Securing e-business applications using smart cards

by E.-M. Hamann
H. Henn
T. Schäck
F. Seliger

As the Internet is used increasingly as a platform for business transactions, security becomes a primary issue for Internet applications. Some applications are too sensitive for software-only security mechanisms. Higher levels of protection can be achieved with smart-card-based authentication schemes and transaction protocols. In this paper, we provide examples of typical banking applications implemented with smart cards using symmetrical (DES) and asymmetrical (RSA) cryptography. We present a pure Java™ architecture for such applications, which is intended for use on standard Web application servers and client devices enabled for Web browsing and the Java language. It employs applets on the client side to access smart cards via the OpenCard Framework. The applets communicate with authentication servlets or application servlets on the server side and act as a mediator between the smart card and the application logic on the server.

Initially, the Internet was used for academic research and military defense applications. Several years later, the first commercial use was to disseminate information such as company profiles, advertising, product catalogs, and specifications. All business transactions were still performed outside the Internet by means of traditional media—voice, fax, and paper forms. After several more years, interaction with businesses through the Internet became possible. Companies started to allow their customers to order goods or request services via the Internet. Banks introduced home banking and on-line brokerage applications using basic security functions for the Internet such as server authentication and the Secure Sockets Layer (SSL). For identification and authentication, the user had to enter an identification string and a password. However, this traditional level of security is not sufficient for such sensitive business transactions on the Internet as payments and legally binding contracts.

The European Union and the United States have passed legislation to establish the conditions for making a digital signature the legally binding identification and authentication mechanism for contracts on the Internet.[1–3] The legislation requires that the technology employed not allow secret keys to be copied or used by nonauthorized parties. The consequence of this requirement is the need for a secure secret-key storage, if the digital signature is based on public key cryptography.

Smart cards are an ideal means to provide the required level of security. In recent years, smart card technology has quickly advanced and by now reached a state where smart cards easily integrate into public key infrastructures. Today's smart cards provide memory up to 64 KB to store keys, certificates, and information, and they have cryptographic coprocessors that allow them to generate digital signatures using the RSA (encryption algorithm named for its creators: Rivest, Shamir, and Adleman) or DSA (Digital Signature Algorithm) algorithms with key lengths up to 1024 bits or 1984[4] bits. The Global System for Mobile Communications, derived from the Groupe Spécial Mobile (GSM) standard, uses smart cards as

Subscriber Identification Modules (SIMs) for user and service provider identification and authentication. Smart cards are used as an ID (identification) card for banking and credit card customers, company employees, or citizens. Standardized access methods such as PC/SC (Personal Computer/Smart Card) for Microsoft Windows** and OpenCard Framework (OCF) for Java** facilitate integration of smart cards in applications.

In this paper, we first present examples of smart-card-secured applications. To explain the architecture for such applications, we then introduce authentication protocols involving smart cards as cryptographic tokens. Next, we briefly describe the major smart card types that adhere to these protocols and present the OpenCard Framework that enables Java-based applications to use smart cards. Finally, we present a pure Java architecture for development of smart-card-secured Web applications that use smart cards for user authentication and to protect individual transactions.

Examples of new secure Web applications

Many of the existing Web applications protect the confidentiality of communication through encryption using the well-established SSL, Transport Layer Security (TLS), or Wireless Transport Layer Security (WTLS) protocols. This protection of confidentiality is efficient and considered adequate.

For user authentication, the currently established method employs a user identifier and password. A password provides only limited security because it can be stolen in many ways. An additional means of authentication, for example, through biometry ("what you are") or through additional cryptographic hardware ("what you possess"), provides additional protection. A smart card used as a mobile personal cryptographic token is optimally suited for this purpose.

To secure an individual transaction, a password by itself is not sufficient. Less common methods have been applied in using passwords so far, for example, one-time passwords (as provided by the SecurID** token from RSA) or TANs (Transaction Authorization Numbers), both of which have to be manually fetched and entered by the user. To secure individual transactions and to achieve nonrepudiation, a smart card is an attractive option.

In the following subsections, we present two examples of sensitive Web applications in which individual transactions are protected through digital signatures generated on a smart card and in which the users are authenticated using the same smart card.

Obviously, Trojan horse attacks are a threat in both scenarios, especially when smart cards are used on PCs, which allow uncontrolled installation of software by users. However, the use of smart cards decreases the likelihood of successful fraud significantly. A Trojan horse might be able to obtain a PIN (personal identification number) for a smart card, but it can only use that PIN in the presence of the smart card. Two common approaches to deal with the Trojan horse threat are the use of trusted devices to display and sign data and appropriate risk management.

Trusted devices used to display and sign data can entirely eliminate the risk of successful Trojan horse attacks. Such devices are usually tamper-evident, have their own display and PIN pad, and do not allow software updates at all or at least not by unauthorized parties. To generate a signature, the trusted device receives the transaction to be signed, e.g., from a PC, and displays the relevant part of the data to the user. It prompts the user to enter the PIN, passes the PIN to the smart card, and lets the smart card generate the signature. The trusted device ensures that the PIN is never visible to the external world and that the user signs what he or she sees.

To avoid the higher cost for trusted devices, the provider of the system can decide to manage a higher risk instead. After identifying the threats and assessing the likelihood of fraud, he or she can calculate the amount of money at risk and reserve that amount to cover the risk. Because use of smart cards typically decreases the likelihood of fraud significantly, less risk has to be covered. Another option is to set up contracts so that the users of the system have to take the risk, that is, require users to control any software installed on their clients and make them responsible for any fraud that may happen if they fail.

Our solution examples were developed together with the Deutsche Bank AG, a member of the Identrus LLC global e-commerce trust organization.[5] The public key infrastructure (PKI) solutions have been based on the guidelines published by Identrus. This allows digital certificates and digitally signed documents created by these solutions to be exchanged with all Identrus member financial institutions.

The db-markets eTrade Project. The db-markets eTrade Project that an IBM Global Services team im-

plemented in February 2000 is a typical example of a Web application for highly secure transactions for the money market. It provides the function of an electronic currency market and addresses the following requirements.

All currency brokers worldwide must be enabled to participate in a worldwide network using a standard Internet browser on their personal Windows-based systems. For accessing the central trading application, the broker must be identified by a highly secure authentication scheme. All transfer of data must be encrypted using the secure variant of the HyperText Transfer Protocol (HTTP). In initiating an order, for example, to trade 458000.00 euros for 412965.32 U.S. dollars, all details of the transaction must be digitally signed by the broker together with the correct date and time. The eTrade application must send the data and the associated digital signature to the central bank trading site. At the central trading site, the order has to be validated and an acknowledgment sent to the broker within 10 seconds. Thus, both trading partners can be sure that the trade has been successfully processed. The client-side software for the application must be installable from a central server at the bank. The required hardware must be easy to deploy.

The required highly secure authentication scheme and digital signature call for the use of a cryptographic token. The eTrade system uses the IBM Digital Signature for the Internet (DSI) solution[6] with the IBM MFC (Multi-Function Card) 4.22 smart card. DSI provides smart-card-based generation and verification of digital signatures supporting the two predominant Internet browsers, Netscape Communicator** and Microsoft Internet Explorer**. The application logic executed on the client side is a Java applet that uses the Java Native Interface to access the smart card through PC/SC and Windows-specific functions.

In order to initiate an eTrade transaction, a broker inserts his or her personal smart card into a smart card reader. Then the broker starts an Internet browser of choice and selects the db-markets eTrade Web page (www.db-markets.de). If a client certificate is found on the smart card, this information is transmitted to the server and the log-in process begins. The server uses this certificate for client authentication as described later within the next section of this paper. After successful client authentication, the broker receives an input mask, or screen, for a currency trade, which he or she has to complete (see Figure 1).

When the broker has filled in the data for a trade order, the applet creates a message digest of the data and digitally signs this message digest on the smart card using the "signing private key." The order data and the signature are transferred via the encrypted communication channel to the server,[7] where the signature is verified. If it is correct, the business transaction is completed and an acknowledgment is sent back to the broker. For each transaction, a record together with the signature is filed in a database at the server. The record, including the validation and display of the digital signature, can be viewed by the bank and the broker.

Both trading parties benefit from the following advantages when using an e-business solution with digital signatures and certificates: A currency trade is settled within 10 seconds. All details are stored on the server in signed format. Both sides accept the basic terms of the trade and cannot repudiate them at a later time.

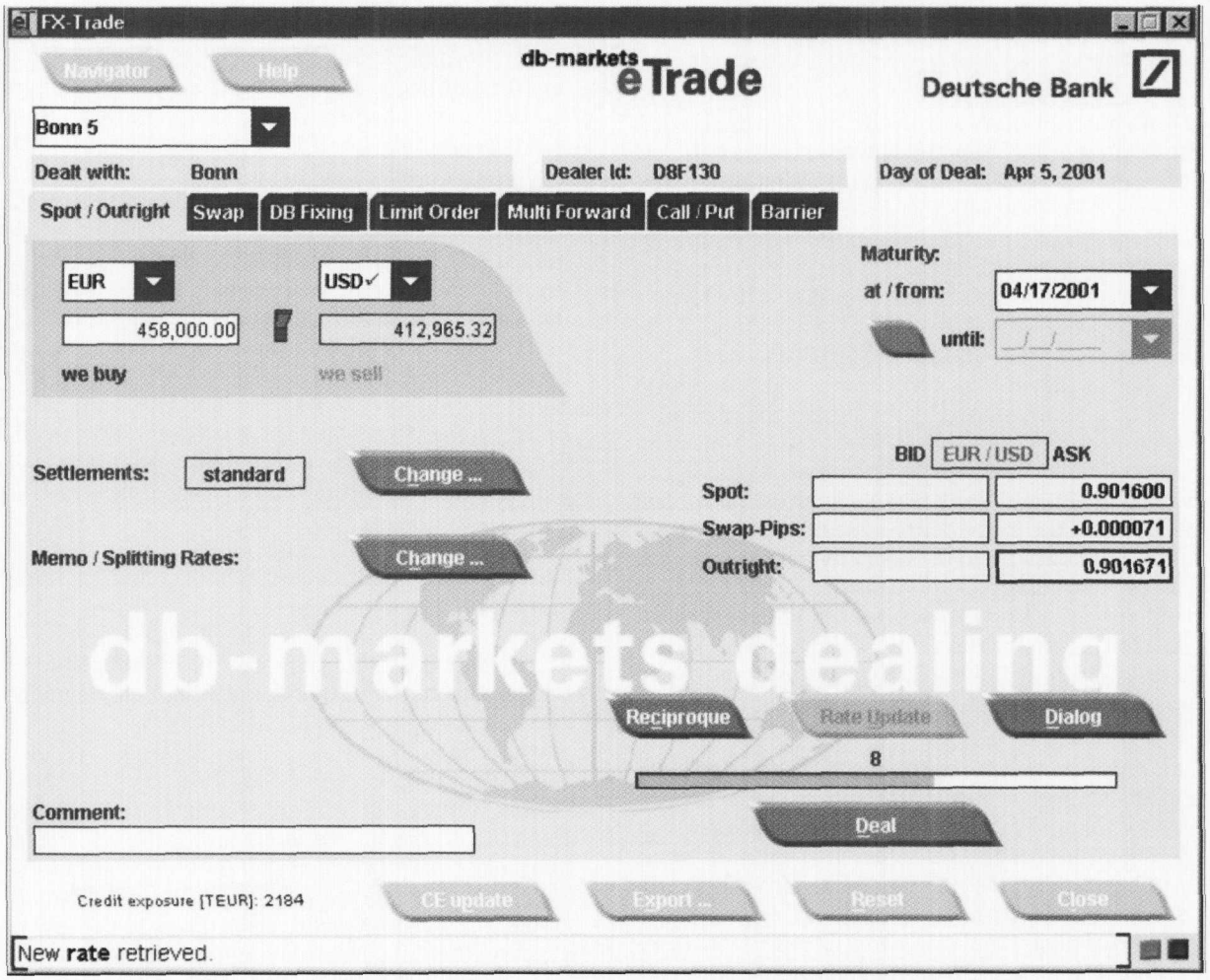
Trojan horse attacks are possible although quite complex to mount. Appropriate risk management must consider this risk.

The e-Safe Project. The e-Safe Project that an IBM Global Services team implemented for Deutsche Bank as a prototype and showcase for the CeBIT 2000 Fair in Hannover, Germany, is another typical example of a Web application that requires use of smart cards. The purpose of the project was to prototype a secure Internet payment system relying on a trusted third party who is responsible for accepting payments on behalf of shops on the Internet. Because the e-Safe system handles all payment-related tasks, there is no need to ever provide the payment information to the shops. The address information of a consumer can be provided to the shops on a need-to-know basis.

Figure 2 shows how the e-Safe system works. Before a consumer can use the e-Safe payment system, he or she has to be registered. Registration will usually be done at the banks that also provide the e-Safe smart cards to consumers.

Once a consumer has been registered and has obtained a smart card, he or she can make e-Safe payments via the Internet. The consumer navigates to a shop site and selects the goods that he or she wants to buy. After filling the "shopping cart," he or she

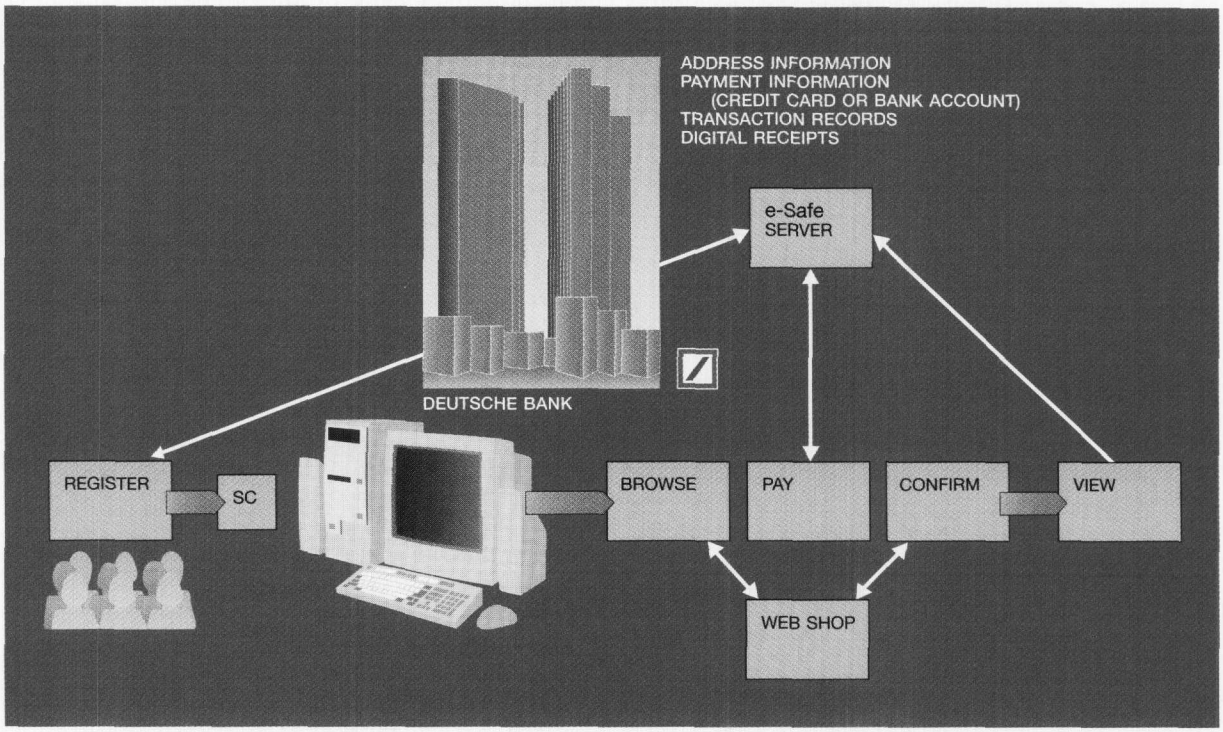
Figure 1 An eTrade client application window



presses a button to start the payment process and is redirected to the payment page of the e-Safe Web site. The payment page summarizes the goods to be bought and shows a payment applet that asks the consumer to enter a PIN to approve the payment. The payment applet then sends the PIN to the smart card to activate the capability of the smart card to generate a digital signature. In the next step, the payment applet requests a challenge from the server, provides it to the card, and lets the card generate a digital signature over the payment transaction record and the challenge. Finally, the payment applet sends the payment record and the digital signature back to the e-Safe server. The server verifies the signature, submits the payment transaction record for clearing, and generates and stores a digital receipt. In addition, it generates a payment confirmation for the shop and redirects the consumer's browser back to the shop site on the Web. The shop receives the payment confirmation and can initiate delivery of the goods.

The consumer can review and check his or her previous payment transactions and digital receipts at any time. To do this, the consumer logs in to the e-Safe server using the smart card. The log-in page contains an authentication applet that uses a protocol similar to the one described in the next section. After successful authentication, the consumer has access to payment transaction statements and digital receipts.

Figure 2 The e-Safe system



Trojan horse attacks against this system are theoretically possible, but since the result of a transaction is always a transfer from a customer's bank account to a merchant's bank account, only registered merchants would be able to obtain money through fraudulent transactions. Because the concept is based on money transfers that can be canceled, customers could reject fraudulent transactions as soon as they realize inconsistencies exist on their bank account statements. In such a case of fraud, the merchant(s) who obtained money deceitfully could easily be identified.

Optionally, trusted devices could be used to display and sign transactions to technically prevent Trojan horse attacks.

Smart-card-based security

As we have seen in the preceding examples of Web applications with smart-card-based security, the smart card provides two types of security services in both cases, user authentication and digital signature generation. Being essentially a tamper-resistant cryptographic token, the smart card is specifically designed to perform these services with a high level of security.[8,9]

Authentication. Authentication of users means proving that users are who they say they are. There are various ways to implement authentication using a smart card; we describe two of them in more detail here.

Authentication using smart cards without public key cryptography. Although the use of public key cryptography allows a more straightforward authentication scheme, smart cards without public key cryptography capability are widely used. These simpler cards have considerably lower prices because they do not require a cryptographic coprocessor that is needed for executing public key cryptographic operations with reasonable speed.

The server gives a random challenge to the smart card and requests a message authentication code (MAC, a kind of signature) generated over the card ID (identifier) and the challenge. Often, a password

Figure 3 Authentication protocol for smart cards without public key cryptography

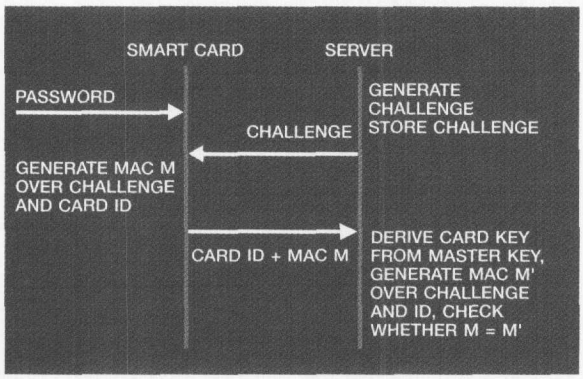
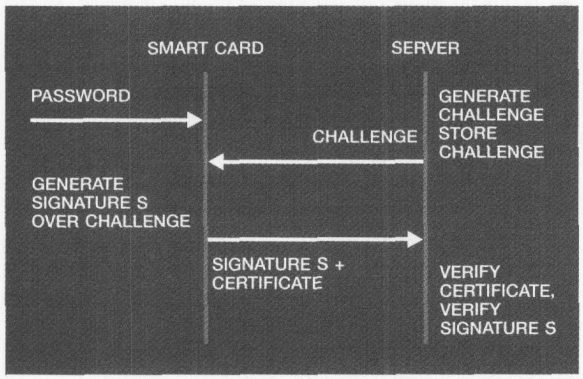


Figure 4 Authentication protocol for public key smart cards



provided by the user has to be given to the smart card before the card generates the MAC. This procedure ensures that a thief or finder of a card cannot use it without knowledge of the password.

The smart card uses a key to generate the MAC over the card ID and the challenge obtained from the server. It sends both the ID and the MAC back to the server. The server uses the card ID to derive the card key from a master key and uses that card key to verify the MAC sent from the card. Figure 3 depicts this protocol.

Authentication using public key smart cards. With smart cards capable of public key cryptography, authentication can be performed as follows: The server sends a random challenge to the smart card. The smart card uses its private key to generate a digital signature over the challenge. The digital signature and the certificate associated with the private key of the smart card are sent to the server. The server verifies the certificate and then uses the public key contained in the certificate to verify the signature (see Figure 4).

Digital signature using smart cards. As we have seen from the two application examples, the eTrade transactions in the money market and the e-Safe transactions of the consumers are digitally signed to protect those transactions against modification and repudiation. Any change to the transaction data would cause the verification of the signature to fail. The person who initiated the transaction cannot reasonably repudiate his or her action, because only the person holding the smart card and knowing the password to unlock the signature generation capability of the card can have initiated the transaction.

A smart-card-based digital signature requires public key cryptography to be installed on the smart card. For any data to be digitally signed, a cryptographic one-way function, for example, SHA-1 (Secure Hash Algorithm-1), is used to create a hash that is signed by the card using the private signature key stored inside the card. Only the cardholder can sign an order or a statement, yet everyone can check the signature using the corresponding public key.

To guarantee nonrepudiation [10] and message integrity, the private key must be stored securely so that only the rightful user can access it. If any other person could obtain a copy of the private key, he or she could impersonate the rightful user's signature.

The most secure place to store such a private key is within a cryptographic hardware unit. A smart card is the most convenient and most portable cryptographic hardware unit. Public key smart cards are able to perform the signing operation inside the card. At the same time, it is not possible to obtain the private signature key without a prohibitively high technical effort. Usually, smart-card-based systems are designed so that obtaining a private signature key would be so expensive that fraud does not pay. To ensure that the private key never exists outside the smart card, legislation in some countries requires that it must be generated inside the smart card.

The emerging legislation for digital signatures has significant regional differences. Several countries require that the device used for the signature must be

tamper-resistant or at least tamper-evident. This requirement not only includes the storage of the private key, but also the hardware and software that displays the content to be signed and prompts the cardholder to initiate the signing by entering the signature PIN.

Common smart card types. In recent years, many brands and types of smart cards have come to market. We can identify several major categories: simple file-system-oriented smart cards without public key capability, advanced file-system smart cards with public key capability, Java Cards**, Windows-powered smart cards, and MULTOS** (multi-application operating system) cards.

Simple file-system smart card. File-system smart cards provide a file system where reading and writing of files can be protected by various access conditions. These cards support only symmetric cryptographic algorithms such as DES (Data Encryption Standard) or Triple DES, for example.

To use such a simple file-system smart card for authentication, a file containing the card ID can be created on the card with special access conditions. These access conditions must allow that file to be read in a way that the result is returned together with a MAC of the result combined with a random challenge passed to the card. This allows running a protocol as explained earlier and shown in Figure 3. Examples of simple file-system smart cards are the IBM MFC 4.1 and the German GeldKarte. Cards of this category are available from every major smart card manufacturer.

File-system card with public key cryptography. File-system cards with public key cryptography capability can store private keys and associated certificates. Key pairs are usually created in the card, and the private key never leaves the card. It is only used internally for generating digital signatures or decrypting session keys or small amounts of data.

To use file-system cards with public key capability for authentication, a private key and an associated certificate must be present in the card. This allows use of a protocol as explained earlier and shown in Figure 4. Examples of file-system smart cards with public key capability are the IBM MFC 4.22, IBM MFC 4.3, Gemplus GPK4000**, Gemplus GPK8000**, and others from Schlumberger, Giesecke & Devrient, etc.

Java Card. A Java Card allows the creation of custom commands on the card. The programs implementing the custom commands are card applets. They are implemented using a subset of the Java programming language, relying on Java libraries tailored for use in smart cards.

A Java Card can host several applets.[11] Off-card applications can select an applet on the card by specifying the application ID of the applet. After that, the off-card applications communicate directly with the selected applet.

To use a Java Card for authentication, the card must contain an applet that exposes an appropriate interface to the external world, e.g., a command that can be parameterized with a challenge, returning a digital signature over the challenge and a command to obtain certificates stored in the card. This functionality allows running the kind of protocol explained earlier and shown in Figure 4.

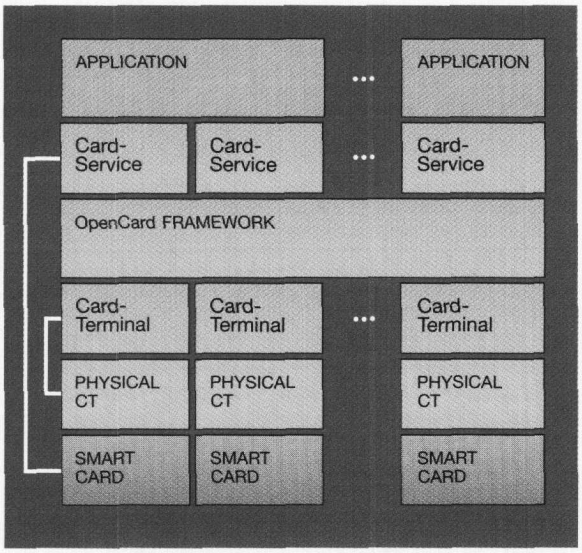
For an in-depth description of the Java Card architecture see Reference 12. An example of a Java Card is the Gemplus GemXPresso** card. A contactless Java Card has recently been developed in the IBM Zurich Research Laboratory.

Windows for Smart Card. Smart cards with the operating system "Windows for Smart Card"[13] allow the implementation of custom commands. It is possible to implement commands that use functions from the internal cryptographic library of the card to provide a function for generating digital signatures, storing, and reading certificates. This functionality also allows for PKI authentication protocols as explained earlier and shown in Figure 4. Windows for Smart Card was developed by Microsoft; cards implementing this operating system will be available from Schlumberger and several other providers.

MULTOS smart card. A MULTOS smart card[14] provides a file system interface and, in addition, an execution environment for custom applications. Application developers can create these custom applications using a new language called MEL (MULTOS Executable Language). Assembler language can be used with MEL; for C and the Java language, a translator to MEL is provided. The MULTOS specification is licensed and controlled by the MAOSCO Consortium.[14]

The OpenCard Framework. An important base for the pure Java architecture that we present in this paper is the OpenCard Framework (OCF), which has become the standard interface for smart card appli-

Figure 5 The OpenCard Framework



cations written in Java. IBM developed the first prototype of the framework in 1997. Also in 1997, IBM, Sun Microsystems, Netscape Communications Corporation, and others founded the OpenCard Consortium to establish the OCF as a *de facto* standard for accessing smart cards from Java. In 2000, the consortium released OCF Version 1.2 and OpenCard for Embedded Devices 1.2. With version 1.2, OCF has reached comprehensive functionality and stability.

OCF permits smart card applications to be implemented in Java. It makes these applications independent from the details of whatever smart card is used and of the device that is used to access the smart card (usually referred to as smart card terminal, smart card reader, smart card acceptance device, or interface device). To achieve this independence, OCF encapsulates the details of diverse smart cards with equivalent function in the abstraction CardService and details of diverse access devices (physical card terminals) in the abstraction CardTerminal (see Figure 5).

Every OCF CardTerminal provides the functions required when accessing a smart card, for example, resetting the card, obtaining the answer-to-reset, sending data packets to the card, and obtaining the response. There are OpenCard card terminal drivers for virtually all PC smart card readers on the market. For several smart card readers, there is a pure-Java card terminal implementation available. All readers that can be used on Win32** platforms through PC/SC[15] can be used from OCF via a generic PC/SC CardTerminal for OpenCard that provides a bridge to the PC/SC card terminal interface.

To achieve independence from the specifications of particular card manufacturers, CardServices are used. Card service interfaces can be defined for particular sets of smart card functions. Two interfaces that are defined in the OpenCard Framework itself are the File Access Interface and the Signature Interface; the first allows files to be accessed on a smart card, the latter allows digital signatures to be generated. Once a card service interface has been defined, various card service implementations that implement this interface can be developed for different makes of cards.

The main purpose of the OpenCard Framework is for use on the client side of Web applications, running in a Java applet. The most advantageous way to deploy the OpenCard Framework in such an application is to install the OpenCard Framework and the required card terminal classes locally on the client by adding the OpenCard JAR (Java archive) files and executables to the paths of the browser. The card services to be used are usually packaged with the applet JAR file that is deployed on the application server.

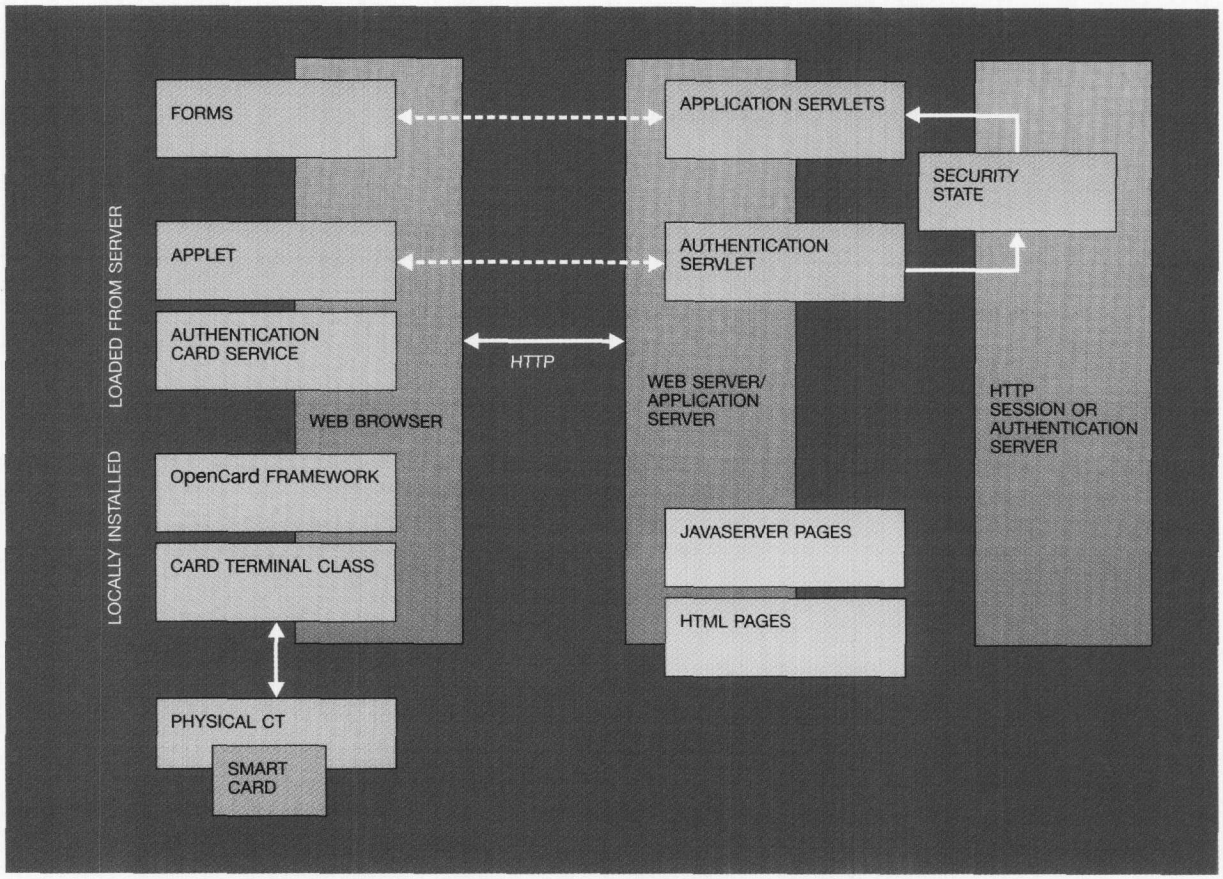
The OpenCard Framework is available from the Web site http://www.opencard.org. The Web site also provides up-to-date information on OpenCard, documentation and user guides,[16] and links to papers and books about the OpenCard Framework. For comprehensive information on developing smart card applications in Java, see Reference 17.

Application architecture

Earlier in this paper, we gave two examples of Web applications that achieve a high level of security through the use of a smart card on the client side, eTrade and e-Safe. Both applications use smart-card-based authentication and digital signatures. Both applications use Java applets on the client side.

The architecture of both applications differs in several respects, however. The eTrade application does not require portability of the client part to operating systems other than Windows. Consequently it uses an architecture that exploits components that are found in Windows operating systems, most no-

Figure 6 Architecture overview

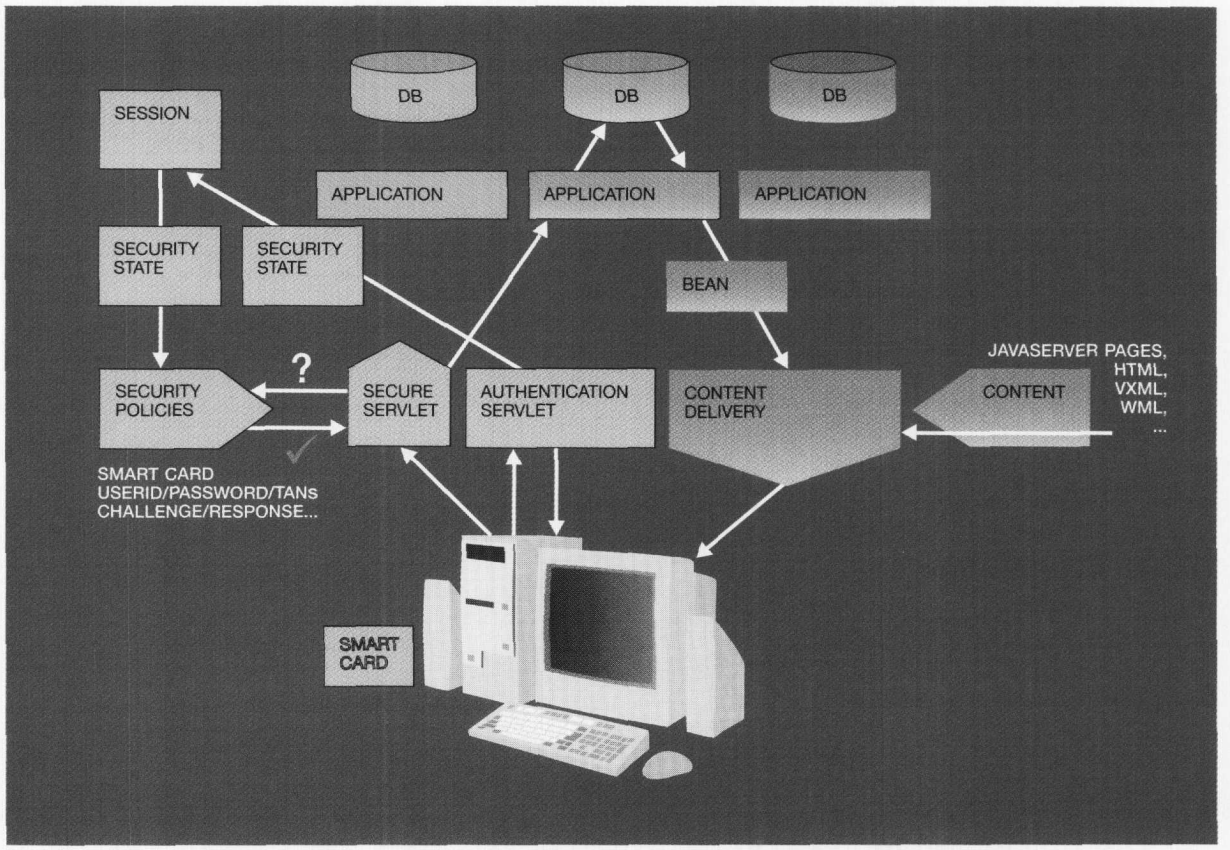


tably the smart card access layers provided by PC/SC.[15] The eTrade application uses the IBM Digital Signature for the Internet (DSI) solution,[6] which internally uses the PC/SC application programming interfaces (APIs). Toward the calling applications, DSI provides the Public-Key Cryptography Standard #11 (PKCS #11) API,[18] a C-language interface that is used by Netscape Communicator and Netscape Messenger**. The PKCS #11 interface is also available to applets through Java wrappers. For the Microsoft Internet Explorer browser, DSI offers the Microsoft Crypto API (CAPI).

The e-Safe application only uses the Java language and does not impose any restrictions on the platform for the clients and for the servers. Therefore, the e-Safe application can easily be offered on such diverse Java-powered devices as, for example, Internet appliances, communicating personal digital assistants, or network computers. Because an HTTP-based protocol is used for communication between client and server, the use of Java on either side does not require using Java on the other side. These decisions are also independent of using a Java Card or any other smart card.

In the following, we focus on this pure Java architecture, which is applicable for Web applications based on the servlet[19] and JavaServer Pages** (JSP**)[20] technologies. Such applications use the following pattern: The browser sends a request for an HTML (HyperText Markup Language) document and displays it to the user. In the document, there may be links or forms that refer to servlets. When the user clicks on such a link or submits such a form, the browser sends a request to the appropriate servlet. The servlet processes the request and invokes a single JSP to display the result to the user. The HTML

Figure 7 Secure access to a Web application



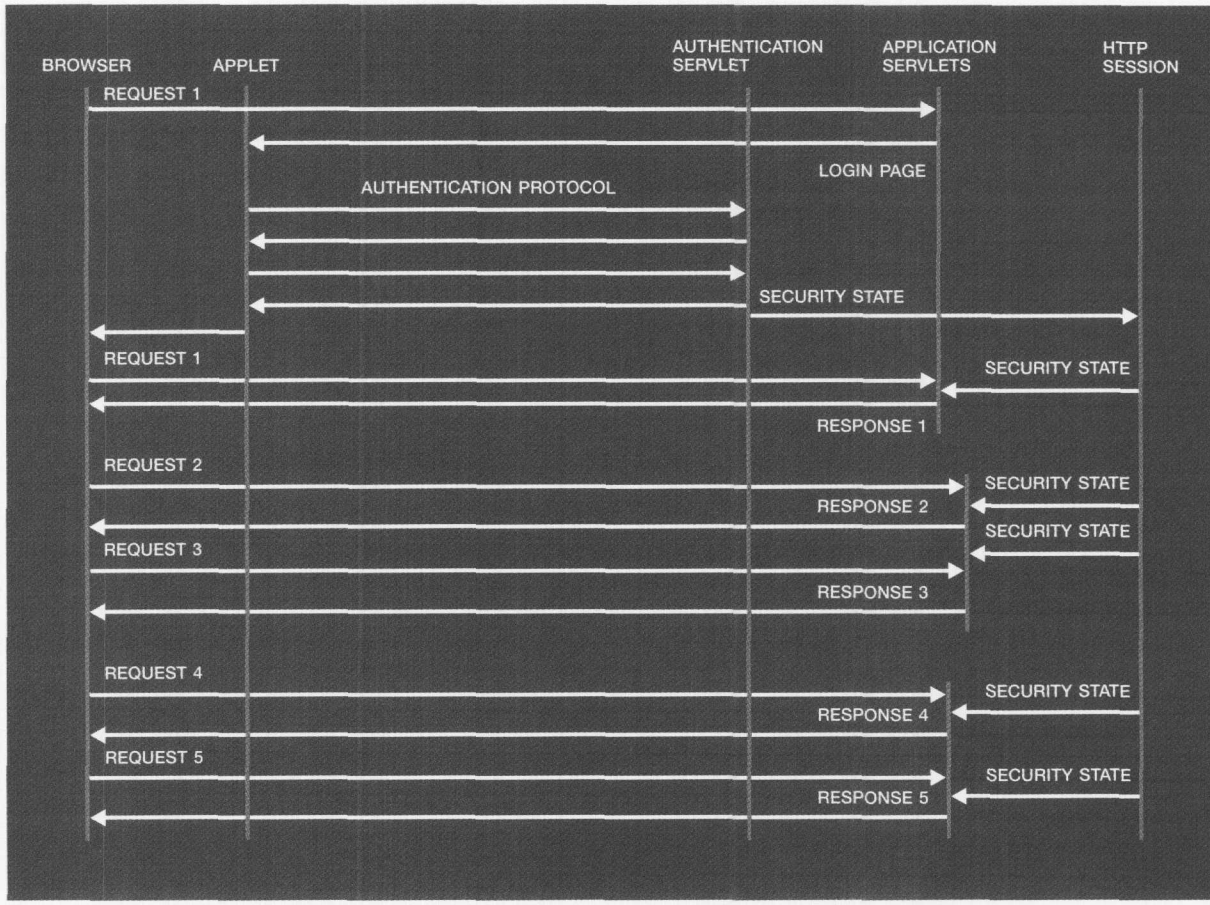
page generated by the JSP is displayed in the browser; it may contain further links or forms. Clicking on these links or submitting these forms starts a new cycle. A page generated by a JSP may contain applets, which in turn can initiate communication with the server from which they originate.

Figure 6 shows a component overview with respect to the smart-card-based authentication. A user who wants to use a smart card for authentication to Web applications must have a smart card reader connected to his or her client device. The client device can be a personal computer or an Internet appliance, for example. The Web browser of the device must be enabled for smart card access. For the Java-based architecture discussed here, the Web browser needs to support Java applets and to enable the applets to access the smart card. The OpenCard Framework or OpenCard for Embedded Devices provides this access for applets. It can be part of the preinstalled software stack of the device, provisioned via the service management framework of the device (as, for example, specified by the Open Services Gateway Initiative OSGi[21]), or user installed. In addition, the device must have appropriate driver software installed to make the card reader available to applications. This software will in most cases be the associated pure Java CardTerminal. For PCs running one of the Win32 operating systems, alternatively PC/SC plus a PC/SC interface device can be installed.

Card services that encapsulate the application protocol of the smart card are packaged in a JAR file, together with the applets that use them. This JAR file is deployed on the Web application server and will be downloaded on demand.

On the server side, in addition to the application servlets (see Reference 19), HTML pages, and JSP (see Reference 20), an *Authentication Servlet* must be de-

Figure 8 Interaction between the Authentication Servlet and application servlets via the session



ployed. This servlet implements the server-side authentication protocol logic and provides a *Security State* upon successful authentication of a user, for example, by putting it into the HTTP session. Security-aware application servlets can then access the Security State before performing sensitive functions. Figure 7 shows the mechanism that is executed.

When the user logs in, a page with the authentication applet is displayed in his or her browser. If the smart card requires a password, the authentication applet prompts the user to enter the password and provides it to the card before starting the actual authentication protocol. The authentication protocol is executed between the smart card and the Authentication Servlet, mediated by the authentication applet. If the user has been successfully authenticated, the Authentication Servlet adds a Security State to the session to indicate that the user has been authenticated.

When the user navigates to a page that invokes a security-aware servlet, the servlet checks whether the Security State stored in the session is sufficient to perform the operation. If it is, the servlet invokes the appropriate application logic. Figure 8 shows an example where several application servlets access the Security State in a session once the Authentication Servlet has established it.

WebSphere* Application Server allows for single sign-on (SSO) using custom authentication methods. A servlet that implements custom authentication can call the appropriate API function to perform a single sign-on log-in. Thus, a slight change in the authentication servlet allows taking advantage of the Web-

Sphere single sign-on and clustering features. Instead of accessing the Security State in the session, the Authentication Servlet would have to use the SSO API to call on the WebSphere Lightweight Third Party Authentication (LTPA) mechanism.

Conclusion

In this paper, we presented examples of applications that require user authentication and transaction authorization with a very high level of security. More and more Web applications with similar security requirements will emerge as the volume of financial transactions conducted via the Internet increases steadily. The pure Java architecture presented in this paper allows such applications to be secured in an elegant and flexible way, using smart cards to provide a higher level of security.

*Trademark or registered trademark of International Business Machines Corporation.

**Trademark or registered trademark of Sun Microsystems, Inc., Microsoft Corporation, RSA Security, Inc., Netscape Communications Corporation, MAOSCO Limited, or Gemplus, SA.

Cited references and notes

1. *Electronic Signatures in Global and National Commerce Act*, issued as United States public law 106-229 on June 30, 2000, effective October 1, 2000, http://www.access.gpo.gov/index.html (search for "106–229").
2. *Directive 1999/93/EC of the European Parliament and of the Council*, European Union (13 December 1999), http://europa.eu.int/eur-lex/en/index.html.
3. *Gesetz über Rahmenbedingungen für elektronische Signaturen/Signaturgesetz* (German signature law) (August 2000), http://www.bmwi.de/Homepage/download/infogesellschaft/Signaturgesetz.pdf.
4. Established smart card communication protocols limit the size of data exchanged in one single command to 255 bytes, which leads to the limitation of keys used with the single command signature operations.
5. Identrus LLC, 140 East 45th Street, New York, NY 10017, http://www.identrus.com/.
6. E.-M. Hamann, *Digital Signature for the Internet (DSI) White Paper*, IBM Pervasive Computing Laboratory, Boeblingen (2000).
7. Authentication of the server was done during communication setup.
8. S. B. Guthery and T. M. Jurgensen, *Smart Card Developer's Kit*, Macmillan Technical Publishing, Indianapolis, IN (1998).
9. W. Rankl and W. Effing, *Smart Card Handbook*, John Wiley & Sons, Inc., New York (1998).
10. Nonrepudiation is guaranteed for an action if the actor cannot reasonably deny having done this action.
11. Smart cards capable of hosting several applications are often called "multiapplication smart cards." The IBM Multi-Function Card (MFC) family, Java Cards, Windows-powered smart cards, and MULTOS cards are all multiapplication smart cards.
12. Z. Chen, *Java Card Technology for Smart Cards: Architecture and Programmer's Guide*, Addison-Wesley Longman, Inc., Reading, MA (2000).
13. *Microsoft Smart Card for Windows*, Microsoft Corporation, Redmond, WA (1999), http://www.microsoft.com/smartcard/.
14. *MULTOS Version 4*, MAOSCO Limited, 47–53 Cannon Street, London, EC4M 5SQ, United Kingdom, http://www.MULTOS.com/.
15. *Interoperability Specification for ICCs and Personal Computer Systems 1.0* (1997), http://www.pcscworkgroup.com/.
16. *OpenCard Programmer's Guide*, OpenCard Consortium (2000), http://www.opencard.org/.
17. U. Hansmann, M. S. Nicklous, T. Schäck, and F. Seliger, *Smart Card Application Development Using Java*, Springer-Verlag, Berlin (1999).
18. *PKCS #11—Cryptographic Token Interface Standard, Version 2.10*, RSA Laboratories, RSA Security, Inc., http://www.rsasecurity.com/rsalabs/pkcs/pkcs-11/index.html.
19. *Java Servlet Specification Version 2.2, Final Release*, Sun Microsystems, Inc., Palo Alto, CA (1999).
20. *JavaServer Pages Specification Version 1.1, Final Release*, Sun Microsystems, Inc., Palo Alto, CA (1999).
21. Open Services Gateway Initiative (OSGi) Specification 1.0 (January 2000), http://www.osgi.org/.

Accepted for publication April 6, 2001.

Ernst-Michael (Mike) Hamann *IBM Pervasive Computing Division, Schoenaicher Strasse 220, 71032 Boeblingen, Germany (electronic mail: mhamann@de.ibm.com).* Mr. Hamann is a consultant solutions architect in the Pervasive Computing Division. He is responsible for digital signature solutions using public key techniques. He developed the architecture of IBM's Digital Signature for the Internet (DSI) solution using PKI smart cards. He is currently specializing in the security components of the Wireless Application Protocol (WAP) and represents IBM at the WAP Forum's Security Group (WSG). He has worked in the development laboratories of IBM for the last 33 years in various positions in software and hardware development. During this time he developed IT solutions in a wide range from large hosts (System/390®) and networking products (IBM 3270, IBM Token Ring, Systems Network Architecture) to smart cards and mobile equipment. Mr. Hamann received an engineering degree (Diplom Ingenieur) in applied physics from the PTL Wedel/Hamburg, Germany, in 1968. He holds several patent submissions in the area of IT security components.

Horst Henn *IBM Pervasive Computing Division, Schoenaicher Strasse 220, 71032 Boeblingen, Germany (electronic mail: hhenn@de.ibm.com).* Dr. Henn is lead consultant in the Pervasive Computing Division. He received his diploma in computer science and his Ph.D. in electrical engineering from the University of Stuttgart. He joined IBM in 1975, working on design software development for System/370™ and 801 system development. He managed a series of hardware, software, and system design projects in the areas of System/390 CMOS sysplex, communication adapters, image processing, and neural-network-based signature validation, as well as the initial development of the IBM MFC smart card. Dr. Henn was involved in standardization activities for PC/SC, OCF, Java Card, and G8 health card. In his current position he is working with development teams and customers to create first-of-a-kind multimode portal systems. He has published a series of papers and filed many patents.

Thomas Schäck *IBM Pervasive Computing Division, Schoenaicher Strasse 220, 71032 Boeblingen, Germany (electronic mail: schaeck@de.ibm.com).* Mr. Schäck is an architect in the Pervasive Computing Division. He started working for IBM in 1996 after obtaining his diploma in computer sciences from the University of Karlsruhe. After joining IBM, he worked in Java and C++ development projects centered on the smart card technology. These projects include the OpenCard Framework that became the standard API for smart card applications in Java, payments via the Internet, and work on a PKCS #11-based digital signature solution. He was the architect of a first-of-a-kind e-business project for a large German bank and is now working as a portal architect in the Pervasive Computing Division. He has published various papers in his field and filed numerous patents. His prior publications include the book *Smart Card Application Development Using Java*.

Frank Seliger *IBM Pervasive Computing Division, Schoenaicher Strasse 220, 71032 Boeblingen, Germany (electronic mail: seliger@de.ibm.com).* Mr. Seliger is currently a security architect in the Pervasive Computing Division. In 1978, he joined IBM, where he has been active in various areas of software and firmware development. His latest focus has been on object-oriented software development and on security technology. Since 1990, he has worked on a C++ collection class library, on a Java framework for smart card applications, as consultant in the IBM Object Oriented Technology Center, and as architect in various applications of mobile computing. With his colleagues he captured his experience in *Developing Object-Oriented Software—An Experience-Based Approach*, published by Prentice Hall in 1997, and in *Smart Card Application Development Using Java*, published by Springer-Verlag in 1999.