PCA: Jini-based Personal Card Assistant

Roger Kehr, Joachim Posegga, and Harald Vogt

Deutsche Telekom AG, Technologiezentrum
IT Security Research/FE34
D-64307 Darmstadt
{Kehr|Posegga|VogtH}@tzd.telekom.de

Abstract. We describe the Personal Card Assistant, 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 digital signatures: in particular, we can circumvent the problems involved with untrusted document viewers in this context.
We consider what sort of network infrastructure is required for using the PCA and outline how Jini can be used for integrating the PDA and smartcards into unknown service networks a mobile user is confronted with.

1 Introduction

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 paper we investigate an application area where this problem is very evident, i.e.: the presentation of a document that is to be digitally signed.

Digital signatures for applications like electronic commerce require high security standards. Some countries have already proposed to embed digital signatures into legal frameworks, the most prominent example being the German digital signature law "Signaturgesetz" [1, 2]: This law requires (among other things) the following ITSEC security levels for a system used for dealing with digital signatures:

- The storage component for the secret key (usually a smartcard) must meet the criteria of ITSEC E4 [3], and the
- component for presenting a document (document viewer) must meet ITSEC E2.

Both requirements form the basis for digital 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 ITSEC E2, the PC Software offers little protection against manipulation. This is in particular problematic if the platform used for viewing such a document is not "under control" of the digitally 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 digital signatures in practice.

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

This paper proposes a pragmatic approach that reduces the risks of using digital signatures by integrating a customer's PDA into the creation of digital 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* (PCA). 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 therefore *is under the control of* a person who wishes to apply a digital signature, such a device will *in practice* be more trustworthy for that person than, for instance, a vendor's PC.

We therefore regard our approach as *pragmatically more secure*, making users of digital signature feel more comfortable with the technology. The notion "trust amplifier" for such a PCA covers this quite precisely. We will see in the sequel of this paper that this at first sight very straightforward idea opens up a number of very interesting questions from a technological and research perspective.

The paper is organised as follows: Section 2 introduces the concepts and components behind the Personal Card Assistant, Sect. 3 describes and analyses the cryptographic protocol used in our scenario. Section 4 investigates the requirements of a network and service infrastructure for implementing the PCA scenario and proposes a Jini-based approach for it. We discuss related work in Section 5 and finally draw conclusions from our work in Section 6.

2 The Personal Card Assistant

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 authorised against the card by entering a PIN code (often referred to as card holder verification, CHV). As 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. It is planned to integrate

keyboards, biometric sensors and screens directly on the card, but that is not yet common.

Thus, smartcard-based applications rely upon the trustworthiness of the environment the card is working in. Our PCA scenario aims at improving this situation; the PCA 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 they can be coupled by cryptographic means . We consider the latter case, 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 personalised 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. But it is also possible to run the application on the smartcard and use the PDA simply as a supplementary device; this is comparable to the GSM SIM AT approach [4] where an application running on the smartcard controls the operation of a mobile phone.

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 digital 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.

3 The PCA for Digital Signatures

We present a scenario where the use of the PCA can enhance the process of creating a digital signature. Our 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.

Our 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.

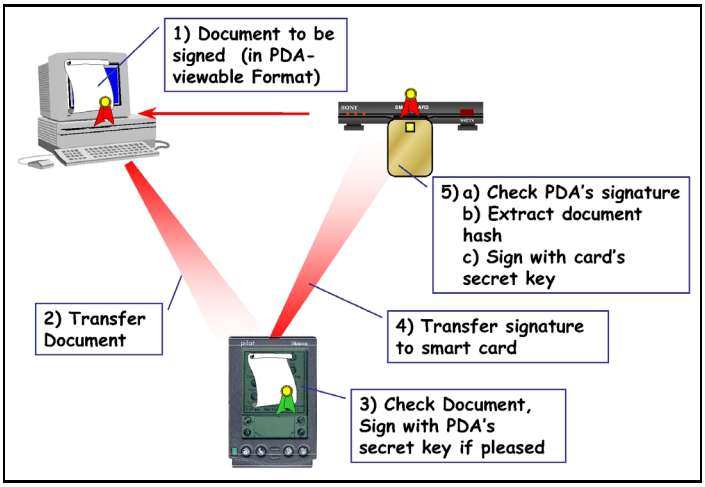

Fig. 1. The PCA in the Context of Signing Documents

We require that each of 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 figure using the PDA's infrared interface.

3.1 A Bird's Eye View of the Scenario

Figure 1 outlines the interworking of the components forming up our approach:

1. 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.
2. The document is transferred to the PDA.
3. The user checks the document on the PDA and approves it by signing the document's hash with the PDA's secret key.
4. The document hash is transferred to the smartcard, which extracts the document's hash value again and creates the final signature.

This procedure differs from the standard approach to using digital signatures in two important points: First, we "route" the document over the PDA for being checked by the signing person; second, we assume 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.

3.2 The Underlying Cryptographic Protocol

Hereafter, we will use the identifiers E_{Crd} and D_{Crd} for denoting the smartcard's public and private keys respectively, and similarly E_{PDA} and D_{PDA} for the

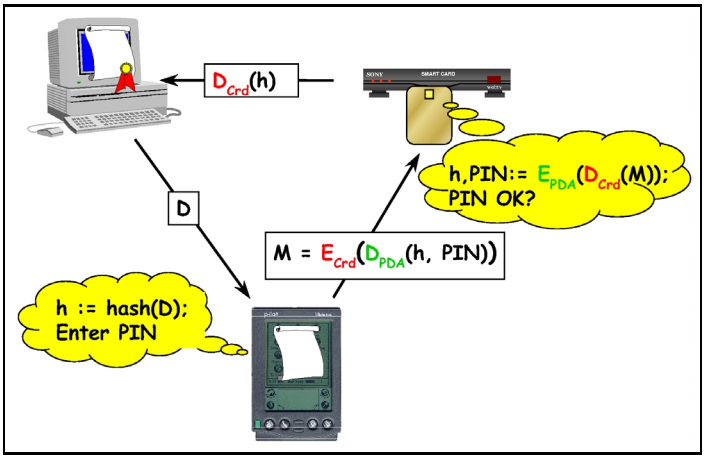

Fig. 2. Cryptographic View of Information Flow

PDA. The application of a key K to a message M, e.g. encrypting the message, will be denoted by $K(M)$.

Figure 2 visualizes the communication between the components forming our approach:

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

PDA's Function: 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.[1]

PDA → Smartcard: The PDA sends the message

$$M = E_{\text{Crd}}\big(D_{\text{PDA}}(h, \text{PIN})\big)$$

to the card.
In plain English: 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's Function: The card deciphers the message by computing

$$(h, \text{PIN}) = E_{\text{PDA}}\big(D_{\text{Crd}}(M)\big)$$

[1] 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.

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 $D_{\mathrm{Crd}}(h)$ 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.

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

3.3 Informal Threat Analysis

Under the assumption that the PDA and the card of the scenario described in Section 3.2 are trustworthy, the protocol can only be attacked by manipulating data sent between the components:

PC → 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 → Card. Under the assumption of secure cryptographic algorithms and sufficient key lengths, the contents of the message M is not reconstructible[2]. 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 → PC. Since the completely generated signature is transmitted only, there is no meaningful attack left. The signature can be easily verified by the vendor or anybody who is interested.

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,

[2] Note that encrypting the PIN alone without the hash h 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.

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.

4 Infrastructures for the PCA

In the previous sections we have shown the usefulness of the personal card assistant by giving an example application – digitally signing documents in untrusted environments. Still missing is an evaluation of the practical feasibility of our approach and suggestions how the infrastructure for environments that support PCAs could look like: the PCA scenario is supposed to be used in environments, where network architecture, names and location of servers, card readers, etc. are unknown, so we need to have means to integrate the PCA into a local service network.

4.1 Jini

In distributed and mobile scenarios like ours it is important that communication partners such as the PDA and a vendor terminal find each other seamlessly and efficiently. Jini [5, 6] is a recently released technology from Sun Microsystems for federating network devices and services. It explicitly addresses many of the problems involved around establishing spontaneous client-server interaction in *a-priori* unknown environments.

Jini is based on the Java facilities for code shipping among different Java virtual machines (JVMs). A so-called *lookup* service [7] acts as the central registration authority for services. Arbitrary services register with the lookup service using a bootstrapping protocol [8] and provide a serialised Java object called *service proxy* with some additional descriptive information. Potential Jini-clients contact the lookup service to query for services they are interested in and download and invoke the associated proxy objects. These proxies are then executed in the virtual machine of the client and implement the communication with the (mostly) remote Jini-service they where launched from.

We have chosen Jini as a middle tier for dynamically establishing connections between the different services that comprise our scenario since it seems to offer much of the functionality needed.

4.2 PCA-Scenario with a Jini Infrastructure

In our PCA scenario we envision a Jini-enabled infrastructure consisting of a lookup service, a PDA, a card reader, and some other service such as a vendor's terminal. The PDA and the card reader must themselves register with the lookup service to offer the services relevant for the application.

In our scenario the PDA would offer a *Document Signing Service* that displays the contract document to the customer (cf. Figure 3, upper part). If she accepts the document after reading it on the PDA, she inserts her signature smartcard

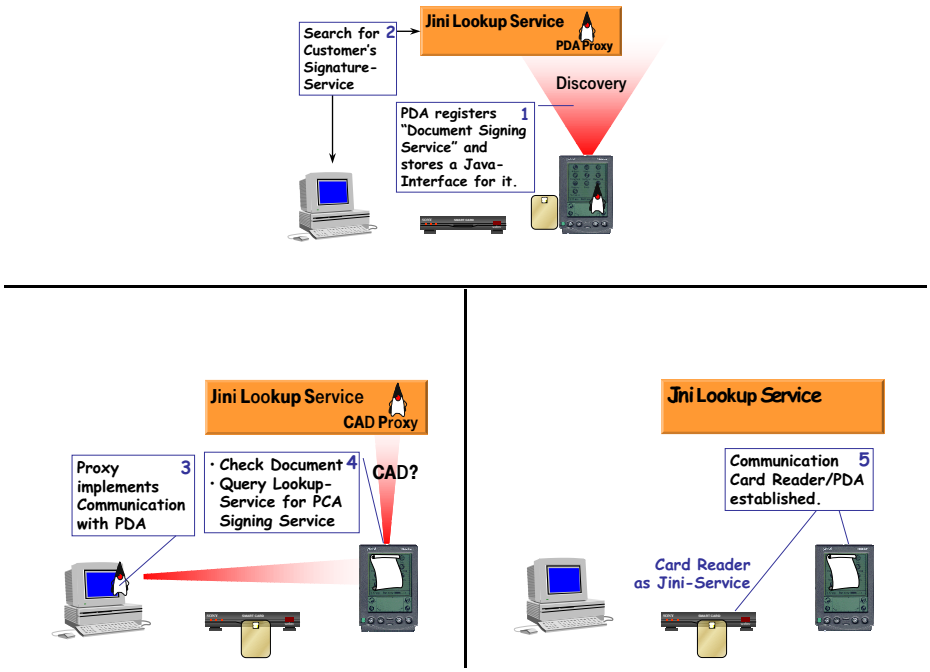

Fig. 3. PCA and Jini

into the vendor's card reader. The reader recognises the smartcard and registers a *PCA Signing Service* with the lookup-service (cf. Figure 3, bottom left). The PDA receives the respective proxy object and passes the encrypted and signed document's hash value to the proxy. The proxy encodes the data and sends it to the smartcard (cf. Figure 3, bottom right).

The smartcard reconstructs the document's hash value from the received data. It applies the signature key to it – thus creating the digital signature – and returns it to the PDA's proxy object, which in turn returns the final signature to its client.

4.3 Integration of the PDA into the Jini Federation

We have chosen the 3COM Palm Pilot III for implementing our prototype with since it is in widespread use and many tools are available on the net. It offers a serial connection via the cradle and an infrared device that is compliant to the IrDA standard [9]. Future PDAs might also use wireless technologies such as Bluetooth [10, 11].

On the hardware level the local net must offer an entry point for the PDA. In any case there must be a point-of-presence the PDA can be connected to on the hardware level which in our case is a standard PC.

Since to our knowledge there doesn't exist a fully Java 2-compliant JVM for the Pilot yet, we needed to separate the functionality of the Document Signing Service into a Jini-part which runs on the host the PDA is connected to, and the application on the Pilot implementing the display engine and the encryption algorithms. Both parts of the service communicate via a protocol running over the serial line. Hence, the registration of the service with the lookup service is done on the host, whereas the security-critical part of the application runs on the Pilot.

4.4 Integration of Smartcards into the Jini Federation

A smartcard reader device can support Jini either by directly integrating a Java VM within the reader or by simply attaching it to a device bay –a technique for making non-Java devices available to Jini federations [12]– or a workstation where a dedicated process performs Jini tasks. Thus, it is able to act as an ordinary Jini device, registering services to employ smartcards.

The types of services offered can range from basic interfaces accessing the primitive functions of the reader to sophisticated services corresponding to the functionality of inserted smartcards. This is similar to approaches like the OpenCard Framework [13, 14] and PC/SC [15]. These frameworks hold static information about smartcards and their services offered. Both supply high-level interfaces to applications for accessing smartcards as well as direct access to readers.

For Jini, it is desired to detect the functionality of a smartcard dynamically, when it is inserted into the reader. After that, corresponding services can be registered with the lookup service. The service proxies should offer application-level services which hide the characteristics of their implementation on the smartcard.

Especially Java smartcards have dynamic functionality, i.e. new applets can be downloaded to the card while others might be deleted from the card. Unfortunately, directory services keeping track of the installed applets are manufacturer-specific. A generic service detection facility must take this into account. The ATR (answer-to-reset) string sent by a card when connected gives at least static information about the card, e.g. the manufacturer, card type etc. This information can be used to load a procedure (implemented by a Java class) from a dedicated location which could, for instance, be a unique Web-address depending on the card's ATR-string. Such an executable could then run a card-specific protocol to create a directory of the installed applets. This information can be passed to the reader device which registers corresponding proxies for the applications inside the card with the lookup service.

We have realized a Jini smartcard service which implements a primitive interface to a card reader attached to a workstation. It allows for the exchange of card APDUs (data packets used in communication with smartcards) which is the least common interface to all smartcards. Applications must be aware of the APDUs a certain smartcard understands and encode requests according to the card's own protocol itself. This heavily limits the range of cards an application can interact with. However, this primitive interface is the base upon which higher-level services can be built.

The smartcard service is registered with the lookup service by the workstation the card reader is connected to. Proxy objects communicate through Java RMI to a server object on that workstation which passes the requests on to the card reader.

4.5 Protection of Services

Providing access to smartcards is potentially dangerous in an open environment like a Jini federation, especially when offering a simple service like the exchange of APDUs. A malicious client may perform a brute force attack on the authentication codes to the card which will either reveal the codes or lock up the card, i.e. make it unusable even to the legitimate user. It is therefore desirable that only authorised clients can make use of such services. There is no standard mechanism in Jini which provides this kind of security, yet. A possible solution might be a Kerberos-like authentication service [16] that makes services available to trusted clients only.

The PCA has the advantage that the PDA (the client) and the smartcard (the server) are tightly coupled. The smartcard accepts only authorised requests which must be signed by the PDA. A separate authentication service is not required in this case.

5 Related Work

PDAs are becoming popular devices, but PDA-based applications to solve security-related problems are sparsely described in the literature.

Pfitzmann et. al. [17] discuss portable end-user devices (POBs) and security modules and define a number of requirements to be made for such devices. They observe that trustworthy POBs do not exist and conclude that therefore "the development of secure applications should concentrate on protocols and procedures"; we presented an instance of such an approach.

Daswani and Boneh [18] consider PDAs for performing cryptographic computations for Electronic Commerce applications. Their approach differs from ours in that they describe a scenario where the PDA replaces the smartcard, rather than complement the card as we propose.

Recently, the Safe Internet Programming Group at Princeton university published on the Web information about a project that aims at integrating smartcards and PDAs [19]; this approach aims in a similar direction as our approach, but no results seem to have been published yet.

Our approach to integrating smartcards into a Jini-based infrastructure is related to the OpenCard Framework (OCF) as well as the PC/SC architecture. Within these frameworks, the mapping from ATR strings (card types) to services is triggered from a separate installation process which introduces new smartcards or changed functionality. Both are restricted to the domain of a single workstation or PC, rather than offering the corresponding services in a local network.

A similar, but more low-level approach is followed by the Direct Method Invocation (DMI) mechanism proposed by Gemplus [20]. When a Java card applet is designed, an interface is fixed from which a stub object is created. A call to the stub is translated into APDUs which are sent to the smartcard. There, the method call is reconstructed and the respective method is executed. This approach hides the nasty details of creating APDUs from applications but is only applicable to new applications implemented on Java cards. Stub objects could, e.g. be used to implement service objects in OCF. An extended approach would be useful for Java cards in Jini environments: a proxy object could be automatically created from a Java card applet description. As mentioned in our scenario description, that object would have to be initialised with a channel to the card reader.

6 Conclusion and Future Work

In this paper we have presented the personal card assistant (PCA) which is comprised of two different devices – PDA and smartcard – that together implement a security-sensitive application. Both devices are tightly bound together by the public/private keys they share: the smartcard does not perform its task without the PDA and the PDA cannot perform the task without the help of the smartcard.

We have shown the applicability and usefulness of our approach with the scenario of digitally signing documents. Our prototype does not only verify the underlying cryptographic protocol and its implementation on the PDA and smartcard but furthermore aims at a general solution in terms of establishing "spontaneous" networking among the participating devices using Jini. We believe that networking infrastructures that solve similar problems as Jini such as the Service Location Protocol [21] or the Secure Directory Service [22] will be of considerable importance in the future to enable the widespread use of PCA-based applications.

Subject to further research is the question if the PCA can be applied to problems in the domains of electronic commerce and electronic cash.

Acknowledgements.

We would like to thank Dr. Klaus Huber for useful comments and suggestions on an earlier version of this paper.

References

1. 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.

2. 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.
3. Commission of the European Communities. Information technology evaluation criteria. Directorate XXIII/F, SOG Information Security, 1991.
4. European Telecommunications Standards Institute. Specification of the SIM Application Toolkit (GSM 11.14), 1998. http://www.etsi.org.
5. Sun. *Jini Architecture Specification – Revision 1.0*. Sun Microsystems Inc., January 1999.
6. Joachim Posegga. Die Sicherheitsaspekte von Java. *Informatik-Spektrum*, 21(1):16–22, 1998.
7. Sun. *Jini Lookup Service Specification – Revision 1.0*. Sun Microsystems Inc., January 1999.
8. Sun. *Jini Discovery and Join Specification – Revision 1.0*. Sun Microsystems Inc., January 1999.
9. Infrared Data Association. http://www.irda.org/standards/specifications.asp.
10. 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.
11. Bluetooth Technology Overview. http://www.bluetooth.com.
12. Sun. *Jini Device Architecure Specification – Revision 1.0*. Sun Microsystems Inc., January 1999.
13. Dirk Husemann and Reto Hermann. OpenCard Framework. Technical report, IBM Corporation, 1998.
14. OpenCard Forum. http://www.opencard.org.
15. Specifications for PC–ICC Interoperability. http://www.smartcardsys.com.
16. B. Clifford Neuman and Theodore Ts'o. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, 32(9):33–38, September 1994.
17. A. Pfitzmann, B. Pfitzmann, M. Schunter, and M. Waidner. Vertrauenswürdiger Entwurf portabler Endgeräte und Sicherheitsmodule. In H. H. Brüggemann and W. Gerhardt-Häckl, editors, *Verläßliche IT-Systeme*, Braunschweig, 1995.
18. Neil Daswani and Dan Boneh. Experimenting with Electronic Commerce on the PalmPilot. In *Financial Cryptography '99, Conference Pre-Proceedings*, Anguilla, BWI, 22 Februar 1999.
19. Safe Internet Programming Group Princeton University. Smarter Smartcards – Using Devices That Support User Interaction. http://www.cs.princeton.edu/sip/-projects/handheld/, 1999.
20. Jean-Jacques Vandewalle and Eric Vtillard. 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.
21. J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service Location Protocol (SLP). Internet RFC 2165, June 1997.
22. 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.