-

Java Card†: An analysis of the most successful smart card operating system to date

Version 1.1 (13)

Eduard de Jong♠, Pieter Hartel♦, Patrice Peyret♣, Peter Cattaneo•

Abstract

To explain why the Java Card operating system has become the most successful smart card operating system to date, we analyze the *realized* features of the current Java Card version, we argue it could be enhanced by adding a number of *intended* features and we discuss a set of *complementary* features that have been suggested. No technology can be successful without the right people and the right circumstances, so we provide some insights in the personal and historical historic aspects of the success of Java Card

1 Introduction

In June 2005, one billion Java Card cards had been sold; more than any single smart card operating system has achieved before. This result is due partly to socio-economic circumstances, serendipity, and, as we will argue, in a large part to the technical excellence of Java Card.

In the early nineties, conceived as a third party smart card OS under the code name MASS and later marketed as 'Macsime,' before adopting its present name, the Java Card concept was endowed with a number of features. Some of these have been *realized* in the product [Che00] and have contributed to the commercial success. Other features, present in the implemented prototype of Macsime [Jon95], have yet to be introduced in Java Card products, and, as we will argue, are entirely within the Java Card spirit: To deliver the core of a card operating system that is easy to use, secure and amenable to formal analysis. A third set of *complementary* features has been proposed, which may be useful for specific applications. We discuss each of these three categories of features.

Java Card products have *realized* a number of innovative features, including the ability to program smart card applications in a high level language (a subset of the Java language), the ability to upgrade a Java Card smart card with new applications after the card has been issued, and strong security mechanisms such as fine-grain access control and strict applet separation. These mechanisms rely on the fact that Java has built-in concepts of security, and that it is object oriented; a Java Card programmer thus uses object oriented design methods to develop the applet code and possibly also the terminal code, supported

---

† Sun, Sun Microsystems, the Sun logo, Java and Java Card are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

♠ Sun Microsystems, Inc.
♦ University Twente
♣ Mobile365.

by programming tools, such as emulators and debuggers. These features have been described elsewhere in detail [Che00]

Java Card technology was *intended* to be used in what is now called a model-based approach. This means that the starting point for any development is a conceptual model of the system under development, in this case which the model consists contains concepts such as persistence, transactions, authentication, authorization and integrity and the establishment of trust in a business transaction. The foundations for the model based analysis of Java Card applications have already been laid in our earlier work on the transaction model [Har00b], the three-phase interaction protocol [Har94f] and the card/terminal co-design development model [Jon00c].

Depending on the area of deployment, the Java Card specifications could be enhanced with further, *complementary* features, such as the provision of an IP address [Gut00] or the addition of web-server functionality [Ree02], interfacing to peripherals such as keyboards and displays [Pra01], or the integration of Biometrics on the card [Hen00].

Our contribution is threefold. Firstly, we present an analysis showing that Java Card technology as it exists today is the object-oriented core of a larger conceptual model. The object-oriented core is well understood and fully embraced by the industry. The main elements of the conceptual model have been proposed earlier, but the present paper emphasizes the integration of these elements into a coherent model. Secondly, we offer a comparison of smart card operating systems and the features they provide showing how the developments of each system contribute to Java Card features. Thirdly, we present a time line showing when Java Card technology and its conceptual model emerged, and who were responsible.

In subsequent sections we provide an analysis of the realised, intended and complementary features of Java Card products. We also provide a brief history of the Java Card development. The last section concludes and provides suggestions for future work. We prefer to give a top down presentation and will begin with the *intended* features.

2 Intended features: Model based development

A smart card is component in an information system, of which the mantra can be described as: *a smartcard is a slave to the terminal but the master of its data*. This reflects that a smart card does not initiate communication, it responds to requests for (security) services. The smart card is wholly in control of access to stored data; it is up to the card applet to respond to requests from the terminal in a way that may vary from application to application. Or to say it differently, the application in the card has the sole discretion in determining that the necessary security conditions for its collaboration with the outside world have been met: this is the foundation of trust.

2.1 Card/terminal co-design model

Following its mantra, we propose a card/terminal co-design model for smart cards [Jon00c] where the card and the terminal application are developed in a single intellectual exercise. Support tools separate the card (applet) part from the terminal (application) part. While our co-design model has not yet been embraced in the smart card world,

essentially the same idea has been widely accepted in the word of embedded systems as hardware/software co-design. We are therefore confident that eventually the smart card world will embrace this idea also.

Currently a significant amount of work is devoted to developing methods and tools to implement the card side correctly. For an earlier survey we refer to our paper [Har01]. Current developments are concentrated at Gemplus (in close cooperation with the University of Lille) [Bon04a], Giesecke & Devrient [Phi03], Inria [Pav04], Nijmegen University [Bre05a], the Kestrel Institute [Cog05] etc.

The extension of the tools into the terminal side has not been realized. An obstacle to realizing the terminal side code generation is the lack of a generally accepted model for integration of card security services and hence the absence of a sensible middleware implementation to provide a context to execute the generated code in. We believe that this is a serious problem, which deserves the attention of the research community.

2.2 Three-phase interaction model

Zooming in on the role of the smart card in the card/terminal co-design model, we observe that almost without exception in every non-trivial application the smart card performs in response to a request, the same three functions, which amounts to our three - phase interaction model [Har94f] [Jon94]:

- *Input security phase:* As a slave to the terminal the smart card receives and decodes a command and some data. Then, except in the most trivial applications, security processing takes place, such as the decryption of encrypted parameters, checking signatures etc.
- *Processing phase*: Any data received and data stored on the card must be processed, which amounts, in almost all cases, to transaction processing where the non-volatile memory of the card provides persistent storage.
- *Output security phase*: After data processing, another security phase ensures that any resulting data is encrypted before it is returned to the terminal, that appropriate signatures are applied etc.

Thus sandwiching the ‘pay load’ processing of the smart card between two layers of security is an essential aspect of a smart card applications response: An application programmer following the three-phase interaction model specifies an appropriate method to implement each phase. In a practical implementation, to control the cost of potentially expensive cryptographic operations, an additional pre-flight phase can be used to determine appropriateness of the request and the availability of resources.

2.3 Transacted memory model

Transaction processing, as mentioned in the payload processing phase is another essential feature of a card application. Program-accessible transaction processing is distinct from what is called “anti-tear,” which is a transparent transaction mechanism that is in general provided by smartcards to be able to react to unexpected removal (tear) of the card from the card-reader. Our transacted memory model [Har00b], [Jon00] embodies a transaction mechanism that can be used explicitly in an applet. A transaction can span several

interactions with the terminal, e.g. sweeps of the three-phase interaction protocol and simultaneous transactions are supported. The transacted memory model has its origins in the transaction model from the data base area [Dat95] and work in the smart card area by Rankl and Weiss [Ran96]. An explicit programmed transaction model can enhance the speed and efficiency, as it can replace the relatively expensive, in processing terms, anti-tear mechanisms and minimizes the number of time-consuming updates of the non-volatile memory.

Summarizing, the card/terminal co-design model, the three-phase interaction model and the transacted memory model form key concepts of the modern smart card. The Java Card specifications in their present form do not support these concepts, but there are no technical impediments to add these intended features to Java Card technology. Having described the three conceptual models underlying the Java Card design we now turn our attention to the *realized* features, which are intimately connected with the object orientation of the Java Card technology.

3 Realized features: Object oriented programming

Java Card applets are small ‘programs’, that can communicate to each other and the terminal, that are developed in a high level language, and then either pre loaded, or, more interestingly, loaded into a smart card once it has been fielded. Other smart card operating systems allow development in high level languages (e.g. Multos [Eve01], Basic card [Mil04a]), post issuance update (e.g. Carte Blanche [Pel95]), and secure communication between applets (e.g. Firewall patent [Jon95a]). The innovation of the Java Card technology is that it supports a combination of these features all exploiting the fact that it is object oriented.

3.1 Applet loading is class loading

Loading an applet means loading a class (which once instantiated becomes an object in its own right). The security of applet loading is intimately connected with access control to the class and the resulting object. This idea can be traced directly back to Java itself, which offers secure loading of classes. The case for post issuance update of card functionality has been demonstrated with the Blank Card concept of Peltier [Pel95].

The Java Card specification itself does not cover life cycle management [Gri03]. With the hindsight of its success it is clear that a practical business case for dynamic cards involves an active card issuer that guards access to the card. The Visa Open Platform, later Open Platform, was introduced by Visa international [Kek01] to implement this commercially critical role of the card issuer.

3.2 Applet communication is object sharing

Applets communicate by sharing objects, again linking the access control directly to the (shared) object. Not all objects can be shared by all applets, that is, the Java heap is partitioned in to a number of virtual heaps. This form of access control has become known as the Java Card firewall, which is actually a bit of a misnomer. A more appropriate description would be an ‘execution context’, which is something that main stream operating systems have offered for at least 50 years [Sal75]. The application of

this idea into the smart card world originates from Banerjee's work on virtual cards in OSCAR, Ugon's work on SPOM [Ugo83].

3.3 Byte code verification alone is not enough

As part of applet loading, Java Card run-time environment performs byte code verification. This is conventionally done in two stages, where the verifier runs off-card, and once the applet has been verified it is digitally signed and installed on the card. Lightweight byte code verification for Java Card management performs on-card verification [Ler02]. In both cases the implicit assumption is that once verified, the applet cannot be changed. This assumption may be too strong, especially if native code can be downloaded and executed in the card. Recently an attack on the Java runtime has been shown by inducing memory errors [Gov03], while, as published only applicable to applications that can control a large amount of RAM, which is not available on smart cards it demonstrates that additional defensive mechanism may be needed, for instance computing and verifying checksum over code before executing [Jon04]. Typically a smart card memory is well protected by error correction and tamper detection circuitry, but additional hardware may be needed to support such run-time code verification mechanism. We believe that this is a clear example where the Java Card platform's Java parentage alone is not sufficient for security.

3.4 Summary of features

Table 1 provides an overview of card operating system features and the support in the current version of the Java Card operating system, its precursors and a number of related systems, which have been influential for the development of the Java Card specifications. The card operating systems are presented in the table in chronological order (as indicated by the row 'introduction') to illustrate the development over time (see Table 2 for further historic details). For each operating system instance the various features are summarily indicated, a more expressive description follows the table;

The following card operating systems are presented:

Oscar is one of the first third party smart card operating system implementations made by GIS in Cambridge (UK), which was available on smart cards produced by OKI. It is a file structured storage card. It's main feature is what was called the 'virtual card', which separates zones of storage and a mechanism to initialize each zone securely with its own set of keys. With Oscar, a virtual card could be created after issuance.

Macsime/TOSCA a portable card execution environment designed and implemented in Zaandam, The Netherlands, aimed at the payment market with a focus on high-level security evaluation. The name was changed into TOSCA when the technology was transferred to Integrity Arts in California.

Blue, the card operating system originally built by DigiCash in The Netherlands to support the eCash payment protocol specified in a manner to support high-level security evaluation.

Multos the card operating system originally built by the Nat West Development Team in London to support the MONDEX payment protocol specified and implemented to support high-level security evaluation.

-

Technology / Feature	OSCAR	Macsime / TOSCA	Blue	Multos	Basicard	Java Card
Source	GIS	QC technology / Integrity Arts	Digi-Cash	NWDT	Zeit Control	Sun Microsystems
Country	UK	NL / US	NL	UK	GE	US
Introduction	1990?	1994 / 1995	1996?	1996?	1997	1997[1]
Multi-application / firewall	X	X		X		X
Interpreter		SCIL / CLASP	J-Code	MEL	ZC byte code	Java Card Byte Code
Application programming language		Data model / ADA	J-Code	MEL / C	Basic	Java
Dynamic	X	X	?	X		X
Card Management	master file		?	Multos	One-shot loader	(VISA) Open Platform
Cross-firewall communication		X		Via I/O buffer		API
Program accessible Transactions		X	?			Single
3-phase interaction model		X				

Basicard a programmable smart-card product from ZeitControl in Germany that can be one-time programmed in a specific flavour of BASIC.

[1] The feature are those presented in the 2.0 specifications

-

Java Card, a portable card execution environment that can be programmed in Java that incorporated the Macsime technology.

The features in the table can be summarized as follows:

Multi Application means that the role of the card can be selected to support operations for a specific purpose, conventionally called the card application. Partitioning the card into independent execution contexts is a necessary mechanism in multi-application cards.

Interpreter means that the card operating system incorporates a virtual machine interpreter, either to interpret application code, e.g. Java Card platform, to interpret the OS implementation, e.g. Blue or both, e.g. Macsime.

Application programming language The language in which application program can de implemented and in which an application-programming environment (API) is available.

Dynamic the property that a multi-application card can be modified after issuance by removing or adding applications using a card management mechanism

Card Management the particular mechanism used for card management implemented on cards with a particular operating system. In the Java Card framework management is realized as a card application in its own right allowing for alternative card management schemes, however Open Platform is used in almost all cards.

Cross firewall communication a mechanism that allows applications to collaborate securely inside the card.

Progam-accesible transaction model a model of transactions and persistence specifically tailored to smart cards.

3-phase interaction model a model of the typical interaction between the smart card and its environment.

Summarising, the marriage of Java and the smart card has created an elegant and powerful platform for smart card application development. This concludes the presentation of the *realised* features.

4. Complementary features

Several proposals have been made to enhance the functionality of Java Card, and more importantly, a significant number of experiments have been carried out to evaluate such proposals for effectiveness and efficiency. We discuss a limited selection of the proposals for *complementary* features, which we hope give a good coverage of the field.

4.1 Addressing the card by a unique number

One widely suggested complementary feature is the provision of an IP address to a smart card with the implementation of a stripped down TCP/IP communication stack in the card [Ree02]. Without an IP address a Java Card can act as what could be called an anonymous server. This has benefits for security and the protection of privacy. Anonymity of the card is an essential property of a multi application card and not

"security by obscurity" [Ker1883]; instead the anonymity of the Java Card smart card enables a card applet to take the appropriate security measures required by its application purpose without concern for identity-based attacks on security or privacy.

In some applications there is little harm in addressing the smart card by a unique identification (such as an IP address). For example the SIM card, one of the most popular Java Card applications, is embedded in a handset that is already replete with identification (phone number, IMEI etc), in a sense the damage has already been done..

In other applications, addressing a smart card by a unique identification would be harmful. For example, we would not like our banking cards to leave a clearly visible trail of our payments to shops, hotels, restaurants etc. A banking card does carry an account number that is diligently guarded by the card application, and not used to advertise the presence of the card to the world. Like an IP address, a card should not advertise its public key certificate: such a certificate is by its construction a unique ID.

4.2 Enhanced interaction with the environment

The Java Card basic processing model is independent of the communication method between card and the terminal, and supports a packet transport protocol specified in an international standard [ISO97] and the application-level command response protocol in [ISO05]. To hide details of APDU processing a variation on the remote procedure call has been proposed by Gemplus [Van98] which was adopted in the Java Card specification [Sun03] using the syntax of Java RMI to indicate a card specific serialization and object referencing mechanism. While providing some of the intended abstraction for the communication details, the RPC realization actually exposes many of the Java Card Java-language restriction to the terminal application designer. When, additionally security processing is part of the RPC handling the net benefits for the implementer seem very small.

4.3 Freeing the card from slavery

Lecomte et al [Lec99] and Deville et al [Dev03] argue that the card should be freed from slavery, giving the card a more active role. The reasoning is that an active card can take initiative, serve applications concurrently and generally perform multiple tasks. A typical application is a SIM card issuing proactive SIM commands. However, we believe that this inherently weakens the security of the smart card. Multi tasking (as opposed to supporting multi application), creates an opportunity for interference in the execution of the applets, which might be exploited in one applet o spy on the actions of another. For example using the timing attack [Koc96] an applet observes that some computations that normally take a short amount of time (when the current applet is the only applet running), can also take larger amounts of time. The history of this kind of attack traces back to the 70s with the infamous TENEX password attack [Gol99]. So we believe that more research is needed to make sure that active cards do weaken the security of the smart card.

4.4 Adding new APIs

The Java Card specifications provides a number of API's, for cryptography [Gui01] [Bor01], vendor specific functions etc. Java Card currently does not provide APIs for

-

other functions. It is notably lacking an appropriate API for Biometrics [Lop04]. There is, however, a Biometric API from the Java Card Forum (JCF) with some limited functionality, which will be incorporated in the next specifications. One of the results of the European Inspired project (See http://www.inspiredproject.com/) should be an API for Biometrics.

1989		In France, Patrice Peyret with Gemplus, then the leading card manufacturer initiatesd a long lasting term research collaboration project called "RD2P[2]" with the university of Lille, initially engaging Vincent Cordonier, Pierre Paradinas and others in work on the software suitable for a smart card for health card applications.
	Autumn	Ram Banerjee and others at GIS in Cambridge (UK) release the OSCAR operating system.
1990	Mar	In the Netherlands, Digicash BV starts development of a smart-card operating system to host the e-cash ultra-fast payment protocol. In that development effort Jelte van der Hoek and Jurjen Bos work on an interpreter, the J-Code engine in the card while Eduard de Jong designs the card OS, implemented by interpreted code, with a focus on security evaluation.
	Sep	Pierre Paradinas, Eduard Gordons and Georges Grimonprez complete work on an SQL implementation engine on for a smart card that results in a C-language interpreter in the card to interpret the C implementation of a subset of SQL based on interpreted C and file a patent on interpreting a high level language program in a card [Gor03]
1991	Oct	Conception by Eduard de Jong of the transacted non-volatile memory model
1992	Mar	Conception by Eduard de Jong of the three-phase interaction model
	Mar	Start of the European-funded ESPRIT-II project named CASCADE with participation form the RD2P team, Gemplus, ARM Ltd, Nokia and others with the objective to research new hard and software architectures for cards, including RISC and high level languages.
	Oct	QC Technology founded by Eduard de Jong and Boudewijn de Kerf, after both had left Digicash BV.
1993		Conception by Eduard de Jong of a model based development system using data definition models
	Feb	Jurjen Bos starts working for QC and begins developing the SCIL interpreter as implementation of a prototype for card OS Macsime.

[2] Not the cute Star Wars droid, but an acronym, in French, standing for "research and development on portable data"

-

1994	Feb	Eduard de Jong files a patent covering Execution contexts, the Object Oriented access control model and three-phase interaction protocol.
		Prototype of Macisme technology built as 2-CPU demo board, Marketing of Macsime technology started with visits to IBM, Gemplus, Gieseck&Devrient presenting the CPU-level interoperability and its capacity for high-level formal security evaluation.
1995	Mar	In California, Patrice Peyret and Phil Trice, with strong backing from Gemplus start Integrity Arts Inc with purpose to develop and commercialize an interpretive secure structure for cards and other portable devices.
	May	SUN announces Java at SunWorld '95
	May	At National Westminster bank in London, the Nat West Development Team, led by David Peacham and David Everett and engaged in developing the Mondex pre-paid purse card system, begin to develop the Mondex card operating system as a distinct project: Multos.
	Jun	Integrity Arts acquires the Macsime technology, and Eduard de Jong joins its ranks.
	Aug	Eduard de Jong with help from Phil Trice files a second Macsime technology patent that focuses on model-based development.
	Dec	Visa international signs a first contract with Integrity Arts.
1996	Spring	The Integrity Arts engineering team decides to use ADA as the application programming language for the TOSCA card OS because the Java language, while attractive is deemed unavailable due to its explicit restrictions on sub-setting.
	Sep	Conception by Eduard de Jong of the split Java VM with a perpetual processing model, and the Integrity Arts engineering team adopts Java as its application language target.
	Oct	Sun Microsystems, Inc. initiates discussions on the acquisition of Integrity Arts.
	Nov	Announcement of Java Card technology and release of the specifications 1.0 developed by Scott Guthery and Krisna at Schlumberger's Austin microprocessor development team that defines the Java Card Virtual Machine as add-on to a file structured smartcard with a per-session execution model. [Sun96??]
1997	May	At Java One, Rinaldo Di Giorgio presents basic features of the Java Card 2.0 specifications.
	Sept	Sun Microsystems, Inc. acquires Integrity Arts.

-

	Nov	Visa International introduces the Open Platform for post issuance management.
	Nov	Java Card 2.0 specification released by Sun Microsystems based on the split VM, perpetual processing models and incorporating some key technical features from the Macsime prototype as presented here.
1998	Nov	Release of the specifications 2.1 for the Java Card execution platform.
1999	Dec	Sun Microsystems, Inc. announces that 24 Million Java Card cards have been sold.
2001	May	Sun Microsystems, Inc. announces that 100 million Java Card cards have been sold
2002	April	Sun Microsystems, Inc. announces that 200 million Java Card cards have been sold and releases the specification 2.2 for the Java Card execution platform.
2003	Sept	Sun Microsystems, Inc. announces that 400 million Java Card cards have been sold.
	Nov	Release of the specifications 2.2.1 for the Java Card execution platform.
2005	June	Sun Microsystems, Inc. announces that 1 billion Java Card cards have been sold.
2006	April	Final release of the next specification of the technology.

Table 2 : History of Java Card technology in a nutshell

5 Related Work

There are a number of historical accounts on smart cards. Quisquater provides a brief survey [Qui97]. Jurgensen and Guthery [Jur02] provide an overview of multi-application card technology with an historic perspective.

6 Conclusions

The Java Card execution platform is a commercial successful product, basically because it fits the needs of mobile-phone operators in their relationship with card manufacturers and their need for short product lead times.

As witnessed by our references and our earlier survey [Har01], Java Card technology has attracted a significant number of researchers from main-stream areas such as operating systems, software engineering, programming languages and formal methods. They have discovered it as an attractive target for applying new ideas or for a testing ground for old ideas analyzing Java Card features and implementations. The Java Card execution

platform is a realistic application domain and yet its specification is relatively simple and small enough to manage in a research context.

A Java Card system has a number of powerful features, some of which are tried and tested (like Object Orientation), and others are truly innovative, such as the applet firewall.

The Java Card framework as a development tool is still in its early days. Life-cycle management is separate, and card/terminal co-development is still in the experimental stage. Model-based development, code generation and testing are currently receiving attention, thus raising hope that co-development might evolve also via this route.

Java Card development as a historical phenomenon is interesting in the sense that just a handful of people have generated most of the ideas, and a small number of companies proved instrumental in fielding the first systems in a relatively short period of time.

References

[Bon04a] S. Bonnet, O. Potonniée, R. Marvie, and J.-M. Geib. A model-driven approach for smart card configuration. In G. Karsai and E. Visser, editors, Generative Programming and Component Engineering (GPCE), volume LNCS 3286, pages 416-435, Vancouver, Canada, 2004. Springer-Verlag, Berlin.

[Bor01] J. Borst, B. Preneel, and V. Rijmen. Cryptography on smart cards. Computer Networks, 36(4):423-435, Jul 2001.

[Bre05a] C.-B. Breunesse, N. Catano, M. Huisman, and B. Jacobs. Formal methods for smart cards: an experience report. Science of Computer Programming, 55(1-3):53-80, Mar 2005.

[Che00] Z. Chen. Java Card Technology for Smart Cards: Architecture and programmer's guide. Addison Wesley, Reading, Massachusetts, 2000.

[Cog05] A. Coglio and C. Green. A constructive approach to correctness, exemplified by a generator for certified Java Card appplets. In IFIP Working Conference on Verified Software: Tools, Techniques, and Experiments, Zürich, Switzerland, Oct 2005.

[Dat95] C. J. Date. An introduction to data base systems. Addison Wesley, Reading, Massachusetts, sixth edition, 1995.

-

[Dev03] D. Deville, A. Galland, G. Grimaud, and S. Jean. Smart card operating systems: Past, present and future. In 5th USENIX/NordU Conf., pages 14-28, Vasteräs, Sweden, Feb 2003. Unpublished.

[Eve01] D. B. Everett, S. J. Miller, A. D. Peacham, I. S. Simmons, T. P. Richards, and J. C. Viner. Multi-application IC card with delegation feature. United States Patent and Trademark Office, Apr 2001. Patent Nr. US6220510.

[Gol99] D. Gollmann. Computer Security. John Wiley & Sons, Chichester, UK, 1999.

[Gor03] E. Gordons, G. Grimonprez, and P. Paradinas. Portable support with easily programmable microcircuit and method of programming this microcircuit. European Patent Office, Mar 1992.

[Gov03] S. Govindavajhala and A. W. Appel. Using memory errors to attack a virtual machine. In 24th Symp. on Security and Privacy (S&P), pages 154-165, Berkeley, California, May 2003. IEEE Computer Society Press, Los Alamitos, California.

[Gri03] G. Grimaud and J.-J. Vandewalle. Introducing research issues for next generation Java-based smart card platforms. In Smart Objects Conf. (SOC), pages 138-141, Grenoble, France, May 2003. France Telecom and CNRS.

[Gui01] L. C. Guillou, M. Ugon, and J-J. Quisquater. Cryptographic authentication protocols for smart cards. Computer Networks, 36(4):437-451, Jul 2001.

[Gut00] S. Guthery, R. Kehr, and J. Posegga. How to turn a GSM SIM into a web server. In J. Domingo-Ferrer, D. Chan, and A. Watson, editors, 4th Int. IFIP wg 8.8 Conf. Smart card research and advanced application (CARDIS), pages 209-222. Kluwer Academic Publishers, Boston, Massachusetts, Sep 2000.

[Har00b] P. H. Hartel, M. J. Butler, E. K. de Jong, and M. Longley. Transacted memory for smart cards. In J. N. Olivieira and P. Zave, editors, 10th Formal Methods for Increasing Software Productivity (FME), volume LNCS 2021, pages 478-499, Berlin, Germany, Mar 2001. Springer-Verlag, Berlin.

[Har94f] P. H. Hartel and E. K. de Jong. Towards testability in smart card operating system design. In V. Cordonnier and J.-J. Quisquater, editors, 1st Smart card research and

advanced application (CARDIS), pages 73-88, Lille, France, Oct 1994. Univ. de Lille, France.

[Har01] P. H. Hartel and L. A. V. Moreau. Formalizing the safety of Java, the Java virtual machine and Java card. ACM Computing Surveys, 33(4):517-558, Dec 2001.

[Hen00] N. J. Henderson and P. H. Hartel. Pressure sequence - A novel method of protecting smart cards. In J. Domingo-Ferrer, D. Chan, and A. Watson, editors, 4th Int. IFIP wg 8.8 Conf. Smart card research and advanced application (CARDIS), pages 241-256, Bristol, UK, Sep 2000. Kluwer Academic Publishers, Boston, Massachusetts.

[ISO95] ISO/IEC. 7816-4:1995 Information technology - Identification cards - Integrated circuit(s) cards with contacts part4: Inter-Industry commands for interchange. Int. Standards Organization, Geneva, Switzerland, 1995.

[ISO97] ISO/IEC. 7816-3:1997 Information technology - Identification cards - Integrated circuit(s) cards with contacts part 3: Electronic signals and transmission protocols. Int. Standards Organization, Geneva, Switzerland, 1997.

[Jon95a] E. K. de Jong. Data exchange system comprising portable data processing units. European Patent Office, Aug 1995. Patent Nr. EP0666550.

[Jon95] E. K. de Jong. Objects in smart cards. In B. Struif, editor, 5th GMD-Smart card Workshop, pages 12.1-12.6, Darmstadt, Germany, Jan 1995. GMD, Darmstadt.

[Jon00] E.K. de Jong and J. N. E. Bos US 6769053 Arrangement storing different versions of a set of data in separate memory areas and method for updating a set of data in a memory

[Jon00c] E. K. de Jong. Data exchange system comprising portable data processing units. United States Patent and Trademark Office, Jul 2000. Patent Nr. US6094656.

[Jon04] E. K. de Jong. Run time code integrity checks. United States Patent and Trademark Office, Jul 2004. Patent Application Nr. US20040143739.

[Jon94] E. K. de Jong and J. N. E. Bos. An application tool kit for smart cards: Security ``as you like it''. In Smart Card 1994, Day one: Market overview of Leisure, finance and

-

security, pages 76-81, London, UK, Mar 1994. Lowndes Exhibition Organisers, Peterborough, UK.

[Jur02] T. M. Jurgensen and S. B. Guthery. Smart Cards: The Developer's Toolkit (Paperback). Prentice Hall, Upper Saddle river, New York, 2002.

[Kek01] M. Kekicheff, F. Kashef, and D. Brewer. Open platform security. In I. Attali and T. P. Jensen, editors, 2st Int. Workshop Java on Smart Cards: Programming and Security (JavaCard), volume LNCS 2041, pages 98-113, Cannes, France, Sep 2001. Springer-Verlag, Berlin.

[Ker1883] A. Kerckhoffs. La cryptographie militaire. Journal des Sciences Militaires, 9:5-38, Jan 1883.

[Koc96] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS and other systems. In N. Koblitz, editor, 16th Advances in Cryptology (CRYPTO), volume LNCS 1109, pages 104-113, Santa Barbara, California, Aug 1996. Springer-Verlag, Berlin.

[Lec99] S. Lecomte, G. Grimaud, and D. Donsez. Implementation of transactional mechanisms for open smartcard. In Electronic Proc. Gemplus Developer Conf (GDC), Paris, France, Jun 1999. Gemplus SA, France.

[Ler02] X. Leroy. Bytecode verification on Java smart cards. Software - Practice and Experience, 32(4):319-340, 2002.

[Lop04] C. López-Ongil, R. Sánchez-Reillo, J. Liu-Jimenez, F. Casado, L. Sánchez, and L. Entrena. FPGA implementation of biometric authentication system based on hand geometry. In J. Becker, M. Platzner, and S. Vernalde, editors, 14th Int. Conf. on Field Programmable Logic and Application (FPL), volume LNCS 3203, pages 43-53, Leuven, Belgium, Aug 2004. Springer-Verlag, Berlin.

[Mil04a] B. Millier. BasicCard 101: Program your first smartcard. Circuit Cellar, the Magazine for Computer Applications, 164, Mar 2004.

[Pav04] M. Pavlova, G. Barthe, L. Burdy, M. Huisman, and J.-L. Lanet. Enforcing High-Level security properties for applets. In J.-J. Quisquater, P. Paradinas, Y. Deswarte, A. A. El Kalam, J.-J. Quisquater, P. Paradinas, Y. Deswarete, and A. A. El Kalam, editors, 6th

-

Int. Conf. on Smart Card Research and Advanced Applications (CARDIS), pages 1-16, Toulouse, France, Aug 2004. Kluwer Academic Publishers, Boston, Massachusetts.

[Pel95] T. Peltier. La carte blanche: un nouveau systeme d'exploitation pour objets nomades. PhD thesis, Univ. de Lille, France, Dec 1995.

[Phi03] J. Philipps, A. Pretschner, O. Slotosch, E. Aiglstorfer, S. Kriebel, and K. Scholl. Model-Based test case generation for smart cards. Electronic Notes in Theoretical Computer Science, 80:1-15, Aug 2003.

[Pra01] D. Praca and C. Barral. From smart cards to smart objects: the road to new smart technologies. Computer Networks, 36(4):381-389, Jul 2001.

[Qui97] J. J. Quisquater. The adolescence of smart cards. Future Generation Computer Systems, 13(1):3-7, Jul 1997.

[Ran96] W. Rankl and D. Weiss. System for conducting transactions with a multifunctional card having an electronic purse. United States Patent and Trademark Office, Jul 1996. Patent Nr. US5534683.

[Ree02] J. Rees and P. Honeyman. Webcard: a Java card web server. In J. Domingo-Ferrer, D. Chan, and A. Watson, editors, 4th Int. IFIP wg 8.8 Conf. Smart card research and advanced application (CARDIS), pages 197-208. Kluwer Academic Publishers, Boston, Massachusetts, Sep 2000.

[Sal75] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems. Proceedings of the IEEE, 63(9):1278-1308, Sep 1975.

[Sun03] Sun Microsystems, Inc., Java Card Platform, version 2.2.1, Runtime Environment Specification, http://java.sun.com/products/javacard/specs.html March 2003

[Ugo83] M. Ugon. Single chip microprocessor with on-chip modifiable memory. United States Patent and Trademark Office, May 1983. Patent Nr. US4382279.

[Van98] J.-J. Vandewalle and E. Vétillard. Developing smart Card-Based applications using Java card. In J.-J. Quisquater and B. Schneier, editors, 3rd Smart card research and advanced application (CARDIS), volume LNCS 1820, pages 167-182 and 105-124, Louvain la Neuve, Belgium, Sep 1998. Springer-Verlag, Berlin.