RC 22816 (W0306-052) 11 June 2003
Computer Science

IBM Research Report

Developers and Evaluators in Composite Evaluations Need Full Information

Paul A. Karger
IBM Research Division
Thomas J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598, USA

Helmut Kurth
atsec information security GmbH
Steinstr. 68
D-81667 Munich, Germany

Research Division
Almaden – Austin – Beijing – Delhi – Haifa – T.J. Watson – Tokyo – Zurich

This paper has been submitted to the 19th Annual Computer Security Applications Conference to be held in December 2003 in Las Vegas, NV. If accepted, it will appear in final form on the conference web site at http://www.acsac.org

Developers and Evaluators in Composite Evaluations Need Full Information

Paul A. Karger
IBM Corp., Thomas J. Watson Research Center
PO Box 704, Yorktown Heights, NY 10598, USA
karger@watson.ibm.com

Helmut Kurth
atsec information security GmbH
Steinstr. 68, D-81667 Munich, Germany
helmut@atsec.com

ABSTRACT

Four Common Criteria Certification agencies from France, Germany, the Netherlands and the UK have developed a concept of composite evaluations in which software developers and evaluators would not receive the full Evaluation Technical Report (ETR), but instead would only received an abbreviated ETR-lite. While ETR-lite is acceptable at low assurance levels, this paper argues that at high assurance levels, such an abbreviated report violates the basic principles of systems engineering and high assurance evaluation, and demonstrates that serious undetected security vulnerabilities can be the result.

1 Introduction

Several of the certifying bodies in the Common Criteria (BSI (Germany), CESG (UK), DCSSI (France), and NLNCSA (The Netherlands)) have developed a series of Supporting Documents [2-5, 7, 16] to assist in the evaluation of integrated circuits and embedded software, with smart cards particularly in mind. These supporting documents represent a good first step in developing methodologies for performing combined hardware and software evaluations. However, as they are only a first step, there are a number of potential problem areas in the documents, some of which could result in serious difficulties when performing such combined evaluations, particularly in the case where the hardware and software developers are different companies and some information is regarded as proprietary by one or the other of the developing companies or as proprietary by one of the CLEFs (commercially licensed evaluation facilities).

The Supporting Documents introduce the concept of a reduced Evaluation Technical Report, called ETR-lite, that omits large amounts of the technical information developed during the evaluation. The assertion of [4] and [5] is that this information is sufficient for the software developers and software evaluators.

However, at high assurance levels (EAL6 and above), the lack of information flow to the software developers and evaluators could introduce serious undetected security flaws due to insidious interactions between software and hardware. Evaluated individually, there may be no security flaws found, but only when viewed as a total system do the flaws become visible. The paper will show that composition of isolated evaluations could easily violate basic principles of systems engineering and basic principles of high assurance evaluation as defined by the Common Criteria.

The purpose of this paper is to identify some of these problem areas and to propose possible solutions that should be incorporated into future versions of the documents.

2 Lack of Information Leads to Undiscovered Vulnerabilities

There is an increased need for "secure" hardware as it can form the basis for highly reliable security functions implemented in software. While in the past it was often valid to assume that highly critical systems would operate in some kind of physically secure environment, this is no longer true for a number of today's embedded systems or in general for smart cards. This results in new kinds of attacks that use vulnerabilities introduced by specific effects a piece of software has when executed on specific hardware. In addition to Trojan Horse attacks that may exploit side effects of the hardware that can be observed by software operating on this hardware, the new kind of attacks use methods to observe hardware side effects in the environment of the system. This includes attacks based on externally measurable timing, power consumption, electromagnetic radiation or high signal frequencies on external interfaces.

Smart cards have especially suffered from these kind of attacks recently. Timing and power analysis attacks have received significant attention in recent years and have been the reason to withdraw some smart card based systems. As a consequence, both hardware and software required significant changes to counter these types of attacks requiring a suitable combination of hardware and software countermeasures. The cost of replacing a large number of issued smartcards with such a vulnerability can be extremely high, not to mention the loss of image and customer confidence in the technology.

In the US, the FIPS 140-1 [14] and the new FIPS 140-2 [13] standards for cryptographic products (especially at the level 4 defined there) also include an assessment of the hardware. But this standard focuses more on the physical protection features of the hardware. Vulnerabilities introduced by the combination of hardware and soft-

ware have been completely neglected in FIPS 140-1 and are only briefly mentioned in FIPS140-2. Although FIPS 140-2 looks at the combination of hardware and software, the potential vulnerabilities introduced by that combination are not addressed.

Common Criteria evaluations of smart card hardware have been become quite popular, especially in Europe. A significant number of new smartcard processors have successfully passed a hardware evaluation at quite high Common Criteria levels with a special emphasis on a sophisticated vulnerability assessment. But those evaluations are performed for the smart card chip alone, without any software included as part of the evaluation.

The current practice is to evaluate a combination of hardware and software by basically performing a vulnerability assessment of the software which takes the results of the hardware vulnerability analysis as the basis for the vulnerability analysis of the combination. The following section show a set of examples, where this approach is likely to fail, because the hardware evaluation could not foresee the way the software was using the functions of the hardware and where neither the hardware developer nor the evaluator might have considered some effects even worthy to mention to the software developer but where the combination of hardware and software introduces disastrous security problems.

That decomposing complex systems into separately designed and evaluated modular units could introduce reliability and/or security problems should not be a surprise. Leveson [36] argues that systems engineering was introduced into the aerospace industry, precisely to deal with these types of problems in complex aircraft design. While components would be designed in isolation and would appear to function properly, only when they were combined into a complex aircraft would problems arise from the interactions. Rao [39] argues that traditional software and systems design paradigms such as separation of concerns break down when system-wide concerns such as security need to be addressed..

This paper discusses hardware side effects that may result in vulnerabilities and addresses effects that can be exploited by observing and exploiting hardware side effects external to the system as well as internal within the system by Trojan Horses.

Those side effects can be categorized as follows:

1. "Internally visible" side effects
   - Incorrect implementation resulting in side effects of standard instructions
   - Timing side effects
   - Problems with pipeline architectures and caching

2. "Externally visible" side effects
   - Timing side effects
   - Power Consumption
   - Radiation
   - "Noise" signals on external interfaces

Those side effects can be used in attacks where we have to consider software based attacks as well as attacks on the hardware or a combination of those. This leads to new kinds of attacks not feasible for "classical" IT systems operating in a friendly or physically secured environment.

The examples we present in this paper are mainly "internally visible" side effects. The main reason is that vulnerabilities related to "externally visible" side effects are being analyzed in the last years (except for the problem of radiation, which has been known for some decades to be a critical problem).

3 Examples of Hardware/Software Interaction Vulnerabilities

This section will describe a number of examples of how complex interactions between hardware and software can result in vulnerabilities. Both the developer and the evaluator of a high-assurance software system need to have as much information as possible in these areas to avoid these vulnerabilities. All of these examples are of "internally visible" problems – they deal with issues in the architecture of the processor or the implementation as visible from software.

3.1 Flaws in Hardware Architecture

Design flaws in the hardware architecture can prevent an operating system from effectively enforcing security properties on running applications. The simplest such example are traditional smart card processors that have no supervisor state or memory protection. Any application can issue any instruction and touch any location in memory. The only hope for achieving security on such a processor is to either ensure all applications are totally trustworthy or to confine all applications to a hopefully secure interpreter, such as Multos [26] or JavaCard [27].

The more subtle issue is whether a hardware architecture that has a supervisor state and memory management has actually been designed without flaws. A good example of this can be found in the Philips *XA* architecture. The basic *XA* processor [18] has supervisor state and memory protection, but certain critical special function registers can be accessed from user mode. The *XA* is intended for automotive applications for which this is not a problem. However, the Philips Smart*XA* [11], designed for smart card applications, has a revised architecture, that resolves these security problems.

These types of security issues will be evident from careful reading of the relevant processor manuals, and such issues should be covered in the ETR-lite [4].

3.2 Flaws in Hardware Implementation

Merely having a processor architecture that is secure is, of course, insufficient. The implementation of that architecture must also be secure.

There was early concern on whether CPUs could fail during operation in a way that could lead to undetected security problems. Molho [38] studied the IBM System 360/50 as part of the ADEPT-50 project [42], and his work led to the development of the first so-called "subverter" program for Multics, described in section 3.2 of [30] by Karger and Schell.

The subverter was designed to test various combinations of illegal opcodes, illegal operands, and illegal memory references to ensure that the hardware protection mechanisms were actually implemented correctly and to ensure that random failures did not cause problems. The first subverter that Karger and Schell implemented for the Honeywell 645 processor [24] ran a series of fixed tests in background mode, one test per minute. After 1100 hours of testing, the subverter found no random failures, but found two implementation errors in the processor. One was an undocumented but benign instruction. The other was a way to bypass the hardware protection mechanisms to allow an unprivileged user to execute privileged instructions. They were able to construct a simple exploit program that allowed them to take complete control of the Multics operating system. Running the 645 subverter on the Honeywell 6180 processor [17] discovered another implementation flaw that permitted a denial of service attack on the 6180 mainframe. Hennigan [28] then developed an improved subverter for the Honeywell 6180 processor that did more extensive testing, but it did not find any further implementation flaws.

These early subverters were useful tools, but because they only had a fixed set of test cases, their test coverage was still rather limited. That they still found flaws only showed that traditional CPU testing focused almost exclusively on showing that legal operations were implemented correctly, rather than that illegal operations were correctly prevented from causing damage. The problem was that an enormous number of test cases would be required to ensure good coverage.

Digital Equipment Corporation developed [25] a better technique for ensuring broad test coverage in the early 1980s. They developed a program called AXE that generated random sequences of instructions and ran the same sequences on two different models of the same processor architecture. For example, their earliest tests compared the VAX-11/780 to the VAX-11/750. The AXE program did not check to see whether the instruction sequence executed correctly. Rather it checked to see if two different implementations gave identical results. If the results differed, then you knew that either one or the other implementation was flawed (or less likely that the AXE program was flawed). AXE would not catch cases in which two different implementations had identical flaws, but by running literally millions of test cases, AXE did a remarkable job of finding implementation flaws – much better than any human written set of test sequences could ever have accomplished. AXE was used for testing Digital's VAX VMM security kernel [32], including adding special subverter-type tests to the AXE test suite. More information about AXE can be found in [23] and [41]. IBM has developed similar test strategies as described by Aharon, et. al. in [19, 20].

Subverter-type testing is recommended in the Common Criteria [10, section 13.3, pg. 158] in the ATE_FUN section, but the actual requirement does not require it in all cases. "Functional testing is not limited to positive confirmation that the required security functions are provided, but may also include negative testing to check for the absence of particular undesired behavior (often based on the inversion of functional requirements)." If the ETR-lite does not say whether subverter-type testing was actually performed, the software developer and the software evaluation agency will have no way to know whether such testing was actually performed or not. This could lead to needless duplication of testing effort on the part of software developers and evaluators and that could significantly increase the costs of performing high assurance evaluations. Furthermore, what is the software developer or evaluator to do if their testing reveals problems that had not been tested for during the hardware evaluation?

3.3 Covert Channel Issues

There can be a variety of covert channel issues in the design of computer hardware. Frequently, these issues are not visible from the normal computer hardware user manuals, but can only be found, either by testing or by reading detailed information about the internal designs. Covert channel issues could be a particular problem in an ETR-lite concept, because many software systems do not consider covert channels at all. Only if the software system intends to implement mandatory access controls at high EAL levels will covert channels become an issue. As a result, it would be very easy for the hardware evaluation to completely omit any covert channel analysis or to do a cursory analysis and report in ETR-lite that there are no problems. Only when a later software development actually starts to implement mandatory access controls will the need for a detailed covert channel analysis become clear, but with ETR-lite, neither the software developer nor the software evaluator will be able to determine what if any covert channel analysis was actually done on the hardware. The next two subsections describe how such hardware covert channels could be serious problems.

3.3.1 Disk Arm Covert Channel in Controller

Disk drivers in operating systems have frequently implemented the so-called elevator algorithm to optimize disk arm motions. Schaefer, et. al. [40] identified an exploitable covert storage channel in such algorithms and developed alternate driver algorithms that avoided the

channel for the KVM/370 system. That covert channel had no relation to a hardware evaluation, because it was purely a software artifact. However, as disk controllers became more sophisticated over the years, the elevator algorithm and other similar optimizations moved from the operating system drivers into the disk controllers themselves, and the controller designs did not permit the operating system to directly prevent the covert channel. Worse still, the existence of these optimizations in the disk controllers were frequently undocumented, so the developer would not know of the covert channel without detailed knowledge of the internals of the controller. Karger and Wray showed how to deal with such covert channels in the disk controller in [31], but their countermeasures were only possible, because they had detailed knowledge of the internals of the controllers, knowledge that did not appear in the normal user manuals. Under the ETR-lite concept, there would no obligation for either the disk controller developer or the hardware evaluator to reveal that information to the software developer, assuming that the disk controller developer was even aware of the potential problem.

3.3.2 Covert Channel in SMP Interlock Hardware

Covert timing channels can exist in memory controllers for symmetric multi-processor (SMP) systems. Hu [29] shows how the fact that the VAX 8800 processor [37] uses only a single interlock in its memory controller, can be exploited as a high bandwidth covert timing channel. Nothing in the normal VAX documentation [35] would suggest that there was only a single interlock in the entire memory controller. Indeed, other VAX processor models do not suffer from this particular covert channel problem. Without this detailed internal design information about how the memory controller was built, a software developer would have no reason to expect such a covert channel. Once again, such information would be considered proprietary to the hardware developer and likely be excluded from the ETR-lite. Also, the hardware designer would not likely consider this to be an issue at all, and close communication between the hardware and software developer would be needed to address the issue properly. There would be a distinct possibility that the hardware evaluation could do no covert channel analysis at all, while the software developer would assume that an analysis had been done and that no problems had been found.

3.3.3 Covert Channels in Unpredictable/Undefined Operations in VAX and Alpha

Many processor architectures define certain illegal operations to be either unpredictable or undefined. For example, the value stored after division by zero might be defined to be unpredictable, in addition to generating a zerodivide exception. Leaving such items as unpredictable or undefined gives the hardware designer more freedom to design any particular implementation in the best way. The VAX architecture [35] has many such unpredictable and undefined operations. Invalid instruction prefixes are an example on the x86 architecture [12, section 11.1.1]. However, most processor architecture specifications do not allow for the possibility that an unpredictable result should depend in any way on information to which the current process does not have access. Such a store of a stale value from before the most recent context switch would be perfectly legal in the VAX architecture, yet it would constitute a quite serious covert channel.

By contrast, the Alpha architecture [1, section 1.6.3] explicitly declares that such covert channels are forbidden. In the Alpha, this would likely be a visible security function in the security target and the ETR-lite, but in most processors, the issue would not even be mentioned in the user manuals and would likely not be mentioned in the ETR-lite, yet this could be a critical security issue to a software developer, closely related to the subverter testing issues described in section 3.2.

3.4 Instruction Prefetch Queue Length

About 1981 the following problem was identified with a program running on an IBM minicomputer of the IBM Series/1 family [8]. The program was executing without a problem on computers with an older version [15] of the processor while it failed on computers with the new version of the processor. The problem could not be explained, since the manual claimed that both processors where "fully binary compatible". Investigating the problem in detail resulted in the following:

The developer of the program had used self-modifying code to save some space and time. So the code included an instruction that under some rare conditions modified another instruction near to it (just 3 instructions ahead).

As part of the investigation it turned out that the new version of the processor was enhanced by increasing the size of the instruction prefetch queue by two instructions. The hardware developer had considered this as a modification that would not affect any program and therefore did not consider this fact to be relevant to be mentioned in the user documentation. They also did not consider implementing any check in the processor to verify that instructions they had already cached in the instruction prefetch queue had not been modified. Self-modifying code was not foreseen by the hardware developers!

As a result, the instruction that was modified by the program was already in the instruction prefetch queue of the new version of the processor and the program then failed in some cases. If this code had been in security or safety critical software, the results could have been disastrous. Fortunately, it was not in such critical software.
As late as 1986, versions of the Series/1 manuals [9] still did not discuss the problem with self-modifying code.[1]

[1] The authors are uncertain if the particular Series/1 models and manuals cited were those involved in the original incident, as the detailed records are no longer available. However, we are grateful to Dawn Stanford of the IBM Archives in Somers, NY for locating these manuals for us.

3.5 Cache Vulnerabilities

The previous example described a potential vulnerability in a specific kind of cache: the instruction prefetch queue. But there are other examples of critical vulnerabilities introduced by caching.

Such a problem was identified just recently in a logical partitioning architecture. In this architecture, the hardware was designed in such a way that two processors always shared a common second level cache. The logical partitioning architecture allowed the allocation of the two processors to different logical partitions.

It is easy to see that Trojan horses in the two partitions could easily set up a covert channel by "measuring" the amount of the second level cache allocated to their own processor (via the time used for memory access). No measurement of the bandwidth of this channel was performed but from other examples it can be assumed that this is quite high.

3.6 Virtual Machine Timing Dependencies

When running an operating system on top of a virtual machine, similar problems arise as when running the operating system on a different "binary compatible" hardware platform. Also in this case, security problems may turn up that did not exist when the operating system is executed on the bare machine. To make it very clear: we are not talking here about the problems of interference between different virtual machines operating in parallel on the same physical hardware. Those problems need to be addressed in the security evaluation of the virtual machine monitor. Here we just focus on security problems arising within an operating system running on top of a virtual machine monitor instead of the bare hardware. Those problems exist even when there is just a single virtual machine started by the virtual machine monitor.

The example where these problems become immediately obvious is the case where a secure operating system that has been carefully analyzed for potential covert channels is executed within a virtual machine environment that provides virtual memory management. Regardless of whether the operating system itself uses virtual memory management techniques or not, new covert timing channels with potentially very high bandwidth are introduced with high probability within the operating system by the virtual memory management of the virtual machine monitor. It should also be obvious to the reader that an analysis of those covert channels can only be done with detailed knowledge about the virtual memory management strategy and algorithms implemented by the virtual machine monitor. In this respect, the channels introduced are similar to those introduced by caching or bus arbitration as part of hardware architectures.

Virtual memory management is just one example of an area where additional security problems can show up when an operating system runs in a virtual machine environment. One also has to consider that the timing characteristics of emulated instructions, I/O, interrupt handling, caching etc. are very much different from the bare machine and may become dependent on the state of the virtual machine where it is constant on the bare machine. As a result, one can deduce that high assurance for an operating system intended to execute in a virtual machine environment requires a combined analysis of the operating system, the virtual machine and the underlying hardware.

4 Smart Card Examples

Attacks on smart cards have received wide attention recently, especially since smart cards are applied in highly critical areas like electronic purses and for the generation of digital signatures. In addition, the assumption of a physically protected environment for the hardware commonly made in most vulnerability assessments and evaluations does not hold any more. For example, in the case of an electronic purse, the legitimate owner of the smart card may have the highest interest to penetrate its functions to set up his own money generating system.

A number of active and passive attacks against smartcards have been published in the last years, and some of them required significant modifications to both the hardware and the software to counter those attacks. The most widely recognized attacks were probably the simple and differential power analysis attacks as described by Kocher [33]. Attacks requiring active manipulation of the smart card hardware also have been published. [22, 34] give an overview on some of those attacks and their effects. We do not intend with this paper to go into further details on those kind of attacks.

What has not yet been analyzed in full detail in the existing papers are the effects of combining hardware and software related attacks. Those kind of attacks become critical with new multi-application smart cards that allow the download of applications in the field. This might allow an attacker to use a combination of hardware and software attacks provided he manages to get his application downloaded and executed on the smart card.

In a Common Criteria evaluation with a target level of AVA_VLA.4 for the vulnerability assessment one would expect that those combined attack methods are analyzed in sufficient detail to ensure that they require highly specific equipment and a detailed knowledge of the operation of both hardware and software. This of course requires the evaluator to analyze hardware and software in parallel with the same rigor. Focusing just on one of them, as is required under the ETR-lite concept, has the danger of missing a highly critical attack in the analysis.

An example of a document focusing just on the software aspects of a security function and neglecting potential hardware related vulnerabilities is [6], which provides very detailed guidance on how to test random numbers generated by a physical random number generator to ensure a high quality of those numbers. Based on the guidance provided in that guidance, one could easily conclude that the best random numbers to use are those that have undergone these extensive statistical tests. On the other

hand, when one performs those tests on a smart card, one be almost sure that the total set of random numbers tested can be obtained using differential power analysis techniques. Post processing by a pseudo random number generator does not help, since one can assume that the attacker knows the algorithm used. Careless use of the thoroughly tested random numbers e. g. for the generation of cryptographic keys, could make those keys known to the attacker before they are used the first time for a cryptographic operation. Unfortunately, the document [6] does not provide any guidance on how to securely use the thoroughly tested random numbers such that the attacker cannot extract critical information by differential power analysis or by the analysis of electromagnetic radiation. An easy fix of this problem would be to generate additional (non-tested) random numbers and use those to select the bits or bytes used from the large set of tested random numbers as the basis for the generation of cryptographic keys. One would expect guidance taking into account potential hardware vulnerabilities would be presented to help software developers avoid the risk of being easily attacked using a hardware vulnerability when they try to counter a potential software related attack.

5 Conclusion

While this paper has focused extensively on some of the drawbacks of the ETR-lite concept as part of composite evaluation, it is important to note that there are a large number of very good and useful concepts in composite evaluation. The supplementary documents lay out very well how hardware evaluations and software evaluations are likely to be separate, because the underlying technical issues are different. Furthermore, a single hardware evaluation may be used as the basis for several different software evaluations. The supplementary documents also make clear that information has to flow from the hardware evaluation to both the software developer and the software evaluator. They recognize that there may be legitimate proprietary interests held by the hardware developer and hardware evaluator, and lays out guidelines for information that must flow, regardless. At lower levels of assurance, there is much less of a problem with ETR-lite. For example, Albertson and Forge [21] report good results with ETR-lite, but the software in that case was only evaluated to EAL4.

The major problem is that the ETR-lite concept was developed without adequate consideration of that at higher assurance levels, the software developer and evaluator need considerably more information than would be the case at lower levels. If the software security target is claiming at least EAL5 or specifically "AVA_VLA.4 Highly Resistant", then the information specified in ETR-lite is almost certainly insufficient. The supplementary documents should specify that further information, including the complete ETR is likely to be essential for such an evaluation to be successful.

6 Acknowledgements

This paper has benefited greatly from comments by J. R. Rao and Elaine Palmer of IBM.

7 References

1. *Alpha Architecture Handbook*, Order Number: EC-QD2KC-TE, October 1998, Compaq Computer Corporation. URL: http://www.support.compaq.com/alpha-tools/documentation/current/alpha-archt/alpha-architecture.pdf

2. *Application of Attack Potential to Smartcards*, Version 1.1, July 2002, Bundesamt für Sicherheit in der Informationstechnik (BSI): Bonn, Germany. URL: www.commoncriteria.org/supporting_docs/2002-08-001.pdf

3. *The Application of CC to Integrated Circuits*, Version 1.2, July 2002, Bundesamt für Sicherheit in der Informationstechnik (BSI): Bonn, Germany. URL: www.commoncriteria.org/supporting_docs/2002-08-002.pdf

4. *ETR-lite for Composition*, Version 1.1, July 2002, Bundesamt für Sicherheit in der Informationstechnik (BSI): Bonn, Germany. URL: www.commoncriteria.org/supporting_docs/2002-08-003.pdf

5. *ETR-lite for composition: Annex A Composite smartcard evaluation : Recommended best practice*, Version 1.2, March 2002, Direction Centrale de la Sécurité des Systèmes d'Information (DCSSI): Paris, France. URL: www.commoncriteria.org/supporting_docs/2002-07-017A.doc

6. *Functionality classes and evaluation methodology for physical random number generators*, AIS 31, Version 1, 25 September 2001, Bundesamt für Sicherheit in der Informationstechnik (BSI): Bonn, Germany. URL: http://www.bsi.bund.de/zertifiz/zert/interpr/ais31e.pdf

7. *Guidance for Smartcard Evaluation*, Version 1.1, March 2002, Direction Centrale de la Sécurité des Systèmes d'Information (DCSSI): Paris, France. URL: www.commoncriteria.org/supporting_docs/2002-07-019.pdf

8. *IBM Series/1 - 4942 Processor Models A and B and Processor Features Description*, GA34-0157-0, April 1981, IBM Corporation: Boca Raton, FL.

9. *IBM Series/1 - Processor Models 30D, 31D, 60D, and 61D Description*, GA34-0253-2, January 1986, IBM Corporation: Boca Raton, FL.

10. *Information technology - Security techniques -- Evaluation criteria for IT security -- Part 3: Security assurance requirements*, ISO/IEC 15408-3, 1999, International Organization for Standardization.

11. *P16WX064 SmartXA-Family: Secure 16-bit Smart Card Controller*, Short Form Specification Revision 1.1, February 2001, Philips Semiconductors. URL: http://www-us.semiconductors.philips.com/acrobat/other/identification/smartxa_ls.pdf

12. *Pentium Pro Family Developer's Manual - Volume 2: Programmer's Reference Manual*, Order No. 242691, December 1995, intel Corporation.

13. *Security Requirements for Cryptographic Modules*, FIPS PUB 140-2, Change Notice 1, 10 October 2001, National Institute of Standards and Technology: Gaithersburg, MD. URL: http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf

14. *Security Requirements for Cryptographic Modules*, FIPS PUB 140-1, 11 January 1994, National Institute of Standards and Technology: Gaithersburg, MD. URL: http://csrc.ncsl.nist.gov/publications/fips/fips140-1/fips1401.pdf

15. *Series/1 Model 5 - 4955 Processor and Processor Features Description*, GA34-0021-0, November 1976, IBM Corporation: Boca Raton, FL. URL: http://www.spies.com/~aek/pdf/ibm/series1/GA34-0021-0_series1mod5proc.pdf

16. *ST-lite*, Version 1.1, July 2002, Bundesamt für Sicherheit in der Informationstechnik (BSI): Bonn, Germany. URL: www.commoncriteria.org/supporting_docs/2002-08-004.pdf

17. *Summary of the H6180 Processor*, 22 May 1973, Information Processing Center, Massachusetts Institute of Technology: Cambridge, MA.

18. *XA User Guide*, 1998, Philips Electronics North America Corporation. URL: http://www-us.semiconductors.philips.com/acrobat/various/XA_USER_GUIDE_1.pdf

19. Aharon, A., A. Bar-David, B. Dorfman, E. Gofman, M. Leibowitz, and V. Schwartzburd, *Verification of the IBM RISC System/6000 by a dynamic biased pseudo-random test program generator.* **IBM Systems Journal**, 1991. **30**(4): p. 527-538.

20. Aharon, A., D. Goodman, M. Levinger, Y. Lichtenstein, Y. Malka, C. Metzger, M. Molcho, and G. Shurek. *Test Program Generation for Functional Verification of PowerPC Processors in IBM*. in **Proceedings of the 32nd ACM/IEEE Design Automation Conference**. June 1998, San Francisco, CA Association for Computing Machinery. p. 279-285.

21. Albertson, H.-G. and F. Forge. *The modular approach: A composite product evaluation for Smart Cards*. in **3rd International Common Criteria Conference - Common Criteria: Delivering Information Assurance Solutions**. 13-14 May 2002, Ottawa, Ontario, Canada. URL: http://www.expotrack.com/iccc/proceedings/pdf/proceed/english/track3/pr041_e.pdf

22. Anderson, R. and M. Kuhn. *Tamper Resistance - a Cautionary Note*. in **Second USENIX Workshop on Electronic Commerce Proceedings**. 1996, Oakland, CA USENIX Association. p. 1-11.

23. Anderson, W. *Logical Verification of the NVAX CPU Chip Design*. in **1992 International Conference on Computer Design: VLSI in Computers and Processors**. 1992, IEEE Computer Society Press. p. 306-309.

24. Andrews, J., M.L. Goudy, and J.E. Barnes, *Model 645 Processor Reference Manual*, revision 4, 1 April 1971, Cambridge Information Systems Laboratory, Honeywell Information Systems, Inc.: Cambridge, MA.

25. Bhandarkar, D., *Architecture Management for Ensuring Software Compatibility in the VAX Family of Computers.* **Computer**, February 1982. **15**(2): p. 87-93.

26. Brown, M.D., *MULTOS Version 3 on Hitachi H8/3112 integrated circuit card*, UK ITSEC Scheme Certification Report No. P130, 13 September 1999, UK IT Security Evaluation and Certification Scheme, Certification Body: PO Box 152, Cheltenham, UK.

27. Chen, Z., *Java Card (tm) Technology for Smart Cards: Architecture and Programmer's Guide*. 2000, Boston: Addison-Wesley.

28. Hennigan, K.B., *Hardware Subverter for the Honeywell 6180*, December 1976, The MITRE Corporation, Bedford, MA: HQ Electronic Systems Division, Hanscom AFB, MA.

29. Hu, W.-M. *Reducing Timing Channels with Fuzzy Time.* in **Proceedings of the 1991 IEEE Symposium on Research in Security and Privacy**. 20-22 May 1991, Oakland, CA IEEE Computer Society. p. 8-20.

30. Karger, P.A. and R.R. Schell, *Multics Security Evaluation: Vulnerability Analysis*, ESD-TR-74-193, Vol. II, June 1974, HQ Electronic Systems Division: Hanscom AFB, MA. URL: http://csrc.nist.gov/publications/history/karg74.pdf

31. Karger, P.A. and J.C. Wray. *Storage Channels in Disk Arm Optimization*. in **Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy**. 20-22 May 1991, Oakland, CA p. 52-61.

32. Karger, P.A., M.E. Zurko, D.W. Bonin, A.H. Mason, and C.E. Kahn, *A Retrospective on the VAX VMM Security Kernel.* **IEEE Transactions on Software Engineering**, November 1991. **17**(11): p. 1147-1165.

33. Kocher, P., J. Jaffe, and B. Jun. *Differential Power Analysis: Leaking Secrets*. in **Proceedings of Crypto '99**. August 1999, Santa Barbara, CA:Lecture Notes in Computer Science Vol. 1666. Springer Verlag. p. 388-397.

34. Kömmerling, O. and M. Kuhn. *Design Principles for Tamper-Resistant Smartcard Processors*. in **Proceedings of the USENIX Workshop on Smartcard Technology (Smartcard '99)**. 10-11 May 1999, Chicago, IL USENIX Association. p. 9-20.

35. Leonard, T.E., *VAX Architecture Manual*. 1987, Bedford, MA: Digital Press.

36. Leveson, N.G., *Safeware: System Safety and Computers*. 1995, Reading, MA: Addison-Wesley.

37. Mishra, S.N., *The VAX 8800 Microarchitecture.* **Digital Technical Journal**, February 1987(4): p. 20-33.

38. Molho, L.M. *Hardware Aspects of Secure Computing*. in **1970 Spring Joint Computer Conference**. 5-7 May 1970, Atlantic City, NJ Vol. 36. AFIPS Press. p. 135-141.

39. Rao, J.R., *On the role of formal methods in security.* **Information Processing Letters**, 28 February 2001. **77**(2-4): p. 209-212.

40. Schaefer, M., B. Gold, R. Linde, and J. Scheid. *Program Confinement in KVM/370*. in **Proceedings of the 1977 ACM Annual Conference**. 16-19 October 1977, Seattle, WA p. 404-410.

41. Taylor, S., M. Quinn, D. Brown, N. Dohm, S. Hildebrandt, J. Huggins, and C. Ramey. *Functional Verification of a Multiple-Issue, Out-of-Order, Superscalar Alpha Processor - The DEC Alpha 21264 Microprocessor*. in **Proceedings of the 35th Annual Conference on Design Automation**. June 1998, San Francisco, CA Association for Computing Machinery. p. 638-644.

42. Weissman, C. *Security Controls in the ADEPT-50 time sharing system*. in **Fall Joint Computer Conference**. 1969, Vol. 35. AFIPS Conference Proceedings, AFIPS Press, Montvale, NJ. p. 119-133.