Proof Requirements in the Orange Book: Origins, Implementation, and Implications

Garrel Pottinger

Mathematical Sciences Institute
409 College Avenue
Cornell University
Ithaca, NY 14850

February 11, 1994

A report prepared for the Naval Research Laboratory under contract N000173-93-P-G934. Facts and opinions contained in this document are the responsibility of the author and not the Naval Research Laboratory.

Naval Research Laboratory, Code 5540, Washington, D.C. 20375-5337

For additional copies of this report, please send e-mail to
landwehr@itd.nrl.navy.mil, or retrieve postscript via
www.itd.nrl.navy.mil/ITD/5540/publications/index.html
(e.g., using Mosaic).

Proof Requirements in the Orange Book: Origins, Implementation, and Implications

Garrel Pottinger
Mathematical Sciences Institute
409 College Avenue
Cornell University
Ithaca, NY 14850

garrel@msiadmin.cit.cornell.edu

Final Deliverable under Contract N000173-93-P-G934.

February 11, 1994

Light NOT heat.

Contents

List of Tables vi

Preface vii

Acknowledgements viii

I INTRODUCTION AND OVERVIEW 1

1 Introduction 3
- 1.1 General Impressions 4
  - 1.1.1 Heterogeneous Complexity 4
  - 1.1.2 Forced Choices 6
- 1.2 Aims and Limitations 7
- 1.3 Structure of the Report 8

2 Overview 8
- 2.1 Narrative, 1961–1983 8
- 2.2 Flashback: High Policy and High Politics, November 1977–July 1990 12
- 2.3 Recent Trends 17

II TECHNICAL CONTEXT 19

3 Time-sharing 21

4 The Classical Computer Security Problem 22

5 The Software Crisis 26

6 Program Verification 28

7 Structured Programming and Design Verification 32

8 Security Kernels and the Reference Monitor Concept 35

9 Modeling Security **40**
9.1 Classification, Security Levels, and Clearances 41
9.2 Bell/La Padula 43
9.2.1 Basic Features 43
9.2.2 The *-property and Trojan Horses 45
9.2.3 Trusted Subjects 45
9.2.4 Covert Channels 46
9.3 Other Security Models 47

III CREATING THE CENTER AND ORANGE BOOK **49**

10 The DoD Computer Security Initiative **51**

11 Locating the Center **53**

12 Establishing the Center **56**

13 Writing the Orange Book **57**
13.1 Lineage 57
13.2 Proof Requirements 57

IV USING THE ORANGE BOOK **59**

14 A Guide to Sections 15 through 18 **61**

15 Information and Expertise **62**

16 Availability of Products **63**

17 Design Verification **64**

18 Proof Requirements and Endorsed Tools **68**

V COMMUNING WITH CLIO **69**

19 Historiographic Questions **71**

VI APPENDICES AND REFERENCES **73**

A Summary of Evaluation Criteria Classes, from the Orange Book, pp. 93–94 **75**

B The Center's Publications **77**
B.1 Final Evaluation Reports 77
B.2 The Rainbow Series 77
B.3 Other Publications 82

C Products Evaluated by the Center **83**
C.1 Operating System Products Rated C1–A1 86
C.2 Other Products 91

D Learning to Do It Right — From Autodin II to Blacker **93**

References **98**

List of Tables

1 Final evaluation reports published by year. 77
2 Kinds of Evaluated Products List product entries. 83
3 Evaluated Products List product entries by kind and year. . . 86
4 Operating system products by level of trust and year, including products for which evaluation has been completed and products in Formal Evaluation. 87
5 Rating Maintenance Phase Evaluated Products List entries included in data used to prepare table 4. 87
6 Result of removing the effect of Rating Maintenance Phase Evaluated Products List entries from table 4. 88
7 Result of excluding Evaluated Products List entries for products in Formal Evaluation from data used in preparing table 4. 89
8 Result of excluding Evaluated Products List entries for products in Formal Evaluation from data used in preparing table 6. 89
9 Result of excluding Add-on Evaluated Products List entries from data used in preparing table 8. 90
10 Bottom line comparison of tables 4, 6, 7, 8, and 9. 90
11 Comparison of percentages of products rated at levels C1–A1 according to tables 4, 6, 7, 8, and 9. 91
12 Comparison of yearly averages of operating system products evaluated derived from tables 4, 6, 7, 8, and 9. 91
13 Other products rated C1–A1, including those for which evaluation has been completed and those in Formal Evaluation. . 92
14 Products in Vendor Assistance Phase and Design Analysis Phase. 92

Preface

This report was written with three audiences in mind: specialists in computer security who wonder how their field came to be as it is, computer professionals mostly unacquainted with computer security who wonder what it's about and how it came to be, and historians of technology who wonder what kind of technology is involved in computer security and whether enough information about the area can be gathered to form the basis for a well grounded historical study.

Attempting to make the report intelligible to all three audiences, I have sometimes explained things for the benefit of one of the audiences (*e.g.*, for the benefit of historians of technology, notes explain what the integers are and what the assignment operator does) at the risk of boring or irritating members of the other two. I have tried to spread this kind of thing around pretty evenly (historians of technology need no explanation of who Clio is, but one is included at the beginning section 19 for the possible benefit of the other two intended audiences), and I hope readers will not be much bothered by it. The aim of being informative to as many as possible is worthwhile, and, really, it costs the reader little to skip material that explains things already known.

I alone am responsible for the factual accuracy, interpretive soundness, and clarity and style of what follows, and am to blame for whatever deficiencies the report may exhibit in respect of these qualities.

Garrel Pottinger

Ithaca, New York

Acknowledgements

The research resulting in this report was supported in part by the UK Economic and Social Research Council's Programme on Information and Communication Technologies (PICT) and the University of Edinburgh. Additional support was provided by the United States Navy through the Naval Research Laboratory and the United States Army Research Office through the Mathematical Sciences Institute of Cornell University.

This document could not conceivably have been written had not those who granted interviews, endured unannounced telephone calls, and responded to (sometimes exigent) messages sent by electronic mail been kind enough to do so. I am grateful to them all. It should be noted that the views they have expressed in response to my questions are their own, and not those of any organization by which they are or have been employed or which they otherwise represent or have represented.

Donld MacKenzie inspired the effort that led to the report's being written, made it possible for the research to be undertaken by arranging the support provided by PICT and the University of Edinburgh, and sustained me throughout the effort with friendship and advice. Carl Landwehr, by helping arrange the support provided by NRL, giving unstintingly of time, knowledge, and advice, and being patient beyond all expectation, made the report's completion possible.

I, and the report, have benefited greatly by the quality of Cornell's libraries and the diligence and skills of their staffs, including particularly the diligence and skills of Mary Patterson, Engineering Library.

My brother, Marion G. Pottinger, arranged through his employer, Smithers Scientific Services, Inc., to facilitate the support provided by PICT and NRL. Marion also made it possible for me to devote time to research by bearing much more of the burden of filial duty stemming from the illnesses and deaths of our mother and father than was his due.

Rachel Maines, my wife and a trained historian of technology, helped me both with the research and writing involved in producing this document and in ways that go beyond what I can appropriately and adequately express.

Part I
INTRODUCTION AND OVERVIEW

1 Introduction

The Orange Book (officially, Department of Defense *Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD) defines a hierarchy of security classes for computer systems. The hierarchy has seven levels — D, C1, C2, B1, B2, B3, A1, listed in order of increasing security. Beginning with class B2, some of the requirements used in defining assurance for the hierarchy of security classes mandate that the system design and/or specification be proved to have certain specified properties, and these proof requirements become increasingly stringent in passing from level B2 through level B3 to level A1.

This report addresses three main questions, focusing in each case on issues related to proof requirements. The questions are:

(1) Origins – Why and how was the Orange Book written?

(2) Implementation – What kinds of ambiguities and vagueness associated with the requirements defining assurance for the hierarchy of security classes came to light when producers of computer systems tried to build systems meeting these requirements, and how were ambiguities and vagueness resolved?

(3) Implications – What implications do the answers to questions **(1)** and **(2)** have for future attempts to specify assurance for computer systems and evaluate them according to the resulting criteria?

The report is based on a combination of documentary evidence and interviews with persons involved in the processes mentioned in **(1)** and **(2)**.

Although the report is focused on issues that arise from imposing highly technical and esoteric requirements on computer system development, the actions and interactions of a great variety of people and institutions have influenced the course of events discussed in what follows: technical specialists in logic and computer science, business executives, librarians, members of the defense and intelligence communities, the American Civil Liberties Union (ACLU), the United States Congress, and those involved in making high level policy in the Executive Branch of the Federal Government. The discussion raises questions that require answers based on a correspondingly diverse range of knowledge and expertise.

1.1 General Impressions

The breadth of the range of persons, institutions, and issues mentioned in the preceding paragraph suggests that computer security is a difficult subject. Furthermore, one would not expect, say, the defense and intelligence communities to interact amicably with the American Civil Liberties Union. In many cases, the duties of members of the defense and intelligence communities require them to proceed on the basis of the authoritarian rule that everything not explicitly permitted, including access to and dissemination of information, is forbidden. In contrast with this, the ACLU is generally committed to the libertarian principle that everything not explicitly forbidden is permitted and specifically committed to defending rights the exercise of which, according to the Constitution of the United States, *may not* be forbidden. The presence of such inherent conflicts between parties to disputes involved in deciding what to do about computer security problems suggests that, besides being difficult to understand, these disputes are likely to be heated.

Both suggestions are correct, and some of the reasons for this are clear from the example of the preceding paragraph and similar cases considered below. The following two subsubsections argue that, in addition, the difficulty of understanding computer security and the heat involved in discussions of the subject arise, in part, from essential features of the technical enterprise of mechanized computation.

1.1.1 Heterogeneous Complexity

It is commonplace to say that computers and computing are hard to understand and deal with because they are complex.[1] This, no doubt, is true. But it is not very informative, unless something is said about the *kind* of complexity involved — after all, in common speech, to say that something is complex is nearly synonymous with saying that it is hard to understand and deal with. Absent an account of the kind of complexity to which the difficulty of understanding and dealing with computers and computing is attributed, saying that the difficulty arises because computers and computing are complex is uncomfortably like saying that opium causes sleep because it has a dormative potency.

Two kinds of complexity involved in dealing with computers and computing and their effect on our understanding of computational issues have

[1] *E. g.*, [Sha90, *passim*.].

been well specified and scientifically studied: complexity due to the enormous number of possibilities that arise for combinatorial reasons in many common well defined problem solving situations [NS72] and complexity involved in hierarchical systems composed, ultimately, from elements that are, in themselves, very simple [Sim81].

But there is another kind of complexity that bears on the difficulty of understanding and dealing with computers and computing that seems quite different from the kinds of complexity mentioned in the preceding paragraph — to understand mechanized computation we must avail ourselves of information drawn from an extraordinarily broad range of disciplines, and, in fact, the range is so broad that no single person can encompass all that is necessary. In what follows, I will call this kind of complexity "heterogeneous complexity".

Heterogeneous complexity manifests itself, to begin with, in the technical enterprise of computer system design, construction, and operation. The technical base of this enterprise rests on information drawn from, at least, the following fields: electrical, electronic, optical, and systems engineering; various branches of mathematics, including combinatorics, computational complexity theory, algebra, automata theory, and numerical analysis; logic; linguistics; psychology; and ergonomics.

Since heterogeneous complexity is an inherent feature of the technical enterprise of computer system design, construction, and operation, the enterprise is essentially social — the Lone Ranger need not apply. Furthermore, due to the quite different viewpoints represented by the participants in the enterprise, communication among them is often difficult. In effect, the same factors that require cooperation and communication also *make them* difficult.

So far, discussion of heterogeneous complexity has been confined to the technical enterprise of mechanized computation. But, in general, the discussion cannot be so confined, and, certainly, it cannot be so confined in considering computer security, as the sequel makes abundantly clear. Consequently, understanding issues of computer security requires drawing on the resources of an even wider range of disciplines than those suggested by the list given two paragraphs above. In turn, this requires achieving cooperation and communication among persons with even more diverse viewpoints than those involved in the technical enterprise of dealing with computers and computation. The result of all this is that the same factors that require cooperation and communication make them very difficult indeed.

1.1.2 Forced Choices

It is not uncommon for technical disputes about mechanized computation to become heated. The preceding discussion of heterogeneous complexity furnishes a partial explanation of this fact — participants in such disputes often approach the issues from quite different points of view, and, consequently, misunderstand and talk past each other. But this cannot be a complete explanation of the sometimes heated character of technical disputes about computers and computation, because such disputes often involve people with very *similar* overall viewpoints. To non-disputants, at least, the issues may appear to be about matters of detail, but, clearly, the details *matter* to the disputants.

In fact, the details do matter. In such technical discussions, details may be, and often are, enough to distinguish different programs of research and development, and people become passionate in arguing for programs of research and development that they favor and against those that they do not.

When someone competent to understand the technical issues sits down in a cool hour to reflect on the merits of the competing positions, the rational conclusion often is that there are no conclusive reasons for singling out any of the approaches under consideration as being better than the others. But technical practitioners do not indulge in this kind of reflection very often, for two reasons. First, pursuing a program of research and development, which is an integral part of being a technical practitioner, requires commitment, and commitment involves passion. Second, it is not feasible to let a hundred flowers bloom here.[2] Programs of computational research and development require large resources, and resources are limited. Often, an approach *must* be singled out, even if the choice cannot be determined fully by rational consideration of the competing alternatives. Clearly, arguments about economically forced choices involving people passionately committed to different alternatives are not likely to be calm and measured affairs.

Forced choices are an inevitable feature of dealing with computer security, because deciding very large questions about research and development policy is an essential part of planning ways to deal with the problems involved. It will become clear as the discussion proceeds that computer security problems arise because of a need to share information processing

[2] And elsewhere. For example, such an approach is infeasible in research and development efforts aimed at producing particle accelerators of great power, passenger rail systems, and systems for use in the space program.

resources among users of very different degrees of trustworthiness. No piecemeal approach to dealing with a such a situation can succeed, and the heat arising in technical discussions involving the forced choices required by general approaches adds itself to that generated by conflicts of principle and other conflicts to be described in what follows.

Another feature of technical discussions driven by forced choices is worth noting. Of necessity, such discussions are partisan, and believing what one is arguing for is a wonderful help in such cases. Commitment to a program of research and development tends, in itself, to foster belief in the superiority of that program over competing alternatives. If one is technically competent and articulate, belief is likely to enhance one's effectiveness in argument, which, in turn reinforces the belief, thereby leading to further enhancement of effectiveness in argument, a further strengthening of belief, greater effectiveness in argument, and so on. This kind of feedback process can lead a technical community to prefer a program of research and development over its competitors and to see the choice as being determined by the merits of the competing programs in cases where dispassionate reflection on the evidence adduced in support of the community's preference leads to the conclusion that it is insufficient to warrant the choice.

Depending on whether a program chosen due to the sort of process just described succeeds, in hindsight its proponents may appear to have been either insightful or disingenuous. If the program succeeds, the reaction may be something like "Remarkable that they were so persistent and forceful; they must have realized the weakness of their position!", and if not, it may run along the lines "Deplorable that they persisted so; they sold the other parties to the discussion a bill of goods!" But if, in fact, the course of a discussion driven by a forced choice is determined by the sort of process described in the preceding paragraph, the proper conclusion is that the appearance of foreknowledge or chicanery, as the case may be, is an illusion. Feedback, rather than foresight, drives the process, and it needs no assistance from knavery to determine the outcome. This point must be borne in mind in thinking about the events discussed below.

1.2 Aims and Limitations

Due to constraints on available support, the aims of the research effort that resulted in writing this document were limited to producing a report that would be informative and accurate in itself and would also furnish a sound basis for further research devoted to the topics discussed here. A number of

ways in which the research done so far could, and should, be extended are mentioned in the body of the document.

1.3 Structure of the Report

Section 2, the remaining section of part I, provides an overview of the events discussed in parts II–IV. Part II lays out the technical context in which the Orange Book was written. Part III describes the writing of the Orange Book, and Part IV discusses what happened when it was used and draws out some of the implications of the events involved.

Part V discusses how well some of the models currently used by historians of technology fit the course of events described and analyzed in parts I–IV.

Part VI contains appendices and the report's references.

2 Overview

Subsection 2.1 is a narrative outline of the events of central concern in this document. Subsection 2.2 is a flashback that fills in the context of high level policy and politics within which the events of central concern occurred, and subsection 2.3 rounds off the overview by briefly discussing recent trends related to the events recounted in subsections 2.1 and 2.2.

2.1 Narrative, 1961–1983

It was observed in subsubsection 1.1.2 that computer security problems arise because of a need to share information processing resources among users of very different degrees of trustworthiness. Section 3 describes how this sort of resource sharing first became a prominent possibility through the introduction of time-sharing operating systems, and section 4 discusses how the security problem posed by sharing computational resources in the context of time-sharing operating systems — the classical computer security problem[3] — came to be recognized by the United States defense community.

Crudely put, the classical computer security problem amounts to saying "How can you allow users who should be able to get at sensitive information that's in a computer system to do it, while preventing users who shouldn't from doing it too?" This is a hard question, because answering it requires solving a combination of thorny technical and conceptual problems.

[3] The terminology is drawn from [L+80, p. 8-25].

An operating system is a collection of software that controls the resources of a computer system and provides users and programs access to those resources. Solving the classical computer security problem requires figuring out how to build appropriate operating systems, building them, and providing evidence that the resulting systems, in fact, behave as they should. Since an operating system is a collection of software, it is clear that prospects for solving the classical computer security problem cannot be better than prospects for making intelligible, reliable software and showing that there is good reason to think it reliable.

As we shall see in section 5, at approximately the same time the classical computer security problem was recognized by the defense community, the computer community at large became acutely conscious that prospects for making intelligible, reliable software and providing reasonable assurance of its reliability were very dim, given the software development techniques that were then known and used. This situation was dubbed the "software crisis", and the need to devise techniques for dealing with it was keenly felt by computer professionals and sophisticated computer users. Embryonic forms of techniques intended for this purpose were soon forthcoming. Two such techniques had an extremely strong influence on planning approaches to handling the classical computer security problem — program verification and structured programming. Sections 6 and 7 recount aspects of the origins of these techniques that are particularly relevant to the purposes of this report.

Program verification and structured programming promised to help with the general problem of making intelligible, reliable software and providing reasonable assurance of its reliability, but, in themselves, they provided no information about the specific problems of operating system design and construction peculiar to the classical computer security problem. In order to devise a plan for dealing with the classical computer security problem, it was necessary to produce a precise doctrine about what features an operating system should have in order to provide information to users entitled to it, while denying information to users who were not. Beyond this, given the necessary doctrine, it was also necessary to invent techniques for designing and building systems that would behave in accordance with the doctrine. Sections 8 and 9 discuss the doctrine that was, in fact, produced by the defense community pursuant to the need to plan a way of dealing with the classical computer security problem, and part III includes information about the associated design and implementation techniques and the initial period of experimentation that provided evidence of their practical utility.

The developments discussed so far can be dated as follows.

Time-sharing operating systems: The first experimental general purpose time-sharing system had been built and demonstrated at MIT by November 1961 [FC71, p. 79].

Classical computer security problem: Recognized during 1967–1972. "In October 1967, a task force was assembled ... to address computer security safeguards that would protect classified information in remote-access, resource-sharing systems," [NCS85b, p. 1]. This resulted in [War70], which is always cited in the Orange Book literature as the first report on the classical computer security problem. See [War70, p. vii] for detailed information on the origins of the report.

According to [Jel85, pp. 68–69], "1967 appears to be the year when computer security began to receive some official attention." The same passage notes that, in addition to the United States Department of Defense (DoD), the Central Intelligence Agency become concerned about computer security at this time, and the Advanced Research Projects Agency "initiated funding for the development of the ADEPT-50, the first recorded general purpose operating system designed to implement DoD security policy." [War70] and its origins are also discussed.

Understanding of the problems involved in providing adequate security safeguards for computer systems was deepened and clarified by [And72].

Software crisis: Recognized during 1967–1968. The software crisis was a major topic at the NATO conference on software technology planned in 1967 and held at Garmisch, Germany in 1969, [NR69]. See [Sha90, pp. 100–103].

Program verification: First influential papers 1967, 1969. [Flo67] is the first consequential paper on the subject. [Hoa69] applies the ideas of [Flo67] to program texts, rather than flow charts. The approach of [Hoa69] has been very influential in subsequent work on program verification.

Early work in the area — [GvN47] and [Tur49] — had no influence on subsequent developments.[4] In particular, neither [Flo67] nor [Hoa69] cites either of [GvN47] and [Tur49].

Structured programming: Fundamental papers 1971, 1972. [Wir71] and [Par72] are the seminal papers in the area. See [Sha90, pp. 111ff.] for discussion of the subsequent developments.

Doctrine: Developed during 1972–1973. The relevant publications are [And72] and [BP74b]. [And72] is cited in [NCS85b, pp. 1 and 66]. [BP74b] is

[4] I am indebted to Donald Good for these references.

often cited as the other main source for the essential ideas (see, for example [Nib79a], [Lan81], and [Sch89]), though [NCS85b, p. 66] refers, instead, to the later [BP76]. See also [BP74a, BP75].

Techniques and Experimentation: Ideas developed during 1971–1973.

According to [Jel85, p. II-73], the ideas did not originate with the panel that produced [And72] — the Anderson panel, but "grew out of some earlier work at ESD [Electronic Systems Division, Air Force Systems Command] involving Roger R. Schell." On the other hand, [Wal80, figure 1, p. 656] gives 1973 as the year of origin for security kernels, which are the fundamental components of the design and implementation techniques that were devised.

Interview evidence resolves this apparent conflict — the *ideas* did, indeed, grow out of Schell's work at ESD, which began in 1971 [Scha, p. 6], and antedated the Anderson panel, but the first attempt to *implement* the ideas, which took place at MITRE [Sch73], [Sch75], resulted from a recommendation made by the Anderson panel [Sch93b, A202-224, A321–A332].

It is clear from [Wal80, p. 655] that [Wal80, figure 1, p. 656] refers to the MITRE security kernel implementation effort, which began a period of experimentation that continued through 1976 [Wal80, pp. 655-656], [Jel85, p. II-74].

By 1977, the time was ripe to formulate a systematic plan for dealing with the classical computer security problem, and the United States Department of Defense did so forthwith. The resulting DoD Computer Security Initiative is discussed in section 10.

An important part of the plan laid out by the DoD Computer Security Initiative was to produce a standard for rating computer systems with regard to security and to establish an evaluation center that would rate computer systems according to the standard and publish the results. Section 11 describes the process that led to deciding to locate the evaluation center at the National Security Agency (NSA), and section 12 describes how this decision led to the formation of the DoD Computer Security Evaluation Center and discusses the Center's[5] original charter. Section 13 discusses the writing of the Orange Book, the standard used in the Center's evaluation of computer systems. Section 14 is a guide to sections 15–18, which discuss four aspects of processes and products associated with the creation of the Orange Book and the activities of the Center that have been chosen for analysis in this

[5] From now on, I will use the phrase "the Center" to refer to both the DoD Computer Security Evaluation Center and its descendant the National Computer Security Center.

report.

Dates for the developments described in the preceding two paragraphs are the following.

DoD Computer Security Initiative: Started, 1977 [NCS85b, p. 1]. Manifesto, 1978 — [L+80], described by [NCS85b, p. 1] as "a definitive paper on the problems related to providing criteria for the evaluation of technical computer security effectiveness." The description is correct, but there is much more to [L+80] than a technical discussion of evaluation criteria, as section 10 shows.

Locating the evaluation center: 1979–August 1980 [Jel85, pp. II-78–II-81], [Wal93a]. The decisive event was a meeting between Stephen T. Walker and Vice Admiral Bobby R. Inman, Director, NSA, that took place on August 4, 1980.

Establishing the Center: September 1980–October 1982. [Jel85, pp. II-81–85] recounts the astonishingly swift series of events leading from the agreement reached by Walker and Inman on August 4, 1980 to the signing of the memorandum that established the Computer Security Evaluation Center on January 2, 1981. [DoD82b], issued on October 25, 1982, provided the Center with an official charter.

Writing the Orange Book: 1978–1985. The technical discussion of evaluation criteria in [L+80] is, in effect, the first stage in the writing of the Orange Book, and [Nib79a] is the second, according to [Wal93a]. [CSE82] is the first version of the final document produced by the Center. Revisions led to [CSE83b] and then to [CSE83c], the first non-draft version of version of the Orange Book issued by the Center. The changes made in moving from [CSE83c] to [NCS85b] were minor.

Evaluating products: 1983–1993. Evaluations began in late 1982 or early 1983 and were "pretty informal, at first," according to Bret Hartman [Har93a]. According to [Bon93a], "evaluations" of a sort were taking place prior to 1982. [Jel85, p. III-31], citing [CSE83a], notes that a formal evaluation was under way by April 15, 1983, prior to the issuance of [CSE83c]. Information about recent evaluations can be found in [NSA93a].

2.2 Flashback: High Policy and High Politics, November 1977–July 1990

The issues of policy and politics involved in the story outlined so far relate primarily to events at or below the level of cabinet departments. The relevant policy and political developments at the higher levels of the executive

and legislative branches of the United States Government divide naturally into three periods.

First period: November 1977–August 1984. The salient documents are Presidential Directive 24 (PD 24) [PD77] and the Paperwork Reduction Act of 1980 [Con80a].

PD 24 mandated a split in responsibility for security of U.S. government communications [Jel85, p. II-52]. DoD's long-standing responsibility for protecting communication of information related to national security was to continue. But responsibility for protecting communication of "government-derived unclassified information (excluding that relating to national security)" was assigned to the Department of Commerce (DoC), along with responsibility for "dealing with the commercial and private sector to enhance their communications protection and privacy."[6]

Despite its innocuous title, the Paperwork Reduction Act of 1980 "can certainly be interpreted as mandating a rather vigorous program in COMPUSEC [computer security] within the U.S. government" [Jel85, p. II-91], with the Office of Management and Budget (OMB) having high-level responsibility for the program in question. In fact, the Act was so interpreted by the General Accounting Office [Jel85, p. II-91] and others.[7]

Responsibility for computer security within DoD was shortly to be given to NSA, with the Center having primary responsibility for the conduct of NSA's program activities in the area [DoD82b]. Since the Paperwork Reduction Act of 1980 was interpreted as requiring a computer security program for the government as a whole, and NSA's computer security activities were confined to DoD, the effect of the Act was to split responsibility for computer security along roughly the same lines as the split in responsibility for communication security instituted by PD 24. But there was a significant difference — PD 24 at least assigned responsibility for non-DoD communication security to a definite cabinet department, DoC, but no assignment of responsibility for non-DoD computer security at a similar level was established by the Paperwork Reduction Act of 1980.

As events unfolded in the period currently under discussion, the organization formed by DoC to discharge its responsibility for communication security, the National Telecommunications and Information Administration, proved to be unsuccessful and short-lived [Jel85, pp II-55–II-66], and no

[6] The quotations are as given in [Jel85, p. II-52] and are from from [PD77, p. 4].

[7] "Security guidance for Federal automated information systems is provided by the Office of Management and Budget," [CSE83c, p. 70] and [NCS85b, p. 72].

organization was charged with actually doing something about non-DoD computer security. Not to put too fine a point on it, the result was that, although DoD needs were being seen to, by 1984 the overall situation of the U.S. government, as regards both communication security and computer security, was a mess [Jel85, chapter 7].

Second period: September 1984–December 1987. The point of National Security Decision Directive 145 (NSDD 145) [NSD84] and National Telecommunications and Information Systems Security Policy 2 (NTISSP 2, the Poindexter memorandum) [SSS86] was to do something about the mess.

NSDD 145 unified high-level responsibility for communication security and computer security by forming the National Telecommunications and Information Systems Security Committee (NTISSC) to formulate operational policy for both. Top-level responsibility for carrying out policies formulated by NTISSC was assigned to the Secretary of Defense, acting in the newly created capacity of Executive Agent of the Government for Telecommunications and Information Systems Security. In turn, the Director, NSA, was designated National Manager for Telecommunications Security and Automated Information Systems Security, and was given broad powers to act under the authority of the Secretary of Defense in prosecuting NTISSC policies. A Systems Security Steering Group was also instituted to have general oversight of the structure so far described and to inform the President about its performance via the National Security Council.

One positive effect of this structure was to give NSA full scope to apply its technical capabilities to the general problem of seeing to communication and computer security throughout the government. But the Agency[8] needed guidance as to how those capabilities were to be employed — in particular, the kinds of information that were to be protected had to be defined, and it had to be determined who would apply the definition to the individual communication systems over which information was transmitted and the individual computer systems in which information resided and was processed. The job of NTISSP 2 was to spell this out. In particular, the job of NTISSP 2 was to spell it out for "sensitive, but unclassified information", as the document's title indicates [SSS86].

"Sensitive, but unclassified information" was defined by NTISSP 2 as follows [Rep87a, p. 39]:

> Sensitive, but unclassified information is information the disclosure, loss, misuse, alteration, or destruction of which could

[8] From now on, I will use the phrase "the Agency" to refer to NSA.

> adversely affect national security or other Federal Government interests. National security interests are those unclassified matters that relate to the national defense or the foreign relations of the U.S. Government. Other government interests are those related, but not limited to the the wide range of government or government-derived economic, human, financial, industrial, agricultural, technological, and law enforcement information, as well as the privacy or confidentiality of personal or commercial proprietary information provided to the U.S. Government by its citizens.

Responsibility for applying this definition was assigned to the heads of the government's various departments and agencies [*ibid.*]:

> This policy assigns to the heads of the Federal Government Departments and Agencies the responsibility to determine what information is sensitive, but unclassified and to provide systems protection of such information which is stored on telecommunications and automated information systems.

And the following duties were assigned to the Director, NSA, acting in the new capacity created by NSDD 145 [Rep87a, p. 40]:

> The National Manager shall, when requested, assist the Federal Government Departments and Agencies to assess the threat to and vulnerability of targeted systems, to identify and document their telecommunications and automated information systems and protection needs, and to develop the necessary security architectures.

Falling between the institution by NSDD 145 of changes in NSA's overall role in the Government and the signing of NTISSP 2 were some changes in NSA's internal structure. In December 1985 the Agency's computer security and communication security programs were "merged ... into an integrated organization called Information Security" [NSA86], and, concomitantly, the DoD Computer Security Evaluation Center became the National Computer Security Center.

These changes harmonized NSA's internal structure with its expanded external role and gave the Center a more imposing name, but they entailed retreating from one of the basic principles agreed to by Walker and Inman when the Center was formed, namely, that it should be independent of the

Agency's communication security program [Wal93a], [Jel85, pp. II-81–II-83]. Nevertheless, *pace* Walker and Inman, NSDD 145's unification of computer security and communication security under the policy and administrative structure the Directive brought into being led directly to corresponding changes in NSA's structure [NSA86]:

> The responsibilities of the new organization [Information Security] are being broadened under the auspices of National Security Decision Directive 145 to include all computer security and communication security for the Federal Government and private industry, including the protection of classified information; unclassified, national security sensitive information; and non-national security sensitive information. This broadening of responsibility was requested by the Department of Treasury and approved on December 20th [1985] at a meeting of the [Systems] Security Steering Group that was organized under NSDD 145.

Organizational changes at NSA weren't the sort of stuff of which national political issues are made, but the changes in top level Government policy and structure that brought them about were. NSDD 145 itself caused political rumblings. Congressman Jack Brooks, for example, reacted as follows [Bro85]:

> I believe that NSDD 145 is one of the most ill-advised and potentially troublesome directives ever issued by a President.

The addition of NTISSP 2, together with visits to libraries and firms providing database services by various Federal agents acting pursuant to and/or concurrently with implementation of NSDD 145 and NTISSP 2 [Rep87b, Rep87a], produced a full-blown Congressional tempest that swept away NTISSP 2 just four and one half months after its issuance [Car87a], shook the administrative structure instituted by NSDD 145 [Car87b], and left a new piece of legislation in its wake.

Third period: January 1988–July 1990. The Computer Security Act of 1987 [Con88] assigned to the National Bureau of Standards (NBS) — subsequently the National Institute of Standards and Technology (NIST) — responsibility for producing computer security standards and guidelines applicable to Federal computer systems other than those that handle classified information. The Act let stand NSA's responsibility for the corresponding functions regarding Federal computer systems that process classified information and mandated close cooperation and resource sharing between the

two agencies. Although the Act does not explicitly mention communication security, it is interpreted as providing for a similar cooperative division of responsibility in that area [Rep92, p. 3].

To promulgate a law is one thing, to obtain compliance another. Obtaining compliance with the Computer Security Act of 1987 involved seeing to it that the planning, training, and standards making activities required by the Act proceeded satisfactorily, on the one hand, and making sure that NSDD 145 was revised to bring Executive Branch policies into line with the Act's provisions, on the other. Congress kept close watch on both aspects of the compliance problem [Rep89, Rep90, Rep91, Rep92].

Harmonization of Executive Branch policy with the Computer Security Act of 1987 was achieved by the signing on July 5, 1990 of a National Security Policy Directive that removed the features of the policy instituted by NSDD 145 that were inconsistent with the Act's provisions [Rep90, p. 61], [SPD90], [Dan90a]. Simultaneously, NSA's internal structure was again altered [Dan90c, Dan90b] by fully integrating the computer security and communication security aspects of the Agency's Information Security organization, thereby completing the the process begun in the 1985 reorganization and moving the Center still further from the form originally intended by Walker and Inman.

2.3 Recent Trends

Viewed historically, consideration of what happened after July 1990 brings the overview given in this section to current events. The remainder of the section briefly discusses recent trends.

In the short term, the effect on computer security of the Computer Security Act of 1987 was probably negative. But the Act made excellent political sense, in the deepest sense of the word "politics" — the principle that civilian authorities control the military, and *not* the other way around, is so deeply established in American law, beginning with the Constitution, and in American political tradition that the scheme embodied in NSDD 145 and NTISSP 2 was almost certain to fall apart eventually. Very likely, sooner was better than later would have been.

Be that as it may, reports of the Center's death [Dan90c, Dan90b] turned out to be greatly exaggerated — its structure may have been hidden almost completely by NSA's traditional mantle of secrecy, but, as far as function is concerned, the Center seemingly just keeps rolling along, chthonian, perhaps, but enduring. (See sections 15 and 16.)

The cooperation between NSA and NIST mandated by the Computer Security Act of 1987 has been sufficiently vigorous to produce the draft *Federal Criteria for Information Technology Security* [NIS92a, NIS92b]. And there is more cooperation to come, much more, and not just between the Agency and NIST [NIS93, pp. 507-508]:

> The United States, Canada, and the European Community have agreed to work together to develop the Common Criteria (CC) which will harmonize all the existing criteria. This effort is expected to begin in early fall of 1993 and be completed in the spring of 1994.[!] Specific inputs include: 1) the Information Technology Security Evaluation Criteria (ITSEC)[9] and the experience gained to date with ITSEC in the form of suggested improvements; 2) the Canadian Trusted Computer Product Evaluation Criteria (CTCPEC);[10] [3)] the draft Federal Criteria for Information Technology (FC) and the comments received on the draft FC document, including the results of the FC invitational workshop; and 4) the Trusted Computer System Evaluation Criteria (TCSEC) and experience gained over the past ten years in conducting trusted product evaluations. The resulting CC will then undergo extensive international review and testing before becoming an international standard.

[9] Author's note: [EC91].
[10] Authors note: [CSS89].

Part II
TECHNICAL CONTEXT

3 Time-sharing

In the bad old days [FC71, p. 79]:[11]

> The large, expensive computing machines had become far removed from their users, both in time and distance. An applicant in effect had to deliver his problem or program to a receptionist and then wait hours or sometimes days for an answer that might take the machine only seconds or even less time to produce. The computer, working on one program at a time, kept queue of users waiting for their turn. If, as commonly happens, a submitted program contained a minor error that invalidated the results, the user often had to wait several hours for resubmission of his corrected program.

Time-sharing changed all that. To begin with, it provided "a means of allowing fuller use of the machine by more people and of saving time for the users" by making it effectively possible for a computer to work on many programs at the same time [*ibid.*]

And there was more. Although "At first thought time-sharing seems simply a convenience ... It has created an unexpected new order of uses for the computer" [*ibid.*] Taken together with the development of peripherals, such as terminals and on-line random access storage devices, needed to take advantage of the new capabilities inherent in time-sharing, the development of collections of application programs and utilities available on-line to system users, and the development of operating systems to manage these resources, the effect was to transform computers into a new kind of machine.

Certainly, the relationship of the users of a computer system to the system and each other was new [FC71, pp. 86–87]:

> In a sense the system [*i.e.*, M.I.T.'s Compatible Time-Sharing System (CTSS)] and its users have developed like a growing organism. Most striking is the way the users have built on one another's work and become dependent on the machine. ... in conventional computer installations [without time-sharing] one

[11] R. M. Fano and F. J. Corbató, authors of [FC71], were time-sharing pioneers [FW71, pp. 272–273].

> hardly ever makes use of a program developed by another user, because of the difficulty of exchanging programs and data ...
>
> ...
>
> All in all, the mass memories of our machines are becoming more and more like a community library.

Remarkable things were being done, and still more remarkable things could be foreseen [FC71, p. 87]:

> The facility actually goes beyond a library's usual services. It already has a rudimentary mechanism whereby one person can communicate with another through a program in real time ... it does not take a long stretch of the imagination to envision an entire business organization making and executing all its major decisions with the aid of a time-shared computing system. In such a system the mass memory at all times would contain an up-to-date description of the business.

4 The Classical Computer Security Problem

Businesses weren't the only organizations that might take advantage of the possibilities opened up by time-sharing — it didn't take a long stretch of the imagination to envision an entire military organization making and executing all its major decisions with the aid of a time-shared computing system, and the imaginations of the American military services and Department of Defense were equal to the task. But there was a problem, and it was *serious*.

In order to take full advantage of the possibilities offered by the new technology, it would be necessary to have classified information of two or more security levels on systems where some users were not cleared for at least one of the levels present.[12] In September 1966, when [FC71] was originally published in *Scientific American*, nobody had any idea how to combine information of differing security levels with users of differing clearances on a single time-sharing system and have any real assurance that users could not gain access to information for which they were not cleared. It does not take a long stretch of the imagination to see why persons and organizations duty bound to support and defend the Constitution of the United States against

[12] Classification, security levels, and clearances are discussed in subsection 9.1.

all enemies, foreign and domestic,[13] would require such assurance for Top Secret information [CSE85c, p. 29]:

> ...information, the unauthorized disclosure of which reasonably could be expected to cause exceptionally grave damage to the national security.

Secret information [*ibid.*]:

> ...information, the unauthorized disclosure of which reasonably could be expected to cause serious damage to the national security.

and even Confidential information [*ibid.*]:

> ...information, the unauthorized disclosure of which reasonably could be expected to cause damage to the national security.

Without doubt, there were foreign enemies, and there had been cases [Bam82, pp. 133–154] in which, despite even NSA's stringent vetting procedures [Bam82, pp. 118–130], their minions had obtained clearances permitting access to extremely sensitive information. This showed that the system of classifications and clearances was not foolproof, even if the prospective technology enabled by time-sharing systems was not involved. But it was an essential component of the *only* established way of preventing unauthorized disclosure of sensitive information, and to undertake the envisioned transformation of the military's methods for dealing with information without extending the scheme of classifications and clearances to that context was unthinkable. Furthermore, the difference between applying the classification/clearance system to information held in paper files and applying it to information held in a time-sharing computer system was *radical*.

For many people, the thought of having access to sensitive information conjures up something like an image of John Le Carré's character George Smiley — than whom no one is more highly cleared — engaged in a mole hunt, legitimately delving into paper files stamped with labels denoting the most rarefied security levels.[14]

[13] All members of the armed forces of the United States have such a duty, undertaken at the time of their induction [Arm59], [Arm90, p. 123].

[14] An American analog of this example can be produced by imagining a similar situation involving Charles McCarry's character Paul Christopher, Smiley's approximate peer in the fictional security services of the United States. But Smiley is a better case for the example given in the text — the United Kingdom leads the world in spy fiction, both as regards quality and as regards renown.

Besides adequate lighting, the only instrumentalities Smiley requires as he digs through files are his glasses. Smiley can be certain that simply looking through his glasses at the papers he is reading will not transmit the information written on the documents to somebody in the employ of his Soviet opposite number, Karla. But, most emphatically, he could *not* be certain that "simply reading a file" would not have such an effect, were he working with "files" held in a time-sharing computer system. In the latter case, Smiley would be able to "read" such a "file" only by invoking a program that caused the information contained in the "file" to be displayed by an output device, and the program might do this and *more* — it might also, either by design or by accident, transmit the information to a "file" that could be "read" by one of Karla's agents.[15]

A program written with the malicious intention of causing the kind of behavior described in the preceding paragraph is called a Trojan horse. There are other possibilities for writing software that can have malign effects. For example, software that behaves benignly in ordinary circumstances may contain a trap door — a hidden mechanism, activated in some seemingly innocent way, that causes the software's behavior to become malignant.[16,17]

Smiley, apparently living in a low tech world, doesn't have to worry about Trojan horses and trap doors, but, still, there is more to be learned by considering what he *doesn't* do. Despite his stratospheric clearance, Smiley *doesn't* go digging through files down in Registry unless his need to know what's in them has been established.

This is a case of art imitating life that applies as well on the west side of the Atlantic as on the east. According to long-standing Department of Defense policy for handling sensitive information, being cleared for information classified at a given security level is a necessary condition for having authorized access to that information, but, by itself, the clearance is *not* a sufficient condition for having authorized access to the information. Accord-

[15] The example and my understanding of the point it makes grew out of thinking about material from [McC93, A270–A319 and A456–A486].

[16] As [NCS88b, p. 48] makes clear, mechanisms embedded in hardware may also be trap doors.

[17] Trap doors were genuine worry for the Air Force during the early to middle 1960's [Sch93b, A060–A091], because of the fear that someone might use a trap door to enable an unauthorized ballistic missile launch. One of the effects of this was that "You didn't use commercial software on those machines in any way, shape, or form." But that didn't solve the problem — someone on the programming staff, despite being a member of the Air Force, might still "for whatever reason" use malicious software to cause an unauthorized launch.

ing to [DoD82a] as quoted in [NCS85b, p. 76], for example:

> ... no person may have access to classified information unless ... access is necessary for the performance of official duties.

To have authorized access to sensitive information requires *both* a high enough clearance *and* a need to know. Thus, in order to extend standard military methods for handling sensitive information to the context of time-sharing computer systems, it was necessary to devise policies, procedures, and mechanisms that would give appropriate expression to both the principles underlying the classification/clearance system and the need to know principle.[18]

Taken together, the Ware and Anderson reports [War70, And72] gave a clear statement of the basic problems discussed in this section[19] and located the crucial technical area in which progress had to be made in order to solve them — operating systems, the collections of software that control computer system resources and provide users, utilities, and application programs access to those resources, including access to files and input/output devices, which is crucial to the problems under discussion.

If, following [Lan81, p. 247], we say that a computer system's mode of operation is multilevel if some information in the system has a security level higher than the clearances of some of the system's users, then the classical computer security problem can be characterized as the problem of building time-sharing operating systems that, when operating in multilevel mode, appropriately implement the principles of the security level/clearance system and the need to know principle. More succinctly put, the classical computer security problem is the problem of building multilevel secure time-sharing operating systems.

---

[18] There is a good deal more to military security practices than the system of classifications and clearances and the need to know principle, though these are the features that are of primary concern in computer security. For a general discussion of military security and other aspects of its relation to computer security, see [Lan81, pp. 248–253].

[19] On the subject of trap doors, see, for example, [War70, p. 8]. The relevant usage of the phrase "Trojan horse" was introduced by D. J. Edwards during the writing of [And72]. (See the note found in [And72, Vol. II, Appendix II, p. 62]. I am indebted to Carl Landwehr for this reference, which corrects [Lan81, note 2, p. 252].)

It should be noted that, although the Ware report received "its impetus from the concern that has been generated by the increasing number of time-sharing systems" [War70, p. xii], both the Ware and the Anderson reports address the more general topic of security safeguards for resource-sharing computer systems, as described in [War70, Introduction and pp. 1–3].

5 The Software Crisis

Characterization of the problem that needed to be solved in order to allow the hoped for transformation of the military's methods of handling information was a notable achievement, but the characterization provided *very* little comfort.

According to [NCS85b, p. 1], the work that resulted in the Ware report began in October 1967. Assessing the state of computing technology was in fashion that fall. In late 1967, the NATO Science Committee began an effort aimed at assessing the field as a whole by forming a Study Group on Computer Science.[20] The first concrete result of the Study Group's activities was a conference on software engineering held in Garmisch, Germany in October of 1968. It became abundantly clear at the conference that, even neglecting the problem of time-sharing and security that was the province of the Ware panel, the state of the art in production of large software systems was, in general, not good and was very bad indeed in the case of operating systems.

Few attending the Garmisch converence disagreed with E. E. David of Bell Labs when he said [NR69, p. 67]:

> ... production of large software has become a scare item for management. By reputation it is often an unprofitable morass, costly and unending. This reputation is perhaps deserved. No less a person than T. J. Watson said that OS/360 cost IBM over 50 million dollars a year during its preparation, and at least 5000 man-years' investment. TSS/360 is said to be in the 1000 man-year category ... The commitment to many software projects has been withdrawn. This is indeed a frightening picture.

David's jeremiad continued [NR69, pp. 68–69]:

> ... the uninitiated sometimes assume that the word 'scale' refers entirely to the size of code ... This dimension is indeed a contributory factor to the magnitude of the problems, but there are others. One of increasing importance is the number of different, non-identical situations which the software must fit. Such

[20] The date for the formation of the Study Group is taken from [Sha90, p. 100], and the remainder of this section follows the account of [Sha90, pp. 100–102] very closely. In particular, it should be noted that quotations from [NR69] are as given in [Sha90, pp. 100–102].

> demands complicate the tasks of software design and implementation, since an individually programmed system for each case is impractical.

and continued further [NR69, p. 69]:

> ... there is no theory which enables us to calculate limits on the size, performance, or complexity of software. There is, in many instances, no way even to specify in a logically tight way what the software product is supposed to do or how it is supposed to do it.

There was some disagreement over whether the software situation really merited the name "crisis" [Sha90, pp. 100-101], but it was agreed that the there were serious problems with software and that, as far as underlying causes were concerned [NR69, p. 122]:

> ... a basic one lies in the unfortunate telescoping of research, development and production of an operational version within a single project effort. This practice leads to slipped schedules, extensive rewriting, much lost effort, large numbers of bugs, and an inflexible and unwieldy product.

J. W. Smith observed that there was a tendency among designers to "use fuzzy terms like 'elegant' or 'powerful' or 'flexible'" [NR69, p. 38], and continued by saying:

> Designers do not describe how the design works, or the way it may be used, or the way it would operate. What is lacking is discipline, which is caused by people falling back on fuzzy concepts ... Also designers don't seem to realize what mental processes they go through when they design. Later they can neither explain, nor justify, or even rationalize, the processes they used to build a particular system.

Even assuming it possible to devise a reasonable scheme for dealing with the problems peculiar to computer security the Ware panel was examining, it would be impossible to build multilevel secure time-sharing operating systems and be assured of their security if significant progress was not made in the direction of dealing with the general problems of large software systems responsible for the urgent tone of the Garmisch conference's proceedings.

Two salient points emerged from what had been said there: (1) precise concepts and exact methods for use in specifying software systems and describing and predicting their behavior were needed, and (2) so were better techniques for desigining software systems and organizing the process of designing, building, and maintaining them.

6 Program Verification

If precise concepts and exact methods were wanted, it seemed reasonable to think they might be found in logic. It had seemed so to Floyd [Flo67], and it certainly seemed so to C. A. R. Hoare, who built on Floyd's work [Hoa69]. The introduction to Hoare's paper setting out the calculus he devised with the aim of exploring the "logical foundations of computer programming" [Hoa69, p. 576] began as follows:

> Computer programming is an exact science in that all the properties of a program and all the consequences of executing it in any given environment can, in principle, be found out from the text of the program itself by means of purely deductive reasoning.

This was a visionary statement, and the vision it expressed was shared by others.

Donald Good, for example, in 1967 a graduate student in computer science at the University of Wisconsin, Madison, had been working in a branch of numerical analysis called interval analysis. And then one day [Goo91a, tpt, p. 1]:

> ... it dawned on me that "Gee! All this mathematics isn't worth anything unless the program works right", and that's what brought me into [program verification].

After finding out that there was nothing he could "just go out and buy and use [to verify programs] and be on my way [in numerical analysis]" [*ibid.*], being introduced to Floyd's ideas via a draft version of [Flo67] given to him by one of the faculty, Ralph London, and publishing a paper with London as a result of joint work based on what Floyd had done [*ibid.*, pp. 1–2], "I switched my graduate work from numerical analysis to working in [program verification], right at the time when I did my dissertaion" [*ibid.*].

Thus, Good had enough confidence in the vision articulated by Hoare to switch from numerical analysis, an established field, to program verification, a brand new one, when he began work on his dissertation. A switch of this kind isn't made unless the person making it thinks the new field is going somewhere, going somewhere interesting and important.

More will be said below about the work Good's change of direction led him into and how it became enmeshed in DoD's effort to solve the classical computer security problem. As for Hoare, careful reading of [Hoa69] is enough to show asserting that computer programming is an exact science was easier than backing up the claim was going to be.

In the paragraph immediately following the one that began with the clarion sentence quoted above, it became clear that subtlety would be needed in achieving the wanted exactitude [Hoa69, p. 576]:

> Unfortunately, in several respects computer arithmetic is not the same as the arithmetic familiar to mathematicians, and it is necessary to exercise some care in selecting an appropriate set of axioms [about the elementary arithmetic operations a program may invoke].

The main difference between computer arithmetic and ordinary arithmetic is, as Hoare observed [*ibid.*], that the familiar number systems — *e.g.*, the natural numbers and integers[21] — are infinite, but computers and, *a fortiori*, the sets of numbers represented in them are finite. Hoare [*ibid.*] proposed some candidate axioms for characterizing various ways the finite sets of natural numbers represented in computers might behave, and there certainly seemed to be no reason this could not be done for integers, rational numbers,[22] and so on. But the situation was worrisome. Much of the power of mathematics would be lost, because many familiar theorems would not apply in the setting provided by such axioms, and, even ignoring that, the resulting axioms might be very tricky to deal with.[23]

Moving from problems of computer arithmetic to the main business of the paper, explaining the calculus intended to help make programming an

[21] The natural numbers are the non-negative whole numbers 0, 1, 2, 3, ... The integers are the natural numbers together with the negative whole numbers ..., -3, -2, -1.

[22] Rational numbers are those represented by fractions that have integers as numerators and integers different from 0 as denominators.

[23] They were tricky enough to trip up Hoare himself — candidate axiom $A11_S$ [Hoa69, p. 576] is the negation of a theorem of classical first order predicate logic with identity [Men79], which seems to be the underlying logical system Hoare had in mind.

exact science, Hoare began with the assignment operator, which is used to change values of program variables.[24] He prepared the reader for the statement of the calculus's assignment axiom by saying [Hoa69, p. 577]:

> Assignment is undoubtedly the most characteristic feature of programming a digital computer,[25] and one that most clearly distinguishes it from other branches of mathematics.[26] It is surprising therefore that the axiom governing our reasoning about assignment is quite as simple as any to be found in elementary logic.

This was good news, but it wasn't *all* the news. The formal preamble of the axiom's statement, which expressed the conditions under which the axiom could be used, ran as follows [*ibid.*]:

> Consider the assignment statement:[27]
>
> $$x := f$$
>
> where
>
> x is an identifier for a simple variable;
>
> f is an expression of *a programming language without side effects* [emphasis added], but possibly containing x.

The statement of the axiom followed this in Hoare's text, but, as far as the issues under discussion here are concerned, it need not be gone into. The trouble with Hoare's treatment of assignment, considered as part of an attempt to show that programming is an exact science, was that, contrary to the emphasized phrase in the last clause of the axiom's preamble, there

[24] The assignment operator, :=, is the main operator in commands of the form $x := f$, where x is the name of a program variable and f is an expression with values of a type appropriate to the variable named by x. On encountering such a command, the machine attempts to compute the value of f and then make it the value of the variable named by x.

[25] Author's note: A reasonable assertion in 1969, but less so now. See note 28.

[26] Author's note: Note the implicit assertion that programming *is* a branch of mathematics.

[27] Author's note: See note 24 for an explanation of the notation $x := f$ that follows in Hoare's text.

Hoare calls $x := f$ a statement, which is a standard, but unfortunate, usage in speaking about programming languages. Since the notation has imperatival force (it tells the machine to *do* something), it is better to call $x := f$ a command, as in note 24.

weren't any programming languages without side effects.[28] Moreover, although Hoare did not note the problem, in order for his assignment axiom to hold, aliasing[29] had to be ruled out as well, and there were no languages in which this was done.

There are other doubtful points in the text of [Hoa69], but to consider them in detail would be otiose. The computer arithmetic and assignment axiom examples are sufficient to point up a tension in Hoare's thinking that is symptomatic of a general tension inherent in the program verification enterprise. On the one hand, realism demands recognition of the fact that computers are *not* abstract entities, obeying the powerful and familiar laws of mathematics. On the other, the aim of demonstrating program correctness demands treating computers *as if* they were subject to mathematical laws, laws perhaps novel but, nevertheless, elegant and powerful.

Tensions of this kind are not unique to program verification, and they are not necessarily fatal to fields of endeavor that are subject to them. Aeronautical engineers, for example, have had impressive success in building aircraft, despite the fact that viscous and turbulent flows resist mathematical treatment and despite use of mathematical models that involve considering wings of infinite length and other unlikely objects [Vin90, chapter 2].[30] But the tension present in program verification does argue strongly against accepting the notion that computer programming is an exact science and supposing

[28] A side effect is a change in the value of a program variable that occurs as a result of computing the value of some expression used in the program. A case where programmers sometimes introduce side effects on purpose provides a good example. In dealing with stacks, it is fairly common to use a function *pop* written so that computing $pop(s)$ both (1) returns the element that is at top of the stack s prior to the computation and (2) removes the element in question from s, thereby causing a side effect.

Purely functional languages, which do not involve side effects, have been developed since Hoare wrote the paper under discussion [Tur85, Tur86, Tur87a, HF92, H$^+$92], but they are so different from the languages Hoare had in mind when he devised the calculus of [Hoa69] that the assignment operator is not part of their apparatus. The question of using functional languages to write operating systems has been considered [Hen82, Sto86, Tur87b, Tur90], but, so far, only as a research problem.

[29] Aliasing occurs in a program context where two variable names refer to the same program variable. For examples that show why aliasing makes straightforward use of Hoare's assignment axiom impossible, see [Ten91].

[30] [Vin90, p. 167] characterizes the problems of turbulent flow as "ubiquitous but still scientifically intractable," and observes in the accompanying note 103, p. 302, that "an especially important example [of turbulent flow], evident in ancient times but still unsolved in a basic scientific sense, is that of flow in a pipe in most conditions of practical operation ..." Evidently, in some cases engineers must get along for a very long time without adequate models of phenomena that are of fundamental concern to them.

that, by itself, program verification is going to solve "the software problem".

It seems that, on sitting down in a cool hour in 1969 or 1970, it should have been possible to see this. But sitting down in a cool hour wasn't what people were into during the late 60's and early 70's, even people primarily concerned with operating systems, software, and computer security [Sch93b, A554–A564]:

> ... in the later 60's, a system which I was the system engineer for ... dealt with very sensitive information, having selected things retrieved from it and passsed into less sensitive systems. And this ... was in a wartime ... environment, where people really ... were dying as a result of the consequences of this. And it's a place where I probably first most directly saw what people would regard as a significant part of the computer security problem.

Operating systems, software, and computer security presented serious problems, as did much else, and people wanted to *do* something about them.

7 Structured Programming and Design Verification

David Parnas [Par72] wanted to do something about the problem of improving techniques for designing software systems and organizing the process of designing and building them. In particular, he wanted to do something about the problem of saying what software modules, connected program segments that were used as subassemblies in building up a whole software system, were supposed to do.

Although Hoare had said nothing about the matter in [Hoa69], it could be argued that his enterprise was, with regard to actual application of the calculus for proving theorems about program code, dependent on the success of enterprises like Parnas's. In order to apply the calculus, one had to know *which* theorems to prove, and that would be determined by a statement of what the code was supposed to do — by a specification.

It was becoming clear that process of designing and building software systems required looking at systems from the different vantage points provided by a hierarchy of system descriptions, with more abstract descriptions

occupying higher levels than those that were less abstract.[31] In this hierarchy, program code occupied the level just below the level Parnas was concerned with, and the kinds of theorems Hoare wanted to prove would connect the two.

The point of the theorems Hoare had in view was to provide assurance that the code did what the specification said it should, and the same was true of traditional software tests. So these assurance techniques were intended to provide a connection running from the code level to the more abstract specification level.

Of course, before assurance techniques that established a connection running from the code level to the specification level could be brought into play, it was necessary to proceed from the higher level of abstraction to the lower by writing the code. Indeed, it was necessary to write the code if a system was to be built at all, and the primary function of specifications was to guide the code writing process.

It should be clear from the foregoing that the problem Parnas was concerned with was of great importance both for the general enterprise of building software systems and for the specific enterprise of building multilevel secure time-sharing operating systems. Moreover, the Ware panel had asserted "Probably the most serious risk in system software is incomplete design" [War70, p. 8], and this was borne out by the "tiger team" penetration studies undertaken by DoD in the early 70's to assesss the security of available computer systems [Jel85, pp. II-70–II-71], [Sch93b]. The tiger team studies furnished convincing evidence that multilevel security *couldn't* be achieved through a "penetrate and patch" approach.

Correct design was essential for security, and correct specification was essential for correct design. Therefore, solving the problem Parnas was concerned with was of crucial importance in attempting to solve the classical computer security problem.

Parnas described his goals briefly as follows [Par72, abstract, p. 330]:

> This paper presents an approach to writing specifications for parts of software systems. The main goal is to provide specifications sufficiently precise and complete that other pieces of software can be written to interact with the piece specified without additional information. The secondary goal is to include in

[31] Dijkstra had advocated this as a way of dealing with the problems that animated the Garmisch conference [Dij69], and the general features of his view were gaining widespread acceptance.

the specification no more information than necessary to meet the first goal.

Giving a more detailed statement of his goals, Parnas broke them down into a four item list. The beginning of item three is of salient interest for the purposes of the present document [Par72, p. 330]:

> 3. The specification must be sufficiently formal that it can conceivably be machine tested for consistency, completeness (in the sense of defining the outcome of all possible uses [of the module specified]) and other desirable properties of a specification.

Later in the paper, under the heading "Using the Specifications," Parnas explained what he meant by saying specifications should be machine testable. Ultimately, the ideas involved in the explanation became an essential part of the Orange Book proof requirements. The passage is worth quoting at length [Par72, pp. 334–335]:

> Our aim has been to produce specifications which are in a real sense just as testable as programs. We will gain the most in our system building abilities if we have a technique for usage of the specifications which involves testing the specifications *long before* the programs specified are produced. The statements being made at this level are precise enough that we should not have to wait for a lower level representation in order to find the errors.
>
> Such specifications are at least as demanding of precision as are programs; they may well be as complex as some programs. Thus they are as likely to be in error. Because specifications cannot be "run," we may be tempted to postpone their testing until we have programs and can run them. For many reasons such an approach is wrong.
>
> We are able to test such specifications because they provide us with a set of axioms for a formal deductive scheme. As a result, we may be able to prove certain "theorems" about our specifications. ...
>
> By asking the proper set of ... questions, the "correctness" of a set of specifications may be verified. The choice of the questions, therefore the meaning of "correctness," is dependent on the nature of the object being specified.

Parnas did not insist on the use of machines in verifying specifications, but he did regard it as essential that "system builders develop the habit of verifying the specifications whether by machine or by hand before building and debugging the programs" [Par72, p. 335].

Parnas's ideas offered hope for a solution to the problem of ensuring design correctness. But there were also reasons for caution.[32]

First, as in the case of Hoare's program verification scheme, application of Parnas's idea of verifying specifications — an essential part of what came to be called design verification — required a way of deciding which theorems to prove. That would a require a level of system description more abstract than the specification level, and if, in turn, things worked out so that another level was required beyond that, and so forth, specification verification, and, more generally, design verification, would be revealed as chimeras. An indefinite piling up of levels of abstraction had to be avoided.

Second, the tension between realism and the need for elegant and powerful laws for use in proofs did not vanish in moving from the setting of program verification to the more abstract situation involved in design verification. If, on the one hand, the machine's finitude was taken into account, the resulting inability to apply familiar mathematical laws might make it impossible to prove the desired theorems. And if, on the other, such considerations were ignored, the theorems might provide false assurance of design correctness, which would be very bad thing indeed — living in a fool's paradise wouldn't do much to promote the general goal of building of better computer systems and *certainly* wouldn't lead to building computer systems that were secure.

It remained to be seen whether these difficulties could be overcome, but, as things turned out, impressive resources would be put into trying the experiment.

8 Security Kernels and the Reference Monitor Concept

At about the time Parnas was writing [Par72], the United States Air Force, too, wanted to do something about a computing problem — getting a time-

[32] Parnas, exhibiting the decent regard for empiricism that is requisite in computer science, pointed out some of them [Par72, pp. 335–336]. The text of this section points out others that are specifically relevant to present purposes.

sharing operating system for use by the Air Force Data Services Center at the Pentagon that would support secure processing of Secret and Top Secret information, while serving a user community that included users cleared only at the Secret level. More generally, the Air Force wanted solutions to the problems of multilevel security set out in the Ware report or, at least, a well defined research program aimed at producing solutions to those problems. So the Air Force did what military organizations typically do with problems of this kind. It ordered an officer who seemed to have reasonable qualifications in the area to get busy solving them.

Major Roger R. Schell, armed with a brand-new Ph.D. in computer science from MIT and five years' experience with with Air Force computer systems prior to matriculating at MIT in June 1968 [Scha], got the job, though he didn't want it and tried to avoid it [Sch93b, A175]: "I was a very reluctant draftee."

There were good reasons for Schell's reluctance. First, he had considered computer security as a dissertation topic during his graduate work at MIT and had been told that the problems involved were unsolvable and that he would never finish his degree if he took them on [Sch93b, A136ff]. Second, the assignment took him away from management information systems, the area he wanted to work in. And, third, research wasn't his area. Schell's background was in engineering and management, where the point was to get systems delivered on time and within budget [Sch93b, A110]. He had strong misgivings about being involved with projects that had a large research component [Sch93b, A195]: "The R&D people were people you stayed away from, because if you let them in your project, they would sink you and your project."

Nevertheless, in August 1971, on assumption of his duties as project officer for computer security within the Directorate of Information Systems Technology, Electronic Systems Division, Air Force Systems Command [Scha], Schell found himself running projects of exactly this kind.

Schell had two crucial problems to deal with. To begin with, he had to define a viable technical approach to the problem of building multilevel secure time-sharing operating systems. Having done that, an impossible task according to his erstwhile mentors at MIT, he had to find a way to make the ideas involved in the technical approach practically effective by selling them to people who counted. Schell solved the core technical problem by creating the idea of what came to be called a security kernel, in Schell's words a "subset of the hardware and software that was sufficient to provide security even if the remainder of the system" had been produced "by an

adversary."[33] The Anderson panel, formed as a result of the work being done at ESD, provided Schell with a way of solving the remaining problem of selling the security kernel approach to people who could get things done.

Schell cultivated the Anderson panel assiduously,[34] introducing and explaining his ideas to its members and, in particular, spending hours with James Anderson, the panel's chair, trying to persuade him to include the ideas in the panel's report [Sch93b, A214ff].

As things turned out, Schell made his sale, insofar as it involved selling his ideas to the Anderson panel. The panel endorsed Schell's ideas in the following language [And72, Vol. I, p. 9]:

> The basic concept upon which multilevel secure computing systems can be based is that of *controlled sharing*. Explicit control must be established over each user's (program[']s) access to any system resource which is shared with any other user or (system) program. Essential to this concept is the requirement that each subject of the system (viz. system entities such as a user or a program which can access system resources) and each

[33] Quotation as given in [Jel85, p. II-73], citing an interview with Schell conducted October 28, 1982. The passage goes on to describe the origin of the phrase "security kernel":

> In search of a name for this bold new concept, Schell went to Dr. John B. Goodenough, then an applied mathematician at ESD. By Schell's own account, he asked Goodenough, "What would you call this?" Goodenough replied, "Well, that seems to be a lot like the notion of a kernel in mathematics and since it relates to security, why don't you call it a security kernel?" Since neither Schell nor fellow researchers at the MITRE Corporation had any other ideas and since Schell was due to submit an abstract for an upcoming conference, he accepted the Goodenough proposal.

The abstract was titled "Abstract of a Virtual Memory Security Kernel" and was submitted by Schell as a participant's position paper to the chair of the IEEE Workshop on Privacy and Protection in Operating Systems, June 13–14, 1972 [Sch92]. A sketch of Schell's ideas was published soon after the workshop [Sch72].

[34] Acting on the advice of a colonel he worked for, who had much experience in research and disagreed with Emerson's supposed dictum "If a man can write a better book, preach a better sermon, or make a better mousetrap than his neighbor, though he builds his house in the woods the world will make a beaten path to his door" [Bar80, p. 496, note 1]. Schell, not yet having built a better computer security mousetrap, was preaching, with the idea of a security kernel as his theme. The colonel opined that (1) in the real world, you can't give ideas away, but (2) Schell could feed ideas to the panel, and people would listen to them because they were too busy to think of ideas on their own [Sch93b, A202].

The colonel's advice may strike some as being a bit cynical, but the course of action he recommended certainly worked for Schell.

> object (viz. system entities such as data, programs, peripheral devices, main memory and subjects which can be accessed by other subjects) must be identified and interrelated according to their authorized accessibility.
>
> One of the most promising developments of this idea is the concept of a *reference monitor*[2] which enforces the authorized access relationships between subjects and objects of a system. ... An implementation of the reference monitor concept is a *reference validation mechanism*[3] that validates each reference to data or programs by any user (program) against a list of authorized types of reference for that user.
>
> ...

---

[2][Note in original:] Reference monitor concept — the notion that all references by any program to any program, data or device are validated against a list of authorized types of reference based on user and/or program function.

[3][Note in original:] Reference validation mechanism — the combination of hardware and software which implements the reference monitor concept.

Although the panel did not use the phrase "security kernel" in the preceding passage, the phrase was used in a later passage commenting on the labels of a figure [And72, Vol. I, Figure 3, p. 15] depicting the advanced development plan recommended as an approach "for achieving the objective of a secure, open-use, multilevel resource sharing system" [And72, Vol. I, p. 14]:

> The access control, reference validation mechanism and security related functions are referred to as the 'Security Kernel'.

A footnote appended to the sentence just quoted seemed to define "security kernel" as referring, in particular, to "the software portion of the reference monitor and access control mechanisms" [And72, Vol. I, p. 14, note 5]. If taken seriously as a definition of the phrase "security kernel", the footnote would have given the phrase a more restricted sense than Schell, who included both hardware and software elements in his original notion of a security kernel, had intended. On the other hand, taken literally, and ignoring the note, the sentence to which the note was appended gave "security kernel" a sense that was a close match for what Schell originally had

in mind and, accordingly, made "security kernel" either a near synonym for the more exalted sounding phrase "reference validation mechanism" or a more inclusive term, depending on whether "security related functions" other than those involved in access control and reference validation were considered to be part of a system's security kernel.

As things turned out, the footnote was ignored. In later discussions of computer security, the usage of "security kernel" conformed to Schell's original intention, including both hardware and software elements of computing systems in its reference, and "security kernel" served, for the most part, as a way of expressing what the Anderson panel had included under the phrase "reference validation mechanism".[35]

According to the usage established subsequent to the Anderson report's publication, the following passage [And72, pp. 9–10] can be taken as further specifying the requirements a security kernel must satisfy:

> Accompanying the concept of Reference Monitor are other essential design requirements. They are:
>
> **a.** The reference validation mechanism must be tamper proof.
>
> **b.** The reference validation mechanism must *always* be invoked.
>
> **c.** The reference validation mechanism must be small enough to be subject to analysis and tests, the completeness of which can be assured.

The ultimate effect of Schell's selling his ideas to the Anderson panel was to sell them to the Department of Defense. The section of the Orange Book that explains the rationale for the requirements defining the various security classes [NCS85b, pp. 65–69] begins with a passage consisting mainly of an abridged version of the passages quoted here from the Anderson report's account of the reference monitor concept and reference validation mechanisms.

The security kernel/reference monitor approach to building multilevel secure time-sharing operating systems, as described in the Anderson report,

[35] See, for example, the Orange Book's definition of "security kernel" [NCS85b, p. 115]. [Lan83, p. 87], perhaps referring to the portion of the Anderson report's text discussed in the two paragraphs preceding the one to which this note is appended, observes that "the term has also been used to denote all security-relevant system software." However, this usage, which makes "security kernel" a term for the software portion of what [Nib79b] and the Orange Book [NCS85b, p. 116] call a "trusted computing base", is much less common than the usage that conforms to Schell's original idea by including hardware elements of a system in the reference of "security kernel", but limits application of the phrase to system elements related to access control and reference validation.

opened up the prospect of reducing the negative problem of showing that a potentially unlimited collection of bad things couldn't happen — Who knew how many different ways of attacking a time-sharing operating system could be devised? — to the positive problem of building a piece of software that did a well defined job. The appeal of turning a seemingly impossible negative problem into possibly solvable positive problem was obvious and had, in fact, been obvious to Schell from the outset. Reducing the multilevel security problem to making a software component with specified functionality was what Schell had in mind when he thought up the security kernel/reference monitor approach — given his orientation, the only way he could think of to attack the problem "was to go off and build something that would help" [Sch93b, A233].

A second attraction of the security kernel/reference monitor approach was that, at bottom, it relied on the familiar idea that the way to maintain confidentiality of information was to control access to containers of information. (Think of the clerks down in Registry zealously guarding paper files against access by those who, unlike Smiley, are not entitled to it. The clerks guard the files containing information, not the information itself.) The assumption that confidentiality of information should be seen to by controlling access to containers of information had been made tacitly in the Ware report — [War70] gave no hint that there might be another way of handling the confidentiality problem — and the Anderson report did nothing to change this. There was comfort in relying on such a familiar idea and, barring some obvious problem, it was only reasonable for the Ware and Anderson reports to do so. But working out in detail how to maintain confidentiality of information via access control in the context of a time-sharing operating turned out not to be simple, and the ideas that came out of the effort were a long way from being as familiar as the ideas that went into it.

9 Modeling Security

The security kernel/reference monitor scheme was predicated on the assumption that controlling access to containers of information was a reasonable approach to solving the multilevel security problem, but did not, in itself, answer the question of just which kinds of access were to be permitted and which were to be forbidden. This gap in the doctrine evolving at ESD was filled by producing an abstract mathematical model of access control policy

that could be specialized in the case of a particular operating system to fit the institutional security policy defined by the body of laws, regulations, and administrative and procedural documents applicable to the system in question.

Subsection 9.2 discusses the the Bell/La Padula model, which came to be the third person of the security kernel/reference monitor/security model trinity that inspired projects based on the finished form of the ESD doctrine. Subsection 9.3 discusses rivals of Bell/La Padula. Subsection 9.1 prepares the way for the discussions of subsections 9.2 and 9.3 by providing background information on classification, security levels, and clearances that is relied on in subsections 9.2 and 9.3 and the remainder of this document.

Before proceeding with the discussion of subsections 9.1–9.3, it should be noted that, besides its specific role in the ESD security doctrine, Bell/La Padula served a more general conceptual purpose — it exemplified a way of terminating the potentially vicious piling up of levels of abstract system description that threatened to make design verification a nullity, as noted at the end of section 7.

If Bell/La Padula could serve this purpose, so might other security models, and the purpose wanted serving and still does. As the exposition of the report proceeds, it will become clear that there was, and is, considerable disagreement as to whether design verification can provide sufficient assurance of high levels of computer security, but no doubts about its necessity for this purpose will come to light. Accordingly, neither will doubts about the necessity of having abstract security models. As far as security models are concerned, the unchallenged consensus is that the question is not whether to have one, but which one to have.

9.1 Classification, Security Levels, and Clearances

In the general sense of the term, classification is the authorized assignment of a security level to information. The more restricted legal usage of the word refers only to assigning one of the legally defined classification levels Confidential, Secret, and Top Secret discussed in section 4. Discussions of computer security use "classification" in its general sense, rather than in the restricted legal sense explained in the preceding sentence.

[NCS88b, p. 41] defines a security level as:

> The combination of a hierarchical classification and a set of non-hierarchical categories that represents the sensitivity of information.

A category is defined as [NCS88b, p. 8]:

> A restrictive label that has been applied to classified or unclassified data as a means of increasing the protection of the data and further restricting access to the data.

NOFORN (Not Releasable to Foreign Nationals) and PROPIN (Caution – Proprietary Information Involved) are examples of categories [CSE85c, p. 9].

The structure of security levels involved in handling sensitive information (see [CSE85c, Appendix B]) is much more complicated than the hierarchy of legally defined classification levels. The practice of augmenting the legal classification hierarchy by adding the security level Unclassified at the bottom (so that, from the bottom up, the hierarchy runs Unclassified, Confidential, Secret, Top Secret) is ubiquitous. There does not seem to be a well established way of referring to the hierarchical component of a security level when the legal classification hierarchy is augmented by adding Unclassified and, perhaps, other hierarchical levels. The phrase "sensitivity level", used for this purpose in [Lan81, p. 248], is a reasonable choice and will be so used in what follows.

Using this terminology, the definition of "security level" quoted above can be glossed as saying that a security level is the combination of a sensitivity level and a set of nonhierarchical categories that represents the sensitivity of information. The terminology and the resulting gloss of the definition drawn from [NCS88b, p. 41] have the advantage of simultaneously fitting the usage customary in discussions of computer security and cutting down the confusing overloading of the term "classification".

In addition to a hierarchy of sensitivity levels that augments the legally defined hierarchy of classification levels, real cases almost always involve categories. Security levels are ordered by saying that level ℓ_1 dominates ℓ_2 if, and only if, the sensitivity level associated with ℓ_1 is at least as far up the hierarchy of sensitivity levels as the sensitivity level associated with ℓ_2 and every category in the set of categories associated with ℓ_2 is also in the set of categories associated with ℓ_1.

In the sense of the term applicable to discussions of computer security, clearance is the authorized assignment of a security level to an individual. In order for a someone cleared at level ℓ_1 to have authorized access to information classified at level ℓ_2, it is necessary that ℓ_1 dominate ℓ_2.

The legally and institutionally defined sense of the word "clearance" is related to the usage just defined, but somewhat different [CSE85c, pp. 27–28].

9.2 Bell/La Padula

The discussion of this subsection presents neither a detailed analysis of the Bell/La Padula model's development nor a technical analysis of any particular version of the model. The aim, instead, is to give a clear enough description of the Bell/La Padula model's characteristics to enable the reader to follow the discussion of the following sections of the report, particularly the subsequent discussion of the Orange Book proof requirements.[36]

Subsubsection 9.2.1 describes the Bell/La Padula model's basic features, including particularly the simple security property and the *-property,[37] and subsubsection 9.2.2 explains the role of the *-property in combating the threat of Trojan horses. Subsubsection 9.2.3 explains why the *-property, despite its role in dealing with Trojan horses, cannot be imposed with full generality, and subsubsection 9.2.4 discusses kinds of computer security threats that the Bell/La Padula model cannot describe and must, therefore, be dealt with on some other basis.

9.2.1 Basic Features

The Bell/La Padula model can be decomposed into representational elements called states, operational elements called rules, and prescriptive elements called axioms or requirements. In the remainder of this document, "requirement" is used to refer to model elements of the third kind, except in cases where the usage of the Orange Book and related documents that refer to model elements of the third kind as "axioms" is discussed explicitly.

A state represents the entire collection of information present in a computer system at a given stage of computation, and the process of computation is viewed as a sequence of transitions from one state to another according to the model's rules. The requirements define what it is for a state to be secure. When the Bell/La Padula model is applied to a particular computer system, the system is regarded secure with respect to the aspects of its behavior represented by the instance of the model involved in that application if, for the states, rules, and requirements of the model instance, every state in every computation starting with a secure state and proceeding according to the rules is secure.

[36] The Orange Book criteria for security classes where a formal security policy model is required do *not* mandate use of the Bell/La Padula model, but Bell/La Padula and the methods evolved for using it in system design and construction furnished the paradigm for what was to be expected of security modeling and design verification.

[37] "*-property" is pronounced the same as "star-property".

The Bell/La Padula model's requirements cover two different aspects of computer security: mandatory security requirements have to do with representing and enforcing the classification/clearance system, and discretionary security requirements have to do with the need to know principle.

Mandatory security requirements: The Bell/La Padula model's distinctive mandatory security requirements are commonly expressed in the slogan "Read down! Write up!" The slogan is easy to remember, but every word in it must be explained, if it is to be understood correctly. The explanation begins with a more detailed description of what states are like.

Following the lead of the first passage from [And72, Vol. I, p. 9] quoted in section 8, the Bell/La Padula model distinguishes certain components of states as objects — containers of information — and other components as subjects, which act on objects and may act on each other. Files (in the computing sense of the word) and output devices (such as printers and video displays) are typical examples of objects. Subjects are programs in execution, consisting of a computational process and the objects it relies on as it runs (its domain of execution.) Since subjects contain information and may be acted on by other subjects, subjects also count as objects.

The various different ways subjects can act on objects are called modes of access. A subject that has read access to an object is able to obtain information from the object, but having read access does not imply being able to change or add to the information the object contains. A subject that has write access to an object is able to change or add to the information the object contains, but having write access does not imply being able to obtain information from the object.

Other modes of access can be considered, but understanding what read access and write access are is enough to allow explanation of half the words in the "Read down! Write up!" encapsulation of the Bell/La Padula mandatory security requirements. It remains to find out about "down" and "up".

Besides distinguishing subjects and objects and the modes of access subjects may have to objects, the model also includes an assignment of security levels to both. Thus, the security level assigned to a subject is analogous to a clearance and the security level assigned to an object is analogous to a classification. The "Read down!" requirement, officially called the simple security property, provides that a subject may have read access to an object only if the security level of the subject dominates the security level of the object. The "Write up!" requirement, known officially as the *-property, provides that a subject may have write access to an object only if the object's security level dominates that of the subject.

Discretionary security requirements: In the Bell/La Padula model, each state includes a table representing a record of decisions made on a case by case basis by users, the system manager, etc. to grant chosen subjects various kinds of access to selected objects. This table provides the basis for the model's discretionary security requirements — a subject may have access to an object in a given mode only if the table records that the subject is permitted such access to the object.

Secure states: Combining the model's representations of the classification/clearance system and the need to know principle, Bell/La Padula defines a secure state as a state that satisfies both the mandatory security requirements and the discretionary security requirements.

Proofs of design correctness: The Bell/La Padula model provides the basis for a straightforward method of proving that a system is secure with respect to aspects of its behavior represented by an instance of the model: prove that, for the states, rules, and requirements of the model instance, all initial states are secure and each rule transforms secure states to secure states.

9.2.2 The *-property and Trojan Horses

Nothing like the *-property is involved in the world of paper files where all this began, so why is it in the model?

The answer is that the *-property rules out Trojan horses. In order for a subject to have read access to an object assigned a given security level, the level of the subject must dominate the level of the object ("Read down!") But then, due to the logical properties of the dominance relation and the *-property, the subject can only have write access to objects that have security levels that dominate the security level of the object to which the subject has read access ("Write up!") A subject cannot, therefore, transfer information from an object to which it has read access to an object with a lower security level, and, hence, cannot act as a Trojan horse.

9.2.3 Trusted Subjects

While it's a good thing to rule out Trojan horses, it turns out, unfortunately, that the *-property rules out certain kinds of things that must be done to get an operating system to work. For example, all users must, as a practical necessity, be able to send files to printers, which means that the subject that controls the initial phase of the printing process — the print spooler —

must be able to read information from files of all security levels and write it to the queue that controls the order in which things are printed — the print queue. So far, so good. The simple security property and *-property will be satisfied if the print spooler and the print queue are assigned the highest security level available on the system. But then, unless everything printed is to be labeled with the highest security level available, the subject that actually controls the printer or printers — the print driver — must be permitted to violate the *-property.[38]

It's good for systems to be secure, but they *must* work. Consequently, selected subjects, called trusted subjects, are exempted from the requirement of satisfying the *-property. This has the effect of enlarging the collection of software that must be subject to stringent assurance requirements beyond what is involved in a security kernel. The resulting collection of software and associated harware that *must*, assuredly, perform correctly if the system is to be regarded as secure is called the system's trusted computing base.[39]

In effect, allowing a new trusted subject into the trusted computing base amounts to introducing a potential trap door. It must be shown that real trap doors are not thereby introduced, and the Bell/La Padula model provides no systematic guidance about how to demonstrate this.

9.2.4 Covert Channels

There is another problem to be dealt with, besides the need for trusted subjects. The Bell/La Padula model is based on a scheme for controlling expected ways of transmitting information in a computer system. [Lam73] pointed out that there may be *unexpected* ways of transmitting information that fall outside the purview of Bell/La Padula. These came to be called covert channels.

Covert channels come in two varieties. Covert storage channels use system storage locations that are not counted as objects in the process of doing the dirty work (and, for practical reasons, there must be some locations that are not counted as objects.) Covert timing channels do without storage entirely, allowing the signaling of information from one subject to another by modulating the rates at which system processes occur.

In order to build a secure system, covert channels must be found and eliminated, or, if their entire elimination is impractical or impossible, the

[38] See [LHM84] for other examples of subjects that typically must be allowed to violate the *-property.

[39] See [Nib79b], [NCS85b, p. 116], and note 35.

rate at which information can be transmitted via the covert channels that remain — their bandwith — must be reduced to an acceptable level. As in the case of trusted subjects, Bell/La Padula provides no systematic guidance as to how to do this with assurance.

9.3 Other Security Models

For a survey of other security models put forth prior to 1981, see [Lan81]. Prominent later models include those described in [GM82], [Sut86], and [McC87, McC88].

Part III
CREATING THE CENTER AND ORANGE BOOK

10 The DoD Computer Security Initiative

Stephen T. Walker described the DoD Computer Security Initiative's origins as follows [Wal93a, A032–A053]:

> Starting about 1974 I was at DARPA, and I was in charge of the computer security research activities, among other things, that were going on at DARPA. And we produced a number of technology demonstration systems that were pretty good for the time.[40]
>
> In 1978 I was invited to come over to the Assistant Secretary of Defense for C^{3}I's office[41] and do something about the political side of the computer security problem[42] — having demonstrated on the technical side what could be done, how do we get these ideas into the Department on a broad scale basis? Because very little had been done to incorporate these ideas into ongoing systems at that point. So in '78 I moved over.
>
> ...
>
> ... And I started what was called the Computer Security Intiative, which was basically three thrusts: To get the Defense Department to act together relative to computer security R&D and usage. To get industry involved in figuring out how to build some trusted systems, so that we'd have some products to deal with, basically to get them to build systems that were better than the then existing commercial practice. And the third piece was to get there to be in the Defense Department, or in the Government somewhere, a center of excellence that could evaluate how well industry had done in producing their products.

[40] Author's note: See [Wal80, pp. 655–657].

[41] Author's note: "C^{3}I" is short for "Command, Control, Communicationss, and Intelligence".

[42] Author's note: According to the Orange Book [NCS85b, p. 1], "The DoD Computer Security Initiative was started in 1977 under the auspices of the Under Secretary of Defense for Research and Engineering to focus DoD efforts addressing computer security issues." Walker was brought to DoD to turn this idea into a program and implement it.

In the research on which this report is based, information was not sought about how the 1977 decision to launch the DoD Computer Security Initiative was made. This should be looked into.

> So we wanted to get the Defense Department organized both in its R&D and its usage, we wanted to get industry involved, and we wanted to get a center someplace that could do evaluations. And the term "evaluation center" became an integral part of that discussion.

The program Walker had in view was expressed as follows in a paper that resulted from an NBS invitational workshop held at Miami Beach, November 28–30, 1978 [L+80, p. 8-9]:

> The specific tasks that we recommend be performed are:
>
> - From available literature and people's experience, prepare a series of reports that characterize the current state of the art, including both the state of the technology and the state of current systems.
> - Formulate a detailed security policy, including especially nomenclature and marking schemes, for any and all sensitive information not covered by the relevant national security policies and guidelines.
> - Establish a formal security evaluation and accreditation process, including the publishing of an "approved products list", to guide specification and procurement of systems intended to handle sensitive information.

Only the third part of this tripartite program was carried out, resulting in the Center and the Orange Book.

The difficulty of carrying out the program's second part can be appreciated from the discussion of NSDD 145, NTISSP 2, and the Computer Security Act of 1987 given in subsection 2.2.

The purpose and character of one of the reports called for in the program's first part was made clear as follows [*ibid.*]:

> To deal with the lack of awareness of the nature of the computer security problem, and its reality ... we recommend that the results of all past efforts to penetrate and repair operating systems be assimilated into a single report. ... For this effort to serve its purpose it must ... employ great candor and identify specific techniques used to break specific systems. Without this, the report will not be sufficiently credible to perform the necessary consciousness-raising function.

The need for a compilation of horrors of this kind is a current issue [Cou91, pp. 36, 163–164], and its implications outrun consciousness-raising [Pet92]. [LBMC93] appears to be the first public DoD document aimed at producing such a compilation.

11 Locating the Center

How the Center came be located at NSA is Walker's story, and it is best to let him tell it with minimal authorial intervention [Wal93a, A053–A091, A297–A354]:

> ... There was a great deal of energy spent trying to figure out where [the evaluation center] should be.
>
> Interviewer: ... when you look at some of the policy objectives that are stated in the back end of the Orange Book and you say "And they handed the job of trying to see to all of this to an arm of NSA", that seems quite peculiar.[43]
>
> Walker: [Wry chuckle.] There were interesting times with that.
>
> There were no obvious candidates in the Defense Department. There was the Defense Communications Agency, which could have done the job but really didn't have a computer orientation at that point. There was the Defense Intelligence Agency, which had a very specific mission and didn't really have any extra resources to apply to the problem. There were the services. The Air Force was leading the technology exploration at that point.
>
> ...
>
> Roger Schell had started a lot of the stuff in the Electronic Systems Division up at Hanscom Field in Massachusetts. From there, from about the mid 70's, that whole effort was falling out

[43] Author's note: At the time the question was asked, handing the job to NSA seemed peculiar in view of passages like "A major goal of the National Computer Security Center is to encourage the Computer Industry to develop trusted computer systems and products, making them widely available in the commercial market place" [NCS85b, p. 58], taken together with the general account of the Agency's history given in [Bam82]. Subsequent to the interview, reading the description given in [Jel85, chapter 4] of the kind of procurement strategy to which NSA was accustomed reinforced the initial impression of peculiarity.

of favor within the Air Force. So it would have been hard to put the center there.

...

... So we spent a lot of time talking about alternatives. We talked about a program office at NSA that would involve people from other agencies coming together. My management at OSD[44] didn't want anything as complex as that.

For a while we went off and explored "This is really a Government-wide problem, not just a Defense Department problem, even though the Defense Department has the biggest initial stake in it."

So we actually talked to Jim Burrows, who was ... the head of the computer science side of NIST, in early 1980 about establishing a Federal computer security evaluation center which would be located at NIST, probably funded and staffed about 2/3 by Defense Department people, that would be a Government-wide thing and would be in an open environment such as NIST would provide.

And ... I actually wrote a charter for the Federal computer security evaluation center,[45] and we talked to folks in Congress about it. And various folks, including the Director of NSA didn't like that idea, came down pretty hard against it for some historical reasons and some turf reasons.

...

I had been advocating the Federal center. In early '80 my boss, Gerald Dineen, would go out to NSA, and he'd come back and say, "Steve, I don't know what's going on, but those guys really don't like what you're doing."

Interviewer: Do you have a sense for what was going on? I mean, why were they so adamant?

Walker: They wanted that turf. They didn't want it to be anywhere else.

...

[44] Author's note: Office of the Secretary of Defense.

[45] Author's note: See [Jel85, Appendix A].

... when I went to see Inman[46] in August, in August of '80, and sat in his office and told him the background of what I was trying to do, he just sat there. And, finally, at one point he rolled up in his chair, he pounded his fist on his desk, and he said, "I will never let that happen! I will go to the President to keep that from happening!" ... these moments remain vivid in my mind, even yet.

...

I was sitting in this room which was about four times bigger than this room,[47] we were at the far end with his desk, and I was sitting in the chair next to his desk. And I'm sitting there thinking "How can I get out that door?" [Wry, nervous chuckle.]

And then he turned after he said that, and said, "I understand what you're saying and I agree with you and I want to do that at NSA." The problem was they didn't want NBS or anybody else involved. They wanted to own this technology. And what I had been doing up to that point was advocating that somebody else be involved.

Well, my first reaction was "I've been snookered, because this guy now wants to grab this thing." So that's when I made the argument "You've got to do it separate from the way COMSEC is done."

And Inman believed me, and this was in August. He said, "If you will write a letter to me", you, Gerry Dineen, "I will get back to you about how we will do this, and we will do it separate from the COMSEC organization."

So I went back to the Pentagon the next day, and I wrote a three liner from Dineen to Inman: "Tell me how you would do this." And in September came back a classified document that said "This is how we would do it."

Then began all kinds of discussion between me and other folks at NSA. Inman was sticking by his word. This was going to be a different organization.

...

[46] Author's note: For information on Inman, see *The New York Times*, December 17, 1993, pp. A1, B12, and B13, [NSA77], [Ben77], [Bam82, pp. 80–85, 113–114, 306, and 353–363], and [Jel85, pp. II-53, II-62–II-65, and II-80–II-87]. See also [Inm80].

[47] Author's note: Walker's office at Trusted Information Systems, where the interview took place, is approximately 15' × 20'.

> ... See what happened was: before it was NSA trying to grab this. Now it became: within NSA the guys who wanted to grab it were being excluded from it.
>
> ...
>
> And so I actually succeeded to a very large extent, because I got a completely independent organization with its own budget. NSA had, at that point I think NSA had only three funding elements. A funding element is a recognizable place that you can get money from, and, because of the nature of their business, there were only three in the entire Agency. Whereas a comparable agency would have hundreds or thousands. And what this did was create a fourth one that was separate. Now it wasn't a huge budget, but it was [a separate thing].
>
> ...
>
> And that was a very significant point. I viewed that, you know, as a major victory in this whole exercise.

12 Establishing the Center

As the Center's first Deputy Director, Roger Schell again had a job he didn't want [Sch93b, A603–A628]: "I was again an unwilling volunteer." Schell's appointment came about as a result of a personal note from Admiral Inman to General Jones, Air Force Chief of Staff, who, in turn, wrote a note to the appropriate Air Force personnel people saying: "This Air Force colonel will be assigned to Ft. Meade." And that was that. "I'd been asked several times by Steve [Walker] and others if I wanted to volunteer, but I said 'No'. So I was drafted."

There was a lot of doubt in the Air Force about whether formation of the Center was the best thing for the interests of the individual services, and Schell had been strongly of this opinion. "So I ended up implementing and having to make successful what I had advocated should not be done."[48]

[48] [Bam82, p. 86] describes the roles of Director and Deputy Director, NSA, as follows:

> DIRNSAs ... are ... selected ... to be senior bureaucratic managers who are supposed to balance budgets, settle squabbles, crack whips, pat backs, extinguish fires ... The day-to-day running of the Puzzle Palace is left to the deputy director ...

This pattern is replicated in the Agency's suborganizations. Consequently, as Deputy

There were two basic parts to the job: to provide a technically sound point of view and to build a sound administrative structure. When Schell was assigned to the Center (1981) "there were thirtyfive disjoint people, who didn't even know who each other were, that were assigned as the Center" [Sch93b, A629]. When Schell left (1984) there were about two hundred people and a good organization.

13 Writing the Orange Book

13.1 Lineage

The direct line of documents leading to the Orange Book is as Follows: [L+80], [Nib79a], [CSE82], [CSE83b], [CSE83c], [NCS85b].

13.2 Proof Requirements

The Orange Book proof requirements build up as follows. The B2 criteria require a formal security policy model and a detailed top level specification (DTLS) of the trusted computing base (TCB) [NCS85b, pp. 31 and 32]. The B3 criteria require, in addition, a convincing argument, but not a proof, that the detailed top level specification is consistent with the formal security policy model [NCS85b, p. 40]. The A1 criteria go beyond this by requiring a formal top level specification (FTLS) of the trusted computing base, together with an argument combining formal and informal techniques to show that the formal top level specification is consistent with the formal security policy model [NCS85b, p. 50]. Furthermore, it is required that formal methods be used in the mandatory covert channel analysis [NCS85b, p. 49].

Director, DoD Computer Security Evaluation Center, Schell was responsible for making things work.

Part IV
USING THE ORANGE BOOK

14 A Guide to Sections 15 through 18

Sections 15–18 discuss four aspects of processes and products associated with the creation of the Orange Book and the activities of the Center that have been chosen for analysis. This section provides a brief introduction to each of these sections.

Information and Expertise: Section 15

The Center's product evaluations and associated activities have led to publication of an impressive and useful collection of documents. Also, taken together, the Center's operations and the processes involved in creating the Center and the Orange Book have produced a valuable pool of computer security expertise that has spread through a substantial part of the computing industry. Section 15, referring to appendix B for an analysis of the Center's publications, concentrates on discussing this industrial diffusion of computer security expertise.

Availability of Products: Section 16

According to the Orange Book, "A major goal of the National Computer Security Center is to encourage the Computer Industry to develop trusted computer systems and products" [NCS85b, p. 58]. Relying on appendix C for an aggregate analysis of data on systems evaluated by the Center, section 16 discusses the extent to which this goal has been met.

Design Verification: Section 17

The Orange Book depends, in large part, on design verification to furnish the assurance required for multilevel secure systems. Section 17 examines the question "To what extent is this reliance justified?" in the light of experience with development efforts aimed at producing systems satisfying the A1 evaluation class criteria.

Proof Requirements and Endorsed Tools: Section 18

Section 18 discusses the quality of the endorsed tools available to vendors and its effect on projects aimed at producing highly secure systems.

15 Information and Expertise

The Orange Book effort has produced a substantial body of published computer security information and a valuable pool of computer security expertise. Both the publications and the expertise have been important resources in developing the draft *Federal Criteria for Information Technology Security* [NIS92a, NIS92b]. Appendix B discusses the National Computer Security Center's publications. This section discusses the pool of expertise resulting from the Orange Book effort.

The pool of expertise resulting from the Orange Book effort has spread through a substantial part of the computer industry, as those involved in developing the Orange Book and former members of the Center's staff have taken positions in the private sector and development of systems designed to meet the Orange Book requirements has led to the growth of indigenous knowledge about techniques for building secure systems within the technical staffs of manufacturers involved in such efforts.

That the diffusion and development of expertise attributed to the Orange Book effort in the preceding paragraph have occurred seems clear,[49] but the body of evidence gathered in preparing this report provides no way of accurately estimating their magnitude.[50] Progress could be made toward providing such estimates through detailed examination of the participant lists of the *Proceedings of the National Computer Security Conference* and *Proceedings of the IEEE Conference on Security and Privacy*, and other methods of estimation could be devised. But these are jobs for another time.

---

[49] See, example, the following parts of [Rep90]: (1) Stephen Lipner's letter and the statement by DEC, pp. 3–10, (2) Stephen Walker's oral and written testimony, pp 85–102, and William R. Whitehearst's letter and accompanying attachment, pp. 169–176.

[50] Culling my `.mailrc` file (the Unix file that controls the behavior of the `mail` program and allows users to defines aliases for electronic mail addresses) and sending an electronic mail query to ten people about former Center staff now working in industry produced a list of twentynine names relevant to the diffusion of expertise, but the actual diffusion figure must be several times this. (Only two recipients responded to the query, and force of circumstance prevented their spending much effort on answering it.)

The Agency was queried about the number of former Center staff now working in industry [Pot93d], [Pot93c], but this information is not tracked [Ano93b].

16 Availability of Products

In full, the goal referred to in section 14 is for trusted computer systems and products to be made "widely available in the commercial market place" [NCS85b, p. 58]. This section discusses the extent to which this goal has been met.

Appendix C analyzes aggregate data on products evaluated, relying primarily on data drawn from the Evaluated Products List (EPL). The analysis reveals that considerable progress toward realizing the goal of widespread commercial availability of trusted computer systems and products has been made at the C2, B1, and B2 levels, but there is a very sharp break at the B3 level. As far as operating systems, the initial focus of the Orange Book effort, are concerned, each of the B3 and A1 levels is limited to a single product.

Comparing the analysis of EPL given in appendix C with [CSE85c, tables 5 and 7] strongly indicates that, so far, the Orange Book effort has not led to commercially based satisfaction of the Department of Defense's need for multilevel secure systems. Consequently, the problem that drove the whole effort — running from the Ware and Anderson reports, through the fundamental work done by ESD, via the DoD Computer Security Initiative and the system building efforts reported in [Wal80] and [Lan83], to the creation of the Center and Orange Book, and beyond — has not been solved by the means chosen for its solution.

The following possible reasons for failure to solve the multilevel security problem on a commercial basis have come to light in the course of the research on which this report is based.

First, building a B3 system requires starting from scratch — you can't build one by modifying a B2 system [Sch93b].

Second, a number of economic factors have tended to work against production of B3 systems. (1) DoD has a weak record in maintaining procurement requirements specifying highly secure systems [Sch93b], [Bon93b]. This tends to destroy the competitive advantage builders of such systems were supposed to derive from the evaluation process. (2) Export controls, which come into play at the B3 level, discourage vendors from offering high end secure systems as part of their reguar product line [Rep90], [Sch93b], [Lip93]. (3) Shifts in the market for operating systems have tended to make the existing scheme for evaluating systems less relevant to actual demand than it once was [Lip93], [Bon93b]. (4) Government behavior has often been at odds with the stated policy for procuring secure systems, preferring

what are, in effect, bespoke systems built at the Government's expense to commercially developed systems [Sch93b], [Bon93b].

Clearly, building a B3 or A1 system is a difficult and costly task, and the combination of economic factors (1)–(4) is not likely to encourage manufacturers to take on the job. This is a plausible explanation for the B3 break, and there the matter must rest, as far as this report is concerned.

17 Design Verification

Both documentary evidence [Sch89, *passim.*] and interview evidence indicate that the design verification activities required at the A1 level tend to become dissociated from system construction. This tendency manifests itself in varying degrees, from an extreme where design verification is essentially epiphenomenal (the verification activity proceeds and so does system construction, but the former has little or no influence on the latter), through intermediate cases where the design and verification group and the system construction group drift apart, despite strong efforts to integrate their activities, to cases where the tendency, though present, seems to have been overcome.

Assessing the Evidence

Eight development efforts have aimed at producing A1 systems or DoD systems intended to meet the A1 requirements. The information about these development efforts that formed the base of evidence suggesting and supporting the thesis stated above can be outlined as follows:

1. Blacker
   (a) General references: [Wei92].
   (b) References bearing on dissociation tendency: [Wei93] (see appendix D), [FH93].
   (c) Remark: [Wei93] has a strong general bearing on the concerns of this section.
2. Boeing secure LAN
   (a) General references:[NSA93a, pp. 4-162–4-163], [Sch85], [NCS91a].

(b) References bearing on dissociation tendency: None.

3. DEC A1 prototype

   (a) General references: [KZB+90].

   (b) References bearing on dissociation tendency: [Lip93].

4. GEMSOS

   (a) General references: [NSA93a, pp. 4-42–4-43], [LPS89], [LTP90], [STH85].

   (b) References bearing on dissociation tendency: [LPS89], [LTP90], [STH85], [Sch93b].

5. LOCK

   (a) General references: [Boe88], [Cou91, pp. 251–252], [HKMY86], [HY86].

   (b) References bearing on dissociation tendency: [Hai93].

6. Multinet Gateway

   (a) General references: [BDF+86], [CGR93a, CGR93b].

   (b) References bearing on dissociation tendency: [CGR93a, CGR93b].

7. SACDIN

   (a) General references: [ITT78].

   (b) References bearing on dissociation tendency: None.

8. SCOMP

   (a) General references: [CSE83a], [NSA93a, pp. 4-53–4-54], [Fra83], [NCS85a].

   (b) References bearing on dissociation tendency: [Bon93b], [Har93c], [Har93b], [Sch89], [Sch93a], [Sch93b].

   (c) Remarks:

      i. [Sch93b] gives SCOMP as an example of a case where design verification worked well, but the other references cited under (b) tend to show that a very high degree of dissociation was present in the SCOMP development.

ii. [Bon93b] suggests a way of resolving the evidentiary conflict noted in (2): Roger Schell's direct experience with the project stemmed primarily from the KSOS 6 phase of the development, not the later construction of a general purpose operating system.

Evidence was too slight to support a well grounded judgment relevant to the thesis stated at the beginning of this section in the case of the following development efforts: Boeing secure LAN, Multinet Gateway, and SACDIN.

The SCOMP development was judged to have involved a high degree of dissociation. Although less information about the DEC A1 prototype is available than in the case of SCOMP, the degree of dissociation present seemed to fall somewhere between the middle and high end of the range.

A noticeable tendency toward dissociation was found in the LOCK effort, but this was recognized by the developers and considerable effort was put into overcoming it. In the case of Blacker, the developers were strongly aware of the possibility of dissociation, took steps to forestall and combat it, and seem largely to have succeeded. A similar judgment appears fair in the case of GEMSOS.

In summary, the evidence in hand is sufficient to conclude tentatively that the thesis of this section is correct. The remainder of the section attempts to explain why the tendency for design verification activities to become dissociated from system construction exists.

Explaining the phenomenon

The dissociation tendency seems arise partly due to a kind of cultural clash between those primarily concerned with design verification and those primarily engaged in actual system construction. A second contributing factor is the existing structure of managerial values, procedures, and institutional structures into which design verification is injected in attempts to meet the A1 requirements.

The general idea behind design verification is that getting very clear about what needs doing will be helpful in getting it done correctly. This is very plausible considered relative to a single individual and fairly plausible in the case of an organization where people in the organization share a common body of concepts and goals and communicate in a common language. Even in the latter case, however, an organization — *e.g.*, a military organization — may have to put sustained effort into fostering communication between those

involved in planning and those at the sharp end, if the potential benefits of clear planning are to be realized.

These factors are often lacking in the case of design verification. Due to their differing backgrounds, project members working with formal methods as part of the design verification process and project members engaged in writing code are likely to lack a common body of concepts, have differing goals, and not share a language adequate to express what they need to say to each other, if they are to cooperate effectively. Beyond this, as [Tie92, p. 259] observes, referring to [Qui91]:

> A development process centered on formal methods pushes its costs upstream: the specification and design stages cost more, while maintenance costs are presumably reduced. Indeed, this very change of emphasis from downstream to upstream is one of the major promised benefits of formal methods, for they are better able to catch those ambiguities of specification which become increasingly costly to correct the further down the lifecycle the software goes. However, there is a snag. For this benefit to be felt, client organizations must change the way projects are budgeted for. As Quintas has pointed out, budgeting for formal methods runs against normal practice. The Project Managers responsible for commissioning or building software are not normally the same people — and sometimes not even the same departments — responsible for maintaining the software later. On this account, Project Managers in client organizations have a problem justifying the higher initial costs of formal methods developments, when the benefits are are difficult to assess in advance, and anyway, they probably fall into somebody else's patch.

This does not bode well for the chances of putting in the time and effort necessary to foster communication between project members working on design verification and those engaged writing code.[51]

[51] It is worth pointing out that Roger Schell expressed very strong agreement with the ideas of the passage just quoted [Sch93b]. [CGR93a, CGR93b] contains information on the Multinet Gateway that may be relevant to these ideas, but, although the survey reported asked about the cost of formal methods use to the project, it did not ask specifically about their effect on the longitudinal profile of system cost.

18 Proof Requirements and Endorsed Tools

According to the A1 proof requirements, verification evidence showing that the formal top level specification is consistent with the formal security policy model must be "consistent with that provided within the state-of-the-art of the particular National Computer Security Center-endorsed formal specification and verification system used" [NCS85b, pp. 50–51]. The effect of this is to make the Endorsed Tools List [NSA93a, pp. 4-191–4-197] part of the A1 proof requirements' definition.

One would expect from this that populating the Endorsed Tools List (ETL) must have been one of the Center's initial concerns, and interview evidence bears this out. But there is strong evidence that really satisfactory tools were never produced [Goo93], [Har93c], [Sch93a], [Sch93b], [LTP90].

The deficiencies of the tools that were built and placed on the ETL have two aspects: (1) the tools available simply are not up to production quality standards, and (2) little seems to have been done to enhance tools in ways required in order to deal with the assurance requirements of levels beyond A1. It is clear that aspect (1) has had a negative impact on past and current projects aimed at building A1 systems. Aspect (2) is noteworthy for two additional reasons. First, according to [CSE85c, tables 5 and 7], systems satisfying requirements more stringent than those of the A1 level are necessary for a full solution of the multilevel security problem. Second, the discussion of section 17 tends toward the conclusion that, in general, the degree of assurance provided by design verification may have been overestimated.

Part V
COMMUNING WITH CLIO

19 Historiographic Questions

Clio, muse of history, fascinates and frustrates. Appropriately, so does the attempt to apply existing analytical historical models to the subject matter of this report.

Using the term "anomaly" in the general sense derived from the usage of [Kuh62] by [Con80b, chapter 1], it is fair to say that the work on computer security ultimately resulting in the creation of the Center and the Orange Book was driven by recognition of an anomaly. But recognition of this anomaly involved realizing that time-sharing systems, as they had evolved by 1967 and seemed likely to evolve in the absence of intervention by the defense community, would fail in the context of the existing institutional practices for handling sensitive information. Consequently, the anomaly in question was not a presumptive anomaly in the sense defined in [Con80b, chapter 1], because recognition of a presumptive anomaly must be due to *scientifically based* realization that an existing technology will fail, or perform unacceptably, in unrealized, but specifiable and possible, circumstances. Nor was the anomaly a case of functional failure or technological co-evolution, in the sense of these terms intended in [Con80b]. Furthermore, in a fairly straightforward sense, *the techniques devised with the aim of producing a technology that would resolve the computer security anomaly were aimed at building into the relevant artifacts* — operating systems — *the institutional practices that were the basis for recognizing the anomaly in the first place.*

There does not seem to be an analytical historical model that covers the situation described in the preceding paragraph.

Also, [Con80b, p. 275, note 33] begins by stating roundly:

> The individual does not design a new "tradition" [of technological practice]; he designs a new device. He may be fully aware that his device is different from the conventional system, but his goal is a thing, not a tradition of practice.

But what went on in the case of computer security looks very much like an effort to design a new tradition of practice and, certainly, became such an effort by the time Walker got the DoD Computer Security Initiative up and going.

Colleagues have suggested that the approach to studying technology found in [BHP87], particularly in the papers [Hug87], [Law87], and the ideas

of [Hug83] may be appropriate for analyzing the history the DoD Computer Security Initiative and Walker's role in it. Certainly, Charles Stark Draper, chief progatonist of [Mac90] — which developed from [Mac87], one of the contributions to [BHP87] — is a better analog of Stephen T. Walker than is, say, Frank Whittle, "the" inventor of the turbojet engine and a central figure in [Con80b].

Walter Vincenti's account of engineering espitemology, given in [Vin90, chapters 7 and 8], applies tolerably well to the case of trusted computer systems, as far as design of particular systems is concerned, but there is a glaring exception — numerical methods do not, and, probably cannot, play the kind of role in software engineering that Vincenti documents in the case of aeronautical engineering and, no doubt, can be documented for other areas of engineering not aimed at producing computer systems. The reason for the "cannot" is essentially the one Dijkstra [Dij82, *passim*] and Parnas [Par85] have emphasized: software is discrete. So, for example, talk about a model of software behavior's being "correct as a first order approximation" is, at best, metaphorical, and, further, the method of paramater variation, prominently discussed by Vincenti [Vin90, chapter 5], is inapplicable.

These features both account for the yen for proofs that seems to be characteristic of software engineering, as opposed to other kinds of engineering, and, also, show why the software verification enterprise is inherently problematic. Testing can't provide the same kind of assurance it furnishes in other branches of engineering, so proofs are very desirable. But proofs require idealized models, and we have no satisfactory way of expressing the inevitable question of how well the models describe what is actually being built, let alone having satisfactory ways of answering it.

Part VI

APPENDICES AND REFERENCES

A Summary of Evaluation Criteria Classes, from the Orange Book, pp. 93–94

The classes of systems recognized under the trusted computer system evaluation criteria are as follows. They are presented in the order of increasing desirability from a computer security point of view.

Class (D): Minimal Protection

This class is reserved for those systems that have been evaluated but that fail to meet the requirements for a higher evaluation class.

Clas (C1): Discretionary Security Protection

The Trusted Computing Base (TCB) of a class (C1) system nominally satisfies the discretionary security requirements by providing separation of users and data. It incorporates some form of credible controls capable of enforcing access limitations on an individual basis, i.e., ostensibly suitable for allowing users to be able to protect project or private information and to keep other users from accidentally reading or destroying their data. The class (C1) environment is expected to be one of cooperating users processing data at the same level(s) of sensitivity.

Class (C2): Controlled Access Protection

Systems in this class enforce a more finely grained discretionary access control than (C1) systems, making users individually accountable for their actions through login procedures, auditing of security-relevant events, and resource isolation.

Class (B1): Labeled Security Protection

Class (B1) systems require all the features required for class (C2). In addition, an informal statement of the security policy model, data labeling, and mandatory access control over named subjects and objects must be present. The capability must exist for accurately labeling exported information. Any flaws identified by testing must be removed.

Class (B2): Structured Protection

In class (B2) systems, the TCB is based on a clearly defined and documented formal security policy model that requires the discretionary and mandatory access control enforcement found in class (B1) systems to be extended to all subjects and objects in the ADP system. In addition, covert channels are addressed. The TCB must must be carefully structured into protection-critical and non-protection-critical elements. The TCB interface is well-defined and the TCB design and implementation enable it to be subjected to more thorough testing and more complete review. Authentication mechanisms are strengthened, trusted facility management is provided in the form of support for system administrator and operator functions, and stringent configuration management controls are imposed. The system is relatively resistant to penetration.

Class (B3): Security Domains

The class (B3) TCB must satisfy the reference monitor requirements that it mediate all accesses of subjects to objects, be tamperproof, and be small enough to be subjected to analysis and tests. To this end, the TCB is structured to exclude code not essential to security policy enforcement, with significant system engineering during TCB design and implementation directed toward minimizing its complexity. A security administrator is supported, audit mechanisms are expanded to signal security-relevant events, and system recovery procedures are required. The system is highy resistant to penetration.

Class (A1): Verified Design

Systems in class (A1) are functionally equivalent to those in class (B3) in that no additional architectural features or policy requirements are added. The distinguishing feature of systems in this class is the analysis derived from formal design specification and verification techniques and the resulting high degree of assurance that the TCB is correctly implemented. This assurance is development in nature, starting with a formal model of the security policy and a formal top-level specification (FTLS) of the design. In keeping with the extensive design and development analysis of the TCB required of systems in class (A1), more stringent configuration management is required and procedures are established for securely distributing the system to sites. A system security administrator is supported.

B The Center's Publications

This appendix breaks the Center's publications down into three main categories. Subsection B.1 deals with final evaluation reports, subsection B.2 covers the Rainbow Series, and subsection B.3 lists kinds of publications not incuded under the other two heads. The discussion is based mainly on [NSA93b] and [NSA93a].

B.1 Final Evaluation Reports

Table 1 shows the number of final evaluation reports published by the Center for the years 1985–1992. The data aggregated in the table are drawn from [NSA93b, pp. 10–13] and [NSA93a, pp. 4-1–4-199].[52]

1984	1985	1986	1987	1988	1989	1990	1991	1992	Total
2	3	7	8	10	10	8	8	7	63

Table 1: Final evaluation reports published by year. Mean = 7 reports/year, median = 8 reports/year, mode = 8 reports/year.

B.2 The Rainbow Series

The members of the Rainbow Series are especially prominent among the Center's publications. Listed in order of publication, they are [NSA93b, pp. 1–4]:[53]

(1) *Password Management Guidelines* [CSE85b],

(2) *Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments* [CSE85a],

[52] Dates of reports listed in [NSA93b, pp. 10–13] were inferred from document numbers — *e.g.*, CSC-EPL-92/002, *DEMAX Software Incorporated SECUREPAK 3.2*, p. 13, was assigned 1992 as its date, in accordance with [Ano93a].

[53] The following rules were used in dating the members of the Rainbow Series and constructing the list. In cases where there was a conflict between the cover date and the internal date, the internal date was assigned to the document. If there was no such conflict but one date was more specific than the other, the more specific date was assigned. If the internal and external dates were the same, documents were ordered according to document number.

(3) *Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements* [CSE85c],
(4) *Trusted Computer System Evaluation Criteria* [NCS85b],
(5) Advisory Memorandum on Office Automation Security Guideline [NMT87],
(6) *A Guide to Understanding Audit in Trusted Systems* [NCS87a],
(7) *Trusted Network Interpretation* [NCS87c],
(8) *A Guide to Understanding Discretionary Access Control in Trusted Systems* [NCS87b],
(9) *A Guide to Understanding Configuration Management in Trusted Systems* [NCS88c],
(10) *Computer Security Subsystem Interpretation of the Trusted Computer System Evaluation Criteria* [NCS88a],
(11) *A Guide to Understanding Design Documentation in Trusted Systems* [NCS88d],
(12) *Glossary of Computer Security Terms* [NCS88b],
(13) *A Guide to Understanding Trusted Distribution in Trusted Systems* [NCS88e],
(14) *Guidelines for Formal Verification Systems* [NCS89b],
(15) *Rating Maintenance Phase Program Document* [NCS89c],
(16) *Trusted UNIX Working Group (TRUSIX) Rationale for Selecting Access Control List Features for the UNIX* System* [NCS89d],
(17) *A Guide to Understanding Trusted Facility Management* [NCS89a],
(18) *Trusted Product Evaluations — Guide for Vendors* [NCS90b],
(19) *Trusted Network Interpretation Environments Guideline* [NCS90a],
(20) *Trusted Database Management System Interpretation* [NCS91f],
(21) *A Guide to Understanding Identification and Authentication in Trusted Systems* [NCS91c],
(22) *A Guide to Understanding Data Remanence in Automated Information Systems* [NCS91b],
(23) *A Guide to Writing the Security Features User's Guide for Trusted Systems* [NCS91e],
(24) *A Guide to Understanding Trusted Recovery in Trusted Systems* [NCS91d],
(25) *Trusted Product Evaluation Questionnaire* [NCS92g],
(26) *A Guide to Understanding Information System Security Officer Responsibilities for Automated Information Systems* [NCS92c],
(27) *Assessing Controlled Access Protection* [NCS92a],
(28) *A Guide to Understanding Object Reuse in Trusted Systems*

[NCS92d],

(29) *A Guide to Understanding Security Modeling in Trusted Systems* [NCS92e],

(30) *Guidelines for Writing Trusted Facility Manuals* [NCS92f], and

(31) *A Guide to Procurement of Trusted Systems: An Introduction to Procurement Initiators on Computer Security Requirements* [NCS92b].

Inspection of the preceding list shows that the Center published an average of about four members of the Rainbow Series per year from 1985 through 1993,[54] with a maximum production of seven members in 1992.

The cover colors of the documents in the Rainbow Series mean nothing [Ano93c], but the document numbers give a rough indication of the order in which the writing projects that produced the documents were begun [Ano93a]. The process was like getting the schedules needed in "doing your income tax" [Ano93a] — when the need for a document was felt, a number was assigned and the writing task was initiated. For these reasons, it is worth listing the members of the Series document number order:[55]

(1) *Trusted Computer System Evaluation Criteria* [NCS85b],

(2) *Password Management Guidelines* [CSE85b],

(3) *Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments* [CSE85a],

(4) *Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements* [CSE85c],

(5) Advisory Memorandum on Office Automation Security Guideline, [NMT87],

(6) *A Guide to Understanding Audit in Trusted Systems* [NCS87a],

(7) *Trusted Product Evaluations — Guide for Vendors* [NCS90b],

(8) *A Guide to Understanding Discretionary Access Control in Trusted Systems* [NCS87b],

(9) *Glossary of Computer Security Terms* [NCS88b],

(10) *Trusted Network Interpretation* [NCS87c],

(11) *A Guide to Understanding Configuration Management in Trusted Systems* [NCS88c],

[54] The mean is 3.875 documents/year, and the median and mode are both 4 documents/year, with modal production occurring in 1985, 1987, and 1989.

[55] The document numbers are given in the report's reference list. There are gaps in the numbering, because, for various reasons, some of the writing efforts were abandoned [Ano93a]. The Orange Book is, of course, listed first, document numbers nonwithstanding.

(12) *A Guide to Understanding Design Documentation in Trusted Systems* [NCS88d],
(13) *A Guide to Understanding Trusted Distribution in Trusted Systems* [NCS88e],
(14) *Computer Security Subsystem Interpretation of the Trusted Computer System Evaluation Criteria* [NCS88a],
(15) *A Guide to Understanding Security Modeling in Trusted Systems* [NCS92e],
(16) *Trusted Network Interpretation Environments Guideline* [NCS90a],
(17) *Rating Maintenance Phase Program Document* [NCS89c],
(18) *Guidelines for Formal Verification Systems* [NCS89b],
(19) *A Guide to Understanding Trusted Facility Management* [NCS89a],
(20) *Guidelines for Writing Trusted Facility Manuals* [NCS92f],
(21) *A Guide to Understanding Identification and Authentication in Trusted Systems* [NCS91c],
(22) *A Guide to Understanding Object Reuse in Trusted Systems* [NCS92d],
(23) *Trusted Product Evaluation Questionnaire* [NCS92g],
(24) *Trusted UNIX Working Group (TRUSIX) Rationale for Selecting Access Control List Features for the UNIX* System* [NCS89d],
(25) *Trusted Database Management System Interpretation* [NCS91f],
(26) *A Guide to Understanding Trusted Recovery in Trusted Systems* [NCS91d],
(27) *A Guide to Procurement of Trusted Systems: An Introduction to Procurement Initiators on Computer Security Requirements* [NCS92b],
(28) *A Guide to Understanding Data Remanence in Automated Information Systems* [NCS91b],
(29) *A Guide to Writing the Security Features User's Guide for Trusted Systems* [NCS91e],
(30) *A Guide to Understanding Information System Security Officer Responsibilities for Automated Information Systems* [NCS92c], and
(31) *Assessing Controlled Access Protection* [NCS92a].

The members of the Rainbow Series can be grouped into three functional categories — criterial documents, advisory documents, and explanatory documents.[56]

[56]Construction of this scheme was informed by [Ano93a], but I devised and named the categories and parceled out the documents among them. So this way of classifying

Criterial documents include the Orange Book and interpretations that adapt the Orange Book criteria to products other than operating systems, together with [NCS89c]:

(1) *Trusted Computer System Evaluation Criteria* [NCS85b],
(2) *Trusted Network Interpretation* [NCS87c],
(3) *Computer Security Subsystem Interpretation of the Trusted Computer System Evaluation Criteria* [NCS88a],
(4) *Rating Maintenance Phase Program Document* [NCS89c], and
(5) *Trusted Database Management System Interpretation* [NCS91f].

Advisory documents contain advice on how to do various things:

(1) *Password Management Guidelines* [CSE85b],
(2) *Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments* [CSE85a],
(3) *Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements* [CSE85c],
(4) Advisory Memorandum on Office Automation Security Guideline [NMT87],
(5) *Guidelines for Formal Verification Systems* [NCS89b],
(6) *Trusted UNIX Working Group (TRUSIX) Rationale for Selecting Access Control List Features for the UNIX* System* [NCS89d],
(7) *Trusted Product Evaluations — Guide for Vendors* [NCS90b],
(8) *Trusted Network Interpretation Environments Guideline* [NCS90a],
(9) *A Guide to Writing the Security Features User's Guide for Trusted Systems* [NCS91e],
(10) *Trusted Product Evaluation Questionnaire* [NCS92g],
(11) *Guidelines for Writing Trusted Facility Manuals* [NCS92f], and
(12) *A Guide to Procurement of Trusted Systems: An Introduction to Procurement Initiators on Computer Security Requirements* [NCS92b].

Explanatory documents provide expository treatments of technical matters:

(1) *A Guide to Understanding Audit in Trusted Systems* [NCS87a],
(2) *A Guide to Understanding Discretionary Access Control in Trusted Systems* [NCS87b],
(3) *A Guide to Understanding Configuration Management in Trusted Systems* [NCS88c],

---

members of the Rainbow series has no official standing.

(4) *A Guide to Understanding Design Documentation in Trusted Systems* [NCS88d],
(5) *Glossary of Computer Security Terms* [NCS88b],
(6) *A Guide to Understanding Trusted Distribution in Trusted Systems* [NCS88e],
(7) *A Guide to Understanding Trusted Facility Management* [NCS89a],
(8) *A Guide to Understanding Identification and Authentication in Trusted Systems* [NCS91c],
(9) *A Guide to Understanding Data Remanence in Automated Information Systems* [NCS91b],
(10) *A Guide to Understanding Trusted Recovery in Trusted Systems* [NCS91d],
(11) *A Guide to Understanding Information System Security Officer Responsibilities for Automated Information Systems* [NCS92c],
(12) *Assessing Controlled Access Protection* [NCS92a],
(13) *A Guide to Understanding Object Reuse in Trusted Systems* [NCS92d], and
(14) *A Guide to Understanding Security Modeling in Trusted Systems* [NCS92e].

B.3 Other Publications

The Center also publishes technical reports, computer security awareness materials, computer security posters, and videotapes [NSA93b] and prepares the Evaluated Products List [NSA93a, pp. 4-1–4-199].

C Products Evaluated by the Center

This appendix analyzes aggregate data on products evaluated by the Center, relying primariy on data drawn from the Evaluated Products List.

Table 2 is a key to abbreviations for kinds of EPL product entries.[57] These abbreviations are used in the EPL itself and in the other tables of this appendix.

PB:	Product Bulletin EPL entry.
OS:	Operating System EPL entry.
AO:	Ad-on EPL entry.
SS:	Subsystem EPL entry.
N:	Network EPL entry.
CMW:	Compartmented Mode Workstation EPL entry.
DB:	Database EPL entry.
RAMP:	Rating Maintenance Phase EPL entry.

Table 2: Kinds of Evaluated Products List product entries.

It should be reasonably clear what OS's, N's, and DB's cover, but the other kinds of EPL entries require explanation, which, in turn, presupposes an account of the Center's Trusted Product Evaluation Program (TPEP) [NSA93a, pp. 4-34–4-35].

The TPEP includes four evaluation phases — Vendor Assistance Phase (VAP), Design Analysis Phase (DAP), Formal Evaluation (FE), and Rating Maintenance Phase (RAMP).[58]

VAP [NSA93a, p. 4-34]:

[57] The phrase "EPL product entries" may look redundant, but it's not. Besides product entries, the Evaluated Products List contains a section titled "Publications Issued by the Standards, Criteria and Guidelines Division" [NSA93a, pp. 4-188–4-190], the Endorsed Tools List [NSA93a, pp. 4-191–4-197], and a section on Dockmaster [NSA93a, pp. 4-198–4-199], "The National Computer Security Center's ... unclassified computer system ... established in 1985 as an Information Security Showplace" [NSA93a, pp. 4-199].

[58] There is also a preliminary Proposal Review Phase [NCS90b, pp. 7–16] during which NSA's Information Systems Security Organization decides whether to devote resources to evaluating a product and, if the decision is affirmative, signs an appropriate legal agreement with the product's vendor and assigns an evaluation team. However, the Proposal Review Phase is not relevant to explaining what the entries of table 2 mean and will be ignored in what follows.

> ... is the first of the three phases of a typical evaluation of an operating system, network or network component. (Subsystems move directly into Formal Evaluation.) During VAP, the NSA serves primarily in an advisory capacity.
>
> ...
>
> During VAP the vendor completes the development of the product, designs security test procedures, and drafts documentation while the NSA ensures that the vendor's documentation of these efforts reflects an understanding of trust technology and evaluation requirements as they are articulated in the *Trusted Computer System Evaluation Criteria* (TCSEC).

During DAP, the second phase of evaluation [NSA93a, p. 4-34]:

> ... the product is largely completed and the evaluation team develops a detailed understanding of the system, its security features and its assurances.
>
> ... Products in Design Analysis must become commercially available within twelve months of the start of this phase (if not already available).

FE is the final phase of evaluation, in which [NSA93a, p. 4-34]:

> ... the evaluation team analyzes and tests the implementation's compliance with the TCSEC requirements for the candidate level of trust (or the requirements of an appropriate interpretation of the TCSEC, such as the Trusted Network Interpretation). The next step of the evaluation is the generation of the Final Evaluation Report.

Beyond this [NSA93a, pp.4-34–4-35]:

> Products evaluated at all levels of trust then continue with the Rating Maintenance Phase. The purpose of this phase is to provide the customer with current versions of trusted products.

The effect of the Rating Maintenance Phase is "Limited to maintenance of a specific rating ..." [NSA93a, p. 4-32].[59]

[59] Originally, RAMP was available only for C1, C2, and B1 evaluation class ratings [NCS89c, p. 3], but the requirements for B2–A1 RAMP were published electronically on September 30, 1992 on Dockmaster's `announce` forum (for Dockmaster, see note 57) and the Center is preparing to circulate the final draft of a revised version of [NCS89c] for comment [Ano93d].

It should be clear from the material of the preceding paragraph what RAMP's are. PB's are "synopses of systems currently undergoing formal evaluation" [NSA93a, p. 4-31]. It remains to explain AO's, SS's, and CMW's.

AO's are relics of an earlier stage in the Trusted Product Evaluation Program's development [NSA93a, p. 4-31]:

> An add-on package is a facility that runs in conjunction with a specific operating system and is not, by itself, a system that performs all the functions traditionally ascribed to an operating system. Initially, the evaluation of an add-on package did not include a complete evaluation of the underlying operating system for which the add-on package was designed. The evaluations which were performed in that manner are identified in the add-on package section of [the EPL]. Evaluations of add-on systems now include an equally thorough analysis of the security-relevant mechanisms contained in the underlying operating system ... These systems are identified in the operating systems section of [the EPL].

As for SS's [NSA93a, p. 4-31]:

> Subsystems are special-purpose products that can be added to existing computer systems to increase security and implement only a subset of the security features identified in the [TCSEC]. Features we evaluate are identification and authentication, audit, access control, and object reuse. Subsystems are evaluated against the *Computer Security Subsystem Interpretation* of the [TCSEC]. The ratings assigned use a special nomenclature to distinguish them from complete system ratings.
>
> ...
>
> A subsystem evaluation is concerned only with the subsystem product, and not any host system that it may support.

CMW's are unique, in that they stem from evaluations based jointly on the Orange Book and the Defense Intelligence Agency's *Security Requirements for System High and Compartmented Mode Workstations.* See [NSA93a, p. 4-166] for further explanation. [CSE85c, pp. 2–3] gives definitions of system high and compartmented security modes.

With these preliminary explanations in hand, the remainder of the appendix examines the Evaluated Products List in detail.

Table 3 shows product entries by kind and year for the January 1993 EPL [NSA93a, pp. 4-39–4-187].[60]

	PB	OS	AO	SS	N	CMW	DB	RAMP	Total
1984	0	1	2	0	0	0	0	0	3
1985	0	1	1	0	0	0	0	0	2
1986	0	3	0	4	0	0	0	0	7
1987	0	2	0	5	0	0	0	0	7
1988	0	4	0	7	0	0	0	0	11
1989	0	3	0	6	0	0	0	0	9
1990	0	2	0	3	1	0	0	1	7
1991	1	1	0	3	1	1	0	3	10
1992	5	4	0	3	0	0	0	2	14
Total	6	21	3	31	2	1	0	6	70

Table 3: Evaluated Products List product entries by kind and year.

C.1 Operating System Products Rated C1–A1

Due to the EPL's complex structure, data on operating system products rated C1–A1 can be aggregated in several different ways. Tables 4, 6, 7, 8, and 9 are the results of applying five such methods of aggregation to the data furnished by the EPL. Tables 10, 11, and 12 compare the different overviews of EPL data provided by tables 4, 6, 7, 8, and 9.

Table 4 shows operating system products by level of trust and year, including products for which evaluation has been completed and products in FE. The data aggregated are from the EPL Indices [NSA93a, pp. 4-3–4-26]. Note that this data set is different from the data set on which table 3 is based. Eventually, this difference leads to a *prima facie* conflict between the OS column total shown in table 3 and the most conservative estimate given in this subsection of the number of operating system products rated C1–A1. The apparent conflict between these figures is resolved in discussing table 9.

[60] It may seem that the total of 63 final evaluation reports shown in table 1 of appendix B, subsection B.1, should agree with the figure 64 obtained by subtracting 6, the number of PB's included in the January 1993 EPL, from the grand total of 70 product entries shown in table 3. But this is not so — some reports listed in [NSA93b, pp. 10–13] refer to products not included in the January 1993 EPL, and some entries in the January 1993 EPL do not cite a formal report and do not correspond to an entry in [NSA93b, pp. 10–13]. Hence, the proposed comparison of tables 1 and 3 is unfounded.

	C1	C2	B1	B2	B3	A1	Total
1984	1	1	0	0	0	1	3
1985	0	1	0	1	0	0	2
1986	0	3	0	0	0	0	3
1987	0	2	0	0	0	0	2
1988	0	4	0	0	0	0	4
1989	0	1	2	0	0	0	3
1990	0	1	2	0	0	0	3
1991	0	1	2	2	0	0	5
1992	0	4	4	1	1	0	10
Total	1	18	10	4	1	1	35

Table 4: Operating system products by level of trust and year, including products for which evaluation has been completed and products in Formal Evaluation.

Table 5 shows the number of RAMP's included in the data used in constructing table 4. The data on which table 5 is based are drawn from the EPL Indices [NSA93a, pp. 4-3–4-26].

	C1	C2	B1	B2	B3	A1	Total
1990	0	0	1	0	0	0	1
1991	0	1	2	0	0	0	3
1992	0	0	2	0	0	0	2
Total	0	1	5	0	0	0	6

Table 5: Rating Maintenance Phase Evaluated Products List entries included in data used to prepare table 4.

Table 6 is the result of removing from table 4 the effect of the RAMP's included in the data on which table 4 is based. The data used in preparing table 6 are drawn from [NSA93a, pp. 4-3–4-26, 4-171, 4-174, 4-177, 4-180, 4-183, and 4-186] and [NSA93b, pp. 10–13].

The 1991 C2 RAMP shown in table 5 stems from a product that received its rating in 1990. Accordingly, in preparing table 6, the 1991/C2 entry of table 4 was decremented by 1 and the 1990/C2 entry was incremented by the same amount. The 1990 B1 RAMP shown in table 5 stems from a product that received its rating in 1989, so the 1990/B1 entry of table 4 was decremented by 1 and the 1989/B1 entry was incremented by 1. One of the 1991 B1 RAMP's shown in table 5 traces back to the 1989 product

rating mentioned in the preceding sentence, and the other 1991 B1 RAMP and both 1992 B1 RAMP's shown in table 5 originate from another product rated B1 in 1989. Consequently, both the 1991/B1 and 1992/B1 entries of table 4 were decremented by 2, and the 1989/B1 entry received an additional increment of 1.

	C1	C2	B1	B2	B3	A1	Total
1984	1	1	0	0	0	1	3
1985	0	1	0	1	0	0	2
1986	0	3	0	0	0	0	3
1987	0	2	0	0	0	0	2
1988	0	4	0	0	0	0	4
1989	0	1	4	0	0	0	5
1990	0	2	1	0	0	0	3
1991	0	0	0	2	0	0	2
1992	0	4	2	1	1	0	8
Total	1	18	7	4	1	1	32

Table 6: Result of removing the effect of Rating Maintenance Phase Evaluated Products List entries from table 4.

Table 7 shows the result of excluding EPL entries for products in FE from the data used in preparing table 4, and table 8 shows the result of excluding EPL entries for products in FE from the data used in preparing table 6. The data used in constructing tables 7 and 8 are drawn from the EPL indices [NSA93a, pp. 4-3–4-26]. In both cases, the 1991/B2 entry was decremented by 1 and the 1992/C2 and 1992/B1 entries were decremented by 2.

Table 9 shows the result of excluding AO's from the data used in preparing table 8. The data used in preparing table 9 are drawn from the EPL Indices [NSA93a, pp. 4-3–4-26] and the Add-On EPL Entries section of the EPL [NSA93a, pp. 4-107–4-113].

The difference between the grand total of 24 operating system products arrived at in table 9 and the total of 21 such products shown in the OS column of table 3 is traceable to the procedure employed in deriving table 6 from table 4. Viewed from the perspective of the data set used in preparing table 3, [NSA93a, pp. 4-39–4-187], the overall effect of the procedure is to *increase* the count of operating system products by 3, since the 6 RAMP's noted in table 5 stem from 3 OS's that were present in earlier versions of the EPL. This increase in the operating system products count carries down

	C1	C2	B1	B2	B3	A1	Total
1984	1	1	0	0	0	1	3
1985	0	1	0	1	0	0	2
1986	0	3	0	0	0	0	3
1987	0	2	0	0	0	0	2
1988	0	4	0	0	0	0	4
1989	0	1	2	0	0	0	3
1990	0	1	2	0	0	0	3
1991	0	1	2	1	0	0	4
1992	0	2	2	1	1	0	6
Total	1	16	8	3	1	1	30

Table 7: Result of excluding Evaluated Products List entries for products in Formal Evaluation from data used in preparing table 4.

	C1	C2	B1	B2	B3	A1	Total
1984	1	1	0	0	0	1	3
1985	0	1	0	1	0	0	2
1986	0	3	0	0	0	0	3
1987	0	2	0	0	0	0	2
1988	0	4	0	0	0	0	4
1989	0	1	4	0	0	0	5
1990	0	2	1	0	0	0	3
1991	0	0	0	1	0	0	1
1992	0	2	0	1	1	0	4
Total	1	16	5	3	1	1	27

Table 8: Result of excluding Evaluated Products List entries for products in Formal Evaluation from data used in preparing table 6.

	C1	C2	B1	B2	B3	A1	Total
1984	0	0	0	0	0	1	1
1985	0	0	0	1	0	0	1
1986	0	3	0	0	0	0	3
1987	0	2	0	0	0	0	2
1988	0	4	0	0	0	0	4
1989	0	1	4	0	0	0	5
1990	0	2	1	0	0	0	3
1991	0	0	0	1	0	0	1
1992	0	2	0	1	1	0	4
Total	0	14	5	3	1	1	24

Table 9: Result of excluding Add-on Evaluated Products List entries from data used in preparing table 8.

through the derivation of table 9 from table 6 via table 8, thus accounting for the difference between the grand total of table 9 and the OS column total of table 3.

Table 10 compares the bottom lines of tables 4, 6, 7, 8, and 9.

	C1	C2	B1	B2	B3	A1	Total
Table 4	1	18	10	4	1	1	35
Table 6	1	18	7	4	1	1	32
Table 7	1	16	8	3	1	1	30
Table 8	1	16	5	3	1	1	27
Table 9	0	14	5	3	1	1	24

Table 10: Bottom line comparison of tables 4, 6, 7, 8, and 9. Table 4 includes products in FE. Table 6 removes the effects of RAMP's from table 4. Table 7 excludes products in FE from data used in preparing table 4. Table 8 excludes products in FE from data used in preparing table 6. Table 9 excludes AO's from data used in preparing table 8.

Table 11 compares percentages of products rated at levels C1–A1 according to tables 4, 6, 7, 8, and 9, and table 12 compares yearly averages of operating system products evaluated derived from tables 4, 6, 7, 8, and 9.

	C1	C2	B1	B2	B3	A1	Total
Table 4	2.86	51.43	28.57	11.43	2.86	2.86	100.01
Table 6	3.13	56.25	21.88	12.50	3.13	3.13	100.02
Table 7	3.33	53.33	26.67	10.00	3.33	3.33	99.99
Table 8	3.70	59.26	18.52	11.11	3.70	3.70	99.99
Table 9	0.00	58.33	20.83	12.50	4.17	4.17	100.00

Table 11: Comparison of percentages of products rated at levels C1–A1 according to tables 4, 6, 7, 8, and 9.

	Table 4	Table 6	Table 7	Table 8	Table 9
Mean	3.89	3.56	3.33	3.00	2.67
Median	3	3	3	3	3
Modes	3	2,3	3	3	1
Modal Years	84,86,89,90	2:85,87,91 3:84,86,90	84,86,89,90	84,86,90	84,85,91

Table 12: Comparison of yearly averages of operating system products evaluated derived from tables 4, 6, 7, 8, and 9.

C.2 Other Products

Table 13 shows other products rated C1–A1, including those for which evaluation has been completed and those in FE. The data used in preparing table 13 are from the EPL Indices [NSA93a, pp. 4-3–4-26].

Table 14 shows products in VAP and DAP. The data used in preparing table 14 are from the EPL Indices [NSA93a, pp. 4-3–4-26].

		C1	C2	B1	B2	B3	A1	Total
N	1990	0	0	0	1	0	0	1
	1991	0	0	0	0	0	1	1
	1992	0	0	0	0	0	1	1
CMW	1991	0	0	1	0	0	0	1
Total		0	0	1	1	0	2	4

Table 13: Other products rated C1–A1, including those for which evaluation has been completed and those in Formal Evaluation. The 1990 N, 1991 N, and 1991 CMW entries are for completed evaluations, and the 1992 N entry is for a product in FE.

	N/A	TBD	C	B	A	Total
OS	*0*	*10*	**1**	**3**	**0**	14
N	*0*	*2*	**0**	**2**	**1**	5
CMW	*0*	*1*	**0**	**3**	**0**	4
DB	*1*	*0*	**2**	**1**	**0**	4
Total	*1*	*13*	**3**	**9**	**1**	27

Table 14: Products in Vendor Assistance Phase and Design Analysis Phase. N/A: Not Applicable. TBD: To Be Determined. VAP figures are in *italics*. DAP figures are in **boldface**. Dates for entering VAP range from September 5, 1990 through September 18, 1992. Dates for entering DAP range from June 1, 1989 through November 10, 1992.

D Learning to Do It Right — From Autodin II to Blacker

The following query, part of an effort to test the thesis of section 17, was sent to Clark Weissman on December 10, 1993, together with a description of the research on which this report is based [Pot93a]:

> Both documentary evidence (Marv Schaefer's paper "Symbol Security Condition Considered Harmful")[61] and interview evidence indicate that the design verification activities required at the A1 level tend to become dissociated from system construction. This tendency manifests itself in varying degrees, from an extreme where design verification is essentially epiphenomenal (the verification activity proceeds and so does system construction, but the former has little or no influence on the latter), through intermediate cases where the design and verification group and the system construction group drift apart, despite strong efforts to integrate their activities, to cases where the tendency seems to have been overcome.
>
> I would appreciate it if you could tell me where, according to your experience, the Blacker development falls within the range of possibilities suggested in the preceding paragraph. (As I understand it, though not a commercial product, the system was intended to satisfy the A1 requirements.) I am particularly interested in the following points. (1) Did the tendency toward dissociation manifest itself? If so: (2) How? (3) Why, in your opinion? (4) What was done to combat it? (5) How successful were the efforts to combat it?

Weissman replied on December 10 [Wei93]:

> First, some preamble.
>
> System Development Corporation (SDC) invented the Formal Development Methodology (FDM) over a 10 year period prior to and including Blacker. It was matured on a number of earlier systems – Autodin II, KVM (Marv Schaefer used to

[61] Note written December 19, 1993: See [Sch89].

work for me at SDC), DTI's HUB, internal R&D projects, and Blacker. During the earlier efforts, we explored different blends of software and trust engineering — independent teams on each thread, integrated teams, and places in between. In Autodin II,[62] we gave courses to Ford Aerospace (now Loral) programmers on FDM who then wrote the formal specs and code. We and they found that a failure. The field of formal notations and mathematical formalism was new and most programmers unaware of its properties. The Ford programmers behaved just like SDC programmers and used the Ina Jo State Description language just as they used Jovial or Algol programming languages. They had great difficulty with existential quantifiers, with precision in the state space and all predicates needed. [Dijkstra] is known for saying that ... programming is too difficult for most of the professionals.[63] We saw exactly that. The formal specs were like code, not math and impossible to comprehend, let alone prove. So [the] "throwing it (formal specs) over the wall" method failed.

Next we tried doing it ourselves with skilled mathematicians on the HUB,[64] a proprietary kernel of a real-time operating system for a 1970's style communications processor for the Defense Communication Agency (now DISA). That was not a success even though we did complete and prove the specs because of your "dissociation" of teams. The FTLS and DTLS-Code were in a race to finish, and code won. As is typical in large programming jobs, the code deviated from the DTLS and the DTLS was not updated. In the end, the FTLS was being developed from the code, a terrible form of "re-engineering." Lesson learned there was that even smart people on both threads — FTLS & DTLS-Code — didn't make for good development.

On KVM[65] we integrated the team and succeeded. The problems were real — new constructs were needed in the FDM tools

[62] Author's note: For information on Autodin II, see [Wal82], [Lan83, Appendix A], and [Jel85, pp. III-84–III-91]. According to [Lan83, Appendix A], "There were many problems in its development, including a court fight over the definition of 'formal specification'." This court action, rather than the suit filed over VIPER [Mac91], seems to be the first instance in which a disagreement about formal methods has led to litigation.

[63] Author's note: Ellipsis in original.

[64] Author's note: See [Lan83, Appendix A].

[65] Author's note: See [Lan83, Appendix A].

(which were being developed concurrently), parts of the KVM architecture were new security approaches — virtualization — and it was just hard to think trust. We had a great team, today's leaders in the field scattered now in many companies. The key to success was doing it right, not just doing it. We had the luxury of a DARPA R&D contract and not a fixed-price contract, as we find in today's market.

Now when Blacker began its A1 trek in 1984, there was much experience in the SDC team in both the formal and design aspects (the final Blacker was based on 10 years of earlier R&D on cryptographic applications to packet communications.) The government also had great knowledge, experience, and desire to achieve A1. Blacker attained A1 evaluation in 1991. [Here are] my answers to your questions.

Q0. Where does Blacker fall ... ?

A0. Blacker development was close to the fully integrated side of the spectrum, but not all the way. There was a distinct formal group and a more traditional software engineering group. However, the two groups worked quite closely as noted below.

Q1. Tendency toward dissociation ... ?

A1. Yes

Q2. How? Q3. Why?

A2. & 3. Schedule pressures tended to force development forward unmercifully following the NSA NACSMs, a DOD-STD 2167A like "waterfall" development model, whereas the trust development was closer to the Spiral Model of Barry Boehm ["A Spiral Model of Software Development and Enhancement", Software Engineering Notes, ACM, Vol 11, No.4, pp. 14–24, 1986].[66] For example, proofs of [the] FTLS (or unproofs actually) would require redesign iteration; code correspondence problems would require DTLS & FTLS changes, Penetration Testing discovered flaws and covert channel analysis (CCA) leaks would spiral design back to affect [the] FTLS, etc. Like falling dominos, each forward step would require some retreat. It was primarily the A1 security requirements that forced these iterations, and the teams

[66] Author's note: Citation and enclosing square brackets in original. Included in this document's references as [Boe86].

would be stressed as management pushed for more progress on deliverables.

Q4. What done to combat?

A4. There were many questions raised as to the worth of all the formalism, i.e., A1 vs B3; even suggestions that an A0 class be invented that had formal specs but no proofs. But many design examples [were] experienced to show that the need to prove the specs forced "due diligence" on the whole process and on team members to eliminate errors and security flaws. Many subtle design [weaknesses] were first uncovered by the proof process. From the beginning, all players on both contractor and government sides were experienced, and the focus was on an A1 secure system. A rigorous and detailed plan was developed with adequate resources and time to do the job. We even built in 2.5 iterations [a la] the spiral model into all deliverables. For the programmers, classes were held on the formal process, the Ina Jo language, the FDM tool suite, etc. not to force them to write formal specs — we learned from Autodin II — but to enable them to read the formal specs and understand the design from a security perspective. This was a key to doing it right. The formal team was trained on the programming tools and methods. In fact a common development environment was mutually agreed upon to use SUN workstations and Unix servers for Configuration Management and documentation. To integrate the teams further, the formal team was required to do quality control on all the specs and design documentation, and later the unit code. Many hours of burned eyeballs were spent by the formal team reading DTLS & code and uncovering problems early. Further, the functional security testing was built into the integration testing of Computer System Configuration Items (CSCIs). Penetration Testing was led by the security team, but included major elements of the software development team. Documentation was extensive and written by all Blacker staff, and read and reviewed by them and by [the] government. Lastly, there were many unusual situations that arose during the five years of development that required going back to first security principles to arrive at solution. We found the contractor and the government cooperating at such times to find the best overall solution — sometimes [the] schedule slipped, sometimes security bent, sometimes [the]

design changed — in a trust engineering give and take.

Q5. How successful?

A5. Very. After 5 years of very detailed NCSC evaluation of the Blacker system and security evidence, Blacker was approved for A1 application, and has served the world as an example that it can be done. [See my paper "BLACKER: Security for the DDN, Examples of A1 Security Engineering Trades," Proc. IEEE Computer Society Symposium on Research in Security and Privacy," May 1992, pp. 286-292.][67]

[67] Author's note: Reference and enclosing square brackets in original. Included in this document's references as [Wei92].

References

[Ada79] J. Adams. Computer security environment considerations. Technical report, IBM Corporation, Arlington, Virginia, August 1979. Contract MDA 903-79-C-0311.

[AGS83] Stanley R. Ames, Jr., Morrie Gasser, and Roger R. Schell. Security kernel design and implementation: An introduction. *Computer*, page 14, 1983.

[And72] J. P. Anderson. Computer security technology planning study. Technical Report ESD-TR-73-51, Electronic Systems Division, Air Force Systems Command, Hanscom AFB, October 1972. Vol. I, AD-758206. Vol. II, AD-772806. Manifesto of security kernel approach to computer security. Computer security usage of "Trojan horse" introduced in Vol. II.

[Ano93a] NCSC staff member Greene, telephone conversation, May 3, 1993. Other party: Garrel Pottinger. "Greene" is a pseudonym.

[Ano93b] NCSC staff member Greenwood, telephone conversation, December 16, 1993. Other party: Garrel Pottinger. "Greenwood" is a pseudonym.

[Ano93c] NCSC staff member Greenwood, telephone conversation, May 3, 1993. Other party: Garrel Pottinger. "Greenwood" is a pseudonym.

[Ano93d] NCSC staff member Holmes, electronic mail, October 20, 1993. Recipient: Garrel Pottinger. "Holmes" is a pseudonym.

[Ano93e] White, telephone conversation, October 28, 1993. Other party: Garrel Pottinger. "White" is a pseudonym.

[Arm59] Department of the Army. OATH OF OFFICE - MILITARY PERSONNEL. Printed by U.S.G.P.O, 1981–341-646/8552, August 1 1959. DA FORM 71.

[Arm90] Department of the Army. Pamphlet 27-21, Update, September 18 1990.

[Bam82] James Bamford. *The Puzzle Palace*. Houghton Mifflin Company, 1982. British edition, *The Puzzle Palace: America's National Security Agency and its Special Relationship with Britain's GCHQ*, Sidgwick & Jackson, 1983, contains preface describing Britain's Government Communications Headquarters and giving account of Geoffrey Prime case. Only publically available general history of National Security Agency as of 1993.

[Bar80] John Bartlett. *Familiar Quotations: A Collection of Passages, Phrases and Proverbs Traced to Their Sources in Ancient and Modern Literature*. Little, Brown and Company, 1980. Fifteenth and 125th anniversary edition, revised and enlarged. Edited by Emily Morison Beck and the editorial staff of Little, Brown and Company.

[BDF+86] Peter C. Baker, George W. Dinolt, James W. Freeman, M. Krenzin, and Richard B. Neely. A1 assurance for an internet system: Doing the job. In *Proceedings of the 9th National Computer Security Conference*, pages 130–137. NBS/NCSC, 1986. Describes Multinet Gateway.

[Bel88] D. Elliott Bell. Concerning 'modeling' of computer security. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 8–13. IEEE Computer Society Press, 1988.

[Ben77] Robert L. Benson. An interview with the Agency's new Director. *National Security Agency Newsletter*, pages 4–5, September 1977. Interview with Vice Admiral B. R. Inman.

[BHP87] Wiebe E. Bijker, Thomas P. Huges, and Trevor J. Pinch, editors. *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology*. MIT Press, 1987.

[Bib77] K. J. Biba. Integrity considerations for secure computer systems. Technical Report ESD-TR-76-372, Electronic Systems Division, Air Force Systems Command, 1977. AD-A039324.

[Boe86] Barry Boehm. A spiral model of software development and enhancement. *Software Engineering Notes*, 11(4):14–24, 1986.

[Boe88] W. E. Boebert. Constructing an Infosec system using LOCK technology. Technical report, National Computer Security Center, October 1988.

[Bon93a] Carles H. Bonneau, telephone conversation, April 30, 1993. Other party: Garrel Pottinger.

[Bon93b] Interview with Charles H. Bonneau, November 20, 1993. Interviewer: Garrel Pottinger.

[BP74a] D. E. Bell and L. J. La Padula. Secure computer systems: Mathematical foundations and model. Technical Report M74-244, MITRE, October 1974. Paper presented at Far Western Conference of Society for General Systems, Sacramento, October 24–25, 1974. Essential source for information on how Bell/La Padula model was developed.

[BP74b] D. E. Bell and L. J. La Padula. Secure computer systems: Mathematical foundations; a mathematical model; a refinement of the mathematical model. Technical Report ESD-TR-73-278, Electronic Systems Division, Air Force Systems Command, November 1973–April 1974. In three volumes. Vols. I and II, November 1973. Vol. III, April 1974. D. E. Bell sole author of Vol. III. Vol. I, AD-770768. Vol. II, AD-771543. Vol. III, AD-780528. Also published as MITRE Technical Report MTR-2547. MTR-2547, Vol. I, March 1973. MTR-2547, Vol. II, May 1973. MTR-2547, Vol. III, December 1973. Simple security property only in Vol. I. Vol. II introduced *-property. Vol. III introduced trusted subjects.

[BP75] D. E. Bell and L. J. La Padula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report MTR-2997, MITRE, July 1975. AD-A020445. Verso of p. 69 blank. Preliminary version of security model intended for use in design of unbuilt Project Guardian version of Multics including security kernel.

[BP76] D. E. Bell and L. J. La Padula. Secure computer systems: Unified exposition and Multics interpretation. Technical Report ESD-TR-75-306, ESD/AFSC, Hanscom AFB, 1976. Revision 1 of MITRE Technical Report MTR-2997. AD-A023588. Security

model intended for use in design of unbuilt Project Guardian version of Multics including security kernel.

[Bro85] Congressman Jack Brooks. Statement of on National Security Decision Directive 145 before the Subcommittee on Transportation, Aviation and Materials, Committee on Science and Technology, June 27, 1985. Reprinted in United States House of Representatives, *Computer Security Act of 1987: Hearings before a Subcommittee of the Committee on Government Operations, House of Representatives, 100th Congress, First Session, on H. R. 145 ..., February 25, 26, and March 17, 1987*, pp. 524–527.

[Car78] Jim Carlstedt. Protection errors in operating systems: A selected annotated bibliography and index to terminology. Technical Report ISI/SR-78-10, University of Southern California Information Sciences Institute, 1978. AD-A053016.

[Car87a] Frank C. Carlucci, Assistant to the President for National Security Affairs. Letter to Congressman Jack Brooks, March 12, 1987. Printed in United States House of Representatives, *Computer Security Act of 1987: Hearings before a Subcommittee of the Committee on Government Operations, House of Representatives, 100th Congress, First Session, on H. R. 145 ..., February 25, 26, and March 17, 1987*, 1987, p. 386.

[Car87b] Frank C. Carlucci, Assistant to the President for National Security Affairs. Letter to Congressman Jack Brooks, March 17, 1987. Printed in United States House of Representatives, *Computer Security Act of 1987: Hearings before a Subcommittee of the Committee on Government Operations, House of Representatives, 100th Congress, First Session, on H. R. 145 ..., February 25, 26, and March 17, 1987*, 1987, pp. 388–389.

[CGR93a] Dan Craigen, Susan Gerhart, and Ted Ralston. An international survey of industrial applications of formal methods, volume 1: Purpose, approach, analysis, and conclusions. Technical report, Computer Systems Laboratory, National Institute of Standards and Technology, March 1993.

[CGR93b] Dan Craigen, Susan Gerhart, and Ted Ralston. An international survey of industrial applications of formal methods, volume 2:

Case studies. Technical report, Computer Systems Laboratory, National Institute of Standards and Technology, March 1993.

[Con80a] United States Congress. The Paperwork Reduction Act of 1980, December 11, 1980. P. L. 96-511.

[Con80b] Edward W. Constant II. *The Origins of the Turbojet Revolution*. Johns Hopkins University Press, 1980.

[Con88] United States Congress. The Computer Security Act of 1987, January 8, 1988. P. L. 100-235.

[Cou91] National Research Council. *Computers at Risk: Safe Computing in the Information Age*. National Academy Press, 1991.

[Cri91] Michael Crichton. *Jurassic Park*. Random House, 1991. Arrow edition. Shows awareness of trap doors has seeped into popular culture. See pp. 175 and 230.

[CSE82] DoD Computer Security Evaluation Center. *Trusted Computer System Evaluation Criteria, 1st Draft*, May 24, 1982. Circulated for comment. Third stage in development of Orange Book criteria, following Lee report drafted in 1978 and October 1979 Nibaldi report.

[CSE83a] DoD Computer Security Evaluation Center. Product evaluation bulletin on the Secure Communications Processor (SCOMP) of Honeywell Information Systems, Inc., April 15, 1983. CSC-PB-01-83.

[CSE83b] DoD Computer Security Evaluation Center. *Trusted Computer System Evaluation Criteria*, January 25, 1983. Final draft. Green cover. Fourth stage in development of Orange Book criteria. Second draft of Orange Book produced by DoD CSEC.

[CSE83c] DoD Computer Security Evaluation Center. *Trusted Computer System Evaluation Criteria*, August 15, 1983. CSC-STD-001-83. Orange cover. Fifth stage in development of Orange Book criteria. Third draft of Orange Book produced by produced by DoD CSEC. First version of Orange Book issued for use in evaluations.

[CSE85a] DoD Computer Security Evaluation Center. *Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments*, June 25, 1985. CSC-STD-003-85. Yellow cover.

[CSE85b] DoD Computer Security Evaluation Center. *Password Management Guidelines*, April 12, 1985. CSC-STD-002-85. Green cover.

[CSE85c] DoD Computer Security Evaluation Center. *Technical Rationale Behind CSC-STD-003-85: Computer Security Requirements*, June 25, 1985. CSC-STD-004-85. Yellow cover.

[CSS89] Canadian System Security Centre. *Canadian Trusted Computer Product Evaluation Criteria*, May 1989. Draft.

[Dan90a] Richard A. Danca. Bush revises NSDD 145. *Federal Computer Week*, page 6, July 16, 1990.

[Dan90b] Richard A. Danca. NCSC affirms shakeup in its structure. *Federal Computer Week*, page 1, August 27, 1990.

[Dan90c] Richard A. Danca. NCSC decimated, security role weakened. *Federal Computer Week*, page 1, July 16, 1990.

[Dij69] Edsger W. Dijkstra. Complexity controlled by hierarchical ordering of function and variability. In Peter Naur and Brian Randell, editors, *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*, pages 181–185. NATO Scientific Affairs Division, 1969.

[Dij82] Edsger W. Dijkstra. *Selected Writings on Computing: A Personal Perspective*. Springer-Verlag, 1982.

[DoD82a] Information Security Program Regulation, August 1982. Directive 5200.1-R.

[DoD82b] United States Department of Defense. Computer Security Evaluation Center, October 25, 1982. Directive 5215.1.

[DS79] J. B. DeWolf and P. A. Szulewski, editors. *Final Report of the 1979 Summer Study on Air Force Computer Security*, Cambridge, Massachusetts, October 1979. The Charles Stark Draper Laboratory, Inc.

[EC91] *Information Technology Security Evaluation Criteria (ITSEC)*, June 1991. Harmonised Criteria of France, Germany, the Netherlands, and the United Kingdom, Version 1.2. Printed and published by the Department of Trade and Industry, London.

[ESD74] ESD 1974 computer security development summary. Interim report MCI-75-1, United States Air Force, Electronic Systems Division, Air Force Systems Command, December 31, 1974.

[Fau81] L. D. Faurer. Keeping the secrets secret. *Government Data Systems*, pages 14–17, November–December 1981.

[FC71] R. M. Fano and F. J. Corbató. Time-sharing on computers. In Robert R. Fenichel and Joseph Weizenbaum, editors, *Computers and Computation*, pages 78–87. W. H. Freeman and Company, 1971. Readings from *Scientific American*.

[FH93] Jon Fellows and Judy Hemenway, electronic mail, December 21, 1993. Recipient: Garrel Pottinger.

[Flo67] R. W. Floyd. Assigning meanings to programs. In *Proceedings of a Symposium on Applied Mathematics (Mathematical Aspects of Computer Science)*, pages 19–32. American Mathematical Society, 1967.

[Fra83] L. J. Fraim. Scomp: A solution to the multilevel security problem. *IEEE Computer*, 16(7):26–34, July 1983.

[FW71] Robert R. Fenichel and Joseph Weizenbaum, editors. *Computers and Computation*. W. H. Freeman and Company, 1971. Readings from *Scientific American*.

[GM82] Joseph A. Goguen and José Meseguer. Security policies and security models. In *Proceedings of the 1982 Berkeley Conference on Computer Security*, pages 11–22. IEEE Computer Society Press, 1982.

[Goo91a] Interview with Donald I. Good, May 16, 1991. Interviewer: Eloína Peláez.

[Goo91b] Interview with Donald I. Good, November 12, 1991. Interviewer: Margaret Tierney.

[Goo93] Interview with Donald I. Good, March 23, 1993. Interviewer: Garrel Pottinger.

[Gor85] M. Gordon. HOL: A machine oriented formulation of higher-order logic. Technical Report 68, University of Cambridge Computer Laboratory, July 1985. Revised version.

[GvN47] Herman H. Goldstine and John von Neumann. Planning and coding of problems for an electronic computing instrument, part ii, volume 1. Technical report, The Institute for Advanced Study, April 1947. Reprinted in A. H. Taub (ed.), *John von Neumann Collected Works: Volume V, Design of Computers, Theory of Automata, and Numerical Analysis*, Pergamon Press, 1961.

[H+92] Paul Hudak et al. Report on the programming language Haskell: A nonstrict, purely functional language version 1.2. *SIGPLAN Notices*, 27, May 1992.

[Hai93] Interview with J. Thomas Haigh, March 12, 1993. Interviewer: Garrel Pottinger.

[Har93a] Bret Hartman, telephone conversation, April 29, 1993. Other party: Garrel Pottinger.

[Har93b] Bret Hartman, telephone conversation, October 11, 1993. Other party: Garrel Pottinger.

[Har93c] Interview with Bret Hartman, February 2, 1993. Interviewer: Garrel Pottinger.

[Hen82] Peter Henderson. Purely functional operating systems. In John Darlington, Peter Henderson, and David A. Turner, editors, *Functional Programming and its Applications*, pages 177–189. Cambridge University Press, 1982.

[HF92] Paul Hudak and Joseph H. Fasel. A gentle introduction to Haskell. *SIGPLAN Notices*, 27, May 1992.

[HKMY86] J. Thomas Haigh, Richard A. Kemmerer, John Mchugh, and William D. Young. An experience using two covert channel analysis techniques on a real system design. In *Proccedings of the 1986 IEEE Symposium on Security and Privacy*, pages 14–24. IEEE Computer Society Press, 1986.

[Hoa69] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–583, 1969.

[HOLa] Cambridge Research Center of SRI International. *The HOL System Description.* Distributed with version 1.12 of the HOL system.

[HOLb] Cambridge Research Center of SRI International. *The HOL System Reference Manual.* Distributed with version 1.12 of the HOL system.

[HOLc] Cambridge Research Center of SRI International. *The HOL System Tutorial.* Distributed with version 1.12 of the HOL system.

[Hug83] Thomas P. Huges. *Networks of Power: Electrification in Western Society, 1880–1930.* Johns Hopkins University Press, 1983.

[Hug87] Thomas P. Huges. The evolution of large technological systems. In Wiebe E. Bijker, Thomas P. Huges, and Trevor J. Pinch, editors, *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology*, pages 51–82. MIT Press, 1987.

[HY86] J. Thomas Haigh and William D. Young. Extending the noninterference version of MLS for SAT. In *Proccedings of the 1986 IEEE Symposium on Security and Privacy*, pages 232–239. IEEE Computer Society Press, 1986.

[Inm80] B. R. Inman. Managing intelligence for effective use. In *Seminar on Command, Control, Communications and Intelligence.* Center for Information Policy Research, Program on Information Resources Policy, Harvard, 1980.

[ITT78] ITT DCD. System specification for SAC Digital Network (SACDIN), 1978. ESD-MCV-1A.

[Jel85] George F. Jelen. *Information Security: An Elusive Goal.* Program on Information Resources Policy, Harvard, June 1985. Stephen T. Walker's draft charter for Federal computer security evaluation center located at NBS printed on pp. V-2–V-9. January 2, 1981 memorandum from Deputy Secretary of Defense W. Graham Claytor, Jr. containing notification of decision to establish DoD Computer Security Evaluation Center at NSA printed on p. V-10. DoD Directive 5215.1 and enclosures reprinted on pp. V-11–V-17. Unclassified version of NSDD 145 reprinted on pp. V-18–V-27.

[Kuh62] Thomas S. Kuhn. *The Structure of Scientific Revolutions.* University of Chicago Press, 1962.

[KZB+90] Paul A. Karger, Mary Ellen Zurko, Douglas W. Bonin, Andrew H. Mason, and Clifford E. Kahn. A VMM security kernel for the VAX architecture. In *1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 2–19. IEEE Computer Society Press, 1990. Describes DEC prototype intended for evaluation at A1. Prototype called VAX SVS (Secure Virtual System) at DEC; called SKVAX by NCSC, but not refered to this way at DEC.

[L+80] T. M. P. Lee et al. Processors, operating systems, and nearby peripherals. In Z. Ruthberg, editor, *Audit and Evaluation of Computer Security II: System Vulnerabilities and Controls*, pages 8–1–8–28. NBS, April 1980. Special publication # 500-57. MD78733. Proceedings of an invitational workshop held November 1978. Manifesto of DoD Computer Security Initiative. Includes first stage in development of Orange Book criteria.

[Lam73] B. W. Lampson. A note on the confinement problem. *Communications of the ACM*, 16(10):613–615, October 1973.

[Lan81] Carl E. Landwehr. Formal models for computer security. *ACM Computing Surveys*, 13, 1981.

[Lan83] Carl E. Landwehr. The best available technologies for computer security. *IEEE Computer*, 16:86–100, July 1983. Tables 1 and

2 and Appendix A summarize base of experience with secure systems available to guide choice of Orange Book proof requirements. Reprinted in Rein Turn (ed.), *Advances in Computer System Security*, Volume II, Artech House, 1984, pp. 76-107.

[Law87] John Law. Technology and heterogeneous engineering: The case of Portuguese expansion. In Wiebe E. Bijker, Thomas P. Huges, and Trevor J. Pinch, editors, *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology*, pages 111–134. MIT Press, 1987.

[LBMC93] Carl E. Landwehr, Alan R. Bull, John P. McDermott, and William S. Choi. A taxonomy of computer program security flaws, with examples. Technical Report NRL/5542–93-9591, Center for Computer High Assurance Systems, Information Technology Division, Naval Research Laboratory, November 19, 1993.

[Lew78] Ronald Lewin. *ULTRA Goes to War.* Pocket Books, 1978. First published by Hutchinson & Co., 1978. Provides valuable illustration of importance of need to know principle.

[LHM84] Carl E. Landwehr, Constance L. Heitmeyer, and John McLean. A security model for military message systems. *ACM Transactions on Computer Systems*, 2:198–222, 1984.

[Lip93] Stephen Lipner, electronic mail, October 22, 1993. Recipient: Garrel Pottinger.

[LPS89] Timothy E. Levin, Stephen J. Padilla, and Roger R. Schell. Engineering results from the A1 formal verification process. In *Proceedings of the 12th National Computer Security Conference*, pages 65–74. Defense Technical Information Center, 1989.

[LTP90] Timothy E. Levin, Albert Tao, and Stephen J. Padilla. Covert storage channel analysis: A worked example. In *Proceedings of the 13th National Computer Security Conference*, pages 10–19. Defense Technical Information Center, 1990.

[Mac87] Donald Mackenzie. Missle accuracy: A case study in the social processes of technological change. In Wiebe E. Bijker,

Thomas P. Huges, and Trevor J. Pinch, editors, *The Social Construction of Technological Systems: New Directions in the Sociology and History of Technology*, pages 195–222. MIT Press, 1987.

[Mac90] Donald MacKenzie. *Inventing Accuracy: A Historical Sociology of Nuclear Missle Guidance*. MIT Press, 1990.

[Mac91] Donald MacKenzie. The fangs of the VIPER. *Nature*, 352:467–469, 1991.

[McC87] Daryl McCullough. Specifications for multilevel security and a hook-up property. In *Proceedings of the 1987 Symposium on Security and Privacy*, April 1987.

[McC88] Daryl McCulough. Foundations of Ulysses: The theory of security. Technical Report RADC-TR-87-222, Rome Air Development Center, July 1988. Interim report.

[McC93] Interview with Daryl McCullough, February 3, 1993. Interviewer: Garrel Pottinger.

[McL87] John McLean. Reasoning about security models. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 123–131. IEEE Computer Society Press, 1987.

[Men79] Elliott Mendelson. *Introduction to Mathematical Logic*. D. Van Nostrand Company, 1979. Second edition.

[Mok90] Joel Mokyr. *The Lever of Riches: Technological Creativity and Economic Progress*. Oxford University Press, 1990.

[NCS85a] National Computer Security Center. *Final Evaluation Report, Secure Communications Processor (SCOMP), STOP Release 2.1*, 1985. CSC-EPL-85/001.

[NCS85b] National Computer Security Center. *Trusted Computer System Evaluation Criteria*, December 26, 1985. DoD 5200.28-STD. The Orange Book. Supersedes CSC-STD-001-83. Orange cover. Sixth, and final, stage in criteria development, preceded by criteria of Lee report drafted in 1978, October 1979 Nibaldi report, DoD CSEC Orange Book drafts of May 24, 1982 and January

25, 1983, and CSC-STD-001, issued August 15, 1983 by DoD CSEC for use in evaluations. Differences between Orange Book and CSC-STD-001-83 are minor.

[NCS86] DoD STANDARD 5200.28: SUMMARY OF THE DIFFERENCES BETWEEN IT AND CSC-STD-001-83, 1986. Enclosure accompanying Sheila L. Brand, DoD 5200.28-STD, Summary of Changes, memorandum for the record, July 3, 1986. Orange Book was other enclosure.

[NCS87a] National Computer Security Center. *A Guide to Understanding Audit in Trusted Systems*, July 28, 1987. NCSC-TG-001. Tan cover.

[NCS87b] National Computer Security Center. *A Guide to Understanding Discretionary Access Control in Trusted Systems*, September 30, 1987. NCSC-TG-003. Orange cover.

[NCS87c] National Computer Security Center. *Trusted Network Interpretation*, July 31, 1987. NCSC-TG-005, Version 1. Red cover.

[NCS88a] National Computer Security Center. *Computer Security Subsystem Interpretation of the Trusted Computer System Evaluation Criteria*, September 16, 1988. NCSC-TG-009. Venice blue cover.

[NCS88b] National Computer Security Center. *Glossary of Computer Security Terms*, October 21, 1988. NCSC-TG-004, Version 1. Aqua cover.

[NCS88c] National Computer Security Center. *A Guide to Understanding Configuration Management in Trusted Systems*, March 28, 1988. NCSC-TG-006, Version 1. Orange cover.

[NCS88d] National Computer Security Center. *A Guide to Understanding Design Documentation in Trusted Systems*, October 6, 1988. NCSC-TG-007, Version 1. Burgundy cover.

[NCS88e] National Computer Security Center. *A Guide to Understanding Trusted Distribution in Trusted Systems*, December 15, 1988. NCSC-TG-008. Lavender cover.

[NCS89a] National Computer Security Center. *A Guide to Understanding Trusted Facility Management*, October 18, 1989. NCSC-TG-015, Version 1. Brown cover.

[NCS89b] National Computer Security Center. *Guidelines for Formal Verification Systems*, April 1, 1989. NCSC-TG-014. Purple cover.

[NCS89c] National Computer Security Center. *Rating Maintenance Phase Program Document*, June 23, 1989. NCSC-TG-013, Version 1. Hot pink cover.

[NCS89d] National Computer Security Center. *Trusted UNIX Working Group (TRUSIX) Rationale for Selecting Access Control List Features for the UNIX* System*, August 18, 1989. NCSC-TG-020A, Version 1. Gray cover.

[NCS90a] National Computer Security Center. *Trusted Network Interpretation Environments Guideline*, August 1, 1990. NCSC-TG-011, Version 1. Red cover.

[NCS90b] National Computer Security Center. *Trusted Product Evaluations — Guide for Vendors*, June 22, 1990. NCSC-TG-002, Version 1. Bright blue cover.

[NCS91a] National Computer Security Center. *Final Evaluation Report, Boeing Space and Defense Group, MSL LAN Secure Network Server System*, 1991. CSC-EPL-91/005.

[NCS91b] National Computer Security Center. *A Guide to Understanding Data Remanence in Automated Information Systems*, September, 1991. NCSC-TG-025, Version 2. Green cover.

[NCS91c] National Computer Security Center. *A Guide to Understanding Identification and Authentication in Trusted Systems*, September 1, 1991. NCSC-TG-017, Version 1. Light blue cover.

[NCS91d] National Computer Security Center. *A Guide to Understanding Trusted Recovery in Trusted Systems*, December 30, 1991. NCSC-TG-022. Yellow cover.

[NCS91e] National Computer Security Center. *A Guide to Writing the Security Features User's Guide for Trusted Systems*, September, 1991. NCSC-TG-026, Version 1. Hot peach cover.

[NCS91f] National Computer Security Center. *Trusted Database Management System Interpretation*, April, 1991. NCSC-TG-021, Version 1. Lavender cover.

[NCS92a] National Computer Security Center. *Assessing Controlled Access Protection*, May, 1992. NCSC-TG-028, Version 1. Violet cover.

[NCS92b] National Computer Security Center. *A Guide to Procurement of Trusted Systems: An Introduction to Procurement Initiators on Computer Security Requirements*, December, 1992. NCSC-TG-024. Purple cover.

[NCS92c] National Computer Security Center. *A Guide to Understanding Information System Security Officer Responsibilities for Automated Information Systems*, May, 1992. NCSC-TG-027, Version 1. Turquoise cover.

[NCS92d] National Computer Security Center. *A Guide to Understanding Object Reuse in Trusted Systems*, July, 1992. NCSC-TG-018. Light blue cover.

[NCS92e] National Computer Security Center. *A Guide to Understanding Security Modeling in Trusted Systems*, October, 1992. NCSC-TG-010. Aqua cover.

[NCS92f] National Computer Security Center. *Guidelines for Writing Trusted Facility Manuals*, October, 1992. NCSC-TG-016, Version 1. Yellow-green cover.

[NCS92g] National Computer Security Center. *Trusted Product Evaluation Questionnaire*, May 2, 1992. NCSC-TG-019, Version 2. Blue cover.

[Nib79a] G. H. Nibaldi. Proposed technical evaluation criteria for trusted computer systems. Technical Report M79-225, MITRE, October 25, 1979. AD-A108-832. Second stage in development of Orange Book criteria, following Lee report drafted in 1978.

[Nib79b] G. H. Nibaldi. Specification of a trusted computing base (TCB). Technical Report M79-228, MITRE, November 30, 1979. AD-A108-831. Apparent source of phrase "trusted computing base" and its definition.

[NIS92a] National Institute of Standards and Technology and National Security Agency. *Federal Criteria for Information Technology Security, Volume I: Protection Profile Development*, December 1992. Version 1.0. Circulated for comment.

[NIS92b] National Institute of Standards and Technology and National Security Agency. *Federal Criteria for Information Technology Security, Volume II: Registry of Protection Profiles*, December 1992. Version 1.0. Circulated for comment.

[NIS93] National Institute of Standards and Technology and National Security Agency. *Proceedings of the 16th National Computer Security Conference*, September 20–23, 1993.

[NMT87] National Manager for Telecommunications and Automated Information Systems Security. Advisory Memorandum on Office Automation Security Guideline, January 16, 1987. NTISSAM COMPUSEC/1-87. White document.

[NR69] Peter Naur and Brian Randell, editors. *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968*. NATO Scientific Affairs Division, 1969. The "software crisis" conference.

[NS72] Alan Newell and Herbert A. Simon. *Human Problem Solving*. Prentice-Hall, 1972.

[NSA77] Vice Admiral B. R. Inman becomes Agency Director. *National Security Agency Newsletter*, July 1977. Page 2.

[NSA86] National Security Agency. NSA Information Security Reorganization, April 24, 1986. Announced first stage in dismantling evaluation center structure worked out by Walker and Inman. Printed in United State House of Representatives, *Computer Security Act of 1987: Hearings before a Subcommittee of the Committee on Government Operations, House of Representatives, 100th Congress, First Session, on H. R. 145 ..., February 25, 26, and March 17, 1987*, 1987, p. 548.

[NSA93a] National Security Agency. *Information Systems Security Products and Services Catalogue*, January 1993. See pp. 4-1–4-199 for the Evaluated Products List.

[NSA93b] National Security Agency INFOSEC Awareness Division. COMPUSEC DOCUMENTS, April 1993.

[NSD84] National Policy on Telecommunications and Automated Information Systems Security, September 17, 1984. National Security Decision Directive 145 (NSDD 145). 10 pp., unclassified in part. Unclassified version reprinted in United State House of Representatives, *Computer Security Act of 1987: Hearings before a Subcommittee of the Committee on Government Operations, House of Representatives, 100th Congress, First Session, on H. R. 145 ..., February 25, 26, and March 17, 1987*, 1987, pp. 528–537.

[Org72] O. Organick. *The Multics System: An Examination of its Structure*. MIT Press, 1972.

[Par72] D. L. Parnas. A technique for software module specification with examples. *Communications of the ACM*, 15(5):330–336, May 1972. A fundamental source for ideas involved in design verification.

[Par85] D. L. Parnas. Software aspects of strategic defense systems. *American Scientist*, 73(5):432–440, 1985.

[PB72] L. J. La Padula and D. E. Bell. Harmonious cooperation of processes operating on a common set of data. Technical Report MTR-2254, MITRE, February 1972–May 1972. In three volumes. Vol. I, February 1972. Vol. II, March 1972. Vol. III, May 1972. L. J. La Padula sole author of Vol. I. D. E. Bell sole author of Vol. II. Vol. III jointly authored. Vol. I, AD-757902. Vol. II, AD-757903. Vo.l III, AD-757904.

[PD77] Presidential Directive/NSC 24 (PD 24). Telecommunications Protection Policy, November 16, 1977. 5 pp., unclassified in part. Unclassified version issued February 9, 1979.

[Pet92] H. Petroski. *To Engineer is Human: The Role of Failure in Successful Design*. Vintage Books, 1992.

[Pla93] Interview with Richard Platek, February 4, 1993. Interviewer: Garrel Pottinger.

[Pot93a] Garrel Pottinger, electronic mail, December 10, 1993. Recipient: Clark Weissman.

[Pot93b] Garrel Pottinger, electronic mail, October 21, 1993. Recipient: Stephen Lipner.

[Pot93c] Garrel Pottinger, facsimile, November 17, 1993. Recipient: Ham, NSA Public Affairs Office, Information and Policy Branch. "Ham" is a pseudonym.

[Pot93d] Garrel Pottinger, telephone conversation, November 16, 1993. Other party: Ham, NSA Public Affairs Office, Information and Policy Branch. "Ham" is a pseudonym.

[Qui91] P. Quintas. Engineering solutions to software problems: Some institutional and social factors shaping change. In *Technology Analysis & Strategic Management*, 1991.

[Rep87a] United States House of Representatives. *The Computer Security Act of 1987: Hearing before the Subcommittee on Science, Research, and Technology and the Subcommittee on Transportation, Aviation, and Materials of the Committee on Science, Space, and Technology, House of Representatives, 100th Congress, First Session, February 26, 1987.* U.S.G.P.O., 1987. NTISSP 2, the Poindexter memorandum, reprinted on pp. 37–40.

[Rep87b] United States House of Representatives. *Computer Security Act of 1987: Hearings before a Subcommittee of the Committee on Government Operations, House of Representatives, 100th Congress, First Session, on H. R. 145 ..., February 25, 26, and March 17, 1987.* U.S.G.P.O., 1987.

[Rep89] United States House of Representatives. *Implementation of the Computer Security Act: Hearing before the Subcommittee on Transportation, Aviation, and Materials of the Committee on Science, Space, and Technology, House of Representatives, 100th Congress, Second Session, September 22, 1988.* U.S.G.P.O., 1989.

[Rep90] United States House of Representatives. *Implementation of the Computer Security Act (Public Law 100-235): Hearing before the Subcommittee on Transportation, Aviation and Materials of the Committee on Science, Space, and Technology, United States House of Representatives, 101st Congress, Second Session, July 10, 1990.* U.S.G.P.O, 1990.

[Rep91] United States House of Representatives. *Computer Security: Hearing before the Subcommittee on Technology and Competitiveness of the Committee on Science, Space, and Technology, United States House of Representatives, 102nd Congress, First Session, June 27, 1991.* U.S.G.P.O, 1991.

[Rep92] United States House of Representatives. *Computer Security Act of 1987.* U.S.G.P.O., 1992. Report prepared by the Subcommittee on Technology and Competitiveness, transmitted to the Committee on Science, Space, and Technology, House of Representatives, 102nd Congress, Second Session, July 1992.

[RM77] Z. Ruthberg and R. McKenzie, editors. *Audit and Evaluation of Computer Security.* National Bureau of Standards, NBS Special Publication #500-19, October 1977. Proceedings of NBS invitational workshop on audit and evaluation of computer security, Miami Beach, March 22–24, 1977.

[Rut80] Z. Ruthberg, editor. *Audit and Evaluation of Computer Security II.* National Bureau of Standards, NBS, 1980. Proceedings of NBS invitational workshop, Miami Beach, November 28–30, 1978.

[Scha] Roger R. Schell. Amplified resume. Received October 1993.

[Schb] Roger R. Schell. Biography. Received October 1993.

[Sch72] Roger R. Schell. Notes on an approach for design of secure military ADP systems. In *Proceedings of the ACM Annual Conference*, pages 665–666. ACM, August 1972.

[Sch73] W. L. Schiller. Design of a security kernel for the PDP-11/45. Technical Report MTR-2709, MITRE, June 30, 1973. Describes preliminary design for MITRE brassboard security kernel.

[Sch75] W. L. Schiller. Design of a security kernel for the PDP-11/45. Technical Report ESD-TR-75-69, Electronic Systems Division, Air Force Systems Command, May 1975. A revision of MITRE Technical Report MTR-2709. Also published as MITRE Technical Report MTR-2934. Describes design used in building MITRE brassboard security kernel.

[Sch77] W. L. Schiller. Design and abstract specification of a Multics security kernel. Technical Report ESD-TR-77-259, MITRE, November 1977. AD-A048576. Describes design for Project Guardian version of Multics. Would have included security kernel, but was never built.

[Sch79] Roger R. Schell. Computer security: The Achilles' heel of the electronic Air Force? *Air University Review*, 30:22–24, 1979.

[Sch85] D. Schnackenberg. Development of a multilevel secure local area network. In *Proceedings of the 8th National Computer Security Conference*. NBS/DoD CSEC, 1985. Describes Boeing A1 LAN.

[Sch89] Marvin Schaefer. Symbol security condition considered harmful. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, pages 20–46. IEEE Computer Society Press, 1989. Good source on reasons for doubting efficacy of design verification.

[Sch92] Roger R. Schell, electronic mail, August 18, 1992. Recipient: Steve Padilla. Forwarded to Garrel Pottinger by Schell, December 9, 1993.

[Sch93a] Interview with Marvin Schaefer, March 23, 1993. Interviewer: Garrel Pottinger.

[Sch93b] Interview with Roger R. Schell, October 10, 1993. Interviewer: Garrel Pottinger.

[Sha90] Stuart S. Shapiro. *Computer Software as Technology: An Examination of Technological Development*. PhD thesis, Carnegie Mellon University, 1990.

[Sim81] Herbert A. Simon. *The Sciences of the Artificial*. MIT Press, 1981. Second edition.

[SPD90] Summary of the National Policy for the Security of National Security Telecommunications and Information Systems, July 5, 1990. Summary of National Security Policy Directive revising policy stated in NSDD 145 so as to remove conflicts with the Computer Security Act of 1987. Second stage in dismantling evaluation center structure worked out by Walker and Inman was contemporaneous with issuance of directive summarized in this document. Printed in United States House of Representatives, *Implementation of the Computer Security Act (Public Law 100-235): Hearing before the Subcommittee on Transportation, Aviation and Materials of the Committee on Science, Space, and Technology, United States House of Representatives, 101st Congress, Second Session, July 10, 1990*, 1990, pp. 69-71.

[SSS86] Systems Security Steering Group. National Policy on Protection of Sensitive, but Unclassified Information in Federal Government Telecommunications and Automated Information Systems, October 29, 1986. National Telecommunications and Information Systems Security Policy 2 (NTISSP 2). The Poindexter memorandum. Reprinted in United States House of Representatives, *The Computer Security Act of 1987: Hearing before the Subcommittee on Science, Research, and Technology and the Subcommittee on Transportation, Aviation, and Materials of the Committee on Science, Space, and Technology, House of Representatives, 100th Congress, First Session, February 26, 1987*, 1987, pp. 37–40.

[STH85] Roger R. Schell, T. Tao, and Mark Heckman. Designing the GEMSOS security kernel for security and performance. In *Proceedings of the 8th National Computer Security Conference*. NBS/DoD CSEC, 1985.

[Sto86] William Stoye. Message-based functional operating systems. *The Science of Computer Programming*, 6:291–311, 1986.

[Sut86] David Sutherland. A model of information. In *Proceedings of the 9th National Computer Security Conference*, pages 175–183. NBS/NCSC, 1986. Author now known as Ian Sutherland.

[Ten91] Robert D. Tennet. Possible-world semantics of Algol-like languages. In M. Okada and P. J. Scott, editors, *MWPLT 91*, pages

84–92. Centre for Pattern Recognition and Machine Intelligence, Concordia University, 1991.

[Tie92] Margaret Tierney. Software engineering standards: The 'formal methods debate' in the UK. *Technology Analysis & Strategic Management*, 4:245–278, 1992.

[Tur49] Alan M. Turing. Checking a large routine. In *Report on a Conference on High Speed Automatic Calculating Machines*. University Mathematics Laboratory, Cambridge, 1949.

[Tur85] David Turner. Miranda: A non-strict functional language with polymorphic types. In Darlington et al., editors, *Proceedings of the IFIP International Conference on Functional Programming and its Application*. Springer, 1985. LNCS 201.

[Tur86] David A. Turner. An overview of Miranda. *SIGPLAN Notices*, 21:158–166, 1986.

[Tur87a] David Turner. An introduction to Miranda. In *The Implementation of Functional Programming Languages*, pages 431–438. Prentice-Hall, 1987.

[Tur87b] David A. Turner. Functional programming and communicating processes. In J. W. de Bakker, A. J. Nijman, and P. C. Treleaven, editors, *Proceedings PARLE, Parallel Architectures and Languages Europe*, volume 259 of *LNCS*, pages 54–74. Springer-Verlag, 1987.

[Tur90] David A. Turner. *Research Topics in Functional Programming*. Addison Wesley, 1990. University of Texas Year of Programming Series.

[Vin90] Walter G. Vincenti. *What Engineers Know and How They Know It*. Johns Hopkins University Press, 1990.

[W+73] J. Whitmore et al. Design for MULTICS security enhancements. Technical Report ESD-TR-74-176, Honeywell Information Systems, Inc., December 1973. Describes design for version of Multics installed in 1974 at Air Force Data Services Center in Pentagon (AFDSC Multics). Included security enhancements called Access Isolation Mechanism (AIM), but no security kernel.

[Wal80] Stephen T. Walker. The advent of trusted computer operating systems. In *National Computer Conference Proceedings*, pages 655–665. AFIPS Press, May 1980. Important source for information on body of experience that established viability of security kernel approach to computer security.

[Wal82] Stephen T. Walker. Department of Defense Data Network. *Signal*, October 1982.

[Wal93a] Interview with Stephen T. Walker, March 24, 1993. Interviewer: Garrel Pottinger.

[Wal93b] Stephen T. Walker, electronic mail, November 5, 1993. Recipient: Garrel Pottinger.

[War70] W. H. Ware, editor. *Security Controls for Computer Systems: Report of Defense Science Board Task Force on Computer Security.* Rand Corporation, 1970. AD-A076617/0. Reissued October 1979. Provided initial characterization of classical computer security problem. Led to general DoD recognition of problem.

[Wei69] Clark Weissman. Security controls in the ADEPT-50 time sharing system. In *Proceedings of the 1969 AFIPS Fall Joint Computer Conference*, pages 119–133. AFIPS Press, 1969.

[Wei73] Clark Weissman. System security analysis/certification methodology and results. Technical Report SP-3728, System Development Corporation, October 1973. Very influential report on penetration testing. Basis for Orange Book penetration testing requirements.

[Wei92] Clark Weissman. BLACKER: Security for the DDN, examples of A1 security engineering trades. In *Proceedings of the 1992 IEEE Symposium on Research in Security and Privacy*, pages 286–292. IEEE Computer Society Press, 1992.

[Wei93] Clark Weissman, electronic mail, December 18, 1993. Recipient: Garrel Pottinger.

[Wir71] Niklaus Wirth. Program development by stepwise refinement. *Communications of the ACM*, 14:221–227, 1971.

[WOG+75] K. G. Walter, W. F. Ogden, J. M. Gilligan, D. D. Schaeffer, S. I. Schaen, and D. G. Shumway. Initial structured specifications for an uncompromisable computer security system. Technical Report ESD-TR-75-82, Electronic Systems Division, Air Force Systems Command, 1975. AD-A022490. A design document for AFDSC Multics.

[WOR+74] K. G. Walter, W. F. Ogden, W. C. Rounds, , F. T. Bradshaw, S. R. Ames, and D. G. Shumway. Primitive models for computer security. Technical Report ESD-TR-4-117, Electronic Systems Division, Air Force Systems Command, 1974. AD-778467. Security model used in design of AFDSC Multics.

[WSO+75] K. G. Walter, S. I. Schaen, W. F. Ogden, W. C. Rounds, D. G. Shumway, D. D. Schaeffer, K. J. Biba, F. T. Bradshaw, S. R. Ames, and J. M. Gilligan. Structured specification of a security kernel. *SIGPLAN Notices*, 10:285–293, 1975.