment with its originator using the primitive `response`.

Control Flow Primitives

Obviously, control flow is a central issue in any programming language. Although `goto`'s are considered harmful [Dij68], at the level of a simple stack-oriented programming language they have clear benefits. Hence, the platform supports *unconditional* and *conditional* branches in the form of '`goto` *label*' and '`if` *condition* `goto` *label*'. Program termination is indicated by the primitive `exit`.

Boolean Predicates

Boolean predicates are needed to allow for arbitrary control-flow decisions. The usual set of operators for the different data types are needed such as `and`, `or`, `not`, `eq`, `neq`, `lt`, `gt`, `le`, `ge`, etc.

Arithmetic Primitives

Integer arithmetics can be performed using the standard operations `add`, `sub`, `mul`, `div`, and `mod`.

Summing up, we follow a minimal approach by just adding the core set of primitives that is in our opinion necessary to allow for useful applications. Further extensions are possible, but the overall security of the platform must always be guaranteed.

7.4.3 Programming with a Platform

As intended, the platform presented previously is programmable, i.e. it can be used to execute programs or scripts written in our domain-specific language. Basically, this eliminates the problem that user interaction is restricted to one instruction only as it was the case in our WebSIM implementation. Instead, the outlined set of primitives defines a language suitable for implementing basic user interaction combined with cryptographic operations.

```
 1 script {
 2 
 3    provider "bidbiz.com";
 4    name     "bidbiz␣auction␣client";
 5    id       "20011223/24356";
 6 
 7    implementation {
 8       playtone;
 9       push("News␣from␣bidbiz.com:\n
10             Bid in auction #3576 (Antique watch): EUR 63.");
11       display;
12 
13       push(mark);                       // mark following entries on stack
14       push("Place␣new␣bid?");           // this is the title of the menu
15       push("New␣bid...");               // first item
16       push("Cancel");                   // second item
17       select;                           // pushes number of selected item onto stack
18 
19       push( 2 );                        // number of choice of interest
20       eq?;                              // check for selected item == #2
21       if (true) goto end;               // branch to end if selected "Cancel"
22 
23 enter:
24       push("Enter␣new␣bid␣(>␣EUR␣64):");
25       input:int;                        // require user to enter numeric amount
26       dup();                            // duplicate entered amount
27       push(64);                         // push 64
28       le?;                              // check, if it is less or equal
29       if (true) goto ok:                // then continue
30       playtone;                         // indicate that higher amount is needed
31       push("Please␣enter␣a␣bid␣greater␣than␣EUR␣64.");
32       display;
33       goto enter;                       // repeat input
34 
35 ok:   store(1);                         // store amount in register #1
36       push(mark);                       // push marker on stack
37       push("Action:␣Place␣new␣bid\n");
38       push("Auction:␣#3567\n");
39       push("Bid␣amount:␣EUR␣");
40       load(1);
41       sign();                           // create signature
42 
43       response();                       // send signed data back to bidbiz
44 end:  exit;
45    }
46 }
```

Figure 7.2: Mobile auction client script

Mobile Auction Client

The example in Figure 7.2 is an implementation of a script that notifies mobile auction participants about the current highest bid in an auction and enables them to instantly place a new bid.

A script starts with header information about the name of the script and its provider (lines 3–5). The `implementation` part (line 7) contains the actual program.

Lines 9–10 demonstrate how to display an initial message about the latest news of the online auction. Lines 13–17 show how the arguments for a `select` are pushed onto the stack marked by the initial marker set in line 13. After the selection has been performed the arguments including mark are removed from the stack and the number of the selected item is available on the stack.

Lines 19–21 check, whether the subscriber selected item no. 2 (i.e. “Cancel”) in which case a jump to the label at the end is performed. Otherwise an input dialog is opened in lines 24–25 and the input from the subscriber is returned on the topmost stack position and duplicated in line 26.

Then the entered amount is checked in lines 27–29, whether its is greater than 64. Otherwise a text is displayed in lines 30–32 and execution resumes to the input dialogue.

Finally a return token is compiled onto the stack and the response containing all elements including the amount entered are sent back through the input channel to the script originator.

This example illustrates how programming with our platform actually looks like and user interaction and communication takes place. The next section focuses on the security problems inherited by such a mobile code platform approach and describes appropriate countermeasures.

7.5 Mobile Code Security

According to Carzaniga *et al.* [CPV97] the mobile code model that has been chosen for the smartcard platform belongs to the *remote evaluation* category of mobile code systems that is characterized by a client sending code to a server for evaluation. *Mobile agents* extend this model by integrating state that is migrated to a remote host offering additional means for the agent's itinerary. One of the most problematic issues with mobile code in any form is the overall issue of security which is traditionally characterized by two different objectives:

— **Host security:** The protection of the platform from the mobile code.

— **Code security:** The protection of the mobile code from the platform.

Each of these will now be considered in more detail.

7.5.1 Host Security

A good overview of techniques for implementing host security is given by Fong [Fon98]. Following his categorization host security can be achieved by several different approaches discussed in the sequel.

Discretion

This approach refers "*to the human judgement of the level of trust to be granted to mobile code units.*" Technically, this approach is implemented by electronic signatures as visible in many industrial-strength systems such as Java [KG98; Oak98] or ActiveX [Cha96]. However, the general problems with such *electronically signed code* are twofold:

- First is the semantics of the signature itself which can be formulated also as "*what essentially does the signature state?*" Such an electronic signature is meant to establish trust into the issuer of the mobile code but this process is not only a technical problem as has been discussed earlier in this thesis.
- Second, the access to the signer's key might be compromised.[2] This effectively renders any signature based on this key meaningless.

In the open platform we have in mind code signing is not applicable for at least two reasons:

[2] For example, in March 2001 is was reported [Sla01] that a person was able to "steal" several certificates issued by *Verisign, Inc.* intended for signing code issued by *Microsoft Corp*.

- Potentially, every Internet client should be able to send code to a user's platform. Thus, there would be as many *code signers* as potential clients which also is counter-productive to the user's problem of establishing trust into a signature.
- This problem can be circumvented by a trusted third party that checks the code and issues certificates for any mobile code sent for evaluation. Unfortunately, this breaks any end-to-end security which has been considered as a fundamental requirement of our approach.

Summing up, any form of electronic signature intended to certify the code is not applicable in our system.

Transformation

Transformation relies on the observation that many mobile code languages use different representations for the input language and the shipped code (cf. Java source-code and virtual machine byte-code). Thus shipping the source-code in which no unsafe behaviour can be expressed might have several advantages over shipping a potentially unsafe language.

Transformation then deals with the transformation of the safe language into an executable, but possibly unsafe one, with a suitable compiler under the control of the target platform.

Although appealing, we do not further follow this approach since any form of transformation can be considered as resource-intensive which should be avoided for smartcards.

Arbitration

Arbitration is based on the idea to "[. . .] *protect a host from 'direct' contact with untrusted execution units.*" The most common approach of this kind is *abstract interpretation* found in Java or Safe Tcl [OLW97]. It avoids that critical resources such as memory and communication links can be directly controlled by the executed mobile code. Instead, any computation is done by an interpreter that is able to perform any suitable security checks prior to execution. The *security manager* known from Java is an example of such a security layer.

Clearly, our approach follows the arbitration approach since we use an intermediate byte-code representation that is to be interpreted by the on-card interpreter.

Verification

Verifiers are generally considered as programs that check other programs for particular security properties. In the context of object-oriented languages security properties are often expressed by *object encapsulation* which is often achieved through a suitable *type system*. Thus, *type checking* of programs is vitally necessary to guarantee these security properties. Java uses its byte-code verifier to check code loaded into the JVM for a number of security properties, among others:

— **Stack over- and underflow checks** guarantee that at no point of execution a state is reached in which computation cannot continue due to stack errors.

— **Type-correctness** guarantees that at no point of execution a JVM byte-code instruction finds stack operands of illegal type.

These kind of properties can be checked prior to execution or at runtime. The latter approach usually results in higher execution costs and should generally be avoided.

Other verification approaches, e.g. *proof-carrying code* [NL96] and byte-code verification based on *model checking* [PV98], have been proposed as formal verification techniques, neither of them is currently applicable in the context of smartcards due to resource limitations.

For our platform we think that the verifier approach prior to execution has the general benefit that users are not bothered with applications that fail during execution for reasons that could be checked prior to execution. Thus, a malformed applet is not executed at all and user acceptance is likely to be much higher if script execution cannot fail because of such kind of programming errors.

Besides the stack and typing properties we also consider a problem which is of particular importance with smartcards: *termination*.

7.5.1.1 Verification of Termination Properties

Technically, smartcards operate as servers in a client/server-model. Requests are sent to the card, the card performs its operation, and after computation results are passed back to the client. Unfortunately, smartcards performing a long-running computation are not under the control of an external device. Thus, it is generally unclear in which state the computation is without interrupting the computation. Furthermore, this means that it is not easily possible to interrupt an on-card computation without resetting the card, i.e. to bring it into a well-defined initial state.

Towards a Non-Turing-Complete Language

Resetting the card is not acceptable for our platform and special precautions must be taken to avoid such situations. To solve this problem we start with the observation that it is not necessary that our domain-specific language supports long-running computations based on appropriate control-flow primitives. Instead, as the name suggests, a *script* is much more comparable to a language for advanced user interaction. Thus, constraining the language in a way that it is not *Turing-complete* anymore is worth further discussion.

Considering the *control-flow graph* of a program, any *acyclic* graph obviously leads to *guaranteed termination* as long as all instructions terminate. However, this might be too restrictive for practical use. A relaxation on the other hand could be achieved by taking into account the user interaction occurring during execution. Since we expect an application to perform some kind of user interaction at regular intervals it can be left to the user to decide about termination of a script. Each time the smartcard performs input and output with the user control leaves the smartcard and is given back to the terminal. Thus, at any user interaction taking place a script is potentially interruptible by the user.

Control flow primitives are available with the `goto` statements and `exit` manipulating the control stack. We place the following restrictions on the control-flow graph of an application:

(a) Each possible path of execution in a script must pass an `exit` statement. This ensures proper termination of the program.

(b) The control-flow graph of the application may have cycles, if and only if along each cycle there exists at least one call to an interruptible interaction primitive. Currently, those are the user interaction primitives `display`, `select`, `input`, and `response`. This ensures that an application cannot loop without user's notice.

Figure 7.3 on the next page illustrates the last property. It shows an example of a legal jump (indicated by an arrow) in the control-flow graph. Numbers indicate maximum distance to end of

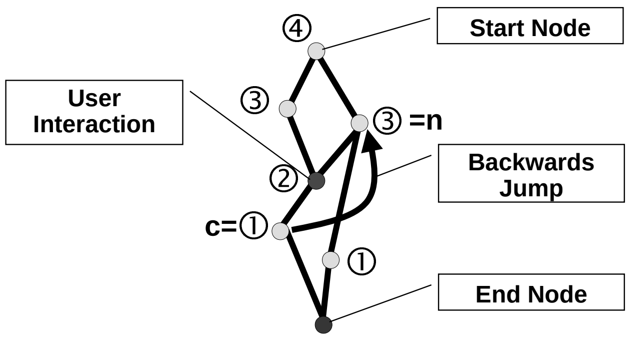

Figure 7.3: Sample platform control flow

application (bottom-most node). Jump of interest is from node *c* to node *n* and is allowed since all possible paths from *n* to *c* pass a user interaction (node 2).

These restrictions yield a constrained domain-specific language that does not allow for, e.g. iteration over a fixed set of numbers without subscriber interaction. Furthermore, there is no support for dynamic data structures such as sets or lists. Despite these limitations we have found the resulting language very flexible for our application domain and we are still looking for interesting examples that cannot be encoded with the current set of language primitives and control flow restrictions and therefore would need further support from the platform.

Rudys *et al.* [RCW01] also consider the problem of termination of possibly untrusted applications. Their *soft termination* approach is based on runtime language support of the mobile code language objects, e.g. *threads*, that is subject to potential termination by an administrator or system resource monitor. They have prototyped an implementation in Java that uses *byte-code instrumentation* to insert additional termination checks into the byte-code. Their approach is generally applicable to our problem domain, but would not contribute any improvements. Further work concerning termination can also be found in the same paper.

7.5.1.2 On Illegal Data Flow

Another issue that is subsumed by the notion of host security is the problem of *information leakage* or *illegal data flow*. This problem has received attention in a number of recent publications [VSI96; Mye99; BCM^{+}00]. However, since scripts run in a safe compartment the only data objects they are in charge of are

— the **script's state**,

— the **persistent state** found in the compartment within the platform at the time of a script's arrival, and

— the **input** received from the user during execution.

Thus, a script cannot obtain any information to which it has no direct access. Further constraints, e.g. controlling whether the security-sensitive information, such as, service provider keys are leaked could be implemented, but does in our opinion not really contribute to the overall security. Basically, the script is free to do whatever it wants with its own information and the information leakage problem does not exist in our domain-specific environment.

7.5.2 Code Security or the Malicious Host Problem

In the previous sections we have extensively discussed the host security issues of mobile code. In contrast, *code security* considers the problem of the security of mobile code from a potentially malicious platform. This problem has been extensively studied in the mobile agents community over the last years and it has been shown to be hard to tackle. Since the literature is exhaustive we only list some of the published approaches:

— **Trustworthy tamper-resistant hardware** such as smartcards have been proposed by Wilhelm *et al.* [WBS98; Wil99], Yee [Yee99], Fünfrocken [Fün99], Pagnia *et al.* [PVGW00], Karjoth [Kar00], and Loureiro *et al.* [LM00] based on the concept of a (mobile agent) platform implemented in tamper-resistant hardware.

— **Encrypted functions** as proposed by Sander and Tschudin [ST98b; ST98a], Loureiro *et al.* [LM99], and Algesheimer *et al.* [ACCK00] operate on encrypted data and produce a result in encrypted form, thus the data is not exposed to the platform. Although encrypted functions are a nice idea this concept is not yet powerful enough to be applied to the kind of applications we consider.

— **Cryptographic traces** have been proposed by Vigna [Vig98] to capture and sign a trace of the execution of a mobile agent on a platform which can be verified afterwards. Such kind of traces will be subject to electronic signature creation described later in the course of this chapter.

— **Code obfuscation** and **encryption** has been proposed by Hohl [Hoh98a] to prevent the inspection of a mobile agent by a malicious host for a certain period of time. However, this approach is also not applicable to our system.

In mobile agent systems the basic scenario is that itinerant agents hop from platform to platform to perform specific tasks such as buying goods, etc. They interact with the platform, often organized as a kind of electronic market, or other agents. In contrast, our scripts are not itinerant and communication occurs between scripts, the platform, and the user only.

Thus, the only code security mechanism that is left is therefore the trustworthy tamper-resistant approach. This means that a script's security could be directly subverted from the following entities:

— The **platform**, but this is eliminated by the fact that our fundamental assumption is that the platform is trustworthy and hence is considered not to be malicious (cf. Sect. 7.3.2).

— The **user**, who has no access to the internals of the platform which is implemented in a tamper-resistant device.

— Some **malicious code**, e.g. viruses that have access to the interface between platform and user.

Thus, the only problem is malicious code that might be able to control the human-device interface. This issue equals the problem of smartcards in hostile environments (cf. Sect. 3.3) and is not directly related to the problem of host and code security.

Nevertheless, this problem is further discussed in Sections 7.6 and 7.7.5 and attacks and possible countermeasures are presented there.

Summing up our approach can be characterized as resting on

- tamper-resistant hardware and an appropriate trust model to solve the malicious host problem, and
- language-based protection and user diligence to solve the malicious code problem.

Hence, both problems have been suitably covered and solutions have been presented.

7.6 Communication Protocols

In the previous sections we have described how the platform is designed according to the mobile code paradigm and what are the fundamental principles on top of which host and code security are built. This section now considers the problem of *end-to-end security* that deals with the cryptographic protocols used for communication and especially discusses the role of the verifier in such scenarios.

7.6.1 Verification and the End-to-End Argument

End-to-end security of a system is commonly understood as a property of the communication channel between the *endpoints* of a multi-tier system. Thus, if a communication channel is established among the nodes $n_1, \dots n_i, \dots, n_m$, then we say that the communication between n_1 and n_m is *end-to-end secure*, if none of the intermediate transport nodes $n_2, \dots, n_{m-1}$ is able to eavesdrop the communication or tamper with messages without being detected. Usually, the intermediate nodes only *transmit* data whereas the endpoints *produce* and *consume* messages.

For the design of a system this means that no intermediate node is able to view or change the contents of the messages without subverting the end-to-end security property. This implicitly means that any architecture providing end-to-end security cannot use *content-recoding* at some intermediate node. Both, the WebSIM approach presented in the previous chapter and the WAP 1.2 architecture relies on some “gateway” to perform recodings. Such encodings on the other hand can be performed on basically two different categories of information a program consists of: the *code* and the *data* on which the code operates.

This implies, that from an architectural point of view we have to consider the following components:

— **Compiler:** The compiler C compiles programs into a code section C and data section D.

— **Code verifier:** A code verifier Vc is responsible for verifying that the code section of a program matches the needed properties, e.g. stack behaviour. Thus, Vc is basically a predicate $v_C(C)$. In this case the code C contains references to data elements in the corresponding data section D and a natural outcome of the type-checking process is some kind of *typing predicate* t that checks, whether the data section matches the requirements from the code to be valid.

— **Data verifier:** A data verifier Vd checks that the data matches the code, e.g. checks for proper data typing and usage. Essentially, this means that besides checking for the integrity of the data it needs to know the typing predicate t to check, whether D satisfies t or not, i.e. Vd is a predicate $v_D(D, t)$ that among others computes $t(D)$.

Table 7.1 on the following page illustrates the most interesting placements for compiler C and the verifiers Vc and Vd.

	(1)	(2)	(3)	(4)
Provider	–	C	C	C
Operator	C,Vc,Vd	Vc,Vd	Vc	–
Platform	–	–	Vd	Vc,Vd

Table 7.1: Compiler and verifier placement

Basically, we have the following placement options:

(**1**) Here, the operator hosts and operates all components. Hence, service provider and platform have to trust the operator and no end-to-end security features are available.

(**2**) This option does not change the situation from a security perspective since the verifiers hosted by the operator have unrestricted access to the code to be verified.

(**3**) *Data confidentiality* can be achieved if the compiler generates an encrypted data pool that is only verified by Vd in the platform after decryption of the end-to-end securely transmitted data has taken place. This is a general improvement over options (**1**) and (**2**) if the data portion of a script is considered to be much more security-sensitive than the corresponding program portion.[3]

(**4**) *Code and data confidentiality* can be achieved if not only the data verifier Vd is placed in the platform as in option (**3**), but also the code verifier Vc. Such an approach is necessary, if the program itself, i.e. the fact that something is reported at all, might be of high confidentiality.

Obviously, the placement of the two verifiers for code and data are critical for the overall end-to-end security guarantees. Depending on the desired level of confidentiality, option (**3**) or even (**4**) must be considered.

7.6.2 Communication Protocols

In the previous section we have investigated the different options of placing a verifier for code and data at particular locations. Consequently, appropriate protocols have to be considered to provide the necessary end-to-end security for code and/or data.

End-to-end security requires that messages are encrypted either by a symmetric encryption using a *shared secret*, e.g. DES, Triple-DES, IDEA, etc., or public key cryptography, e.g. using RSA. Generally, it makes sense to agree on a shared secret to avoid the necessity to extensively use computationally expensive public key algorithms. Several approaches to the exchange of a shared secret could be considered:

(a) Each platform C owns a *platform public key pair* (S_C, P_C). This can be used as follows:

- Transmit a persistent shared secret from the service provider to the platform using the public key S_C. This shared secret would then be used for all subsequent communication.

[3] We argue that this assumption might be true for a reasonable number of applications. Consider for example sales figures reported to the CEO of a global company. Here the fact that a sales figure is reported in the form of a script is of less importance than the actual sales figure numbers.

- Transmit a temporary shared secret for every transmission of a script implying each time the use of public key encryption.

(b) Use another secure communication channel between the service provider and the platform. This could easily be achieved with the help of the platform user as an intermediate node.

A scenario would be that the user is running a Web session over HTTPS with the service provider's Web site S. This session is used to securely transmit a secret to the user U who then types in the secret using a generic application on the platform that transforms the key into an appropriate cryptographic key. For a key of 128 bits of size one could use an alphabet that comprises the symbols $\{$`A–Z,0–9`$\}$, i.e. roughly 5 bits per symbol. Thus a secret of about 25 characters of length could be used to transfer a highly secure key. A service provider then can decide upon the necessary security level by transmitting a shorter secret, if desired.

After deciding the general way how key exchange should be performed the resulting protocols can be developed. We consider both cases in more detail and especially discuss *authenticity*, *confidentiality*, and *integrity*.

7.6.2.1 Public Key Communication Protocols

In the sequel we describe, how communication protocols based on public keys can be used to establish secure communication between service provider and platform.

Public Key Pair

Each platform C owns a secret K shared with the platform operator, an identity id_S, and a public key *platform key pair* (S_C, P_C). This key pair can be installed in the platform at time of issuance by operator M, or later by on-card key generation. In the latter case a user can trigger the key generation and the mobile operator could use the shared secret K, and a random value n to securely obtain the public key P_C from the card and at the same time verify its authenticity as follows:

$$C \rightarrow M : \quad sig_K(id_C, P_C, n). \tag{7.1}$$

The platform operator makes the public key P_C of the platform available as certificate in a central directory, e.g. Web or LDAP server. Hence, the directory provides a mapping $id_c \rightarrow P_C$ from the platform identity id_C to the platform's public key P_C.

Secret Key Installation

The platform offers each service provider S mechanisms to install a secret symmetric key K_S into the platform. Each provider generates a unique key for each individual platform. This would enable confidentiality of communication $S \rightarrow C$ but is not sufficient to prove the authenticity of the sender S that installs a secret key K_S. In the Internet this is basically solved with the use of server certificates used in SSL, i.e. the client verifies that the sender is actually in possession of the secret key belonging to the certificate of the sender. Since the verification process is a rather complex process (challenge-response authentication, certificate revocation list checking, etc.) for practical reasons it should be performed off-card. This implies that another trusted party performs the authentication of the service provider at secret key installation time. In the following protocols we follow this argument and embrace the platform operator M into the installation process.[4]

$$\begin{array}{rl} S \to M: & \big\{id_C, enc_{P_C}(K_S, k, i, n)\big\}, \\ M \to C: & \big\{enc_{P_C}(K_S, k, i, n), id_S, m, \Sigma\big\}, \\ & \text{with } \Sigma = sig_K\big(enc_{P_C}(K_S, k, i, n), id_S, m\big). \end{array} \tag{7.2}$$

First, the service provider sends to the operator the identifier id_C of the target platform, the secret key K_S along with a sequence number i for replay prevention, a key store number k, and a random value n, all encrypted with P_C. The operator completes the message with the identity id_S of the service provider obtained from running a suitable challenge-response protocol with S and a random value m, and signs it appropriately with the shared secret K. Here, id_S could contain some information about the service provider, e.g. its name, address, etc. for information purposes. The message and its signature are then sent to the platform and after signature verification, K_S can be safely stored into the service provider's key store at position k. If necessary, a new compartment for is S is created.

This protocol enables a service provider to upload shared secrets K_S into the platform using public key cryptography. General assumption is that the operator or platform provider has no means to access the service provider's secret keys K_S or the secret key S_C of the card. Otherwise, the whole chain of trust would be compromised.

Secure End-to-End Communication

Upon successful installation of the service provider's key K_S, the service provider communicates with the platform as follows:

$$S \to C: \quad \{id_S, k, enc_{K_S}(prog, i, n)\}, \tag{7.3}$$

i.e. S encrypts its data with K_S and annotates its identity id_S for correct decoding. The platform decrypts the message with the secret key K_S stored in a key table indexed by (id_S, k). This step not only decrypts the message but also achieves provider authentication, since the successful decryption of the message through the platform proves the authenticity of the sender.

One drawback of this protocol, however, is that it does not prevent any sort of replay attack. Replay prevention can be easily solved by sequence number schemes that apply a FIFO-order on the messages transmitted over the channel $S \to C$ which works well, if the channel is sufficiently reliable.

Communication in the opposite direction is performed through the following protocol:

$$C \to S: \quad \{id_C, k, enc_{K_S}(E)\}, \tag{7.4}$$

i.e. data E is simply encrypted with K_S. The platform identifier id_C and k are used to look up the appropriate key for decryption. But this can be equally well left to the script itself, if it contains a fresh secret key that is used to protect the communication back to the provider. Encryption can be achieved with the interpreter operation `encrypt` that performs encryption of the topmost stack elements up to the `mark` with a key from the key store. Obviously, such temporary key management involves a significant amount of house-keeping on the service provider's side. However, since one of our major requirements is to make such security properties subject of the applications and scripts we consider it as an actual advantage over more fixed security frameworks.

[4] Here again, n and m are random values.

Secure End-to-End Communication without Key Installation

If no persistent secret key installation is performed, each time a new script is sent to the platform the trusted third party has to authenticate the service provider and sign the script accordingly. The resulting protocol is similar to 7.2:

$$\begin{aligned} S \rightarrow M: &\quad \big\{enc_{P_C}(K_S, i, k, n), enc_{K_S}(\mathit{prog})\big\}, \\ M \rightarrow C: &\quad \big\{enc_{P_C}(K_S, i, k, n), enc_{K_S}(\mathit{prog}), \Sigma\big\}, \\ &\quad \text{with } \Sigma \leftarrow sig_K\big(enc_{P_C}(K_S, i, k, n), enc_{K_S}(\mathit{prog}), id_S, m\big). \end{aligned} \tag{7.5}$$

Basically, the protocol is adapted to transport the program *prog* in addition to the key K_S.

Communication in the direction $C \rightarrow S$ is achieved with protocol 7.4.

7.6.2.2 User Channel Communication Protocols

In contrast to the public key protocols for shared secret exchange, we already mentioned that the platform user could participate in the transfer of the secret from the provider to the card.

Secret key installation

We use the following protocol to transfer the secret K_S from the provider S to the platform C:

$$\begin{aligned} S: &\quad W \leftarrow f(id_S, k, K_S), \\ S \rightarrow U: &\quad \{W\}, \text{ over a secure channel}, \\ U \rightarrow C: &\quad \{W\}, \text{ user input}, \\ C: &\quad (id_S, k, K_S) \leftarrow f^{-1}(W). \end{aligned} \tag{7.6}$$

Here, f is the transformation of the key into the alphabet $\{\mathtt{A–Z,0–9}\}$. Hence, this protocol agrees upon a shared secret that can be used for confidential communication. However, the identity id_S of the provider is not verified by the platform. Again we use the help of the platform provider or another trusted third party to verify the authenticity of the key as follows:

$$\begin{aligned} S: &\quad \mathit{res} \leftarrow enc_{K_S}(\mathit{rand}), \\ S \rightarrow M: &\quad \{id_S, k, id_C, \mathit{rand}, \mathit{res}\}, \\ M \rightarrow C: &\quad \{id_S, k, \mathit{rand}, \mathit{res}, n, \Sigma\}, \\ &\quad \text{with } \Sigma \leftarrow sig_K(id_S, k, \mathit{rand}, \mathit{res}, n), \\ C: &\quad \mathit{res}' \leftarrow enc_{K_S}(\mathit{rand}), \mathit{res}' = \mathit{res}? \end{aligned} \tag{7.7}$$

First, S generates a challenge *rand* and encrypts it using K_S yielding *res*. The challenge, the response, the identifier id_S, key store number k, and the platform identifier id_C are then sent to the platform provider M over an encrypted and authenticated channel. M signs the data to be forwarded adding a random value n. In the platform, the signature is verified and then the challenge is encrypted with K_S yielding *res*'. If *res* and *res*' are equal, the key can be considered as authenticated, the corresponding compartment can be activated, and the key is registered in the key store of the compartment of provider id_S.

Secure End-to-End Communication

Subsequent communication between the service provider and the platform occurs with the protocols 7.3 and 7.4.

7.6.2.3 Further Protocol Options

In the previous sections we have outlined the basic protocols how end-to-end security between a service provider and the platform can be achieved. More protocols for other purposes can be easily developed based on the presented ones, such as non end-to-end protocols. This communication pattern could be used to transport a script from the service provider to the platform provider if no on-card code verification is available. This corresponds to option (**3**) in Table 7.1 on page 113. The complete protocol would then use a mixture of the end-to-end protocols presented previously to transport the data section, and verify the script at the operator who then appropriately signs the code after verification.

More generally, the actual protocols can be adapted, to perform suitable *key scheduling* and *dynamization* based on additional random values transported in each message to generate temporary session keys derived from master session keys.

Summing up, a diversity of different options for establishing secure communication channels between the service provider and the platform can be established. In particular the difficult case of end-to-end security can be achieved with reasonable protocol overhead and on-card resources.

7.7 Non-Repudiation and Electronic Signatures

In the previous sections we have outlined the architecture of our platform, its language and runtime environment with a focus on the platform security issues, and the end-to-end secure communication protocols.

For business-critical applications, however, end-to-end security is not the only important factor. The problem of *non-repudiation* in the form of electronic signatures is of similar importance, in particular, if legal issues have to be considered.

7.7.1 Motivation

We have motivated the platform approach by a general demand that a service provider should be able to implement application-specific security properties. In a typical *business-to-employee* (B2E) scenario this could mean that the enterprise security policy states that it is not necessary to use legally-binding highly secure electronic signature systems under all circumstances. In particular this means that a signature based on a shared secret key system might be sufficient, if the signature creation device is considered to be trusted.

The tamper-resistance of the smartcard and the trusted platform approach we target could be an ideal platform for such shared secret signature systems since it offers means for a service provider to manage its own keys appropriately.

A significant number of issues related to the creation of electronic signatures have been raised and solutions to them have been patented in [Bro97]. However, our approach does not target mobile phones only, but can be used in a variety of possible end-user terminals supporting a smartcard as security module, e.g. the PCA as presented in Sect. 5.3.

Basic Electronic Signature Protocol

Traditionally, in scenarios for electronic signature creation the most important roles as introduced in Sect. 5.2 are the signer S owning a public key pair (S_S, P_S), the document to be signed D, the signature creation application A, a document viewer V interacting with the signer, a smartcard C, the originator of the document O and the document D. The basic protocol is as follows:

$$\begin{array}{rll} O \rightarrow A: & \{D\}, & \text{document transfer,} \\ A \rightarrow V, S: & \{D\}, & \text{document presentation,} \\ S, V \rightarrow A: & accept/reject, & \text{user choice,} \\ A \rightarrow C: & \{h(D)\}, & \text{hash computation,} \\ C \rightarrow A \rightarrow O: & \{sig_{S_S}(h(D))\}, & \text{signing.} \end{array} \tag{7.8}$$

Based on the above protocol we describe further improvements on the overall security of the electronic signature creation process on top of our platform.

7.7.2 Electronic Signatures with On-Card Hash Computation

With the PCA we have demonstrated how the presentation of a document to be signed and the hash computation can be performed on a device that is more trustworthy than a vendor's terminal. Performing the hash computation on a trusted device such as a smartcard itself might be even more secure than other approaches. However, it is important how the document presentation and hash computation is done in the overall signature protocol. Consider for example the following case:

$$\begin{array}{rll} A \rightarrow C: & \{D\}, & \text{document transfer to card,} \\ C \rightarrow A: & \{sig_{S_S}(h(D))\}, & \text{document signing.} \end{array} \tag{7.9}$$

From a security point of view an intruder I who is in control of A can easily exchange document D with another document D' that is subsequently hashed, sent to the card, and signed. Hence, compared with the basic protocol, no additional benefit can be gained.

On-Card Hash Computation Protocol

Assuming a scenario in which the signature creation application A is located in the security module C, and the viewer in the (less trustworthy) terminal the protocol is as follows:

$$\begin{array}{rll} O \rightarrow A: & \{D\}, & \text{document transfer to card,} \\ A \rightarrow V, S: & \{D\}, & \text{document presentation,} \\ S, V \rightarrow A: & accept/reject, & \\ C, A \rightarrow O: & \{sig_{S_S}(h(D))\}, & \text{hash and signature computation in card,} \end{array} \tag{7.10}$$

Assuming end-to-end secure communication between O and A/C, an intruder is not able to control the hash computation anymore. Only the document presentation and the user's *accept/response* could be manipulated, although the intruder controlling V cannot gain anything from such manipulation, except by mounting the attack presented in the next section.

A Conspiracy Attack on On-Card Hash Computation

A successful attack can be mounted in the above scenario as follows:

- The intruder I and the originator O cooperate.
- O sends the document D' that is the document which the attackers want to be signed by S.
- Upon invocation of V, I presents a fake document D, which S might accept for signing.

- In the card, D' is signed and sent back to O.

Hence, an attack is still possible, if the intruder subverting V and the originator O of the document directly cooperate. Although this attack is of general importance, practically, it means that it is not anymore sufficient to attack the user's terminal only, but also manage to actively send a faked document to the user for signing. Furthermore, possible countermeasures are sketched in Sect. 7.7.5.

As a consequence, we think that shifting the hash computation in the above manner to a tamper-resistant device yields reasonable improvement in the overall security.

7.7.3 Electronic Signatures Assisted by a Trusted Third Party

If on-card hash computation is not feasible, this process can also be delegated to a trusted third party T as the following protocol outlines. It uses the URL url_D to denote some identifier where D can be fetched from. The trusted third party T computes D's hash on behalf of A and signs it. A just forwards the URL to the document viewer V. The remaining protocol steps are the same as in the on-card hash computation protocol (cf. 7.10).

$$\begin{array}{rll}
O \rightarrow A: & \{url_D\}, & \text{transmit URL to sign,} \\
A \rightarrow T: & \{url_D\}, & \text{send URL to TTP,} \\
T \rightarrow A: & \{sig_T\big(h(D)\big)\}, & \text{fetch document from URL, compute hash,} \\
A \rightarrow V, S: & \{url_D\}, & \text{document presentation,} \\
S, V \rightarrow A: & accept/reject, & \\
A \rightarrow C: & \{sig_T\big(h(D)\big)\}, & \text{signature verification and signing,} \\
C \rightarrow A \rightarrow O: & \{sig_{S_S}\big(h(D)\big)\}, & \text{final computation of signature.}
\end{array} \tag{7.11}$$

Similar to the on-card hash computation protocol, it is vulnerable to the conspiracy attack.

7.7.4 Electronic Signatures with Recipient Addressing

Looking at the traditional signature creation protocol it becomes obvious that the authenticity of the document sender is not of particular concern. In electronic business processes, signatures are often used to provide the technical basis for contracts between two parties. Although the identities of the contract partners are usually somehow denoted in the document D, this is by no means cryptographically protected.

To improve the signature process we propose to include the cryptographic identity of the peer into the signature process. In particular we propose the following protocol that is based on the on-card hash computation protocol 7.10 and the public key pair (S_O, P_O) of the originator O:

$$\begin{array}{rll}
O \rightarrow A: & \{D, sig_{S_O}(D)\}, & \text{send document and signature,} \\
A \rightarrow V, S: & \{D, id_O\}, & \text{show document and identity of } O, \\
S, V \rightarrow A: & accept/reject, & \\
C, A \rightarrow O: & \{sig_{S_S}\big(h(D), sig_{S_O}(D)\big)\}, & \text{hash and signature computation in card.}
\end{array} \tag{7.12}$$

This protocol now achieves that an electronic signature is created over both – the cryptographic hash of the document and the identity of the *recipient* or originator of the signature.

To assess the advantages of this approach we consider that in a traditional signature attack scenario an intruder could "hijack" the signing process of an arbitrary document D_O with its intended recipient O to infiltrate another document D' to be signed. The intruder I could then claim that the user has signed this document which is likely of advantage to the intruder. In the above protocol, however, the intruder I is not able to generate a signature $sig_{S_S}\big(h(D'), sig_{S_I}(D')\big)$ since the signature $sig_{S_I}(D')$ cannot be generated. At best $sig_{S_S}\big(h(D'), sig_{S_O}(D')\big)$ could be obtained, but leading to a contradiction between the information available in D' denoting I as the recipient and the envelope signature sig_{S_O}. Therefore, we argue that linking the document and the recipient in the signature gives advantages to standard electronic signature creation.

A Possible Attack on Recipient Addressing

Basically, the same conspiracy attack presented in the on-card hash computation in Section 7.7.2 can be mounted in the recipient addressing scheme. Again, if originator O and intruder I cooperate, the user is not able to distinguish that signature creation occurs with a document that she does not intend to sign. Possible countermeasures are presented in the next section.

7.7.5 Electronic Signatures with Samples

In the previous sections we have proposed on-card hash computation and recipient addressing as possible approaches to improve the security of the electronic signature creation process. However, both suffer from the conspiracy attack described in Section 7.7.2. The underlying assumption of this attack is that the document viewer component V might be vulnerable to different manipulation attacks. Typical examples of this assumption might be that the viewer is running on top of a general-purpose operating system onto which a user can install arbitrary software.

This situation can be corrected with the help of an additional security module V' that is not an open platform, but rather a dedicated device that has a secure link to the platform. This device is used for input and output of only a limited amount of information, e.g. in textual form that can be used as a secondary device to implement the *two-eye principle* for signing data. The overall protocol is as follows:

$$\begin{array}{rll} O \rightarrow A: & \{D\}, & \text{document reception,} \\ A \rightarrow V, S: & \{D\}, & \text{document presentation,} \\ S, V \rightarrow A: & \{accept/reject, \{s_0, \ldots, s_n\}\}, & \text{set of sample selections,} \\ A \rightarrow V', S: & \{s_0(D), \ldots, s_n(D)\}, & \text{presentation of samples,} \\ S, V' \rightarrow A: & accept/reject, & \\ C, A \rightarrow O: & \{sig_{S_S}\big(h(D)\big)\}. & \end{array} \tag{7.13}$$

The basic difference to protocol 7.10 is as follows:

i. The *less trusted* viewer V is not only used to display the document and return the user's response but also offers means to enable the user to select arbitrary *samples* $s_0, \ldots, s_n$ of the document D that are send back to A.

ii. Based on the samples obtained from the first viewer the signing application A extracts the corresponding parts from document D, i.e. $s_0(D), \ldots, s_n(D)$, and sends them to the *more trusted* viewer V' that ultimately decides upon whether the document should be signed or not.

Basically, this approach has the general drawback that a user needs to perform a two-step approach to signing. Furthermore, the first viewer V must offer suitable facilities to enable a user to make arbitrary selections, and finally, it does not give ultimate security about whether the document has been tampered with.

However, in practice we think that this approach can improve the security of the signature creation process in hostile environments to a certain degree if we assume that there exist some parts of a document that are more important than others. Consider, for example, a bill with several information about the items to buy. Nonetheless, the amount a to pay is most likely the most important information on the bill and the user could decide to sample the amount in the bill using an appropriate selection $s_i(D) = “\dots, a, \dots”$ covering the amount. If later the trusted viewer V' displays the same amount the user can be sure that the untrusted viewer V' is at least not able to forge this part of the document and change the amount.

The essence of sampling the data to be signed is the fact that an intruder in control of a document viewer cannot make any assumption, whether or how the sampling information is actually used. Obviously, the intruder knows about the selections performed by the user and could, e.g. change the amount of money in the bill, if no appropriate sample was taken. However, it could be left to the signing application A to decide, how the set of samples $\{s_0, \dots, s_n\}$ is to be interpreted. Possible interpretations could be to take the samples as indexes into the document creating intervals of the form $\{[s_0, s_1], \dots, [s_{2i}, s_{2i+1}]\}$ that are used to extract the appropriate parts from the document. Equally well a projection of the form $\{[s_0, s_{0+x}], \dots, [s_i, s_{i+x}]\}$ with a fixed length x could be made. Hence, it is up to the application A and the user to interpret $\{s_0, \dots, s_n\}$, something that can be used to make attacks arbitrarily hard.

Although signing with samples might not be a solution which is generally applicable to all electronic signature schemes, we consider it a valuable addition to the overall security of a signature scheme in resource-restricted environments, where the presentation of the complete document must be delegated to a less trusted component, e.g. a vendor's terminal, whereas a certain security-critical subset of the whole document can be viewed on a more trusted component.

7.7.6 Electronic Signatures on Interactions

One of the most problematic issues with electronic signatures on mobile devices is the fact that such signatures can be only computed over documents. In particular this means that according to current signature laws, cf. Germany, the document must be presented to the user who then either accepts or rejects the subsequent signature creation. Hence, a document to be signed must be presented as a whole in a suitably rendered form. This problem of encoding and subsequently displaying a document in a reproducible and standardized way has been extensively discussed by Scheibelhofer [Sch01].[5]

To approach this presentation problem which becomes especially crucial on small terminals we consider not only the presentation of a document but also the way the document is created. We argue that a document is often the result of some kind of interaction between a service provider, e.g. who offers goods, and a client, e.g. selecting goods to buy. Finally, after all selections are made, a document containing the complete list of goods is presented and signed accordingly.

However, the platform approach offers advanced options to create such a signature in a much more user-friendly way. If a document is encoded as a script, the execution of the script is *deterministic* as long as all *non-deterministic* input that is received from “outside” the script such as user input, random number generator, persistent variables, etc. is recorded. A “document” over which the signature is computed then consists of

(a) the script,

[5] In his approach he uses XML style sheets defining mappings to a possibly certified rendering engine.

```
script {

   provider "bidbiz.com";
   name     "bidbiz␣auction␣client";
   id       "20011223/24357";
   options  signed-interaction;

   implementation {
      playtone;
      push("News␣from␣bidbiz.com:\n
            Bid in auction #3576 (Antique watch): EUR 63.");
      display;

      push(mark);
      push("Place␣new␣bid?");
      push("New␣bid...");
      push("Cancel");
      select;                            ⟵ User selects option int:'1'

      push( 2 );
      eq?;
      if (true) goto end;

enter:
      push("Enter␣new␣bid␣(>EUR␣64):");
      input;                             ⟵ User inputs new bid amount: (string,'70')
      dup();
      push(64);
      le?;
      if (true) goto end:
      playtone;
      push("Please␣enter␣a␣bid␣greater␣than␣$64.");
      display;
      goto enter;

end:  sign-interaction;
      response;
      exit;
   }
}
```

Figure 7.4: Mobile auction client with interaction signatures

(b) the persistent state used during the computation,

(c) all user input,

(d) all messages received from other communication channels,

(e) the current time and progress of execution,

(f) some platform characteristics such as version numbers, serial numbers, etc.

The signature can be easily verified by executing the script in a simulated environment using the recorded and signed input values. Thus, a signed document is not intended to be human-readable, but rather meant to record and log the interaction that happened between a service provider and a client.

For illustration purposes we provide the mobile auction example introduced in Fig. 7.2 on page 106 in a version that uses our interaction signatures instead of explicit signing. This modified version is listed in Fig. 7.4 and we show the contents of a sample interaction with all corresponding log information on the right.

The only information available in the log is the menu selection the user performed (option no. 1) and the new amount entered (€ 70). The differences to the original version are as follows:

- The script has an additional mode (line 6) indicating with the additional option `signed-interaction` that the script execution should be recorded.
- Upon execution of the primitive `sign-interaction` (line 36) the execution record is signed and pushed onto the stack as a stack object that can be further encrypted or otherwise protected.
- The signed interaction object is then sent back to the originator using the primitive `response` (line 37).

During execution the runtime environment collects the non-deterministic input from the various sources into a log $L = \{i_1, \ldots, i_n\}$ of inputs i_j. In the above example execution thus yields

$$L = \{(\texttt{int}, \text{‘1’}), (\texttt{string}, \text{‘70’})\},$$

i.e. for each input we record the type information and the data. The overall interactive log of an execution of script P with the identifier id_P is computed and returned to the original sender S as follows:

$$C \rightarrow S: \quad \big\{id_P, L, sig_{S_C}\big(hash(id_P, P, L)\big)\big\}. \tag{7.14}$$

The receiver must be able to verify the authenticity of the signature by simulating the execution of the script according to the log L. Based on this simulation, the interaction of the script and the user can be replayed and the user's choices and inputs can be examined to take appropriate action.

Summary

Runtime execution monitoring using an execution log has been investigated by Vigna as a means to protect the execution of mobile agents in hostile environments [Vig98]. The sender of a mobile agent can verify whether the agent has been tampered with while executing on a remote agent platform.

Using signatures on runtime execution audits combined with the recording of user interactions as a means to implement non-repudiation, however, is to the best of our knowledge a novel approach. We consider this approach particularly useful for our application domain for the following reasons:

- Due to the lack of user input and output facilities, performing all possible executions within the trust domain of the smartcard is from a security point of view desirable.
- Interactions that leave the trust boundary of the smartcard are reduced to the bare minimum, i.e. to user interactions only.
- The approach is very flexible, since it offers scripts a full control over the way signatures are built, how encryption is performed, and how interaction takes place. As such it is able to offer applications means to implement security policies as needed.

Thus, our approach allows service providers to take full advantage of the smartcard as an open platform for running security-critical applications in the tamper-resistant context of the physical device.

7.8 Application Scenarios

As motivated already with the WebSIM, the smartcard platform presented in this chapter must be considered as a mere platform for third-party providers such as Internet shops, banks, etc. Ideally, such a platform would be again hosted by a GSM SIM or another smartcard embedded in a wireless card reader. This would offer secure interaction of service providers with their customers in a push-based style of communication over a wireless communication link such as GSM.

The following are examples of the types of applications that could be thought of in such a setting:

— **Push-based mobile auction client:** This application is extensively described in Section 7.4.3.

— **Push-based brokerage, stock trading:** A broker application is similar to the auction client. The subscriber could be informed about new stock watch-points that could be used in addition to stop-loss. It could provide more subscriber feedback on what is going on at the stock market and provide means to react to current trends in a matter of seconds.

— **Internet-based authentication:** Possibly one of the most interesting classes of applications is authentication of mobile subscribers to the Internet. Currently, Internet servers accessed by HTTPS/SSL authenticate themselves via certificates whereas in practice client authentication is not used due to the lack of a global PKI infrastructure. Detailed protocols and approaches to achieve this goal based on the technology presented in this thesis are described in [GKP00] and [KPSW01].

— **Interactive SMS:** This *real* killer means that subscribers can send each other (predefined) interactive messages – mobile originated, or from the Internet.[6] How would you like the interactive message "*Hey honey! Do you want to drink a coffee with me at 15:00 at the Bizarre's?; Yes,No*"

— **Loyalty-points application:** Smartcards offer secure persistent storage that could be used by loyalty point systems that manage the bonus points via GSM. This would provide a complete off-line light-weight loyalty system that uses the distributed secure storage of SIMs for management.

— **Electronic ticketing:** Electronic tickets ordered over the Internet could be downloaded over the air and stored securely in the SIM. At a site's entrance, e.g. the opera, subscribers could show the ticket by browsing the appropriate ticket containing unique identifiers or offer access to the ticket by means of IrDA or Bluetooth in the future.

— **Secure enterprise applications:** Enterprises with a strong demand for secure and application-specific mobile transactions are likely to invest into customizable solutions. Especially flexible mechanisms for end-to-end secure communication between an enterprise and its mobile employees could be of particular importance.

Summing up, the architecture provides an open and secure platform on top of which various other applications can be built.

7.9 Related Work

WAP Push [WAP99c] defines the architecture underlying the push-based technology in the WAP protocol family. Content is pushed via an appropriate gateway to a WAP Push-enabled terminal.

[6] We are sure that this application is much more attractive than the boring security stuff described previously.

However, it is not targeted towards pushing active content from a provider into a smartcard and thus not comparable to the platform approach presented here.

SIM toolkit browsers [SAT00] provide a browsing technology on top of the SIM application toolkit. Applications are driven by a gateway usually hosted by a mobile operator. It also offers means for end-to-end encryption of confidential data between the service provider and the SIM. In contrast, our platform approach does not focus on a browsing-like type of interaction but more on a push-based style of communication driven by an Internet client also aiming at true end-to-end security of transmitted scripts.

The specification of the 3GPP USAT interpreter [3GPP01b; 3GPP01c] under development of the corresponding working group comes most close to our envisioned platform. This group defines the USAT interpreter that makes use of the SIM toolkit commands available in the future UMTS SIM. It does not only define a browser-based model but also allows for push-style communication from the gateway, which in turn can be triggered from the Internet. The byte-codes used there reflect the group's idea of a *page*-based style of user interaction similar to the SIM toolkit browsers. As such they support pages as the elementary containers that can be linked by a web of anchors for navigation. Within a page, variables can be used to store page-local information and exchange data between different pages. Contained in a page are USAT commands encoded as tag-length-value data (TLV) structures providing a generic approach to subscriber and handset interaction.

Interestingly, the architectural descriptions for the USAT interpreter also include true end-to-end security between a service provider and the SIM. Hence, it seems that the general ideas presented in this thesis concerning the smartcard platform are actually considered in the smartcard and telco industry, yet in a completely different manner in terms of protocols and language model. However, we consider the platform model in this thesis as still more flexible and open for future applications.

7.10 Summary

In this chapter we have described what additional role a smartcard can play as personal security module. More precisely, we have shown how an open application platform that is programmable in a domain-specific language can be used to allow for end-to-end secure communication between a service provider and its customer. In the sequel we revisit the main contributions of this chapter.

Platform Approach

Considering the numerous security problems known from mobile agent systems for both – host and agent security – it is at first irritating to provide a security infrastructure on top of a mobile code platform. In contrast to more general-purpose approaches, however, the open platform presented here follows a domain-specific design paradigm starting with a top-down approach identifying the functionality desired and offering a framework that provides solutions to this problem only.

A question that often had to be answered is why not use Java Card for this purpose? Basically, the research community in JavaCard currently tries to formalize a complex language trying to eventually prove certain properties of applications using suitable theorem provers or model checkers (cf. [Ber97; PV98; BCM$^+$00; PvdBJ00]), tools that are currently applicable only in an off-card setting.

The work presented here, tries to follow a top-down approach, i.e. starting from a particular application domain and trying to find a solution that fits the requirements without struggling with unnecessary features. This approach has been proven to be especially suitable for finding solutions to the security problems encountered in our application domain.

End-to-End Security

End-to-end security requires that information is transmitted confidentially and authentic using appropriate cryptographic keys and algorithms. In the context of our personal security module this results in the following fundamental problems:

— **Key management:** Since extensive use of public key cryptography should at least for the near future be restricted to the non-avoidable cases. Thus, we propose to use public key cryptography for key management only, and further communicate using appropriate symmetric cryptographic algorithms.

— **On-card verification:** Verification of code and data comes into play as a result of end-to-end security that does not allow for a third party assisted verification step. Although data verification might be considered as more security critical, though easier to implement in a smartcard, ultimately code and data verification must be envisioned. This has resulted in the identification of security properties to be verified on-card. Some of the properties are well-known from the Java byte-code verification, however, new properties such as the control-flow constraints have been identified and suitable algorithms have been presented.

Thus, adding end-to-end security to this problem domain has resulted in much more complexity on the platform's side, since it alone is responsible for guaranteeing correct behaviour.

Non-Repudiation

The third major contribution of this chapter is the domain of non-repudiation, i.e. the generation of electronic signatures on-card. Basically, we have distinguished two different mechanisms:

— **Application-driven signatures:** This is the traditional method of implementing some kind of `sign` primitive that presents to the user some data to be signed.

— **Interactive signatures:** This novel approach represents some kind of *implicit* signing of the whole execution of an application in the platform. This relieves the application from explicitly requesting the user to sign a specific piece of data and instead relies upon the implicit signature creation using the logging mechanisms in the platform. Signature verification can be easily achieved by a *simulated execution* of the script that is then fed with the information from the log.

Besides mechanisms for non-repudiation, possible attacks on the platform have been described of which the so-called conspiracy attack is rather critical. From a general point of view protecting from this attack, which requires a malicious script and an intruder's access to the interface between card and terminal, is hard to achieve. However, in certain contexts the use of samples that delegate the document viewing and accepting process to an even more secured environment might help.

The platform approach described in this chapter has been published in [KM01]. The electronic signature creation based on runtime execution and interaction is further described in [KP01].

Chapter 8
Conclusion

As foreseen by many researchers and analysts, the mobile user of tomorrow will live and move in a world of many digital services available as part of a larger pervasive and ubiquitous computing environment. In this scenery personalized terminals will play a significant role – be it a device similar to today's mobile phone or something that comes into completely new form factors in the spirit of the "disappearing computer". Acting in this environment and accessing different services at any time and any place most likely requires fundamental security operations to protect both – the mobile user and the provider of a service.

In the information age cryptography has proven to be at the core of many protection mechanisms implemented in many different applications all over. From a human's perspective, however, cryptography has the fundamental drawback that it cannot be done without suitable security modules performing cryptographic operations on possibly large – and therefore non-memorizable – cryptographic data. As a consequence, users need cryptographic support in different shapes.

Smartcards have proven to be such well-suited devices that solve some of the essential problems, i.e. they are tamper-resistant, can perform cryptographic operations, and they are not observable during computation. However, a fundamental problem is their lack of suitable input and output facilities to communicate with their holder.

The underlying idea of this thesis was to find some answers to the general problem of "*how can smartcards become truely personal and ubiquitous security modules?*" Especially, how it can be achieved to make them usable at any time at any place, a question which obviously has a very technical focus. In line with this idea is the paradigm shift to view the smartcard as the central security component around which other components have to be arranged. Thus, the question is also more "*how to find suitable terminals for my smartcards?*" than vice versa.

8.1 Results

This thesis shows in general that small devices require special assistance from their infrastructure. Smartcards are an incarnation of small devices that are particularly interesting because of their widespread use and limited functionality. One of the first results presented in Chapter 4 is a general framework for the integration of such devices that subsequently has been applied to smartcards as a special instance of this category of devices. The approach presented allows a formerly "passive" smartcard that is inserted into a suitably equipped reader to "actively" offer services towards potential clients. As a consequence, the smartcard is actually able to actively search reasonably "secure" terminals for its user through which further interaction can take place.

The most obvious problem with smartcards is that they are vitally dependent on the trustworthiness of the terminal that is used to interact with the card. In Chapter 5 an approach was presented that tries to bring personal terminals such as a user's PDA into the scene. The underlying idea is that of a "trust amplification", i.e. a user is likely to trust his or her "own" device more than others' and thus a pragmatically more secure environment can be created. The PCA essentially consists of a pair of devices – a terminal and a smartcard – that are cryptographically linked to each other. The net side effect of this is that they can only be used together and that none of the two devices is usable without the other. This general theme could be applied to many security-related contexts to make users more comfortable and increase the overall security of a system.

Whereas the two former approaches concentrated more on the usage of the smartcard in a local environment, the WebSIM presented in Chapter 6 follows a completely different approach. It essentially builds upon the metaphor of a "wireless smartcard reader" through which a user's security module can be accessed from anywhere in the Internet using HTTP, i.e. "the" Internet protocol of today. The advantage of this approach is that users can request services through virtually any communication link, though still performing their security-related action through their personal wireless card reader. The advantage of the proposal is that it builds upon an already deployed infrastructure – the existing GSM and future UMTS telecommunication networks – making it a promising candidate for future security architectures.

One of the most urgent problems with electronic businesses is the need for end-to-end confidential transmission of data and the creation of electronic signatures. In Chapter 7 the idea of a "autonomously" acting smartcard was investigated further and the vision of an application platform for mobile code inside a smartcard was created. As a consequence, the tamper-resistant container offered by smartcards cannot only be exploited by the applications brought into the card at issuance time but also by applications wishing to interact with customers. In this model, service providers can build small applications written in a domain-specific programming language tailored for this particular application domain and send them using end-to-end secure communication to the smartcard. Upon receipt, a verifier can check the code whether it fulfills the security properties that allow for safe execution in the target platform. The advantages of this approach are that especially in the context of mobile scenarios significant advantages can be given to both – service provider and user – to securely perform interaction. Although the general problem of the trustworthiness of the users' terminals cannot be solved from a theoretical point of view, new protocols for the creation of electronic signatures have been presented by which significant "practical" security advantages can be gained.

Summing up, this thesis presents new design options that can be used independently or in combination to create new security interfaces for smartcards assisting their mobile users in their daily routine. As smartcards can be expected to continue their growth in terms of memory and computational power, this thesis has hopefully demonstrated that truely personal security modules are ready for implementation.

8.2 Future Work

Obviously, a number of issues has not been further discussed in this thesis and thus can be the subject of future investigation.

The first issue is related to the JiniCard framework considering the problem of mapping device-related identifiers as introduced in the detection event with suitable mobile code objects controlling further interaction with the device. From a privacy perspective this mapping discloses a number of information that can be considered as "private" and thus should not be given away imprudently. First steps towards an anonymization scheme based on devices that are able to perform cryptographic operations have been investigated in [Mol00] where different anonymization protocols have been developed and prototypically implemented on cryptographic devices. However, a scalable and especially secure infrastructure is needed for the mapping from real-world objects to virtual objects, an issue that has not yet been discussed too much in the research community.

Recently, security problems such as global user authentication have been in the focus by different players in the industry aiming at services that are hosting and managing a user's digital identity. New kinds of attacks such as "identity theft" (cf. [US01]) exploiting weaknesses in those technologies are likely to emerge. The work presented here offers countermeasures by the use of strong cryptography based on smartcards. An open research topic is henceforth the investigation how both approaches could benefit from each other and how the smartcard way of thinking can be integrated into such systems to avoid pure software solutions in these security-critical technologies.

Another research topic in conjunction with the smartcard platform approach presented in this thesis is the problem of on-card verification of security properties of mobile code. The formal methods research community is in possession of the relevant technologies needed to solve the problems outlined. Although a true on-card implementation has not been undertaken in the course of this work, a smartcard platform implementation that offers such mechanisms should be the next step towards this direction.

A related issue with the smartcard platform is the question whether the termination problem in the smartcard platform can be solved more elegantly. However, the interaction invariant used to decide upon the execution of the application is a good starting point, improving this mechanism maybe in conjunction with resource control should be tried.

Although the work in this thesis has concentrated on the "classical" smartcards and has to some extend neglected newer devices such as USB tokens these should be taken into future consideration since changes in underlying assumptions, e.g. missing clock, often lead to new design options.

Hence, this thesis has brought smartcards into a new setting and many different aspects related to these settings can be considered as future targets of investigation.

Appendix A
Platform Specification

A.1 Prototypical Implementation of the Smartcard Platform

Based on the conceptual approach presented in Chapter 7 we have prototypically implemented most of the presented concepts to demonstrate the feasibility of our approach in the context of GSM SIM smartcards.

— **Compiler:** The implementation of the compiler was done in Java 2 based on the parser generator toolkit JAVACC [Met01]. It basically takes an script and compiles it into the bytecode form understood by the verifier and interpreter. The compiler consists of roughly 7.000 lines of code (LoC) of which a substantial amount is automatically generated by the parser toolkit.

— **Verifier:** We have implemented an off-card version of the verifier in Java 2. It performs the basic checks for correct typing, stack over- and underflow, and checks for the control flow constraints discussed in Section 7.5.1.1. The implementation is currently about 5.000 LoC.

— **Interpreter:** We have implemented the platform and interpreter in a Java 2 off-card version consisting of roughly 4.000 LoC. An on-card implementation was partially performed on a Bull SIM'n Rock™ GSM SIM JavaCard 2.1 card, which did offer only about 9 kB of EEPROM which was not sufficient for a complete implementation of the interpreter.

Although a Java Card implementation of an interpreter is for performance reasons not comparable to a native implementation on the card's processor, it can be estimated that a native implementation of our platform seems to be possible with the 64 kB EEPROM smartcard generation likely to enter the market in 2002.

Leroy [Ler01] has recently presented first results of an algorithm that facilitates the implementation of a Java byte-code verifier on a smartcard. Since this problem can be considered to be at least as complex as the verifier needed for our platform and the expected growth of smartcard resources in the next years we believe that the overall approach can be considered as reasonably feasible in the near future.

A.2 Language Primitives

For the description of the primitives we denote stack effects in the usual form (*operation*, *before stack — after stack*). We use the notation "!" to indicate a special stack element called *marker* that

Primitive	*Before*	—	*After*	*Description*
push(z)	*x y*	—	*z x y*	
pop	*x y z*	—	*y z*	
swap	*x y z*	—	*y x z*	
dup	*x y*	—	*x x y*	
rot	*x y z*	—	*z x y*	
add, sub, mul, div, mod	*x y*	—	*z*	$z = x \circ y$
not	*x y*	—	*z y*	$z = \neg x$
and, or	*x y*	—	*z*	$z = x \circ y$
eq, neq, le, lt, ge, gt	*x y*	—	*z*	$z = x \circ y$
null?	*x*	—	*z*	$z =$ true *if* x *is* null, *else* false
mark?	*x*	—	*z*	$z =$ true *if* x *is* mark, *else* false
load	*r z*	—	*w z*	$w = reg[z]$
store	*v r z*	—	*z*	$reg[r] \leftarrow v$
kload	*k z*	—	*w z*	$w = key[k]$
kstore	*v k z*	—	*z*	$key[r] \leftarrow v$
encrypt	*k x. . . y ! z*	—	*w z*	$w = enc_k(x\| \dots \|y)$
decrypt	*k x. . . y ! z*	—	*w z*	$w = dec_k(x\| \dots \|y)$
digest	*x. . . y ! z*	—	*w z*	$w = digest(x\| \dots \|y)$
sign	*x. . . y ! z*	—	*w z*	$w = sign(x\| \dots \|y)$
sign-interaction	*z*	—	*w z*	$w =$ *signed interaction*
display	*x. . . y ! z*	—	*z*	
input	*x. . . y ! z*	—	*w z*	$w = input(x \dots y)$
select	*x. . . y ! z*	—	*w z*	$w = select(x \dots y)$
response	*x. . . y ! z*	—	*z*	

Table A.1: Smartcard platform primitives

is pushed onto the stack with '`push(mark)`'. It can be used to denote the end of a variable length number of arguments of a primitive which is denoted in the form '$x \dots y$'. For convenience, if there is no mark on the stack, the number of arguments on the stack used for a particular command is automatically inferred depending on the concrete command.

Furthermore, the set of registers is denoted in the form '*reg*$[i]$' and the keys available in the key store are denoted in the form '*key*$[i]$'. Bindings for variables are denoted using '$=$' and assignments are denoted using '$\leftarrow$'.

The description of the behaviour of the platform primitives manipulating the data stack are listed in Table A.1.

```
⟨script⟩       ::= ‘script’ ‘{’ ⟨header⟩ ⟨impl⟩ ‘}’
⟨header⟩       ::= ( ‘provider’ ⟨STRING⟩ ‘;’ | ‘scriptname’ ⟨STRING⟩ ‘;’
                 |  ‘id’ ⟨STRING⟩ | ‘options’ ⟨option⟩* )*
⟨options⟩      ::= ( ‘signed-interaction’ )
⟨impl⟩         ::= ‘implementation’ ‘{’ ( [⟨label⟩ ‘:’] ⟨stmt⟩ ‘;’ )* ‘}’
⟨stmt⟩         ::= ⟨smpl stmt⟩ [‘(’ ‘)’] | ⟨cplx stmt⟩ | ⟨flow stmt⟩ | ⟨ui cmd⟩
⟨smpl stmt⟩    ::= ‘nop’ | ‘exit’ | ‘pop’ | ‘dup’ | ‘swap’ | ‘rot’ |
                 |  ‘and’ | ‘or’ | ‘not’
                 |  ‘eq’ | ‘neq’ | ‘gt’ | ‘lt’ | ‘ge’ | ‘le’ | ‘null?’ | ‘mark?’
                 |  ‘add’ | ‘sub’ | ‘mul’ | ‘div’ | ‘mod’
                 |  ‘load’ | ‘store’ | ‘kload’ | ‘kstore’
                 |  ‘encrypt’ | ‘decrypt’ | ‘sign’ | ‘digest’
                 |  ‘sign-interaction’
                 |  ‘begin’ | ‘commit’ | ‘rollback’
⟨ui cmd⟩       ::= ‘display’ | ‘playtone’ | ‘select’
                 |  ‘input’ ‘:’ ( ‘int’ | ‘string’ | ‘bool’ )
                 |  ‘response’
⟨cplx stmt⟩    ::= ‘push’ ‘(’ ⟨push args⟩ ‘)’
⟨flow stmt⟩    ::= ‘goto’ ⟨label⟩ | ‘if’ ‘(’ (‘true’ | ‘false’) ‘)’ ‘goto’ ⟨label⟩
⟨label⟩        ::= ⟨IDENTIFIER⟩
⟨push args⟩    ::= ⟨STRING⟩ | ⟨INT⟩ | ‘null’ | ‘mark’
```

Figure A.1: BNF grammar and instruction set of the platform language

A.3 Grammar

The grammar of the platform language is shown in Figure A.1.

Bibliography

[3GPP01a] 3GPP: Third Generation Partnership Web site. *http://www.3gpp.org*, 2001.

[3GPP01b] 3rd Generation Partnership Project. *3GPP TS 31.112 V5.0.0 (2001-09) Technical Specification 3rd Generation Partnership Project; Technical Specification Group Terminals; USIM Application Toolkit (USAT) Interpreter Architecture Description (Release 5)*, September 2001. Available at *http://www.3gpp.org*.

[3GPP01c] 3rd Generation Partnership Project. *3GPP TS 31.113 V5.0.0 (2001-09) Technical Specification 3rd Generation Partnership Project; Technical Specification Group Terminals; USAT Interpreter Byte Codes (Release 5)*, September 2001. Available at *http://www.3gpp.org*.

[ABKL93] M. Abadi, M. Burrows, C. Kaufman, and B. Lampson. Authentication and delegation with smart-cards. *Science of Computer Programming*, 21(2):91–113, October 1993.

[ACCK00] Jay Algesheimer, Christian Cachin, Jan Camenisch, and Günther Karjoth. Cryptographic security for mobile code. Technical Report RZ 3302 (#93348), IBM Research Division, Zürich Research Laboratory, December 2000.

[ADH$^+$99] Gerd Aschemann, Svetlana Domnitcheva, Peer Hasselmeyer, Roger Kehr, and Andreas Zeidler. A framework for the integration of legacy devices into a Jini management federation. In R. Stadler and B. Stiller, editors, *Tenth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management DSOM '99, Zurich, Switzerland*, volume 1700 of *Lecture Notes in Computer Science*, pages 257–268. Springer-Verlag, October 11–13, 1999.

[AK96] Ross Anderson and Markus Kuhn. Tamper Resistance – A Cautionary Note. In *Proceedings of Second USENIX Workshop on Electronic Commerce, Oakland, California*, pages 1–11, November 18–21, 1996.

[AK97] Ross Anderson and Markus Kuhn. Low Cost Attacks on Tamper Resistant Devices. In M. Lomas et al., editor, *Proceedings of 5th International Workshop on Security Protocols, Paris, France*, volume 1361 of *Lecture Notes in Computer Science*, pages 125–136. Springer-Verlag, April 7–9, 1997.

[Amo94] Edward G. Amoroso. *Fundamentals of computer security technology*. Prentice-Hall, 1994.

[And01] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley Computing Publishing, 2001.

[ASBL99] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley. The design and implementation of an intensional naming system. In *Seventeenth ACM Symposium on Operating Systems Principles, Charleston*, pages 186–201, December 12–15 1999.

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Bell Telephone Laboratories Inc., Murray Hill, New Jersey, USA, 1986.

[BBC+94] Jean-Paul Boly, Antoon Bosselaers, Ronald Cramer, Rolf Michelsen, Stig Mjølsnes, Frank Muller, Torben Pedersen, Bigit Pfitzmann, Peter de Rooij, Berry Schoenmakers, Matthias Schunter, Luc Vallée, and Michael Waidner. The ESPRIT Project CAFE – High security digital payment systems. In *Third European Symposium on Research in Computer Security (ESORICS '94)*, volume 875 of *Lecture Notes in Computer Science*, pages 217–230, 1994.

[BCM+00] Pierre Bieber, Jacques Cazin, Abdellah El Marouani, Pierre Girard, Jean-Louis Lanet, Virginie Wiels, and Guy Zanon. The PACAP prototype: A tool for detecting Java Card illegal flow. In Isabelle Attali and Thomas Jensen, editors, *Proceedings of Java Card Workshop (JCW '2000), Cannes, France*. INRIA, September 14th, 2000.

[BE01] Uwe Baumgarten and Claudia Eckert. Mobil und trotzdem sicher? *it+ti Informationstechnik und Technische Informatik, Oldenbourg Wissenschaftsverlag*, 43(5):254–263, September 2001.

[Ber97] Peter Bertelsen. Semantics of Java byte codes. Technical report, Dept. of Information Technology, University of Denmark, March 1997. Available at *http://www.dina.kvl.dk/~pmb/*.

[BF99] Dirk Balfanz and Edward W. Felten. Hand-held computers can be better smart cards. In *USENIX Security '99*, August 1999.

[BGW98] Marc Briceno, Ian Goldberg, and Dave Wagner. GSM Cloning. Avaiable at *http://www.isaac.cs.berkeley.edu/isaac/gsm.html*, April 1998.

[Blu00a] Christof Blum. Elektronisches Ticketing bei der Deutschen Bahn AG. In Matthias Flur, editor, *OMNICARD*, 2000. *http://www.omnicard.de*.

[Blu00b] Oliver J. Blumert. Entwicklung eines portablen Signatursystems. DA-BS-2000-03, Darmstadt University of Technology, Department of Computer Science, July 2000.

[Blu01] Bluetooth Consortium. Specification of the Bluetooth System Version 1.1 – Core. *http://www.bluetooth.com/*, February 2001.

[BM92] Steven M. Bellovin and Michael Merrit. Encrypted Key Exchange: Password-based protocols secure against dictionary attacks. In *Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland*, pages 72–84, May 1992.

[BPR00] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *Advances in Cryptology – Eurocrypt '00*, volume 1807 of *Lecture Notes in Computer Science*, pages 139–155. Springer-Verlag, 2000.

[Bro97] Brokat Informationssysteme GmbH. *German Patent No. DE-197-47-603-A-1: Verfahren zum digitalen Signieren einer Nachricht*, October 1997.

[CEC00] Commission of the European Communities. Working site of eEurope action on smart cards. *http://www.europa.eu.int/istka2/smartcards.html*, 2000.

[CEN01a] CEN/ISSS WS/E-Sign N 136 Berlin/2001-03-01. CEN/ISSS WS/E-Sign Workshop Agreement Group F. Available at *http://www.ict.etsi.org/eessi/EESSI-homepage.htm*, March 2001.

[CEN01b] CEN/ISSS WS/E-Sign N 141 Berlin/2001-03-01. CEN/ISSS WS/E-Sign; PT on Area G1; Draft CWA: "Security Requirements for Signature Creation Systems". Available at *http://www.ict.etsi.org/eessi/EESSI-homepage.htm*, March 2001.

[Cha96] David Chappell. *Understanding ActiveX®and OLE*. Microsoft Press, 1996.

[Che00] Zhiqun Chen. *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*. Addison-Wesley, 2000.

[Con98] HAVi Consortium. *The HAVi Specification: Specification of the Home Audio/Video Interoperability Architecture Version 1.0 beta*, November 1998.

[CPV97] Antonio Carzaniga, Gian Petro Picco, and Giovanni Vigna. Designing distributed applications with mobile code paradigms. In *Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, 1997.

[CV65] F. J. Corbató and V. A. Vyssotsky. Introduction and overview of the Multics system. In *Proceedings of 27th AFIPS Joint Computer Conference*, pages 619–628, 1965.

[CZH$^+$99] Steven Czerwinski, Ben Y. Zhao, Todd Hodes, Anthony Joseph, and Randy Katz. An Architecture for a Secure Service Discovery Service. In *Fifth Annual International Conference on Mobile Computing and Networks (MobiCOM '99), Seattle, WA*, August 1999. Draft version, accepted for publication.

[Dal00] Dallas Semiconductor iButton Homepage. *http://www.ibutton.com/*, 2000.

[DB99] Neil Daswani and Dan Boneh. Experimenting with electronic commerce on the PalmPilot. In *Financial Cryptography '99*, number 1648 in Lecture Notes in Computer Science, pages 1–16, Anguilla, BWI, February, 22nd 1999. Springer-Verlag.

[Det78] Jürgen Dethloff. *US Patent No. 4,105,156: Identification system safeguarded against misuse*, August 8, 1978. Available by search at *http://www.uspto.gov/patft/index.html*.

[DG69] Jürgen Dethloff and Helmut Gröttrup. *German Patent No. DE-19-45-777-C3: Identifizierungsschalter*, February 1969. Patent was granted in 1982.

[DH76] W. Diffie and M.E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, November 1976.

[Dij68] Edsger W. Dijkstra. Go To Statement Considered Harmful. *Communications of the ACM*, 11(3):147–148, March 1968.

[DoD85] DoD. *Department of Defense Standard No: DoD 5200.28-STD: Department of Defense Trusted Computer System Evaluation Criteria*, December 1985.

[Dur99] Durlacher Research Ltd. Mobile Commerce Report. Available at *http://www.durlacher.com*, London, November 1999.

[Dur01] Durlacher Research Ltd. UMTS Report – An Investment Perspective. Available at *http://www.durlacher.com*, London, March 2001.

[Eck00] Claudia Eckert. Mobile Devices In eBusiness – New Opportunities And New Risks. In *Proceedings of SIS '2000 Workshop, Zürich, Switzerland*, 2000.

[Eck01] Claudia Eckert. Zur Sicherheit mobiler persönlicher Endgeräte. In *Proceedings of Kommunikationssicherheit KSI '2001, Rot, Germany*, DuD Fachbeiträge. Vieweg Verlag, March 27–28, 2001.

[Edw99] W. Keith Edwards. *Core JINI*. The Sun Microsystem Press - Java Series. Pentice Hall PTR, 1999.

[ETSI00] European Telecommunication Standardization Institution (ETSI), Sophia Antipolis, France. *ETSI EN 300 089 V3.1.1 (2000-12) Integrated Services Digital Network (ISDN); Calling Line Identification Presentation (CLIP) supplementary service; Service description*, December 2000.

[EUP99] Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures. Available at *http://www.ict.etsi.org/eessi/e-sign-directive.pdf*, 1999.

[EUR01] EURESCOM P1005 Project. Project homepage at *http://www.eurescom.de/public/projects/p1000-series/P1005/P1005.htm*, 2001.

[FHW00] M. Freudenthal, S. Heiberg, and J. Willemson. Personal security environment on Palm PDA. In *Proceedings of 16th Annual Computer Security Applications Conference, New Orleans, Louisiana, USA*, December 11–15, 2000.

[Fin99] Klaus Finkenzeller. *RFID-Handbuch*. Carl Hanser Verlag, 1999.

[FIPS95] National Institute of Standards and Technology, U.S. Department of Commerce. *Federal Information Processing Standards Publication (FIPS PUB) 180-1: Secure Hash Standard*, April 1995.

[FMM99] Stefan Fünfrocken, Friedemann Mattern, and Marie-Louise Moschgath. Die Java-Card als Programmier- und Anwendungsplattform für Verteilte Anwendungen. In C. H. Cap, editor, *Proceedings Java-Informations-Tage JIT'99, Düsseldorf, Germany*, Informatik aktuell, pages 100–109. Springer-Verlag, September 20–21, 1999.

[Fün99] Stefan Fünfrocken. Protecting Mobile Web-Commerce Agents with Smartcards. In *Proceedings of ASA/MA'99, Palm Springs, CA*, pages 90–102. IEEE Computer Society Press, October 3–6, 1999.

[Fon98] Philip W. L. Fong. Viewer's discretion: Host security in mobile code systems. Technical Report SFU CMPT TR 1998-19, School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada, May 1998. Available at *http://www.cs.sfu.ca/~pwfong/personal/*.

[Fox97] Dirk Fox. Gateway: IMSI-Catcher. *Datenschutz und Datensicherheit (DuD)*, 21(9), 1997. Available at *http://www.datenschutz-und-datensicherheit.de/jhrg21/imsicatc.htm*.

[Gas88] Morrie Gasser. *Building a Secure Computer System*. Van Nostrand Reinold Company, New York, 1988.

[Gen99] Wolfgang Gentz. Elektronische Geldbörsen in Deutschland. *Datenschutz und Datensicherheit (DuD)*, 23(1), 1999.

[GJS96] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification*. Sun Microsystems Inc., Mountain View, 1996.

[GKP00] Scott Guthery, Roger Kehr, and Joachim Posegga. How to turn a GSM SIM into a Web server. In Josep Domigo-Ferrer, David Chan, and Anthony Watson, editors, *Proceedings of Fourth IFIP TC8/WG8.8 Smart Card Research and Advanced Application Conference CARDIS'2000*, pages 209–222. Kluwer Academic Publisher, September 20–22, 2000.

[GKPV00] Scott Guthery, Roger Kehr, Joachim Posegga, and Harald Vogt. GSM SIMs as Web servers. In *Proceedings of 7th International Conference on Intelligence in Services and Networks IS&N'2000, Athens, Greece*, February 2000. Short paper.

[GM01a] S. Guthery and S. Marks. IP and ARP over ISO 7816. IETF Internet-Draft *draft-guthery-ip7816-01.txt*, January 2001. Expires 2001-07.

[GM01b] S. Guthery and S. Marks. IP/TCP/UDP Header Compression for ISO 7816 Links. IETF Internet-Draft *draft-guthery-ip7816-00.txt*, January 2001. Expires 2001-07.

[GSM02.19] European Telecommunication Standardization Institution (ETSI), Sophia Antipolis, France. *Digital cellular telecommunications system (Phase 2+, Release 98): Subscriber Identity Module Application Programming Interface (SIM API); Service description; Stage 2 (GSM 02.19 version 7.0.0 Release 1998)*, 2000.

[GSM03.19] European Telecommunication Standardization Institution (ETSI), Sophia Antipolis, France. *Digital cellular telecommunications system (Phase 2+): Subscriber Identity Module Application Programming Interface (SIM API); SIM API for Java Card; Stage 2 (GSM 03.19 version 7.1.0, Release 1998)*, 2000.

[GSM03.20] European Telecommunication Standardization Institution (ETSI). *ETSI-GSM Technical Specification: European digital cellular telecommunication system (phase 1); Security-related Network Functions (GSM 03.20 version 3.3.2)*, January 1991.

[GSM03.40] European Telecommunication Standardization Institution (ETSI). *Technical Specification Digital cellular telecommunications system (Phase 2+); Technical realization of the Short Message Service (SMS); (GSM 03.40 version 7.4.0 Release 1998)*, January 2000.

[GSM03.48] European Telecommunication Standardization Institution (ETSI). *Digital cellular telecommunications system (Phase 2+); Security Mechanisms for the SIM application toolkit; Stage 2 (GSM 03.48 version 8.1.0 Release 99)*, November 1999.

[GSM07.05] European Telecommunication Standardization Institution (ETSI). *Technical Specification Digital cellular telecommunications system (Phase 2+); Use of Data Terminal Equipment - Data Circuit terminating; Equipment (DTE - DCE) interface for Short Message Service (SMS) and Cell Broadcast Service (CBS) (GSM 07.05 version 7.0.1 Release 1998)*, July 1999.

[GSM07.07] European Telecommunication Standardization Institution (ETSI). *Technical Specification Digital cellular telecommunications system (Phase 2+); AT command set for GSM Mobile Equipment (ME) (GSM 07.07 version 7.5.0 Release 1998)*, January 2000.

[GSM11.11] European Telecommunication Standardization Institution (ETSI). *Digital cellular telecommunications system (Phase 2+); Specification of the Subscriber Identity Module – Mobile Equipment (SIM–ME) interface (GSM 11.11 version 8.1.0 release 1999)*, November 1999.

[GSM11.14] European Telecommunication Standardization Institution (ETSI). *Digital cellular telecommunications system (Phase 2+); Specification of the SIM Application Toolkit for the Subscriber Identity Module – Mobile Equipment (SIM–ME) interface (GSM 11.14 version 8.1.0 release 1999)*, August 2000.

[GSTY96] Howard Gobioff, Sean Smith, J. D. Tygar, and Bennet Yee. Smart cards in hostile environments. In *Second USENIX Workshop on Electronic Commerce*, pages 23–28, November 1996.

[Gut01] Scott Guthery (ed.). Frequently asked questions (FAQ) of the Internet newsgroup *news:alt.technology.smartcards*. Available at *http://www.scdk.com/atsfaq.htm*, 2001.

[HH98] Dirk Husemann and Reto Hermann. OpenCard Framework. Available at *http://www.ibm.com/developerworks/library/opencard-framework/*, August 1998.

[HNI$^+$98] Jaap Haartsen, Mahmoud Naghshineh, Jon Inouye, Olaf J. Joeressen, and Warren Allen. Bluetooth: Visions, goals, and architecture. *ACM Mobile Computing and Communications Review*, 2(4), October 1998.

[HNSS99] Uwe Hansmann, Martin S. Nicklous, Thomas Schäck, and Frank Seliger. *Smart Card Application Development Using Java*. Springer-Verlag, 1999.

[Hoh98a] Fritz Hohl. Time limited blackbox security: Protecting mobile agents from malicious hosts. In Giovanni Vigna, editor, *Mobile Agents and Security*, number 1419 in Lecture Notes in Computer Science, pages 92–113. Springer-Verlag, 1998.

[Hoh98b] Michael Hohmuth. The Fiasco kernel: Requirements definition. Technical Report TUD-FI98-12, Dresden University of Technology, Dept. of Computer Science, 1998.

[HvE98] C. Hawblitzel and T. von Eicken. A Case for Language-Based Protection. Technical Report TR98-1670, Cornell University, Department of Computer Science, March 1998. Available at *http://www.cs.cornell.edu/slk/papers/TR98-1670.pdf*.

[IAN01] Internet Assigned Numbers Authority. *http://www.iana.org/*, 2001.

[IBM01] IBM. IBM Pervasive Computing Homepage. Available at *http://www.ibm.com/pvc/*, 2001.

[Ice99] ICEBERG Project Home Page. *http://iceberg.cs.berkeley.edu*, 1999.

[IEEE802.11] IEEE. *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 Adopted by the ISO/IEC and redesignated as ISO/IEC 8802-11:1999(E)*, 1999.

[IFH00] Naomaru Itoi, Tomoko Fukuzawa, and Peter Honeyman. Secure internet smartcards. Technical Report 00-6, Center for Information Technology Integration (CITI), University of Michigan, August 2000.

[IrD01] Infrared Data Association. *http://www.irda.org/*, 2001.

[ISO88] *ISO/IEC 7816-2: Identification cards – Integrated circuit(s) cards with contacts – Part 2: Dimensions and locations of the contacts*, 1988.

[ISO89] *ISO/IEC 7816-3: Identification cards – Integrated circuit(s) cards with contacts – Part 3: Electronic signals and transmission protocols*, 1989.

[ISO90a] *ISO/IEC 8824: Information technology, Open Systems Interconnection, Specification of Abstract Syntax Notation One (ASN.1)*, 1990.

[ISO90b] *ISO/IEC 8825: Information technology, Open Systems Interconnection, Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*, 1990.

[ISO92] *ISO/IEC 7816-3 Amd. 1: Identification cards – Integrated circuit(s) cards with contacts – Part 3: Electronic signals and transmission protocols – Amendmend 1: Protocol type T=1, asynchronous half-duplex block transmission protocol*, 1992.

[ISO94] *ISO/IEC 7816-4: Information technology – Identification cards – Integrated circuit(s) cards with contacts – Inter-industry commands for interchange*, July 1994.

[ISO97] International Standardization Organization, JTC 1/SC 22. *ISO/IEC 15145:1997 Standard Information technology – Programming languages – FORTH*, 1997.

[ISO99a] International Organization for Standardization. *ISO/IEC 15408-1:1999 Information technology, Security techniques, Evaluation criteria for IT security, Part 1: Introduction and general model*, 1999.

[ISO99b] International Organization for Standardization. *ISO/IEC 15408-2:1999 Information technology, Security techniques, Evaluation criteria for IT security, Part 3: Security assurance requirements*, 1999.

[ISO99c] International Organization for Standardization. *ISO/IEC 15408-3:1999 Information technology, Security techniques, Evaluation criteria for IT security, Part 1: Introduction and general model*, 1999.

[Jab96] David P. Jablon. Strong Password-only Authenticated Key Exchange. *ACM Computer Communications Review*, 26(5), October 1996.

[JCF01] Java card forum web site. *http://www.javacardforum.org*, 2001.

[Kar00] Günter Karjoth. Secure mobile agent-based merchant brokering in distributed marketplaces. In David Kotz and Friedemann Mattern, editors, *Proceedings of Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents ASA/MA '2000, Zurich, Switzerland*, number 1882 in Lecture Notes in Computer Science, pages 44–56. Springer-Verlag, 2000.

[KG98] Lora L. Kassab and Steven J. Greenwald. Formalizing the Java Security Architecture of JDK 1.2. In *5th European Symposium on Research in Computer Security (ESORICS)*, Lecture Notes in Computer Science, Louvain-la-Neuve, Belgium, 1998. Springer Verlag.

[KK99] Oliver Kömmerling and Markus G. Kuhn. Design Principles for Tamper-Resistant Smartcard Processors. In *USENIX Workshop on Smartcard Technology*, 1999.

[KM01] Roger Kehr and Hendrik Mieves. SIMspeak: Towards an open and secure platform for GSM SIMs. In I. Attali and T. Jensen, editors, *Proceedings of International Conference on Research in Smart Cards: Smart Card Programming and Security, E-smart 2001, Cannes, France,*, volume 2140 of *Lecture Notes in Computer Science*. Springer-Verlag, September, 19–21 2001.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology (Crypto'96)*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, 1996.

[KP01] Roger Kehr and Joachim Posegga. Verfahren und Vorrichtung zum digitalen Signieren einer Transaktion. German Patent, Deutsches Patent- und Markenamt, 2001.

[KPS95] C. Kaufman, R. Perlman, and M. Speciner. *Network Security – Private Communication in a Public World.* Prentice-Hall, 1995.

[KPSW01] Roger Kehr, Joachim Posegga, Roland Schmitz, and Peter Windirsch. Mobile Security for Internet Applications. In *Proceedings of Kommunikationssicherheit KSI'2001*, DuD Fachbeiträge. Vieweg Verlag, March 27–28, 2001.

[KPV99a] Roger Kehr, Joachim Posegga, and Harald Vogt. PCA: Jini-based Personal Card Assistant. In R. Baumgart, editor, *Proceedings of Secure Networking – CQRE [Secure]'99, Düsseldorf, Germany*, volume 1740 of *Lecture Notes in Computer Science*, pages 64–75. Springer-Verlag, November 30 – December 2, 1999.

[KPV99b] Roger Kehr, Joachim Posegga, and Harald Vogt. *Verfahren zur Erhöhung der Sicherheit bei digitalen Unterschriften, DE 199 23 807 A 1*. Deutsches Patent- und Markenamt, May 19th, 1999.

[KPV00] Roger Kehr, Joachim Posegga, and Harald Vogt. *Kartenterminal und Verfahren zum Betreiben eines Kartenterminals, DE 100 15 775 A 1*. Deutsches Patent- und Markenamt, March 30th, 2000.

[KRV00a] Roger Kehr, Michael Rohs, and Harald Vogt. Issues in smartcard middleware. In Isabelle Attali and Thomas Jensen, editors, *Proceedings of First International Workshop, JavaCard '2000, Cannes, France*, number 2041 in Lecture Notes in Computer Science. Springer-Verlag, September 14th, 2000.

[KRV00b] Roger Kehr, Michael Rohs, and Harald Vogt. Mobile code as an enabling technology for service-oriented smartcard middleware. In *Proceedings of Second International Symposium on Distributed Objects and Applications DOA'2000, Antwerp, Belgium*, pages 119–130. IEEE Computer Society, September 20–23, 2000.

[KVZ99] Roger Kehr, Harald Vogt, and Andreas Zeidler. Towards a generic proxy execution service for small devices. In *Proceedings of Workshop on Future Services for Networked Devices (FuSeNetD'99), Heidelberg*, November 8–9, 1999.

[Lam01] Thierry Lamotte. IP Smart Cards in the (Not So) Distant Future. ETSI Project Smart Card Platform Meeting #5, Palm Springs, USA, March 20–23, 2001. Available at *http://research.gemplus.com/smart/r_d/publications/index.html*.

[Ler01] Xavier Leroy. On-card bytecode verification for Java Card. In *Proceedings of eSmart '2001, Cannes, France*, volume 2140 of *Lecture Notes in Computer Science*. Springer-Verlag, September 2001.

[LM99] Sergio Loureiro and Refik Molva. Function hiding based on error correcting codes. In Manuel Blum and C.H. Lee, editors, *Proceedings of Cryptec'99, International workshop on cryptographic techniques and electonic commerce, Hong Kong*, pages 92–98, July 1999.

[LM00] Sergio Loureiro and Refik Molva. Mobile code protection with smartcards. In *Proceedings of ECOOP '2000 Workshop on Mobile Object Systems Sophia Antipolis, France*, June 13th, 2000.

[LY99] Tim Lindholm and Frank Yellin. *The Java Virtual Machine Specification*. Sun Microsystems Inc., Mountain View, second edition edition, 1999.

[Mat01] Friedemann Mattern. Ubiquitous Computing – Der Trend zur Informatisierung und Vernetzung aller Dinge (in German). In *Der Weg in die mobile Informationsgesellschaft, Tagungsband 6. Deutscher Internet-Kongress*. dpunkt-Verlag, September 2001. Available at *http://www.inf.ethz.ch/vs/publ/*.

[Met01] MetaMata. JavaCC compiler toolkit. Available at *http://www.metamata.com/javacc/*, 2001.

[Mic96] Microsoft Corporation. DCOM Technical Overview. Available at *http://www.microsoft.com/com/*, November 1996.

[Mic00] Microsoft Windows for Smart Cards Homepage. Available at *http://www.microsoft.com/smartcard*, 2000.

[Mic01] Microsoft Visual Basic Homepage. Available at *http://msdn.microsoft.com/vbasic/*, 2001.

[Mol00] Raphael Molina. Delegated exchange of privacy information within context-aware applications. Master's thesis, Darmstadt University of Technology, September 2000.

[MOV97] Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[MP92] Michel Mouly and Marie-Bernadette Pautet. *The GSM System for Mobile Communications*. Cell & Sys, 1992. Available at *http://www.TelecomPublishing.com*.

[mSi00] The Mobile Electronic Signature Consortium. *mSign Protocol Specification Version 1.0*, October 2000. Available at *http://www.msign.org*.

[MT79] Robert Morris and Ken Thompson. Password security: A case history. *CACM*, 22(11):594–597, 1979.

[Mye99] Andrew C. Myers. *Mostly-Static Decentralized Information Flow Control*. Technical report mit/lcs/tr-783, Massachusetts Institute of Technology, January 1999.

[Nid99] Michael Nidd. Service Discovery in DEAPspace. Technical Report RZ 3149 (#93195), IBM Research Division, Zürich Research Laboratory, June 1999. *http://deapspace.zurich.ibm.com*.

[NL96] George C. Necula and Peter Lee. Safe kernel extensions without run-time checking. In *Second USENIX Symposium on Operating Systems Design and Implementation (OSDI '96), Seattle, Washington,* pages 229–243. ACM Press, October 28–31 1996.

[NT94] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. *IEEE Communications Magazine*, 32(9):33–38, September 1994.

[Oak98] Scott Oakes. *Java Security*. O'Reilly, 1998.

[OCF99] OpenCard Consortium. *OpenCard Framework 1.2 Programmer's Guide*, fourth edition, December 1999. Available at [OCF00].

[OCF00] OpenCard Consortium Homepage, 2000. *http://www.opencard.org*.

[OLW97] John K. Ousterhout, Jacob Y. Levy, and Brent B. Welch. The safe-tcl security model. Technical Report TR-97-60, Sun Microsystems Laboratories, 1997.

[OMG00] Object Management Group. *The Common Object Request Broker Architecture Specification Revision 2.4*, October 2000. Available at *http://www.omg.org*.

[Ott01] Heinz Otter. Die österreichische Bürgerkarte mit Sozialversicherungs- und elektronischer Signaturfunktion. In *11. GMD-SmartCard Workshop, Darmstadt, Germany*. GMD – Forschungszentrum Informationstechnik GmbH, SIT, Darmstadt, February 6–7, 2001.

[Ovum99] Ovum Ltd. Smart Card Systems: Multi-application Technologies and Strategies, June 1999. Available at *www.ovum.com*.

[Pay01] Paybox homepage. Available at *http://www.paybox.de*, 2001.

[PC99] Stephan Preuß and Clemens. H. Cap. Overview of spontaneous networking – evolving concepts and technologies. In *Proceedings of Workshop on Future Services for Networked Devices (FuSeNetD'99), Heidelberg*, November 8–9, 1999.

[PCS00] PC/SC Workgroup. *http://www.pcscworkgroup.com*, 2000.

[PKCS#11] RSA Laboratories. *PKCS #11 v2.10: Cryptographic Token Interface Standard*, December 1999. Available at *http://www.rsalabs.com/rsalabs/pkcs/pkcs-11/*.

[PPSW95] Andreas Pfitzmann, Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Vertrauenswürdiger Entwurf portabler Benutzerendgeräte und Sicherheitsmodule. In *Proceedings of Verläßliche Informationssysteme VIS'95*, pages 329–350. Vieweg, 1995.

[PPSW96] Andreas Pfitzmann, Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Mobile user devices and security modules: Design for trustworthiness. Technical Report RZ 2784 (#89262), IBM Research Division, Zurich, May 1996.

[PRS$^+$01] Birgit Pfitzmann, James Riordan, Christian Stüble, Michael Waidner, and Arnd Weber. The PERSEUS system architecture. Technical Report RZ 3335 (#93381), IBM Research Division, Zurich, April 2001.

[PV98] Joachim Posegga and Harald Vogt. Byte code verification for Java smart cards based on model checking. In *5th European Symposium on Research in Computer Security (ESORICS'98), Louvain-la-Neuve, Belgium*, volume 1485 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[PvdBJ00] Erik Poll, Joachim van den Berg, and Bart Jacobs. Specification of the javacard api in jml. In *Proceedings of CARDIS '2000*, Bristol, UK, September 2000. Kluwer Academic Publisher.

[PVGW00] Henning Pagnia, Holger Vogt, Felix C. Gärtner, and Uwe G. Wilhelm. Solving Fair Exchange with Mobile Agents. In *Second International Symposium on Agent Systems and Applications and Fourth International Symposium on Mobile Agents (ASA/MA2000)*, volume 1882 of *Lecture Notes in Computer Science*, pages 57–72, Zurich, Switzerland, 2000.

[RCW01] Algis Rudys, John Clements, and Dan S. Wallach. Termination in language-based systems. In *Network and Distributed Systems Security Symposium '01, San Diego, California*, February 2001.

[RE97] W. Rankl and W. Effing. *Smart Card Handbook*. John Wiley & Sons, Winchester, New York, Heidelberg, 1997.

[RFC0761] J. Postel. DOD Standard: Transmission Control Protocol. Internet RFC 761, January 1980.

[RFC1034] P. Mockapetris. Domain Names - Concepts and Facilities. Internet RFC 1034, November 1987.

[RFC1057] Sun Microsystems Inc. RPC: Remote Procedure Call Protocol Specification Version 2 (ONCRPC). Internet RFC 1057, June 1988.

[RFC1321] R. Rivest. The MD5 message-digest algorithm. Internet RFC 1321, April 1992.

[RFC1661] W. Simpson. The Point-to-Point Protocol (PPP). Internet RFC 1661, July 1994.

[RFC1945] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0. IETF RFC 1945, May 1996.

[RFC2045] N. Freed and N. Borenstein. Multipurpose internet mail extensions (MIME) part one: Format of internet message bodies. Internet RFC 2045, November 1996.

[RFC2131] R. Droms. Dynamic host configuration protocol (DHCP). Internet RFC 2131, March 1997.

[RFC2165] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service Location Protocol (SLP). Internet RFC 2165, June 1997.

[RFC2246] T. Dierks and C. Allen. The TLS Protocol Version 1.0. Internet RFC 2246, January 1999.

[RFC2535] D. Eastlake. Domain Name System Security Extensions. Internet RFC 2535, March 1999.

[RH99] Jim Rees and Peter Honeyman. Webcard: A Java Card Web Server. Technical Report 99-3, Center for Information Technology Integration (CITI), University of Michigan, 1999.

[RH00] Jim Rees and Peter Honeyman. Webcard: A Java Card Web Server. In *Proceedings of CARDIS '2000*, Bristol, UK, September 2000. Kluwer Academic Publisher.

[Roh00] Michael Rohs. Konzeption und Realisierung einer Integration von Smartcards in Jini-Föderationen (in German). Master's thesis, Darmstadt University of Technology, Department of Computer Science, April 2000.

[RV01] Michael Rohs and Harald Vogt. *Smart Card Applications and Mobility in a World of Short Distance Communication, CASTING Project*. Distributed Systems Group, ETHZ Zurich, January 2001. Available at *http://www.inf.ethz.ch/vs/res/proj/casting.html*.

[SA99] Frank Stajano and Ross Anderson. The Resurrecting Duckling: Security issues for ad-hoc wireless networks. In B. Christianson, B. Crispo, and M. Roe, editors, *Proceedings of 7th International Workshop on Security Protocols, Heidelberg*, Lecture Notes in Computer Science. Springer-Verlag, April 1999. Available at *http://www.cl.cam.ac.uk/~fms27/duckling/*.

[Sal97] Salutation Consortium. Salutation Architecture Specification. *http://www.salutation.org*, October 1997.

[SAT00] SIMalliance Limited. *SBC: S@T Byte Code Technical Specification 01.00 v1.0.3*, June 2000. Available at *http://www.simalliance.org*.

[Sch96] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc, 1996.

[Sch00] Bruce Schneier. *Secrets and Lies – Digital Security in a Networked World*. Wiley Computer Publishing, 2000.

[Sch01] Karl Scheibelhofer. What you see is what you sign. In *Proceedings of IFIP conference on Communications and Multimedia Security (CMS '2001), Darmstadt, Germany*, May 21–22, 2001.

[SigG97] Deutscher Bundestag. Gesetz zur digitalen Signatur. *http://www.regtp.de/Fachinfo/Digitalsign/neu/rechtsgr.htm*, 22 July 1997. English Version ("Digital Signature Act") available from *http://www.regtp.de/English/laws/download.htm*.

[SigV97] Deutscher Bundestag. Verordnung zur digitalen Signatur. *http://www.regtp.de/Fachinfo/Digitalsign/neu/rechtsgr.htm*, 22 July 1997. English Version ("Digital Signature Ordinance") available from *http://www.regtp.de/English/laws/download.htm*.

[SK00] Tage Stabell-Kulø. Smartcards: How to put them to use in a user-centric system. In *Second International Symposium On Handheld and Ubiquitous Computing (HUC2k), Bristol, UK*, volume 1927 of *Lecture Notes in Computer Science*, pages 200–210, September 25–27 2000.

[Sla01] Slashdot. "don't trust code signed by 'microsoft corporation". Available at *http://slashdot.org/articles/01/03/22/1947233.shtml*, March, 22nd 2001.

[SLB00] Schlumberger, Inc.: Cyberflex Simera™, Technical Information. Available at *http://www.cyberflex.com/Products/MobileCom/simera/simera.html*, 2000.

[Smi96] Sean W. Smith. Secure coprocessing applications and research issues. Technical Report LA-UR-96-2805, Los Alamos National Laboratory, August 1996.

[SSF99] Jonathan S. Shapiro, Jonathan M. Smith, and David J. Farber. EROS: a fast capability system. In *Seventeenth ACM Symposium on Operating Systems Principles, Charleston*, pages 170–185, December 12–15 1999.

[ST98a] T. Sander and C. Tschudin. Towards mobile cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, 1998. IEEE Computer Society Press.

[ST98b] Tomas Sander and Christian F. Tschudin. Protecting mobile agents against malicious hosts. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 44–60. Springer-Verlag, 1998.

[Stü00] Christian Stüble. Development of a Prototype for a Security Platform for Mobile Devices. Master's thesis, University of Dortmund and University of Saarland, 2000.

[Sun96] Sun Microsystems Inc. *Manifest and Signature Specification*, 1996. *http://java.sun.com/products/jdk/1.2/docs/guide/jar/manifest.html*.

[Sun98] Java Object Serialization Specification. *http://java.sun.com/j2se/1.3/docs/guide/serialization/*, 1998.

[Sun99a] Sun. *Jini Architecture Specification – Revision 1.0*. Sun Microsystems Inc., January 1999.

[Sun99b] Sun. *Jini Discovery and Join Specification – Revision 1.0*. Sun Microsystems Inc., January 1999.

[Sun99c] Sun. *Jini Lookup Service Specification – Revision 1.0*. Sun Microsystems Inc., January 1999.

[Sun00a] Sun. *Java Card 2.1.1 Runtime Environment Specification*, 2000. Available at *http://java.sun.com/products/javacard/*.

[Sun00b] Sun. *Java Card 2.1.1 Virtual Machine Specification*, 2000. Available at *http://java.sun.com/products/javacard/*.

[Sun01a] Sun Microsystems Inc. *Jini Technology IP Interconnect Specification Version 0.7*, March 2001. Available at *http://developer.jini.org/exchange/projects/surrogate/IP/*.

[Sun01b] Sun Microsystems Inc. *Jini Technology Surrogate Architecture Specification Version 0.7*, March 2001. Available at *http://developer.jini.org/exchange/projects/surrogate/*.

[Sur01] Smart Card Project Homepage at Jini.org. Available at *http://developer.jini.org/exchange/projects/smartcard/*, January 2001.

[Swi00] Swisscom. Swiss Patent No. PCT/EP00/09121: Method for securing communications between a terminal and an additional user equipment, 2000.

[Tab00] Hugo Taborda. Analysis and Evaluation of a Server-side Secure Sockets Layer Implementation on Smartcards. Master's thesis, Darmstadt University of Technology, July 2000.

[TCPA00] Trusted Computing Platform Alliance. *Trusted Computing Platform Alliance (TCPA): TCPA Design Philosophies and Concepts Version 1.0*, 2000. Available at *http://www.trustedpc.org*.

[UDDI01] UDDI Consortium. UDDI Web site. *http://www.uddi.org/*, 2001.

[UPn99a] Univeral Plug and Play Device Architecure Specification Version 0.9. Available at [UPn99b], November 1999.

[UPn99b] Universal Plug and Play Homepage. *http://www.upnp.org*, 1999.

[Uri00] Pascal Urien. Internet card, a smart card as a true Internet node. *Computer Communications, Elsevier Science*, 23(17):1655–1666, 2000.

[Uri01a] Pascal Urien. Programming internet smart card with XML scripts. In *Proceedings of eSmart '2001, Cannes, France*, volume 2140 of *Lecture Notes in Computer Science*. Springer-Verlag, September 2001.

[Uri01b] Pascal Urien. *Proposal for a standard smartcard communication stack interface*. Bull CP8 R&D, March 2001. Submitted to the Java Card Forum [JCF01].

[US01] ID Theft: U.S. government's central website for information about identity theft. Available at *http://www.consumer.gov/idtheft/*, 2001.

[UST00] Pascal Urien, Hayder Saleh, and Adel Tizraoui. Internet card, a smart card for Internet. In *Protocols for Multimedia Systems PROMS '2000, Cracow, Poland*, October 22–25, 2000. Proceedings are available from *http://www.kt.agh.edu.pl/research/conf/proms00/*.

[Vig98] G. Vigna. Cryptographic traces for mobile agents. In G. Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*. Springer-Verlag, June 1998.

[VSI96] Dennis Volpano, Geoffrey Smith, and Cynthia Irvine. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187, 1996.

[VV98] Jean-Jacques Vandewalle and Eric Vétillard. Developing Smart Card-Based Applications using Java Cards. In *Proceedings of the Third Smart Card Research and Advanced Application Conference (CARDIS'98)*, Louvain-la-Neuve, Belgium, September 1998.

[W3C99] W3C. *XSL Transformations (XSLT) Version 1.0 W3C Recommendation 16 November 1999*, November 1999. Available at *http://www.w3.org/TR/xslt.html*.

[W3C00] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, and Eve Maler. *Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation 6 October 2000*. W3C, October 2000. Available at *http://www.w3.org/TR/2000/REC-xml-20001006*.

[W3C01] W3C. *XML Schema Part 0–2: W3C Recommendation, 2 May 2001*, May 2001. Available at *http://www.w3.org/XML/Schema/*.

[Wal99] Jim Waldo. The Jini Architecture for Network-centric Computing. *Communications of the ACM*, 42(7):76–82, July 1999.

[WAP98a] Wireless Application Protocol Forum. *http://www.wapforum.org*, 1998. Hosts downloadable specifications.

[WAP98b] Wireless Application Forum Ltd. *Wireless Application Protocol Architecture Specification Version Version 30-Apr-1998*, April 1998. Available at [WAP98a].

[WAP99a] Wireless Application Forum Ltd. *Wireless Application Protocol Wireless Markup Language Specification Version 1.2*, November 1999. Available at [WAP98a].

[WAP99b] Wireless Application Forum Ltd. *Wireless Application Protocol Wireless Transport Layer Security Specification Version 05-Nov-1999*, November 1999. Available at [WAP98a].

[WAP99c] Wireless Application Protocol Forum, Ltd. *WAP Push Architectural Overview Version 1999-11-08*, November 1999. Available at [WAP98a].

[WB96] Mark Weiser and John Seely Brown. The coming age of calm technology. Available at *http://www.ubiq.com/hypertext/weiser/acmfuture2endnote.htm*, October 1996.

[WBS98] U. G. Wilhelm, L. Buttyàn, and S. Staamann. On the problem of trust in mobile agent systems. In *Symposium on Network and Distributed System Security*, pages 114–124, San Diego, CA, USA, March 1998. Internet Society.

[WE99] Jane Kaufman Winn and Carl Ellison. *http://www.ftc.gov/bcp/icpw/comments/revwin~1.htm*, March 1999.

[Wei91] Mark Weiser. The Computer for the 21st Century. *Scientific American*, pages 94–104, September 1991.

[WHFG92] R. Want, A. Hopper, V Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10(1), 1992.

[Wil99] Uwe Wilhelm. *A Technical Approach to Privacy based on Mobile Agents Protected by Tamper-resistant Hardware*. PhD thesis, École Polytechnique Fédérale de Lausanne (EPFL), 1999. Thése N. 1961.

[WSZ01] Konrad Wrona, Marko Schuba, and Guido Zavagli. Mobile Payments – State of the Art and Open Problems. In Ludger Fiege, Gero Mühl, and Uwe Wilhelm, editors, *Proceedings of Second International Workshop on Electronic Commerce, WELCOM '2001, Heidelberg, Germany*, volume 2232 of *Lecture Notes in Computer Science*, pages 88–100. Springer-Verlag, November, 16–17 2001.

[Yee94] Bennet Yee. *Using secure coprocessors*. PhD thesis, Carnegie Mellon University, May 1994.

[Yee99] B. S. Yee. A sanctuary for mobile agents. In J. Vitek and C. Jensen, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Systems*, number 1603 in Lecture Notes in Computer Science, pages 261–273. Springer-Verlag, 1999.

[YKS$^+$01] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen. SSH Connection Protocol. Available as IETF Draft *draft-ietf-secsh-connect-09.txt*, January 2001.

[YT95] Bennett Yee and J. D. Tygar. Secure Coprocessors in Electronic Commerce Applications. In *Proceedings of the First USENIX Workshop of Electronic Commerce, New York*, pages 155–170, July 1995.

[Zob01] Rosalie Zobel. Smart cards for eEurope. In Matthias Fluhr, editor, *Die Chipkarte: Neue Sicherheitskonzepte und Wertschöpfungsmodelle, Proceedings of OMNICARD ’2001, Berlin*, pages 10–18. inTIME Berlin, January 16–18, 2001.

Curriculum Vitae

Name	Roger Kilan-Kehr geb. Kehr
Geburtsdatum	22. Januar 1970 in Marburg/Lahn

Schule

Aug 76 – Jul 80	Grundschule in Hatzbach und Stadtallendorf
Aug 80 – Jun 89	Schwalmgymnasium in Treysa mit Abschluß der allgemeinen Hochschulreife
Jul 89 – Sep 90	Wehrdienst in Stadtallendorf

Studium

Okt 90 – Mär 91	Studium der Mathematik an der TH Darmstadt
Apr 91 – Sept 97	Studium der Informatik an der TH Darmstadt
Sep 94 – Mär 95	Auslandsaufenthalt im Rahmen des ERASMUS-Programms an der Universidade Nova de Lisboa (Lissabon)
Sep 97	Abschluß als Diplom-Informatiker

Berufliche Tätigkeit

Okt 97 – Dez 98	Mitarbeiter bei Net & Publication Consultance GmbH, Rödermark
Jan 99 – Dez 01	Wissenschaftlicher Mitarbeiter am Fachgebiet Datenbanken und Verteilte Systeme und am Information Technology Transfer Office (ITO) des Fachbereichs Informatik, TU Darmstadt
Sep 00 – Dez 01	Kollegiat des Graduiertenkollegs *Infrastrukturen für den elektronischen Markt* and der TU Darmstadt
seit Jan 99	Wissenschaftlicher Mitarbeiter bei T-Systems Nova GmbH, Bereich Informationssicherheit, Technologiezentrum Deutsche Telekom AG, Darmstadt