University of Cambridge

Computer Laboratory

Essays about Computer Security

Prof. E. Stewart Lee
Director
Centre for Communications Systems Research
Cambridge

Preface

The purpose of these essays is to present some of the material that is the basis of security in computer systems. An attempt has been made to include examples in order to make the difficulty of many of the concepts apparent. It is all too easy to state glibly that a system is required to include a reference monitor, or that controls on covert channels will be enforced, or that the network will be secure against unauthorised observation of traffic, or a myriad of other requirements. Usually, such things imply a major development issue, because always the devil hides in the details.

No attempt has been made to include everything. Notable missing areas are security protocols[1], database security[2], the least privilege principle[3], modern work on cryptography and cryptographic algorithms[4], public key cryptosystems[5] and the recent work on the composition problem[6]. There are many other subjects that might have been covered, but it is felt that these essays provide sufficient background so that people who are developing an interest in computer security can appreciate many of the follow-on issues that are the focus of present research activity.

These essays were originally conceived as some handouts to a course in computer security that was started in 1993 in the Department of Electrical and Computer Engineering at the University of Toronto. These essays were drawn from the handouts for the spring of 1995.

In all cases an attempt has been made to include references to original material. However, much of this material is not easily accessible. In these cases rather more detail is included that might be otherwise strictly necessary.

Great thanks must go to those authors who developed the material upon which the essays are based. Several hundred students also merit thanks. Without their attention and probing questions the essays would contain many more absurdities and errors than they still do.

---

1 Bruce Schneier, Applied Cryptography, 2nd ed., Wiley, 1996.

2 S. Castano, M.G. Fugini, G. Martella, P. Samarati, Database Security, Addison-Wesley, 1995.

3 See for example Aris Zakinthinos & E. Stewart Lee, "A Least Privilege Mechanism for User Processes" *Fifth International Working Conference on Dependable Computing for Critical Applications*, IFIP Working Group 10.4 on Dependable Computing and Fault-Tolerance, September 1995, pp. 56-67.

4 See citation 1.

5 See citation 1.

6 Aris Zakinthinos & E. Stewart Lee, "Composing Secure Systems that have Emergent Properties", *11th IEEE Computer Security Foundations Workshop*, IEEE Computer Society, June 1998. A series of papers by these authors, of which this is the latest, discuss the problem.

Table of Contents

1. THE SECURITY VOCABULARY 1

1.1. Security Policies, Principles, and Mechanisms 1

1.1.1. Security Policy 1

1.1.2. Security Principles 2

1.1.3. Security Mechanisms 2

1.2. Access Control 2

1.2.1. Discretionary Access Control (DAC) 3

1.2.2. The Access Matrix 3

1.2.3. Capabilities 4

1.2.4. Profiles 4

1.2.5. Access Control Lists 5

1.2.6. Mandatory Access Control (MAC) 5

1.3. Common Attack Techniques 7

1.3.1. Search for Stupidity 7

1.3.2. Trojan Horse 8

1.3.3. Trapdoor 9

1.3.4. Spoofing 10

1.3.5. Virus 10

1.3.6. Worm 11

1.3.7. Overt Channel 11

1.3.8. Covert Channel 11

1.4. Safety Critical Systems 12

2. GLOSSARY OF TERMS 15

3. OPERATING SYSTEM SECURITY 21

3.1. Confidentiality Security Policy 22

3.1.1. U.S. Confidentiality Policy 22

3.1.2. CLASS ONE – POLICY 22

3.1.3. CLASS TWO – ACCOUNTABILITY 24

3.1.4. CLASS THREE – ASSURANCE 26

3.2. The Orange Book 28

3.2.1 Summary of Evaluation Criteria Classes 28

3.2.2 Requirement Directory 29

3.2.3 Summary of the TCSEC 45

3.3. A Formal Model of Access Systems 46

3.3.1. The Trusted Subjects 47

3.3.2. Command Semantics 49

3.3.3. Access Control 50

3.3.4. Delegated Ownership 51

3.3.5. Avoiding Confinement Problems 51

3.3.6. DAC Assessed 52

4. BUILDING ASSURANCE 53

4.1. The Reference Monitor 53

4.2. The Mechanisms That Must be Trusted 53

4.3. Single Level Operation 54

4.4. System High 55

4.5. Building Assurance 55
4.5.1. Verifying All Programs 55
4.5.2. Certify or Evaluate Critical Programs 56
4.5.3. Journal of All Security Relevant Events 56
4.6. Principles of Trusted Operating Systems....................57
4.6.1. Lattices 57
4.7. The Three Policies 58
4.7.1 Confidentiality 59
4.7.2 Integrity 59
4.7.3. Denial of Service 60
4.7.4. The TCB and the Reference Monitor 60
4.7.5. Specification 60
4.8. The Six Basic Principles....................61
4.8.1. Simple Security: READ DOWN 61
4.8.2. The $*$-Property: WRITE UP 61
4.8.3. Tranquillity 62
4.8.4. No Re-use 62
4.8.5. Accountability 62
4.8.6. Enhanced Privilege 63
4.9. An Example: Privilege in UNIX 63

5. BELL-LAPADULA AND SECURE XENIX 67
5.1. System States 67
5.2. State Transitions 68
5.3. Systems, System Appearance, System Actions....................69
5.4. Model Axioms 69
5.4.1. The Simple Security (SS) Property 70
5.4.2. The $*$ – ***Property*** 70
5.4.3. Discretionary Security (DS) Property 71
5.4.4. Compatibility Property 71
5.4.5. (Sometimes) Tranquillity Principle 71
5.4.6. Activation Axioms 71
5.5. System Security....................73
5.6. Other BLP Theorems....................75
5.7. Assessment of Bell-LaPadula 76
5.7.1 Perceived Advantages of Bell-LaPadula 76
5.7.2 Perceived Disadvantages of Bell-LaPadula 76
5.8. The Secure Xenix Interpretation of BLP....................77
5.8.1. The Interpretation of the System State 78
5.8.2. State Transitions in Secure Xenix 90
5.8.3. Access Control in Secure Xenix 91
5.8.4. Initial State z_0 97
5.8.5. Compatibility in Secure Xenix 98
5.8.6. Tranquillity in Secure Xenix 98
5.8.7. Activation 98
5.9. Conclusion....................99

6. RISK ANALYSIS AND ASSET LABELLING 101
6.1. Our Risk Measure 102
6.1.1. Worth 103

6.1.2. Vulnerability 104
6.1.3. Threat 104
6.2. The Evaluation of Risk 105
6.3. Risk Factors 106
6.3.1. Risk and a Reasonable Authority 107
6.4. Risk in Input Labelling 108
6.5. Risk in Separation 109
6.6. Risk in Output Labels 110
6.7. Output Label Risk Factor 110
6.8. Limiting Cases 111
6.8.1. Completely Reliable Separation 112
6.8.2. Faultless Input Data 112
6.9. The Environment and Maximal Risk 112
6.10. The Environment and Maximal Levels 113
6.11. The Environment and Minimal Reliability 113
6.12. Maintaining Minimal Risk 114
6.13. Discussion of the Results 116
6.14. Inter-System Communications 117
6.15. Summary 118
6.16. Acknowledgement 119

7. TRUSTED LAN NETWORKS 121
7.1. Architectural Overview 123
7.1.1 Hardware 123
7.1.2 Protocols 123
7.1.3 Encryption 125
7.2. Specification and Discussion 126
7.2.1. Hardware Components of the Network 126
7.2.2. Encryption and Keys 130
7.2.3. Protocols 133
7.3. Risk Assessment 136
7.3.1. Common Security Attacks 136
7.3.2. Damage Extent after Key Compromise 137

8. BIBA DATA INTEGRITY 141
8.1. The Elements of a Mandatory Integrity Policy 142
8.1.1 Definitions 143
8.1.2. The Low-Water Mark Policy on Subjects 144
8.1.3. The Ring Policy 148
8.1.3. The Strict Integrity Policy 149

9. CLARK-WILSON COMMERCIAL POLICY 153
9.1. Commercial Security Policy For Integrity 153
9.2. Differences with the Orange Book 155
9.3. Mandatory Commercial Controls 156
9.4. A Formal Model Of Integrity 157
9.5. Properties of Clark-Wilson 162

10. INFERENCE CONTROL IN DATABASE SYSTEMS 165
10.2. Knowledge and Inference 166
10.3. Types of Statistics 167

10.4. Disclosure........168
10.5. Partial Disclosure........169
10.5.1. Information Entropy 169
10.5.2. Disclosure 170
10.5.3. Protection 172
10.6. Control Mechanisms........172
10.6.1. Macrostatistics 172
10.6.2. Query Set Size Control 173
10.6.2. Tracker Attacks 173
10.6.3. General Trackers 174
10.6.3. Linear Equations and Other Attacks 175

11. ELEMENTARY CRYPTOGRAPHY........177
11.1. A Simplistic Introduction to Cryptography........177
11.2. Transposition Ciphers........178
11.3. Substitution Ciphers........179
11.3.1. Simple Substitution Ciphers 180
11.3.2. Homophonic Substitution Ciphers 181
11.3.3. Polyalphabetic Substitution Ciphers 182
11.3.4. Vigenère Cipher 182
11.3.5. Rotor Machines 183
11.3.6. Polygram Substitution Ciphers 186
11.3.7. The Playfair Cipher 186
11.3.8. Block Codes 187
11.3.9. Product Ciphers 187
11.3.10. The Data Encryption Standard 188
11.3.11. The DES Function f 189
11.5. One-Time Pads........195
11.5.1. Developing the Key Stream 196
11.6. Rivest-Shamir-Adelman........197

The SECURITY VOCABULARY

1. THE SECURITY VOCABULARY

The use of personal computers in industry and commerce has expanded dramatically in the last decade. Large gains in employee productivity are possible as a result of this technology. However, ensuring the security of the processes and the privacy of data that these machines access is a very hard problem. Solutions that ensure security by preventing access by legitimate users are inconsistent with the gains in productivity that are possible. The general problem of computer security is being attacked by government and by academic and industrial research with some notable success. The aim of these essays is to review the principles behind these successes, to describe some of the remaining problems and to discuss their application in industry and commerce.

The opening two essays present the terminology used in the computer security world. This terminology is mostly inherited from that used in government. Consequently, it has become common practice to speak of *secret* when *company confidential* might in many cases be a more appropriate term. In the literature, the government terms are (almost) universally used. It is an easy matter to map the governmental vocabulary onto that used in other sectors.

1.1. Security Policies, Principles, and Mechanisms

The objective of a trustworthy computer system is to control access by subjects (users) to objects (data). This control is governed by a set of general goals and objectives called a security policy.

1.1.1. Security Policy

The security policy is a statement of intent about the required control over access to data. For a trustworthy system to be effectively implemented, the security policy it must enforce must be static and must be precisely known. The security policy must represent the pertinent laws, regulations, standards, and general policies accurately. There are three types of policy generally used in secure computer systems:

Confidentiality Policy:

A confidentiality policy typically states that only authorised users are to be permitted to observe sensitive data, and that all unauthorised users are to be prohibited from such observation.

Integrity Policy:

An integrity policy has two facets. The first refers to the quality of the data that is stored in the computer. The integrity policy will state that the data should reflect reality to some degree. How best to do this is the subject of much research activity.

The second facet of integrity policy is associated with the data being available for use when it is legitimately needed. No user, whether he or she is or is not authorised to access some data item, should be able to unreasonably delay or prohibit another authorised user from legitimate access.

Availability Policy: The computer system should be available for use when it is needed, and it should satisfy some specified requirements for its mean-time-to-failure and its mean-time-to-repair.

The first and second of these types of policy will be examined in detail. The third is a separable topic. It will be mentioned only in passing.

1.1.2. Security Principles

Security principles are not policy. They are a collection of generally accepted standards of good practice that is thought to further the enforcement of security policies in general. We shall identify several popular security principles throughout the course.

1.1.3. Security Mechanisms

A security mechanism is a device that is used partially to enforce a security policy. Some mechanisms do this by individually or collectively implementing a security principle. Other mechanisms are implied directly by a security policy. These enforce some component of the policy.

1.2. Access Control

The purpose of access controls is to authorise legitimate access by subjects to objects, and to prohibit illegitimate accesses. The essential notion is that without a legitimate access to an object, the system prohibits anything from happening with or to it. It can not be owned, created, destroyed, observed, altered, appended to, distributed, executed, or searched. There are two forms of access control, those that can be used at the discretion of the owner of an object, and those that are required to be used by the policy.

1.2.1. Discretionary Access Control (DAC)

With Discretionary Access Control the owner of an object is able to specify the types of access other users, or classes of users, can have to that object. Discretionary Access Control[7] is a means of controlling access by subjects to objects. DAC is established by the owner of an object, and is changeable only by that owner. It is based on the identity of the subject and the identity of the object. Usually, the identity of the owner that the subject is an agent for is a significant factor in DAC.

EXAMPLE: UNIX Permission Bits		
Binary	Octal	Comment
100 000 000	400	Read by owner.
010 000 000	200	Write by owner.
001 000 000	100	Execute (search) by owner.
000 100 000	040	Read by group.
000 010 000	020	Write by group.
000 001 000	010	Execute (search) by group.
000 000 100	004	Read by others.
000 000 010	002	Write by others.
000 000 001	001	Execute (search) by others.

In UNIX, the owner of an object is permitted to set independently permissions as shown. There is a 9-bit field associated with every object, with the bits independently settable with the meaning in the table above.

1.2.2. The Access Matrix

Discretionary access controls can be captured in the access matrix. This is a (usually *very* sparse) matrix showing the totality of all subjects (users and processes) and their access permissions to a set of supposedly protected objects. An example is given in the following table. The permissions that are stored at each entry are drawn from the set rwaxdso.

	Objects												
Subjects	1	2	3	4	5	6	7	8	α	β	ξ	δ	ε
1	ox								rwo	r			
2		ox							r	rop	o	xo	
3			ox									x	
4				ox							rw		
5					ox					rop	rw		
6						ox					rw		
7							ox						rxo
8								ox					

[7] U.S. National Computer Security Center, "A Guide to Understanding Discretionary Access Control in Trusted Systems", NCSC-TG-003, Library N0. S-228,576, September 1987.

The members of the set rwaxdso stand for read, write, append, execute, directory, special or sticky (see glossary) and owns.

1.2.3. Capabilities

A capability is an unforgeable permission that is incontestable proof of authorisation for a subject to access an object. The capability is associated with a subject, is protected or otherwise given high integrity, includes the name of the object that it refers to, and specifies the access rights that the subject may have to the object. The set of all capabilities for a given subject is a row of the access matrix. If the access matrix were not so sparse, we might store the whole row with the subject and capture all the subject's access rights at once. This is not practical because it takes too much memory.

A capability-based system can in theory provide dynamically changeable domains for processes to run in. When a process (which is, of course, a subject) is started, it is given the capabilities it needs to do its thing. Dynamic domains are ideal for limiting the damage that a Trojan horse can cause. It is very difficult, however, to build such a system and make it efficient. A structure that attempts to do this is typically called a Least Privilege Mechanism (LPM).

Subjects can have the right to pass on their capabilities to other subjects. Sometimes this right is conditional on, for instance, a special access permission. Making this right conditional on ownership makes the system too secure – so secure that it can not be used for practical work. Because the right is not necessarily confined, it is difficult to make an effective list of all the subjects (users) who have access to a given object. This can make DAC impractical. For instance, one of the things that may be desirable is the ability of the owner of an object to revoke access by all other subjects. This is difficult if the permissions of each other subject has to be inspected.

Most security systems require that an owner of an object be able to specify by name those other users who, either directly or indirectly through processes, are permitted access and those who are prohibited access. This usually must be achievable on the basis of user names. Grouping users is not considered fine enough control. It is hard to do this with capabilities – the control of passing on the capability becomes rather complicated. This is because capabilities are row-based mechanisms. Deletion of an object also presents a problem, because all access capabilities must be deleted as well.

1.2.4. Profiles

A profile is a row of the access matrix, with the empty entries deleted. It is, in effect, a summary of all a subject's capabilities. It suffers from all the difficulties that capabilities suffer from, without some of their advantages.

1.2.5. Access Control Lists

Access control lists are the columns of the access matrix. It is usual to delete unnecessary entries, so that the list is not too large. An access control list is a list of the users who are permitted access to an object, and those who are prohibited access to the object. In this way, the control of access to the object is associated with the object, which is an intuitively satisfying concept. In most implementations, every file has associated with it another file called the ACL, that ordinary users can not observe or alter. Observation of the ACL requires a special access privilege that only the trusted part of the system has and that can not be passed on.

In order to limit the growth of the ACLs, a *wild card* mechanism is common. Let an ACL be composed of a sequence of entries of the form [user, group, permissions]. Let the (user) field be the name of a user. In practice this will be the name of the user who is accountable for the actions of a subject. Let the (group) field be the name of a group of users with a known membership. Both the (user) field and the (group) field can also be the single entry "*". This * stands for any legitimate entry in the field it occupies, and is called a *wild card*. Let the permissions include the usual (rwaxdsop) set, or be the single entry (-). Then an access control list like the following one can be quite compact, and yet quite easy to search, to store, or to alter. Suppose the name of the file is <u>SecretOfHeidi&Peter</u>.

user	**group**	**permissions**
Heidi	Athlete	rwxao
Peter	Athlete	xa
MumOfHeidi	Parent	-
DadOfHeidi	Parent	-
MumOfPeter	Parent	-
DadOfPeter	Parent	-
*	Athlete	r
*	*	-

When a file is created it will be given a default ACL, including something like ownership by the originating user with nobody having any access except him or her. The defaults may be system-wide, user specific, or associated with the directory that contains the file. All of these can co-exist, but there are usually limits on the last two for obvious reasons. Another approach is to allow a user to specify that the ACL of a new file should mirror a given prototype ACL, perhaps the ACL of an existing file.

1.2.6. Mandatory Access Control (MAC)

Mandatory Access Controls (MAC) are means of restricting access to each object based on the sensitivity of the information contained in the object and on the authorisation of each accessing subject. The sensitivity of the object is determined by its classification. The authorisation of the subject is determined by its clearance. Every object has associated with it a label containing its classification. Every subject has associated with it a label containing its clearance.

1.2.6.1. Realms of Knowledge

Not all knowledge is intended to be universal. Some knowledge is private for social reasons – medical records. Some knowledge is secret for functional reasons – when the next police drug-raid will happen. Some knowledge is more secret than other knowledge – the content of the next patent application to be filed by me contrasted with the content of the next paper to be published by you. Some of the difference, in the sense that *this* is more sensitive than *that,* is a matter of degree or of the identity of the observer. Some of the difference is related to the realm of knowledge that the information belongs to. Users need special permission to access information in special realms of knowledge. For instance, Patent Application is a realm of knowledge to which salesman has no access.

A category is an attribute of an object that identifies it as belonging to some group of objects with some common basis – often a realm of knowledge. This basis usually has to do with the contents of the document. A category is a property of the data in the document.

A caveat is an attribute of an object that also identifies it as belonging to some group of objects with some common basis. This basis usually has to do with what a user (a process) can do with the object. A caveat is usually a property of, or a restriction on, the way a document can be handled.

A simple model is that a category is a property of the data and a caveat is a restriction on the handling of the data. If we have a secret research project with the category NuclearAutoEngine we might want to prohibit access by foreigners to some of the critical data. We do this with a caveat Foreigner. Any user with the entry Foreigner in his or her caveat list would be denied access.

For simplicity, from this point on in this course the concept of the existence of a caveat will be ignored.

1.2.6.2. The Dominates Relation and the Lattice

Let there be a subject S and an object O. Let the clearance of S be a sensitivity level $L(S)$. Let the classification of O be a sensitivity level $L(O)$. The sensitivity levels are a totally ordered sequence, with the order relation being $\geq$. Let the categories of S be a set $G(S)$, and the categories of O be a set $G(O)$. The categories are a set. In comparing the categories of two objects the significant thing is whether one set of categories is a subset of the other set. Let the security level of S be $\mathbf{L}(S)$, and the security level of O be $\mathbf{L}(O)$. The security level of an object or subject is the combination of its sensitivity and its categories.

We are interested in being able to determine whether one subject or object is at a higher security level than another. We speak of this as one object dominating another if the first is at a higher level or the same level as the second[8]. In these notes dominates will be written using the symbol $\rhd$. The converse of dominates, called is dominated by, is written $\lhd$. The negatives of these two relations are sometimes useful; they are written $\neg\rhd$ and $\neg\lhd$ respectively. Let the two objects be α and β. $\alpha,\beta \in \{S,O\}$ and

$$\alpha \rhd \beta \quad \Leftrightarrow \quad [\ L(\alpha) \geq L(\beta)\ ] \text{ and } [\ G(\alpha) \supseteq G(\beta)\ ].$$

The dominance relation defines a lattice on the possible security levels. A partial ordering like $\rhd$ or its inverse $\lhd$ defines a structure called a lattice. For subjects or objects $a,b,c,\cdots$, the relation $\rhd$ has the following properties.

Reflexive	$a \rhd a$ is always true.
Antisymmetric	If $a \rhd b$ and $b \rhd a$ then $L(a) = L(b)$ and $G(a) = G(b)$ so that $\mathbf{L}(a) = \mathbf{L}(b)$.
Transitive	$a \rhd b$ and $b \rhd c$ implies that $a \rhd c$.

1.3. Common Attack Techniques

The following sections describe the main techniques used to attack systems.

1.3.1. Search for Stupidity

There are only three ways into a computer system. The first bypasses all security mechanisms, and relies on a direct usage of the raw machine. Normally, this way is blocked because it must be done from the console of the machine, and hopefully that is in a locked room that Machiavelli himself could not enter from a distance.

user name	**password**
root	system
root	god
root	guru
root	wizard
wolf	flow
guest	guest
guest	visitor

The second way concerns normal, authorised users. Normal users have every right to login to the machine. Once they are in, they can deploy any of the devices described below to circumvent the security controls. There is far, far greater security risk from authorised users than there is from any other person. The risk is because these users can legitimately pass the primary security perimeter, the login perimeter.

8 Dorothy E. Denning, "A Lattice Model Of Secure Information Flow", Communications of the ACM, Vol. 19, No. 5, pp. 236-243, May 1976.

Once it is passed, the threat comes from those who are careless, who want to play (often expressed as testing the security controls), or who are just plain evil.

Third, a more covert entry relies on the stupidity of authorised users or system administrators. The primary security perimeter is the login perimeter. Entry is easy if this perimeter is not carefully guarded. For example, if the system permits it, an intruder can try to login using many (often thousands) of user names and corresponding passwords that are frequently used. The table above gives some hint to the kind of (name, password) pairs that might be on the list.

There has been considerable work aimed at making passwords random, and memorable, and typeable. Some aids exist[9].

Because of the world-wide computer network, the malicious invader need only login to one machine that is attached to the network. Then he or she can use its power to try to login to other machines. The first of these other machines that an evil invader will try to invade are those that have their names stored in the files of the machine that has been successfully invaded. These attempts can be in parallel, with the telephone bills being paid by the various machines' budgets. Once a login to any machine is successful, the ability to repeat the login is made sure by one of the techniques below.

Thus, the observation that by far is the most significant and important motto of the computer security professional:

The most important security perimeter is the login perimeter.

1.3.2. Trojan Horse

Other than problems with the login perimeter, Trojan horses are the single most dangerous security problem. They have an additional serious disadvantage when compared to login. The act of performing a login can be recorded by the system, and many systems inform their users of the time and place of their last login. This can be monitored, and can sometimes show up either illegitimate users or misuse of passwords. On the other hand, Trojan horses operate entirely within the security perimeter. In practice they are only detected when they cause a serious breach of security, or by accident.

A Trojan horse is a part of a program that otherwise conforms to the security policy. When this apparently innocuous program is executed it compromises the security of the user (or his agent executing it) by unauthorised violations of the security policy. Trojan Horses are built into the source code of a program by the original programmer, and have

9 Ping Lin and E. Stewart Lee, "An Algorithm for the generation of Pronounceable, Typeable Passwords", Proc. 1st Annual Canadian Computer Security Conf., Ottawa, Ontario, 30 Jan - 1 Feb, 1989. Pages 165-176.

been known to be dormant for a period of time before any attempt is made to overcome security.

The Trojan horse is the single most powerful tool in the armoury of the malicious intruder. DAC is impotent against it. MAC protects against it, but only as long as the Trojan horse is confined to one realm of knowledge. The canonical example is as follows:

Peter: "I really hate this editor. I wish there was a better one."

Heidi: "I wrote a pretty good editor last week. I think it is a great improvement. See – it uses changes in the background colour to highlight misspelled words. It has lots of other fancy features, too!"

Peter: "That looks fantastic. Can I try it?"

Heidi: [Gotcha] "Sure. It's in my file /hack/myed. I will change the permissions so you can run it. I will appreciate any feedback you can give me!"

The editor, when run by Peter in his directory /mysecret, does a wonderful job. It also contains a Trojan horse that copies all of the files from Peter's directory /mysecret to Heidi's directory /hack. There is no security mechanism to prevent this copying. It is all perfectly legitimate, because Peter is running the editor with his access permissions, and the editor is owned by Heidi and she has given it the right to write into /hack.

Trojan horses are the operating mechanism behind a spoof and a virus and sometimes a worm.

1.3.3. Trapdoor

A trapdoor is a feature built into a program such that the provision of specific input data allows the process executing the program to overcome the security policy, usually by directly returning to the calling program with the called program's permissions, completely bypassing all security guards. Trapdoors are built into the source code of a program by the original programmer, and have been known to be dormant for a period of time before any attempt is made to overcome security.

Trapdoors and their establishment can be very sophisticated. Suppose there is a Trojan horse placed into the compiler such that

- When it compiles the login procedure, it will generate unexpected (by the casual user) output code that will let the user ptah login without a password, and

- When the compiler recompiles itself the code to build the Trojan horse into the compiler is itself always included in the compiler.

The original source code for the Trojan horse can then be removed from the compiler, with the knowledge that the Trojan horse will always be included in the login procedure. The only way to detect this is to compare carefully the source and object code of the compiler, a task not frequently undertaken for obvious reasons.

1.3.4. Spoofing

A spoof is a pretence that a subject is some other subject, or a user is some other user. It is possible only if the authentication mechanism fails, either because it is not good enough or because it contains a Trojan horse (e.g.: always authenticate the user ptah as the user root).

There are several spoofs that are often overlooked. It is possible to be able to authenticate subjects to one another with adequate reliability. But how does the user, when he or she wants to login, tell that the software and hardware is what he or she expects it to be? How does the system software determine who it really is that is trying to login, and whether or not he or she is under duress? The answers to these kinds of questions are not obvious!

Most secure computer criteria insist on a trusted path from a login location to the login procedure. There is no agreed best way to do this. Most system software identifies a potential user by some combination of:

- Something he or she knows (a password).
- Something he or she has (a key or badge).
- Something he or she is (biometrics).
- Where he or she is.
- What time it is.

Some users consider a login protocol that involves more than the first two of these factors to be a nuisance (or worse), and are actively hostile about their use.

1.3.5. Virus

A virus is a program that when executed operates entirely within the security policy. When a virus is activated (becomes part of a process) it uses a Trojan horse to search its authorised environment for executable programs that it is entitled to modify, and attaches itself to all such programs. In turn, when they are executed, the virus reactivates, and eventually spreads throughout the executable code in the computer.

The only hard parts of setting up a virus are to substitute the entry point of the virus for that of the program, and then to pass the program's parameters, supplied by the calling program as input to the original program, on to the original program when they appear at the entry to the virus. If this is done correctly the chance of a user detecting a virus is very nearly nil unless it announces its presence. The substitution of entry point can easily be done by renaming the original program with a name unlikely to be noticed, renaming the virus with the original program's name, and arranging that the virus knows about the renamed original program. Parameter passing just means that when the original program is called, the registers must be restored by the virus to their state that they had on entry to the virus.

These programs are easy to create and very difficult to detect. The anti-virus agents in use today accomplish their task by looking for the distinctive patterns that are a representation of the virus's code and looking for new copies of particular files.

1.3.6. Worm

A worm is a program that migrates from a computing environment to another computing environment, often keeping a record of the last few environments it has entered. The name worm comes from analogy with the animal well known to gardeners and fishermen, because each connected environment of the worm is called a segment. There are usually several segments in order to develop fault tolerance. Once the worm has migrated to a new environment, it can do whatever it is entitled to do according to its discretionary and mandatory access controls.

A worm must have the ability to login to a target machine. It is this perimeter, as usual, that is the critical one.

Good worms are used to distribute software, to propagate quick fixes to bugs, to modify system tables to reflect the present state (as when a new machine is added to a network) and generally to help the system managers ensure that all machine software is up-to-date.

Bad worms carry a virus.

1.3.7. Overt Channel

An overt channel is a communication channel that is used in the way it is intended to be used, that legitimately exists and that is consistent with the security policy. It is to be contrasted with a covert channel.

1.3.8. Covert Channel

A covert channel is a mechanism for two processes to communicate, where such communications is a violation of the security policy. There are two forms of covert channel.

In addition to the ones discussed below, some covert channels are, in practice, undetectable. For instance, suppose two processes with good provenance, thought to be secure, in the course of their legitimate function exchange data. They could modulate this legitimate exchange with an illegitimate exchange, for example by varying the message length. No known confidentiality technique, other than turning off the computer, can be proven to eliminate this type of threat. The two processes are permitted to exchange certain kinds of data. If they begin to modulate the data to exchange other kinds of data, detection becomes difficult.

1.3.8.1. Storage Channel

A storage channel uses a storage cell of some kind (for instance, a register, an I/O buffer, or a system parameter) to communicate from one process to another, where such communication should not happen. Storage channels can be completely eliminated in most cases by careful system design using sensible hardware and good software engineering. A common way is to use a blunder in the hardware or software to transfer data, such as reading and writing a register in an I/O device so as to evade the security checks.

1.3.8.2. Timing Channel

A timing channel uses some detectable time-related action to transfer information. Timing channels almost always rely on resource exhaustion. This is the detectable unavailability of (some or all of) a resource because some process is hogging most or all of it. By using all of memory, locking and unlocking a data file, or modulating the use of the CPU, it is possible to transfer information from one process to another. This transfer is not controlled by the security mechanism.

Fortunately, the bandwidth of these timing channels can be made quite low. The timing channels that are capable of high data rates can be either prohibited or controlled as a special case. For instance, locking and unlocking a data file could be done at quite a high rate, but in practice there is little reason for a single process to do this. The system may limit the rate of successive lock commands from one process. The objective is to keep the channel bandwidth very low. Typical permissible bandwidths are in the 0.01 bit/s region.

1.4. Safety Critical Systems

A safety critical system is one where the availability, fault tolerance, accuracy, faithfulness, or integrity of the system needs to be assured. Safety critical systems are closely related to secure systems. They both require that identifiable properties of the system be irrefutably established. Many of the same techniques apply to both types of systems.

There is beginning to be substantial research into the area of safety critical systems. Following the example of security, initially a policy that describes the required safety properties is specified. This policy is then imposed upon a design. Using modelling or other techniques it is proven that the design fulfils the properties. Then an implementation of the system is constructed and thoroughly checked to establish that it is exactly consistent with the design.

The process outlined in the preceding paragraph is an attempt to ensure that the desirable level of assurance is developed in the efficacy of the system conforming to its goals. We will visit this issue frequently in this course.

GLOSSARY

2. GLOSSARY OF TERMS

Access	The ability to make or make use of information.
Access Control List (ACL)	A list of subjects that are authorised to access some object.
Accountable User	The identity of the user that is responsible for the instantiation of a subject or for the existence of an object. An executing process is always an agent of a user, and each executing process will always have some user accountable for it. An object is always owned by or controlled by a unique user, and each object will always have some user accountable for it.
Alter, Write	To modify any part of an object without increasing its size.
Append	To increase the size of an object without in any way modifying its original contents.
Authenticate	Verify the identity of a user, a machine on a network, or of something that is an agent of a user or machine (a process, etc.). This almost always uses an exchange of messages called an authentication protocol. Most such exchanges depend on the sender and receiver sharing a secret, or on the existence of a universally trustworthy authentication server that all entities share a secret with and that can verify their identity.
Authorise	Grant a subject access to an object.
Beyond Top Secret	Whether or not there are any security classifications beyond top secret is a top secret. Words like NATO Top Secret and Cosmic recur.
Boolean Algebra	A lattice with a total ordering instead of a partial ordering.
Capability	An unforgeable permission that is incontestable proof of authorisation for a subject to access an object.
Category	An attribute of an object that identifies it as belonging to some group of objects with some common basis. This basis usually has to do with the contents of the document.
Caveat	An attribute of an object that identifies it as belonging to some group of objects with some common basis. This basis sometimes has to do with the attributes that a user (a process) must have to access the object, or more commonly with some special handling requirement.

Certify	To formally guarantee the accuracy, correctness and completeness of a security mechanism measured against some established evaluation criteria. Compare certify with evaluate.
Classification	A level of sensitivity of an object. A measure of the amount of damage that could happen if the object fell into the wrong hands. In Canada the usual classifications for objects connected with National security are Top Secret, Secret, Confidential, Private and Unclassified. These terms should be looked up in this glossary, along with Beyond Top Secret. In the United States the usual classifications for objects connected with National security are Top Secret, Secret, Classified and Unclassified.
Clearance	A level of trustworthiness of a subject. The subject can be a person or of an agent authenticated to be acting on his or her behalf. The clearance of a subject is a measure of the proven, demonstrated, or assumed trustworthiness that it merits. In Canada, the usual classifications for subjects connected with National security are Top Secret, Secret, Confidential, Private, and Uncleared. The meaning of these terms is similar to the meaning of the levels of classification, except a clearance applies to a subject rather than an object.
Complete Isolation	A system such that no flow of control or data can exist between two or more (abstract) compartments. In operating system terms, the flow of control or data can only be up or down the tree of activated processes, from parent to child or from child to parent. A fundamental principle in containing the damage that might occur in the event of unauthorised access.
Confidential	An object is classified as confidential if unauthorised disclosure would be prejudicial to the interests or prestige of the nation or any governmental activity or be of advantage to a foreign power. Also, individual personal files may be classified confidential.
Confidentiality	The limitation of access to data based on the sensitivity of the data and the trustworthiness of the user. Security is often contrasted to integrity. Security is concerned with access to data, while integrity is concerned with the value of data.
Confinement	The containment of the transitivity of the delegation of an authority or access or privilege. If a user authorises an agent to perform actions on his behalf, and if the agent is powerless to pass this authorisation on to another unauthorised party, then the authorisation is said to be confined. A system in which subjects are so confined is said to have confinement.

Denial of Service	A requirement established by a security policy that there is some criterion that must be met before a subject can be denied legitimate access to the objects that it must access to perform its proper function.
Discretionary Access Controls	Controls on access to an object that may be changed by the user (or his authenticated agent) that created the object.
Distribute	To make a copy of an object with no alteration or observation of it.
Evaluate	To measure the degree to which a security mechanism satisfies some given evaluation criteria. Compare evaluate with certify.
Execute	Execute has the usual meaning, but when used as a permission it does not imply that the executed object can be observed or altered. The transfer of parameters and results, and issues concerning non-local data objects or side effects are all instances of observe, alter, etc., and are all subject to both discretionary and mandatory access controls. Execute is subject to access controls.
Integrity	The integrity of data is the likelihood that the data's value is what it should be. Integrity is often contrasted to confidentiality. Confidentiality is concerned with access to data, while integrity is concerned with the value of data.
Lattice	A partial ordering on a collection of objects such that there exist a Least Upper Bound and a Greatest Lower Bound. Usually, the combination of the clearances, classification, category, and caveat properties of objects and subjects, interpreted according to the Security Policy, forms a lattice.
Liveness	The state in which it can be shown that something eventually happens. See safety; it is not difficult for a system that is not live to be safe.
Mandatory Access Controls	Controls on the access by a subject to an object that are determined by the security level of the subject relative to the security level of the object.
Multilevel	A system that runs with more than one level of subjects and objects is a multilevel system. To be contrasted with a single level system.
Object	Data or programs stored in a computer system. Objects that are programs that are executed become a part of a process, and so are also subjects as long as they are being executed.
Observe	To have access to the value of an object. Depending on the security policy and the system design, it may be possible to see the name of some object but not its value, or it may be possible to see the name of an object only if seeing its value is permissible.

Proof of Correctness	A formal demonstration that the (high level) code of a program agrees with some assertions that are included in it. The assertions are coded as assertion statements. They contain a proposition (a Boolean predicate) that must be demonstrably true for any possible values of the variables that are involved. Using some known formal semantics of the statements in the programming language, it is shown that each assertion must be true (in the terminology of mathematical logic, at that point in the program it is a valid theorem). There are usually four kinds of assertions that are used: • a pre-condition to establish a basis for reasoning; • an in-line assertion, that must always be true based on the situation that was previously known and the semantics of the statements that could have been executed to get to the point of the assertion, for any combination of values of variables; • a loop invariant, that must be true for any and all excursions around the body of a loop; and • a post condition, to establish that the program agrees with its expected semantics.
Read	To observe the value of an object with a destructive write-destroy-restore sequence. Most modern computer storage devices read their contents when they observe.
Restricted	An object is classified as restricted if it should not be published or communicated to anyone except for official purposes.
Safety	The state in which it can be shown that something bad never happens. Something bad is something that violates the security policy. See liveness.
Search	To cause an object to be examined for a specific known value. A search does not allow any observation of the object, except by exhaustive probing. A successful search returns the value and perhaps some associated information. An unsuccessful search returns a failure indication. Search is subject to access controls.
Secret	An object is classified as secret if unauthorised disclosure would endanger national security, cause serious injury to the interests or prestige of the nation, or any governmental activity thereof, or would be of great advantage to a foreign nation.
Security Level	For data, programs, or other files, the combination of classification, category, and caveat that establish the attributes required to access the data in the confidentiality sense. For processes, the combination of clearance, category, and caveat that establish the attributes available to access data in the confidentiality sense.
Security Policy	The set of rules that establish legitimate access, and define accesses to be prevented, if a system is to be secure.

Single Level	A system that runs with only one level for all subjects and objects is a single level system. To be contrasted with a multilevel system.
Sticky Bit	A tenth UNIX permission bit that is interpreted in various ways depending on the nature of the object it is associated with. For instance, on a directory it may mean that only the owner can delete any file in the directory regardless of what permissions other users have.
Subject	A process. An agent of a user in a computer system. An executing process is always an agent of a user, and each executing process will always have some user accountable for it.
System High	System high refers to the highest security level that can possibly occur and that a system must be able to deal with. It is sometimes the case that a system is constrained to operate at system high in order to fulfil operational imperatives.
Top Secret	An object is classified as top secret if unauthorised disclosure would cause exceptionally grave damage to the nation.
Trapdoor	A feature built into a program such that the provision of specific input data allows the process executing the program to overcome the security policy, usually by directly returning to the calling program, completely bypassing all security guards. Trapdoors are built into the source code of a program by the original programmer, and have been known to be dormant for a period of time before any attempt to overcome security is attempted.
Trojan Horse	A part of a program that otherwise conforms to the security policy, such that when this either invisible or apparently innocuous part of the program is executed it compromises the security of the user (or his agent executing it) by illicit violations of the security policy. Trojan Horses are built into the source code of a program by the original programmer, and have been known to be dormant for a period of time before any attempt to overcome security is attempted.
Trusted Computing Base (TCB)	The Trusted Computing Base is the set of protection mechanisms in a computer system that is responsible for enforcing a security policy. Formally, it includes all mechanisms – hardware or software – but in practice the hardware is often considered to be immutable. Frequently, in consequence, a reference to the TCB is taken to be a reference to the software mechanisms that enforce the security policy. These are the mechanisms that must be trusted, because to do their job they must violate the policy that they are enforcing. If they did not need to violate the policy, they need not be trusted, and would not be a part of the TCB. Normally, considerable effort is expended to keep the TCB to a small size.

Unclassified	Anything that is not classified otherwise (higher) is classified as unclassified.
Verify	To use a (usually formal) mechanism to establish the formal semantics of a program, in terms of the formal semantics of its components. This always involves software tools, two in particular that are called a verifier and a truth prover. The verifier knows the formal semantics of the various constructs in the language. Starting with a precondition (often the condition true) it reads the statements of the program, and deduces their semantic effects. The truth prover does a lot of logical reductions on the evolving expression. The whole enterprise concludes with an expression describing the semantics of the program. In appropriate cases, such as subprograms that do not reference non-local variables and do not allow parameters to alias of each other, this expression can then be added to the verifier's dictionary of constructs with known semantics.
Virus	A program that when executed operates entirely within the security policy. When a virus is activated (becomes part of a process) it searches the file system for executable programs that it is entitled to modify, and attaches itself to all such programs. When these in turn are activated, they do the same. The virus can spread throughout the system unless contained by mandatory access controls.
Worm	A program that migrates from a computing environment to another computing environment, often keeping a record of the last few environments it has entered. The name worm comes from analogy with the animal well known to gardeners and fishermen, because each connected environment of the worm is called a segment. There are usually several segments in order to develop fault tolerance. Once the worm has migrated to a new environment, it can do whatever it is entitled to do according to its discretionary and mandatory access controls. Worms can be good or bad.
Write	The same as Alter.

OPERATING SYSTEM SECURITY

3. OPERATING SYSTEM SECURITY

It is desired to enforce a *security policy* that is an organisation's decisions about security. This policy must be agreed in advance and must be stationary.

A policy is a governing objective or goal. In practice, some proposals for security policy arise within an organisation and are approved by the most powerful body or person in the organisation. They then represent the objectives or goals for data security in the organisation. The security policy governs data security in the organisation.

There are three distinct classes of security policy that are used in most organisations.

[1].	Confidentiality Policy:	*Who is permitted to see what?*
[2].	Integrity Policy:	*What quality data values are needed?*
[3].	Availability Policy:	*What service availability is needed?*

The confidentiality policy that we use has been inherited from the pencil-and-paper times before computers. Fortunately, it fits quite well.

Data integrity is an issue that is different in a computer than it is in the pen & paper world. This is because in the P&P world, if data should have high quality it is written in ink and never erased, so that the evolution of its value is immediately evident. This is not the case with computers, in which a change of value erases and overwrites the previous value.

There is no known way to arrange that the design of some computer hardware & software will prevent an anarchist or terrorist from setting off a bomb to deny service. In addition to this type of drama, denial of service attacks can be quite subtle and might be hard to distinguish from momentarily high computational loads. What is possible is not known.

There is no agreement the problem's identity, or about what solutions might exist.

3.1. Confidentiality Security Policy

The objective of a trustworthy computer system is to control access by subjects to objects. The security policy is a statement of intent about the required control over access to data. For a trustworthy system to be effectively implemented, the security policy it must enforce must be static and must be precisely known. The security policy must represent the pertinent laws, regulations, standards and general policies accurately.

3.1.1. U.S. Confidentiality Policy

The situation in the United States of America was, until recently, more mature than that elsewhere and in consequence documentation is much more readily available. For these reasons, this section will concentrate on the US situation.

The confidentiality policy of the government of the USA is based on several laws, Presidential Findings and Departmental Regulations. It can be encapsulated in six simple requirements, arranged in three classes.

3.1.2. CLASS ONE – POLICY

REQUIREMENT ONE
CLASS ONE

SECURITY POLICY

There must be an explicit and well-defined security policy enforced by the system.

For instance, US Department of Defense (DoD) directive 5200.28 requires that *classified material contained in an ADP*[10] *system shall be safeguarded by the continuous employment of protective features in the system's hardware and software design and configuration. [...] [ADP systems that] process, store, or use classified data and produce classified information will, with reasonable dependability, prevent:*

a. Deliberate or inadvertent access to classified material by unauthorised persons and

b. Unauthorised manipulation of the computer and its associated peripheral devices.

---

10 Automatic Data Processing. Sometimes reference is made to Electronic Data Processing (EDP) or other such terms. They are all a euphemisms, presumably intended to assign more force to a document by concealing its content behind a fog of alphabetic symbols. These should not be confused with mnemonics, which are intended as an *aide memoire.*

The idea of classified material that is mentioned above has long been established. The implication is that there is a set of classifications of data and clearances of persons that is totally ordered. Also, there exists a set of realms of knowledge, such that any data item of file can be a member of none, one, or more realms. The rules for access are expressed in terms of this arrangement, as described in the previous essay. The combination of classification or clearance, together with one or more categories or caveats, forms a security level. The normal way of deciding to permit or prohibit an access involves the comparison of the security level of the subject attempting access with the security level of the object potentially being accessed.

The system will be composed of subjects and objects. There must be a set of criteria that will determine whether a proposed access by a subject to an object is to be permitted or to be prohibited. The dominance rule of Denning, explained in previous essay, is one such criterion. Any proposed access must either pass or fail these criteria; no access can be permitted to evade or avoid this determination. The simple rule that existed long before computers had been conceived is that information can not flow downwards in the lattice. This is a restatement of the dominance relation.

Some examples:

<u>Within a Realm of Knowledge:</u>
This means that all relevant subjects and objects have the same category set. A subject cleared to secret can read data classified confidential but is prohibited from reading data classified top secret. A subject cleared to secret can write data classified top secret (but not overwrite such data) but is prohibited from writing data classified confidential. These actions prohibit data from flowing from a higher level to a lower level.

<u>Across Realms of Knowledge:</u>
This means that all relevant subjects and objects have some category set, not necessarily the same. Let the categories be <u>red</u>, <u>white</u> and <u>blue</u>. For simplicity, let all relevant subjects and objects be cleared or classified the same, say secret. A subject with category set (<u>red</u>, <u>white</u>) can read data with category set (<u>empty</u>) or with category set (<u>red</u>) or with category set (<u>white</u>) or with category set (<u>red</u>, <u>white</u>), but is prohibited from reading any data with <u>blue</u> in its category set (even if <u>red</u> or <u>white</u> or both <u>red</u> and <u>white</u> are also in this set). A subject with category set (<u>red</u>, <u>white</u>) can write data with category set (<u>red</u>, <u>white</u>), or with category set (<u>red</u>, <u>white</u>, <u>blue</u>). It is prohibited from writing data with any other category set.

REQUIREMENT TWO
CLASS ONE

MARKING

Access control labels must be associated with objects.

This requirement is quite restrictive, because it eliminates the use of capabilities. There are several reasons for forcing a system to be designed this way. The revocation, destruction and confinement problems touched on in previous essay are certainly among the more important. The requirement associates the security data pertinent to an object with the object, not with several subjects that may possibly or permissibly access the data. This arrangement seems logical and elegant.

Originally, when the revocation, destruction and confinement problems were not as well understood as they now are, it is clear to see that this strong requirement was indicated. Modern research may, however, solve some of these problems so that capabilities become fashionable again – but the DoD requirements are not likely to allow them to be used in the foreseeable future unless they are shown to be singularly better than the present arrangement.

Access control labels include the category set, the classification, the set of users who are to be permitted access and the set of users who are to be denied access.

The classification is a number that can be stored in a few bits of data. The category set can be stored in a bitstring that has a bit for each possible category. Both these are reasonably modest in size. A system capable of 8 sensitivity levels and 256 categories would require only 33 bytes to store this fixed length data. However, the set of users who are to be permitted access and the set of users who are to be denied access can in principle be long and can change dynamically and frequently. Normally this data is stored in a compulsory ancillary file associated with the data. This is the previously mentioned access control list (ACL).

3.1.3. CLASS TWO – ACCOUNTABILITY

REQUIREMENT THREE
CLASS TWO

IDENTIFICATION

Individual subjects must be identified.

Each access to information must be mediated. The mediation will involve who is proposing access and what their access authorisations are. Thus, there must exist a file of user parameters, including data permitting identification with sufficient accuracy. The identification and authorisation information must be securely protected.

This requirement implies the existence of a reference monitor that contemplates any proposed information flow. If the security level of the destination file dominates the security level of every file that is open for reading, then data can be allowed to flow to the destination. Otherwise there is the possibility of an unauthorised leak. For instance, suppose two files are open for reading, a (red) one and a (white) one. Any file that is opened for writing must be a (red, white) one. Otherwise, there is the possibility of directly or indirectly writing (red) data in a (white) file, or conversely.

The security level of the subject that triggers the write is not in principle relevant to the permissibility of the write, unless that subject could manipulate the data in transit. This can arise, for instance, if the subject reads from a file into its memory and then writes the data out. After the write the file will still be in memory. In this case the security level of the subject must dominate the source file and must be dominated by the destination file. This is the normal case.

This set of simple rules was imported almost unmodified from a time long before computers could manipulate data. It has the primary virtue of being simple to understand and simple to use[11]. It can be extended to easily cover systems that include with the dominance determination issues such as the maximum level a user is cleared to, the minimum level a user is allowed to operate at and the actual level he or she has assumed. Users and their agents usually operate all at one level. Few users ever actually read data from a lower level, or write up.

The reference monitor requires the identity of the object that is the source of the proposed access, the identity of the object that is the destination of the proposed access and the identity of the users accountable for the objects as well as the access rights of these entities. It will require the identity of a subject (a subject may be part of a process, as distinct from an executing object) proposing an access. Every access must be mediated in this fashion. The reference monitor must govern all proposed accesses by every subject. For efficiency reasons, the reference monitor normally will do its mediation at the first of possibly multiple references by a subject to an object.

REQUIREMENT FOUR
CLASS TWO

ACCOUNTABILITY

Audit information must be selectively kept and protected so that actions affecting security can be traced to the responsible person.

The journal of security-related events that is kept for audit is the second-most important security protection mechanism in the system, after the login perimeter. In almost all secure systems, governmental or business, it is more important to detect all attempts to breach the security policy, successful or not, than it is to prevent some of them. The

11 Any security mechanism that is not simple to understand and simple to use is doomed to fail.

disclosure of data in an unauthorised way can be countered only if the security officers are aware of it.

Sometimes the journal of security-related events can be monitored in real-time to attempt to identify isolated problems, such as an unusual number of failed login attempts from some source. Certainly, an audit of the journal will require algorithmic assistance, because the volume of data can be daunting and the evidence being sought is likely to be subtle.

3.1.4. CLASS THREE – ASSURANCE

REQUIREMENT FIVE CLASS THREE
ASSURANCE
The computer system must contain hardware and software mechanisms that can be independently evaluated to provide sufficient assurance that the system enforces requirements one through four above.

Assurance usually gets confused with guarantee. An assurance is a believable assertion that things should not go wrong and a commitment to fix the system if they do go wrong. A guarantee is a useless protection against unauthorised observation. That which has been seen, particularly when it is unauthorised, is unlikely to be ignored or forgotten. Documented guarantees are accompanied by emphatic assertions about the good intentions and the resolve of the guarantor. An assurance is a demonstration that some level of confidence can safely be placed in the guarantee. The distinction is important and the assurance is the important thing, because the security policy establishes the required guarantee.

All the Generally Accepted Standards of Good Practice (GASGPS) that have been developed in general engineering practice and in software engineering must be deployed to develop a high level of confidence that the system meets the policy. Without going into much detail, a system that could be considered to have high assurance of enforcing the policy is one that

- has been specified to meet some particular requirement and that
- has been designed to meet the specification and that
- has the right hardware configuration and that
- has been documented thoroughly before it is built, including testing and that

- has been written by disciplined programmers according to the specification and that

- passes all tests and that

- is highly resistant to penetration by a team of experienced security personnel (often called a tiger team, for unknown reasons) even when they are provided with the entire specification, the documentation and the source code.

All this, regrettably, is not enough in the most sensitive case. The assurance developed by the method described above finally boils down to proof by emphatic assertion. No absolutely convincing proof, in the meaning of the word in logic, will have been developed. As will be shown later, in the most demanding cases it is necessary to construct an abstract mathematical model of the system and to show that

- once the abstract model of the system gets into a secure state it can not be driven into a state where insecurity is possible and that

- the actual system is an instance of the abstract model and conversely and that

- the many details that erupt when the abstraction is solidified are not in themselves a problem and can be proven to be consistent with the abstraction and that

- the system can fail and recover from failure securely.

We will look into abstract models in a later essay.

REQUIREMENT SIX
CLASS THREE

CONTINUOUS PROTECTION

The trusted mechanisms that enforce these basic requirements must be continuously protected against tampering or unauthorised changes.

This merely states the obvious. The system must protect itself against meddlers. Much of the material discussed above in requirement five applies directly to this case. Requirement six applies to the entire development cycle and can at the higher levels of assurance involve detailed methodologies to be used at the time software is written to guard against the inclusion of Trojan horses or other unwanted excrescences.

3.2. The Orange Book

The following sections are reproduced more or less verbatim from the TRUSTED COMPUTER SYSTEMS EVALUATION CRITERIA (TCSEC). Recently, these criteria have been replaced by criteria common to many NATO countries. However, the principles remain the same.

3.2.1 Summary of Evaluation Criteria Classes

The classes of systems recognised under the trusted computer system evaluation criteria are given next. They are presented in 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.
Class 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 co-operating 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: Labelled 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 labelling 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 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 tamper proof 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 minimising 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 highly 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 developmental 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.

3.2.2 Requirement Directory

This appendix in the TCSEC lists its requirements alphabetically rather than by class. It is provided to assist the reader in following the evolution of a requirement through the classes. For each requirement, three types of criteria may be present. Each will be preceded by the word: NEW, CHANGE, or ADD to indicate the following:

NEW:	Any criteria appearing in a lower class are superseded by the criteria that follow.

CHANGE: The criteria that follow have appeared in a lower class but are changed for this class. Highlighting is used to indicate the specific changes to previously stated criteria.

ADD: The criteria that follow have not been required for any lower class and are added in this class to the previously stated criteria for this requirement.

Abbreviations are used as follows:

NR: (No Requirement) This requirement is included in this class.

NAR: (No Additional Requirements) This requirement does not change from the previous class.

The reader is referred to TCSEC when placing new criteria for a requirement into the complete context for that class. Figure 1 provides a pictorial summary of the evolution of requirements through the classes.

Audit

C1: NR.

C2: NEW: The TCB shall be able to create, maintain and protect from modification or unauthorised access or destruction an audit trail of accesses to the objects it protects. The audit data shall be protected by the TCB so that read access to it is limited to those who are authorised for audit data. The TCB shall be able to record the following types of events: use of identification and authentication mechanisms, introduction of objects into a user's address space (e.g., file open, program initiation), deletion of objects, actions taken by computer operators and system administrators and/or system security officers and other security relevant events. For each recorded event, the audit record shall identify: date and time of the event, user, type of event and success or failure of the event. For identification or authentication events the origin of request (e.g., terminal ID) shall be included in the audit record. For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the object. The ADP system administrator shall be able to audit selectively the actions of any one or more users based on individual identity.

B1 : CHANGE: For events that introduce an object into a user's address space and for object deletion events the audit record shall include the name of the objects <u>and the object's security level</u>. The ADP system administrator shall be able to audit selectively the actions of any one or more users based on individual identity <u>and/or object security level</u>.

ADD: The TCB shall also be able to audit any override of human-readable output markings.

B2 : ADD: The TCB shall be able to audit the identified events that may be used in the exploitation of covert storage channels.

B3 : ADD: The TCB shall contain a mechanism that is able to monitor the occurrence or accumulation of security events that are subject to audit that may indicate an imminent violation of security policy. This mechanism shall be able to notify immediately the security administrator when thresholds are exceeded and, if the occurrence or accumulation of these security relevant events continues, the system shall take the least disruptive action to terminate the event.

A1 : NAR.

Configuration Management

C1 : NR.

C2 : NR.

B1 : NR.

B2 : NEW: During development and maintenance of the TCB, a configuration management system shall be in place that maintains control of changes, to the descriptive top-level specification, other design data, implementation documentation, source code, the running version of the object code and test fixtures and documentation. The configuration management system shall assure a consistent mapping among all documentation and code associated with the current version of the TCB. Tools shall be provided for generation of a new version of the TCB from source code. Also available shall be tools for comparing a newly generated version with the previous TCB version in order to ascertain that only the intended changes have been made in the code that will actually be used as the new version of the TCB.

B3 : NAR.

A1 : CHANGE: During the entire life-cycle, i.e., during the design, development and maintenance of the TCB, a configuration management system shall be in place for all security-relevant hardware, firmware and software that maintains control of changes to the formal model, the descriptive and formal top-level specifications, other design data, implementation documentation, source code, the running version of the object code and test fixtures and documentation. Also available shall be tools, maintained under strict configuration control, for comparing a newly generated version with the previous TCB version in order to ascertain that only the intended changes have been made in the code that will actually be used as the new version of the TCB.

ADD: A combination of technical, physical and procedural safeguards shall be used to protect from unauthorised modification or destruction the master copy or copies of all material used to generate the TCB.

Covert Channel Analysis

C1 : NR.

C2 : NR.

B1 : NR.

B2 : NEW: The system developer shall conduct a thorough search for covert storage channels and make a determination (either by actual measurement or by engineering estimation) of the maximum bandwidth of each identified channel. (See the Covert Channels Guideline section.)

B3 : CHANGE: The system developer shall conduct a thorough search for <u>covert channels</u> and make a determination (either by actual measurement or by engineering estimation) of the maximum bandwidth of each identified channel.

A1 : ADD: Formal methods shall be used in the analysis.

Design Documentation

C1 : NEW: Documentation shall be available that provides a description of the manufacturer's philosophy of protection and an explanation of how this philosophy is translated into the TCB. If the TCB is composed of distinct modules, the interfaces between these modules shall be described.

C2 : NAR.

B1 : ADD : An informal or formal description of the security policy model enforced by the TCB shall be available and an explanation provided to show that it is sufficient to enforce the security policy. The specific TCB protection mechanisms shall be identified and an explanation given to show that they satisfy the model.

B2 : CHANGE: The interfaces between <u>the TCB</u> modules shall be described. A <u>formal</u> description of the security policy model enforced by the TCB shall be available and <u>proven</u> that it is sufficient to enforce the security policy.

ADD: The descriptive top-level specification (DTLS) shall be shown to be an accurate description of the TCB interface. Documentation shall describe how the TCB implements the reference monitor concept and give an explanation why it is tamper resistant, cannot be bypassed and is correctly implemented. Documentation shall describe how the TCB is structured to facilitate testing and to enforce least privilege. This documentation shall also present the results of the covert channel analysis and the trade-offs involved in restricting the channels. All events that are subject to audit and that may be used in the exploitation of known covert storage channels shall be identified. The bandwidths of known covert storage channels, the use of which is not detectable by the auditing mechanisms, shall be provided. (See the Covert Channel Guideline section.)

B3 : ADD: The TCB implementation (i.e., in hardware, firmware and software) shall be informally shown to be consistent with the DTLS. The elements of the DTLS shall be shown, using informal techniques, to correspond to the elements of the TCB.

A1 : CHANGE: The TCB implementation (i.e., in hardware, firmware and software) shall be informally shown to be consistent with the formal top-level specification (FTLS). The elements of the FTLS shall be shown, using informal techniques, to correspond to the elements of the TCB.

ADD: Hardware, firmware and software mechanisms not dealt with in the FTLS but strictly internal to the TCB (e.g., mapping registers, direct memory access I/O) shall be clearly described.

Design Specification and Verification

C1 : NR.

C2 : NR.

B1 : NEW : An informal or formal model of the security policy supported by the TCB shall be maintained over the life cycle of the ADP system and demonstrated to be consistent with its axioms.

B2 : CHANGE :A formal model of the security policy supported by the TCB shall be maintained over the life cycle of the ADP system that is proven consistent with its axioms.

ADD: A descriptive top-level specification (DTLS) of the TCB shall be maintained that completely and accurately describes the TCB in terms of exceptions, error messages and effects. It shall be shown to be an accurate description of the TCB interface.

B3 : ADD : A convincing argument shall be given that the DTLS is consistent with the model.

A1 : CHANGE: The FLTS shall be shown to be an accurate description of the TCB interface. A convincing argument shall be given that the DTLS is consistent with the model and a combination of formal and informal techniques shall be used to show that the FTLS is consistent with the model.

ADD: A formal top-level specification (FTLS) of the TCB shall be maintained that accurately describes the TCB in terms of exceptions, error messages and effects. The DTLS and FTLS shall include those components of the TCB that are implemented as hardware and/or firmware if their properties are visible at the TCB interface. This verification evidence shall 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. Manual or other mapping of the FTLS to the TCB source code shall be performed to provide evidence of correct implementation.

Device Labels

C1 : NR.

C2 : NR.

B1 : NR.

B2 : NEW: The TCB shall support the assignment of minimum and maximum security levels to all attached physical devices. These security levels shall be used by the TCB to enforce constraints imposed by the physical environments in which the devices are located.

B3 : NAR.

A1 : NAR.

Discretionary Access Control

C1 : NEW: The TCB shall define and control access between named users and named objects (e.g., files and programs) in the ADP system. The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals or defined groups or both.

C2 : CHANGE: The enforcement mechanism (e.g., self/group/public controls, access control lists) shall allow users to specify and control sharing of those objects by named individuals, or defined groups <u>of individuals</u>, or by both and<u> shall provide controls to limit propagation of access rights</u>.

ADD: The discretionary access control mechanism shall, either by explicit user action or by default, provide that objects are protected from unauthorised access. These access controls shall be capable of including or excluding access to the granularity of a single user. Access permission to an object by users not already possessing access permission shall only be assigned by authorised users.

B1 : NAR.

B2 : NAR.

B3 : CHANGE: The enforcement mechanism (e.g., access control lists) shall allow users to specify and control sharing of those objects and shall provide controls to limit propagation of access rights. These access controls shall be capable of specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective modes of access to that object.

ADD: Furthermore, for each such named object, it shall be possible to specify a list of named individuals and a list of groups of named individuals for which no access to the object is to be given.

A1 : NAR.

Exportation of Labelled Information

C1 : NR.

C2 : NR.

B1 : NEW: The TCB shall designate each communication channel and I/O device as either single-level or multilevel. Any change in this designation shall be done manually and shall be audited by the TCB. The TCB shall maintain and be able to audit any change in the security level or levels associated with a communication channel or I/O device.

B2 : NAR.

B3 : NAR.

A1 : NAR.

Exportation to Multilevel Devices

C1 : NR.

C2 : NR.

B1 : NEW: When the TCB exports an object to a multilevel I/O device, the sensitivity label associated with that object shall also be exported and shall reside on the same physical medium as the exported information and shall be in the same form (i.e., machine-readable or human-readable form). When the TCB exports or imports an object over a multilevel communication channel, the protocol used on that channel shall provide for the unambiguous pairing between the sensitivity labels and the associated information that is sent or received.

B2 : NAR.

B3 : NAR.

A1 : NAR.

Exportation to Single-Level Devices

C1 : NR.

C2 : NR.

B1 : NEW: Single-level I/O devices and single-level communication channels are not required to maintain the sensitivity labels of the information they process. However, the TCB shall include a mechanism by which the TCB and an authorised user reliably communicate to designate the single security level of information imported or exported via single-level communication channels or I/O devices.

B2 : NAR.

B3 : NAR.

A1 : NAR.

Identification and Authentication

C1 : NEW: The TCB shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall use a protected mechanism (e.g., passwords) to authenticate the user's identity. The TCB shall protect authentication data so that any unauthorised user cannot access it.

C2 : ADD: The TCB shall be able to enforce individual accountability by providing the capability to identify uniquely each individual ADP system user. The TCB shall also provide the capability of associating this identity with all audited actions taken by that individual.

B1 : CHANGE: Furthermore, the TCB shall <u>maintain authentication data that includes information for verifying the identity of individual users (e.g., passwords) as well as information for determining the clearance and authorisations of individual users. This data shall be used by the TCB to authenticate the user's identity and to ensure that the security level and authorisation of subjects external to the TCB that may be created to act on behalf of the individual user are dominated by the clearance and authorisation of that user</u>.

B2 : NAR.

B3 : NAR.

A1 : NAR.

Label Integrity

C1 : NR.

C2 : NR.

B1 : NEW: Sensitivity labels shall accurately represent security levels of the specific subjects or objects with which they are associated. When exported by the TCB, sensitivity labels shall accurately and unambiguously represent the internal labels and shall be associated with the information being exported.

Labeling Human-Readable Output

C1 : NR.

C2 : NR.

B1 : NEW: The ADP system administrator shall be able to specify the printable label names associated with exported sensitivity labels. The TCB shall mark the beginning and end of all human-readable, paged, hardcopy output (e.g., line printer output) with human-readable sensitivity labels that properly[12] represent the sensitivity of the output. The TCB shall, by default, mark the top and bottom of each page of human-readable, paged, hardcopy output (e.g., line printer output) with human-readable sensitivity labels that properly represent the overall sensitivity of the output or that properly represent the sensitivity of the information on the page. The TCB shall, by default and in an appropriate manner, mark other forms of human-readable output (e.g., maps, graphics) with human-readable sensitivity labels that properly represent the sensitivity of the output. Any override of these marking defaults shall be audited by the TCB.

B2 : NAR.

B3 : NAR.

A1 : NAR.

Labels

C1 : NR.

C2 : NR.

12 The hierarchical classification component in human-readable sensitivity labels shall be equal to the greatest hierarchical classification of any of the information in the output that the labels refer to; the non-hierarchical category component shall include all of the non-hierarchical categories of the information in the output the labels refer to, but no other non-hierarchical categories.

B1 : NEW: Sensitivity labels associated with each subject and storage object under its control (e.g., process, file, segment, device) shall be maintained by the TCB. These labels shall be used as the basis for mandatory access control decisions. In order to import non-labelled data, the TCB shall request and receive from an authorised user the security level of the data and all such actions shall be audited by the TCB.

B2 : CHANGE: Sensitivity labels associated with each <u>ADP system resource (e.g., subject, storage object, ROM) that is directly or indirectly accessible by subjects external to the TCB</u> shall be maintained by the TCB.

B3 : NAR.

A1 : NAR.

Mandatory Access Control

C1 : NR.

C2 : NR.

B1 : NEW: The TCB shall enforce a mandatory access control policy over all subjects and storage objects under its control (e.g., processes, files, segments, devices). These subjects and objects shall be assigned sensitivity labels that are a combination of hierarchical classification levels and non-hierarchical categories and the labels shall be used as the basis for mandatory access control decisions. The TCB shall be able to support two or more such security levels. (See the Mandatory Access Control guidelines.) The following requirements shall hold for all accesses between subjects and objects controlled by the TCB:

A subject can read an object only if the hierarchical classification in the subject's security level is greater than or equal to the hierarchical classification in the object's security level and the non-hierarchical categories in the subject's security level include all the non-hierarchical categories in the object's security level. A subject can write an object only if the hierarchical classification in the subject's security level is less than or equal to the hierarchical classification in the object's security level and all the non-hierarchical categories in the subject's security level are included in the non-hierarchical categories in the object's security level. Identification and authentication data shall be used by the TCB to authenticate the user's identity and to ensure that the security level and authorisation of subjects external to the TCB that may be created to act on behalf of the individual user are dominated by the clearance and authorisation of that user.

B2 : CHANGE: The TCB shall enforce a mandatory access control policy over all resources (i.e., subjects, storage objects and I/O devices) that are directly or indirectly accessible by subjects external to the TCB. The following requirements shall hold for all accesses between all subjects external to the TCB and all objects directly or indirectly accessible by these subjects:

B3 : NAR.

A1 : NAR.

Object Reuse

C1 : NR.

C2 : NEW : All authorisations to the information contained within a storage object shall be revoked prior to initial assignment, allocation or reallocation to a subject from the TCB's pool of unused storage objects. No information, including encrypted representations of information, produced by a prior subject's actions is to be available to any subject that obtains access to an object that has been released back to the system.

B1 : NAR.

B2 : NAR.

B3 : NAR.

A1 : NAR.

Security Features User's Guide

C1 : NEW : A single summary, chapter, or manual in user documentation shall describe the protection mechanisms provided by the TCB, guidelines on their use and how they interact with one another.

C2 : NAR.

B1 : NAR.

B2 : NAR.

B3 : NAR.

A1 : NAR.

Security Testing

C1 : NEW: The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. Testing shall be done to assure that there are no obvious ways for an unauthorised user to bypass or otherwise defeat the security protection mechanisms of the TCB. (See the Security Testing guidelines.)

C2 : ADD: Testing shall also include a search for obvious flaws that would allow violation of resource isolation, or that would permit unauthorised access to the audit or authentication data.

B1 : NEW: The security mechanisms of the ADP system shall be tested and found to work as claimed in the system documentation. A team of individuals who thoroughly understand the specific implementation of the TCB shall subject its design documentation, source code and object code to thorough analysis and testing. Their objectives shall be: to uncover all design and implementation flaws that would permit a subject external to the TCB to read, change, or delete data normally denied under the mandatory or discretionary security policy enforced by the TCB; as well as to assure that no subject (without authorisation to do so) is able to cause the TCB to enter a state such that it is unable to respond to communications initiated by other users. All discovered flaws shall be removed or neutralised and the TCB re-tested to demonstrate that they have been eliminated and that new flaws have not been introduced. (See the Security Testing Guidelines.)

B2 : CHANGE :All discovered flaws shall be <u>corrected</u> and the TCB re-tested to demonstrate that they have been eliminated and that new flaws have not been introduced.

ADD: The TCB shall be found relatively resistant to penetration. Testing shall demonstrate that the TCB implementation is consistent with the descriptive top-level specification.

B3 : CHANGE: The TCB shall be <u>found resistant to</u> penetration.

ADD: No design flaws and no more than a few correctable flaws may be found during testing and there shall be reasonable confidence that few remain.

A1 : CHANGE: Testing shall demonstrate that the TCB implementation is consistent with the <u>formal</u> top-level specification.

ADD: Manual or other mapping of the FTLS to the source code may form a basis for penetration testing.

Subject Sensitivity Labels

C1 : NR.

C2 : NR.

B1 : NR.

B2 : NEW: The TCB shall immediately notify a terminal user of each change in the security level associated with that user during an interactive session. A terminal user shall be able to query the TCB as desired for a display of the subject's complete sensitivity label.

B3 : NAR.

A1 : NAR.

System Architecture

C1 : NEW: The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures). Resources controlled by the TCB may be a defined subset of the subjects and objects in the ADP system.

C2 : ADD: The TCB shall isolate the resources to be protected so that they are subject to the access control and auditing requirements.

B1 : ADD: The TCB shall maintain process isolation through the provision of distinct address spaces under its control.

B2 : NEW: The TCB shall maintain a domain for its own execution that protects it from external interference or tampering (e.g., by modification of its code or data structures). The TCB shall maintain process isolation through the provision of distinct address spaces under its control. The TCB shall be internally structured into well-defined largely independent modules. It shall make effective use of available hardware to separate those elements that are protection-critical from those that are not. The TCB modules shall be designed such that the principle of least privilege is enforced. Features in hardware, such as segmentation, shall be used to support logically distinct storage objects with separate attributes (namely: readable, writeable). The user interface to the TCB shall be completely defined and all elements of the TCB identified.

B3 : ADD: The TCB shall be designed and structured to use a complete, conceptually simple protection mechanism with precisely defined semantics. This mechanism shall play a central role in enforcing the internal structuring of the TCB and the system. The TCB shall incorporate significant use of layering, abstraction and data hiding. Significant system engineering shall be directed toward minimising the complexity of the TCB and excluding from the TCB modules that are not protection-critical.

A1 : NAR.

System Integrity

C1 : NEW: Hardware and/or software features shall be provided that can be used periodically to validate the correct operation of the non-site hardware and firmware elements of the TCB.

C2 : NAR.

B1 : NAR.

B2 : NAR.

B3 : NAR.

A1 : NAR.

Test Documentation

C1 : NEW: The system developer shall provide to the evaluators a document that describes the test plan, test procedures that show how the security mechanisms were tested and results of the security mechanisms' functional testing.

C2 : NAR.

B1 : NAR.

B2 : ADD: It shall include results of testing the effectiveness of the methods used to reduce covert channel bandwidths.

B3 : NAR.

A1 : ADD: The results of the mapping between the formal top-level specification and the TCB source code shall be given.

Trusted Distribution

C1 : NR.

C2 : NR.

B1 : NR.

B2 : NR.

B3 : NR.

A1 : NEW : A trusted ADP system control and distribution facility shall be provided for maintaining the integrity of the mapping between the master data describing the current version of the TCB and the on-site master copy of the code for the current version. Procedures (e.g., site security acceptance testing) shall exist for assuring that the TCB software, firmware and hardware updates distributed to a customer are exactly as specified by the master copies.

Trusted Facility Management

C1 : NR.

C2 : NR.

B1 : NR.

B2 : NEW: The TCB shall support separate operator and administrator functions.

B3 : ADD: The functions performed in the role of a security administrator shall be identified. The ADP system administrative personnel shall only be able to perform security administrator functions after taking a distinct audited action to assume the security administrator role on the ADP system. Non-security functions that can be performed in the security administration role shall be limited strictly to those essential to performing the security role effectively.

A1 : NAR.

Trusted Facility Manual

C1 : NEW : A manual addressed to the ADP system administrator shall present cautions about functions and privileges that should be controlled when running a secure facility.

C2 : ADD: The procedures for examining and maintaining the audit files as well as the detailed audit record structure for each type of audit event shall be given.

B1 : ADD: The manual shall describe the operator and administrator functions related to security, to include changing the security characteristics of a user. It shall provide guidelines on the consistent and effective use of the protection features of the system, how they interact, how to generate securely a new TCB and facility procedures, warnings and privileges that need to be controlled in order to operate the facility in a secure manner.

B2 : ADD: The TCB modules that contain the reference validation mechanism shall be identified. The procedures for secure generation of a new TCB form source after modification of any modules in the TCB shall be described.

B3 : ADD: It shall include the procedures to ensure that the system is initially started in a secure manner. Procedures shall also be included to resume secure system operation after any lapse in system operation.

A1 : NAR.

Trusted Path

C1 : NR.

C2 : NR.

B1 : NR.

B2 : NEW: The TCB shall support a trusted communication path between itself and user for initial login and authentication. Communications via this path shall be initiated exclusively by a user.

B3 : CHANGE: The TCB shall support a trusted communication path between itself and users for use when a positive TCB-to-user connection is required (e.g., login, change subject security level). Communications via this trusted path shall be activated exclusively by a user or the TCB and shall be logically isolated and unmistakably distinguishable from other paths.

A1 : NAR.

Trusted Recovery

C1 : NR.

C2 : NR.

B1 : NR.

B2 : NR.

B3 : NEW: Procedures and/or mechanisms shall be provided to assure that, after an ADP system failure or other discontinuity, recovery without a protection compromise is obtained.

A1 : NAR.

3.2.3 Summary of the TCSEC

Figure 1.: Summary of the TCSEC

	C1	C2	B1	B2	B3	A1
SECURITY POLICY						
Discretionary Access Control	▒	▒	□	□	▒	□
Object Reuse	■	▒	□	□	□	□
Labels	■	■	▒	▒	□	□
Label Integrity	■	■	▒	□	□	□
Exportation of labelled information	■	■	▒	□	□	□
Exportation to Multi-Level Devices	■	■	▒	□	□	□
Exportation to Single-Level Devices	■	■	▒	□	□	□
Labelling Human-Readable Output	■	■	▒	□	□	□
Mandatory Access Control	■	■	▒	▒	□	□
Subject Sensitivity Labels	■	■	■	▒	□	□
Device Labels	■	■	■	▒	□	□
ACCOUNTABILITY						
Identification and Authentication	▒	▒	▒	□	□	□
Audit	■	▒	▒	▒	▒	□
Trusted Path	■	■	■	▒	▒	□
ASSURANCE						
System Architecture	▒	▒	▒	▒	▒	□
System Integrity	▒	□	□	□	□	□
Security Testing	▒	▒	▒	▒	▒	▒
Design Specification and Verification	■	■	▒	▒	▒	□
Covert Channel Analysis	■	■	■	▒	▒	□
Trusted Facility Management	■	■	■	▒	▒	□
Configuration Management	■	■	■	▒	□	▒
Trusted Recovery	■	■	■	■	▒	□
Trusted Distribution	■	■	■	■	■	▒
DOCUMENTATION						
Security Features User's Guide	▒	□	□	□	□	□
Trusted Facility Manual	▒	▒	▒	▒	▒	□
Test Documentation	▒	□	□	▒	□	▒
Design Documentation	▒	□	▒	▒	▒	▒

LEGEND

No additional requirements	□
New or enhanced requirements	▒
No requirements	■

3.3. A Formal Model of Access Systems

The material presented here is modelled on the work of Harrison, Ruzzo and Ullman[13]. Here, the work has been updated and modified to be compatible with modern terminology. The original paper concludes with some analytical results that show that, in effect, a determination of whether an access privilege has been confined is an NP-complete problem. This result is interesting but somewhat dusty — DAC is still used with effectiveness. Work by Sandhu[14] extends the notions developed here.

This descriptive power and elegance of the HRU model of access systems is still significant. A description of an access system using HRU is close to a specification of it.

Definition: An *access system* consists of the following parts:

1. An access matrix A with a row for every subject S and a column for every object O, containing the permitted accesses of subject S to object O. Elements of A are designated by the row-column pair $[X_s, X_o]$. The elements of A are members of a set of privileges R. For our purposes $R = \{\hat{o}, r, w, x\}$. These mean, respectively, own, read, write and execute.

2. A set of (DAC) commands of the form

```
command $\alpha(X_1, X_2, \cdots, X_k)$
  if
    $r_1$ in $[X_{s_1}, X_{o_1}]$ and
    $r_2$ in $[X_{s_2}, X_{o_2}]$ and
    •••
    $r_m$ in $[X_{s_m}, X_{o_m}]$ and
    $X_{o_1}$ has some identified property and
    $X_{o_2}$ has some identified property and
    •••
    $X_{o_m}$ has some identified property
  then
    $op_1$
    $op_2$

    $op_n$
  end $\alpha$
```

In this command, α is the name of the command. Each $r_j \in R$, the X_j are the parameters of the command, the X_{s_j} and X_{o_j} are both

13 M.A. Harrison, W.L. Ruzzo, and J.D. Ullman, "Protection in Operating Systems", *CACM*, Vol. 19, No. 8, 1976 August, pp. 461-71.

14 Ravi Sandhu, Esorics 1994, Brighton.

parameters, respectively identifying a subject and an object and the op_j are each one of the set of primitive operations in the table below. Let $r \in R$.

enter r **into** $[X_{s_j}, X_{o_j}]$	**create object** X_{o_j}
delete r **from** $[X_{s_j}, X_{o_j}]$	**destroy subject** X_{s_j}
create subject X_{s_j}	**destroy object** X_{o_j}

The primitive operations are assumed to be protected, trusted and to be indivisible and uninterruptible. We will see later that the primitive operations are included in the TCB and that many of the commands need to be there too.

The predicate following the **if** in the command is called the condition of the command. It is constructed from an intersection of sub-conditions that are each presumed to be indivisible and uninterruptible. The series of primitive operations is called the body Of the command.

Definition: A *configuration* is the triple (S, O, A).

A configuration of the access system is the triple (S, O, A). Each member of S has a unique member of O that directly corresponds to it. There are additional members of O that can never become subjects, or that can become subjects but are not one at this time. The later are called executable objects.

3.3.1. The Trusted Subjects

The commands that exist form a set of trusted subjects. The following is an example list of possible trusted subjects.

A subject can always create a file:

```
command CREATE_FILE $(X_{s,} X_o)$
    create object $X_o$
    enter $\hat{o}$ into $[X_s, X_o]$
end CREATE_FILE
```

A subject can create another subject if it has the proper access. The created subject needs to be given access to some data before it can do anything. This is one of the places that the confinement problem arises. There has never been a satisfactory rule developed that will properly allocate a domain of execution to confine a privilege. In the following the subject s_2 is the name of the subject created by subject s_1 when it asks for object o to be executed.

```
command CREATE_SUBJECT ($X_{s_1}, X_o, X_{s_2}$)
    if
        $\hat{o}$ in [$X_{s_1}, X_o$] and
        $x$ in [$X_{s_1}, X_o$]
    then
        create subject $X_{s_2}$
end CREATE_SUBJECT
```

A subject s_1 can confer a privilege to access some object o on a subject s_2.

```
command CONFER_READ ($X_{s_1}, X_o, X_{s_2}$)
    if
        $\hat{o}$ in [$X_{s_1}, X_o$]
    then
        enter $r$ into [$X_{s_2}, X_o$]
end CONFER_READ
```

Any of the privileges $\{r, w, x\} \in R$ can be passed on in this way. Any previously conferred privilege can be revoked with an analogous command.

```
command REVOKE_READ ($X_{s_1}, X_o, X_{s_2}$)
    if
        $\hat{o}$ in [$X_{s_1}, X_o$] and
        $r$ in [$X_{s_2}, X_o$]
    then
        delete $r$ from [$X_{s_2}, X_o$]
end REVOKE_READ
```

A subject s_1 can transfer ownership $\hat{o}$ of some object o to a subject s_2. This is limited in the obvious way because only one subject can own any object.

```
command CONFER_OWN ($X_{s_1}, X_o, X_{s_2}$)
    if
        $\hat{o}$ in [$X_{s_1}, X_o$]
    then
        enter $\hat{o}$ into [$X_{s_2}, X_o$]
        delete $\hat{o}$ into [$X_{s_1}, X_o$]
end CONFER_OWN
```

A subject s can always remove a file o that it owns. This leaves dangling the issue of revoking all accesses by other subjects to the file. These will be presumed not to exist. Otherwise the privileges all other subjects would need to be polled. This is an example of the advantages of access control lists, stored with the file and vanishing with the file, over capabilities.

```
command REMOVE_FILE ($X_{s,}X_o$)
    if
        $\hat{o}$ in [$X_{s_1}$, $X_o$]
    then
        destroy object $X_o$
end REMOVE_FILE
```

The removal of the file o above is not without difficulties. Several subjects may have been given access rights to the file. When it is removed there is no obviously efficient way to revoke these rights. In modern systems this deficiency is overcome by keeping with the file an access control list (ACL) that lists all subjects (users) that have access rights. At least in theory if the object is removed the access control list can be used to destroy any outstanding rights.

A subject s_1 can remove another subject s_2 if it has the proper access. This leaves dangling the issue of revoking all accesses by the destroyed subject to any objects it may have permission to access. As long as no subsequently created subject can assume the identity of the removed subject there should be no security problem, at least in theory. Cluttering up the access matrix with latent privileges belonging to dead (perhaps long dead) subjects does not seem sensible. Some method of revocation is needed. Of course, if capabilities are being used they will all disappear with the subject s_2. In the following, the subject s_2 is assumed to have previously resulted from the execution of the object o by s_1.

```
command REMOVE_SUBJECT ($X_{s_1,}X_o, X_{s_2}$)
    if
        $\hat{o}$ in [$X_{s_1}$, $X_o$] and
        $x$ in [$X_{s_1}$, $X_o$]
    then
        destroy subject $X_{s_2}$
end REMOVE_SUBJECT
```

3.3.2. Command Semantics

It is possible to give precise mathematical meaning to each of the primitive operations. The starting point is the configuration (S, O, A). After the operation the configuration becomes (S', O', A'). Let $s \in S$ and $o \in O$. An example for CONFER_READ is given. The rest are left as an exercise.

$(S, O, A) \Rightarrow (S', O', A')$ if op is the primitive operation enter r into $[X_s, X_o]$.

$S' = S, \quad O' = O, \quad A'_{s_i,o_i} = A_{s_i,o_i}$ if $s_i, o_i \neq s, o$ and $A'_{s,o} = A_{s,o} \cup \{r\}$.

3.3.3. Access Control

The command set that can be constructed from these primitive operations can implement an effective DAC access control system. The number of privileges might need to be expanded to include, for example, the privilege append as well as the privilege write. As a second example, often the privilege search (frequently coded as execute) is given to a subject for an object that is a directory. This is in itself meaningless and is used to imply that the subject has the privilege to search the directory. Gligor *et al*[15] use the four privileges read, write, execute and null. Other privileges are contrived from these with considerable difficulty.

To make such a system usable in practice it should be able to confine access privilege propagation. We will reason about this issue here.

Consider a multi-pass compiler. Such a program is commonplace and mundane. It should not need special privileges to do its job. When a user invokes the compiler it must have conferred upon it at least read access privilege for the source program. This is easy to do. The user owns the source program and can perform a command similar to the CONFER_READ command shown above. The first pass of the compiler will be able to read the program it is supposed to be compiling.

The first pass of the compiler produces some intermediate output, intended as input for the second pass. This output must be written, so whoever owns it must grant write privilege to the compiler. Who owns this output? It is not legitimate for the compiler to own it, because ownership must be traceable to a user if accountability is to be effective and the owner of the compiler is the compiler writer or his or her employer, or perhaps the superuser in some guise. The logical owner of the intermediate output is the user for whom the compiler is working. This user must somehow assign appropriate privileges to the succession of compiler passes, presumably without responding to frequent and obscure requests for privilege from the compiler[16]. Thus the compiler must pass on the necessary privileges on behalf of the user.

This need for the ability to pass privilege is the origin of the confinement problem. Just about any program that is more than trivial will need some privilege to receive input, to deal with files, or to return output. If the program is a trusted system program it can be guaranteed with assurance that the privileges will be confined. If the program is homemade, or if it is as complicated as typical compilers are, the guarantee can be asserted but the assurance is doubtful at best.

---

15 Virgil Gligor with eight other authors, "Design and Implementation of Secure Xenix", IEEE *Transactions on Software Engineering*, Vol. SE-13, No. 2, February 1987, pages 208-221. Xenix is a dialect of UNIX System V.

16 These requests would constitute a violation of the principle of ease of safe use. Almost no users would have the knowledge to decide whether any given request is sensible or possibly evidence of an intruder.

3.3.4. Delegated Ownership

A number of proposals[17] have been made to delegate from the owner of an object some of his or her privilege-passing authority to an agent. Thus, the owner of a source program might pass "temporary ownership" to the compiler. The compiler could proceed to do whatever it needed and could renounce its temporary ownership when it is finished.

While this process can be presented as progress, it exacerbates the confinement problem by hiding it behind a veil of delegated ownership. The user can only guess at what the agent is doing behind the veil. Attempts to limit the agent's delegated ownership privileges – the beginning of the delegation of <u>least privilege</u> – have not yet proven successful.

3.3.5. Avoiding Confinement Problems

Can the command set be constructed so that, if confinement can not be escaped problems with it can at least be detected? Recall the general structure of a command:

```
command $\alpha(X_1, X_2, \cdots, X_k)$
    if
        $r_1$ in [$X_{s_1}, X_{o_1}$] and
        $r_2$ in [$X_{s_2}, X_{o_2}$] and
        ...
        $r_m$ in [$X_{s_m}, X_{o_m}$] and
        $X_{o_1}$ has some identified property and
        $X_{o_2}$ has some identified property and
        ...
        $X_{o_m}$ has some identified property
    then
        $op_1$
        $op_2$

        $op_n$
    end $\alpha$
```

If there is only one sub-condition in the predicate of each command to determine whether to execute the command body, then whether to execute the command can be established without fear of interruption. The sequence of primitive operations in the command body can be ordered so as to withdraw privilege before reassigning it, so there need not be an inconsistent or delicate state between primitive operations. Thus, if a set of one-condition commands could be produced, an examination of the allocation decisions for all privileges can be developed. The allocation decisions may not have actually happened at some time, but the delegation of any privilege can be predicted and with some difficulty

17 See Butler Lampson's fine rebuttal of these proposals in his 1973 paper "A Note on the Confinement Problem", CACM, Vol. 16, No. 10, October 1973. Pages 613-615.

controlled. Regrettably, a set of commands that each has only one sub-condition implies a powerset growth in the number of privileges. This is not helpful.

If there are several sub-conditions in the commands, whether to execute the command body can not be established without fear of interruption. The several sub-conditions can change during the examination of the condition. Avoidance of this implies freezing the allocation of the relevant privileges until they have all been examined. How long the examination will take is unpredictable because interruptions can happen between sub-conditions. It is impractical for two reasons to freeze the allocation of a privilege until all conditions that have been invoked are complete. Firstly, deadlock is a danger. Secondly, it would be extremely expensive. An elaboration of the latter argument is the basis of the NP-complete result of Harrison, Ruzzo and Ullman.

3.3.6. DAC Assessed

DAC does have its uses. It is impotent against Trojan horses and the other animals in the attacking zoo. It can, however, be very helpful in preventing accidents and in low-grade confidentiality circumstances. It is easily understood by users. It is and will continue to be an important tool in computer security implementation.

The descriptive mechanism called an access system remains a powerful design, analysis and planning tool for those of us who want to understand how a given system works

BUILDING ASSURANCE

4. BUILDING ASSURANCE

The software bears the responsibility for many of the security requirements. The hardware provides a way that processes can run in independent domains, with secure process switching, no covert storage channels and very slow covert timing channels. Software, and particularly the Trusted Computing Base (TCB) provides the rest.

4.1. The Reference Monitor

Modern security system implementations all use an abstract concept that is called the reference monitor. It is an abstract mechanism that mediates all accesses by subjects to objects. The reference monitor determines if an access being made by the TCB satisfies both the DAC and MAC requirements, or if it is a reference from a trusted subject that has privileges beyond those accorded normal processes. The TCB is the site of all such trusted subjects with enhanced privileges.

In fact there is no one part of any truly secure system that is the reference monitor. It is diffused throughout the software and hardware. The process isolation enforced by the hardware is a part of it. The software mechanism that compares two security levels to evaluate the dominance relation is another part. Parts of the login mechanism is a third.

4.2. The Mechanisms That Must be Trusted

The trusted computing base is the totality of the mechanisms within a computer system, including all hardware and software, which combine to enforce the security policy, and that must be trusted to work properly. They must be trusted because they do not themselves obey the security policy.

The TCB is evaluated to determine its ability to perform its task. The level of confidence that is developed in the ability of the TCB to function properly is dependent on three things:

- the functionality it provides, and the appropriateness of these functions to enforce the policy;

- the degree of confidence that can be developed that the TCB exactly meets its functionality, no more and no less; and
- the degree of security that must be provided.

The TCB is required to be as small as is practical. It consists of those mechanisms that themselves violate the security policy so that ordinary programs need not. That is why it must be trusted. Small things are easier to test for their trustworthiness that big things are. There can be many degrees of trust, dependent on the following kinds of things:

- The functionality that is provided. A system that provides individual logins only, as contrasted to group logins, is probably inherently more trustworthy.
- The way it is specified. Some formal technique is better than natural language.
- The way it is implemented. The configuration must be managed to try to avoid Trojan horses.
- Whether intuition or mathematical proof have been used to demonstrate its security.
- The way it is distributed and maintained.

4.3. Single Level Operation

Most secure computer systems (and this means computer systems that are actually secure, not those that are thought to be secure) that are running today have the following properties:

- They have never been tested to establish their security, and they would almost certainly fail any such test.
- They have a small number of users, all working on the same problem. (This means the data is all from the same realm of knowledge.)
- The users all know each other, and are all peers of each other. (This means they operate at a single level of sensitivity.)
- The computer and its terminals are in a physically secure location.
- There is no telephone attached to the computer — it is completely self-contained.
- There is no network attached to the computer — it is completely self-contained.

This kind of operation is called single level. It is adequate as long as its environment persists. However, every one of the conditions above is inconsistent with modern computer usage.

4.4. System High

A version of single level operation that can cope with a small cluster of computers that are networked together is called system high. The interconnected systems run at the level of the component with the highest classification. Obviously all users must be cleared to at least this level.

System high works provided that the presence of a network is the only condition in the previous section that is false. And the network must be contained, with no communications outside the cluster. Then, provided every user has at least the same clearance as the classification of the data object with the highest classification, the system can be reasonable secure. This means that, for instance, an isolated workstation/fileserver set-up can be reasonably secure in this environment.

4.5. Building Assurance

There are a number of explicit techniques, other than good engineering practice, that can be used to increase both the implementer's and the user's level of confidence that the system is secure.

4.5.1. Verifying All Programs

The formal examination of all programs is an important part of determining the level of confidence to place in it.

Faithfulness	A faithful program is one that will always have a predictable result, regardless of the input stimulation it receives or the computation it performs. It is neither necessary nor sufficient that the result be numerically accurate. Safety implies that the program conclude within some finite time bound, that it not exit with an unpredictable exception, that it not abort, that it not cause the computer to stop, and that it not return a result that might cause the calling program to be not safe.
Evaluation	To evaluate a program is to measure the degree to which a security mechanism satisfies some given evaluation criteria.
Certification	To certify a security mechanism is to formally guarantee the accuracy, correctness and completeness of the security mechanism, measured against some established evaluation criteria.
Verification	To verify a program is to use a (usually formal) mechanism to establish the formal semantics of a program, in terms of the formal semantics of its components. This always involves software tools, two in particular that are called a verifier and a truth prover. The verifier knows the formal semantics of the various

constructs in the language. Starting with a precondition (often the condition true) it reads the statements of the program, and deduces their semantic effects. The truth prover does a lot of logical reductions on the evolving expression. The whole enterprise concludes with an expression describing the semantics of the program. In appropriate cases, such as subprograms that do not reference non-local variables and do not allow parameters to alias of each other, this expression can then be added to the verifier's dictionary of constructs with known semantics.

Correctness A proof of correctness of a program is a formal demonstration that the (high level) code of a program agrees with some assertions that are included in it. The assertions are coded as assertion statements. They contain a proposition (a Boolean predicate) that must be demonstrably true for any possible values of the variables that are involved. Using some known formal semantics of the statements in the programming language, it is shown that each assertion must be true (in the terminology of mathematical logic, at that point in the program it is a valid theorem). There are four kinds of assertions that are used:

- a pre-condition to establish a basis for reasoning;
- an in-line assertion, that is necessarily true based on the situation that was previously known and the semantics of the statements that could have been executed to get to the point of the assertion, for any combination of values of variables;
- a loop invariant, that is necessarily true for any and all excursions around the body of a loop; and
- a post condition, to establish that the program agrees with its expected semantics.

4.5.2. Certify or Evaluate Critical Programs

The critical programs are those that enforce the security policy. These programs are those that are in the TCB. The necessity of developing a level of confidence in the capabilities of the TCB has already been discussed.

4.5.3. Journal of All Security Relevant Events

Regardless of the confidence that can be placed in the TCB, it is important that a journal be kept of all security-related events, and that there be an analysis of the journal both in real time and historically. It is often more important to know that security has been violated that it is to prevent the violation. This one assertion sums up the seriousness of the threat to security that a Trojan horse represents, because if it is properly done it leaves no evidence of its existence behind it.

4.6. Principles of Trusted Operating Systems

This section is concerned with the general problems of specifying, designing, implementing, and evaluating trustworthy operating systems.

4.6.1. Lattices

Let $\mathbf{L}$ be a finite set with elements $a, b, c, \cdots$. A partial ordering is a relation on the members of a set. The usual symbol for a partial ordering is $\geq$. It is convenient for us to use the symbol $\rhd$ and to pronounce it *dominates.*. The relation $\lhd$ is pronounced *is dominated by*. The relation $\rhd$ has the following properties.

Reflexive	If $a \rhd a$ is always true.
Antisymmetric	If $a \rhd b$ and $b \rhd a$ then a and b have the same level. When dealing with security this is written as $L(a) = L(b)$.
Transitive	$a \rhd b$ and $b \rhd c$ implies that $a \rhd c$.

A partially ordered set (poset) consists of a set, say $\mathbf{L}$, and a partial ordering relation, say $\rhd$, on $\mathbf{L}$. This poset is written as $[\mathbf{L}, \rhd]$. If $[\mathbf{L}, \rhd]$ is a poset then $[\mathbf{L}, \lhd]$ is also a poset.

An element M of a poset $[\mathbf{L}, \rhd]$ is maximal if there is no other element a in $[\mathbf{L}, \rhd]$ for which $a \rhd M$. An element m of a poset $[\mathbf{L}, \rhd]$ is minimal if there is no other element a in $[\mathbf{L}, \rhd]$ for which $m \rhd a$. An element $\mathbf{I}$ is the <u>upper universal bound</u> if $\mathbf{I} \rhd a$ for all $a \in [\mathbf{L}, \rhd]$. An element $\mathbf{O}$ is the <u>lower universal bound</u> if $a \rhd \mathbf{O}$ for all $a \in [\mathbf{L}, \rhd]$.

Example: Consider the relation $\div$ (meaning divides evenly into, without a remainder) on the set $S = \{1, 2, 3, 4, 6, 8, 12\}$. The poset $[S, \div]$ has a lower universal bound but no upper universal bound. Why?

For elements $\{a, b\} \in [\mathbf{L}, \rhd]$ the <u>least upper bound</u> (lub) of a and b is an element c of $[\mathbf{L}, \rhd]$ for which $c \rhd a$ and $c \rhd b$, and there exists no other element $x \in [\mathbf{L}, \rhd]$ for which $c \rhd x \rhd a$ and $c \rhd x \rhd b$. In general, there might be several elements that satisfy the lub condition. If there is only one such element it is called the join of a and b and written as $a \vee b$.

Let $\{a, b\} \in [\mathbf{L}, \rhd]$. A join of a and b always exists if and only if $[\mathbf{L}, \rhd]$ has an upper universal bound $\mathbf{I}$.

For elements $\{a, b\} \in [\mathbf{L}, \rhd]$ the <u>greatest lower bound</u> (glb) of a and b is an element c of $[\mathbf{L}, \rhd]$ for which $c \rhd a$ and $c \rhd b$, and there exists no other element $x \in [\mathbf{L}, \rhd]$ for which $a \rhd x \rhd c$ and $b \rhd x \rhd c$. In general, there might be several elements that satisfy the glb condition. If there is only one such element it is called the meet of a and b and written as $a \wedge b$.

Let $\{a, b\} \in [\mathbf{L}, \rhd]$. A meet of a and b always exists if and only if $[\mathbf{L}, \rhd]$ has a lower universal bound $\mathbf{O}$.

A lattice is a poset $[\mathbf{L}, \rhd]$ in which all pairs of elements have a unique join and a unique meet.

All these operations are useful in thinking about the properties of the lattice of security levels. Those students who are seriously interested in trustworthy systems are encouraged to expand their knowledge far beyond this brief introduction.

4.7. The Three Policies

A security policy is a set of principles, which can be used to define the set of permitted actions by active agents within a system. A system that faithfully enforces a given policy can be said to be secure with respect to that policy. For example, the security of a computer system could be defined in terms of its ability to prevent the disclosure of information to unauthorised persons. For another purpose, it might be preferable to define security in terms of the ability to prevent the unauthorised use of system resources. These are two quite different goals: a system that satisfies the first may well not be secure according to the second, and vice versa.

In any single host, the security of subjects (processes, agents or surrogates for users), and objects (programs, data, or files) is a well-studied problem. In the United States, the TCSEC[18] establishes a graded set of measures against which a computer system can be calibrated to establish levels of security confidence. Several other countries have produced
or will produce similar criteria[19,20,21,22].

18 Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, Library No. S225,711, December 1985.

19 Canadian Trusted Computer Product Evaluation Criteria, System Security Centre, Communications Security Establishment, Government of Canada, Ottawa, Ontario. Draft Version 1.0, May 1989. Now in Version 2.0.

20 A five volume set from the United Kingdom. Direct inquiries to Department of Trade and Industry, Kingsgate House, 66-74 Victoria Street, London SW1E 6SW, Attn. A. Matthew, IT4c, Room 8.47.

21 In the Bundesrepublik Deutschland, IT-Sicherheitskriterien, Kriterien für die Bewertung der Sicherheit von Systemen der Informationstechnik (IT), ISBN 3-88784-192-1, Published by, Bundesanzeiger Verlagsgesellschaft, Postfach 10 80 06, 5000 Köln, Germany. Also available in English from the same source with the title IT–Security Criteria, ISBN 3-88784-200-6.

22 A Trusted Network Interpretation (TNI) of the TCSEC, Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, U.S. National

4.7.1 Confidentiality

Various kinds of information policies are of interest in computer systems. Together they form the overall security policy. The kind of policy that is most familiar is the confidentiality policy, that defines which entities are permitted access to what information. The most common formulation of this partitions the policy into a mandatory component and a discretionary component. Further detail is amply available in the literature[23,24]. The essential notion is that every entity in the system must have a security label, and there exists a relation on the set of possible labels that establishes whether a given instance of a given function is to be permitted. Given the labels, the relation can be adapted to fit whatever policy it is desired to enforce.

4.7.2 Integrity

Integrity is not as well understood as its confidentiality[25,26,27,28]. The integrity of data might indicate the degree to which it has the value that it should have. The integrity of a user might indicate our confidence that he or she originates correct data, or how trusted that person is to modify critical files. The integrity of an executing process might reflect the integrity label of its user, or the amount of trust we are willing to place in its program's correctness, or our confidence that the file containing the program has not been modified.

A policy issue related to integrity is identification and authentication. An application must be granted some level of assurance of the identity of a peer process with which it communicates. A reference monitor implementing a security policy on a host must know the identity of a user in order to correctly implement its policy. A policy is necessary to control these aspects of authentication.

Computer Security Center, NCSG-TG-005, Library No. S228,526. 31 July 1987 supposedly applies to networked computer systems.

23 Dorothy E. Denning, "A Lattice Model Of Secure Information Flow", *Communications of the ACM*, Vol. 19, No. 5, pp. 236-243, May 1976.

24 United States National Computer Security Center, *A Guide to Understanding Discretionary Access Control in Trusted Systems*, NCSC-TG-003, 1985 Sep 30.

25 S. Porter and T. S. Arnold, "On the Integrity Problem", Proc. 8th National Computer Security Conference, Gaithersburg, pp. 15-17, 1985.

26 K.J. Biba, "Integrity Considerations for Secure Computer Systems", MITRE Tech. Report 3153, 66 pp., Apr. 1977.

27 Dr. Stuart W. Katzke and Zella G. Ruthberg, editors, "Report of the Invitational Workshop on Integrity Policy in Computer Information Systems (WIPCIS)", Bentley College, Waltham, Mass., 1987 Oct 27-29, United States National Institute of Science and Technology publication 500-160, January 1989.

28 "Reports of the 2nd and 3rd Invitational Workshops on Integrity Policy in Computer Information Systems (WIPCIS II and III)", NIST, Gaithersburg, Md., February 1989 and in 1990.

4.7.3. *Denial of Service*

Denial of service is *the prevention of authorised access to resources or the delaying of time-critical operations* [29]. In this course, a denial of service attack is one where there is no expectation that the trusted computing base can recover in a reasonable time. The objective of the attacker might be to destroy the ability of some user (or all users) to do useful work. One way is to halt the machine. This will be an effective denial of service attack that no amount of clever TCB design can overcome. Protection from denial of service attacks must involve a measure of physical security. Such attacks are immediately evident. A more subtle denial attack involves the deletion, alteration, or insertion of just enough requests for service so the system will eventually recover given enough time, but that much time is never quite available. Such an attack is also immediately evident to the hosts, and would require very fine judgement on the part of the attacker.

4.7.4. *The TCB and the Reference Monitor*

The enforcement of a security policy is based on an interesting contradiction. In order to enforce the security policy, some parts of the system must be trusted with the privilege of violating the security policy. These parts of the system are the trusted computing base (TCB). The TCB is actually a suite of software programs. The TCB must be able to

- read all security labels – it can not decide what is permissible without reading the security labels.
- to manipulate data regardless of its security level – it has to be able to copy, input, and output, all data files. It has to be able to load the system securely.
- to read and write secure files that must have high integrity – for instance, the password file.

The TCB is actually an implementation of a set of mechanisms that decide which kinds of which references are permissible, and which are prohibited. This set of mechanisms can be abstracted to represent a set of principles rather than a pile of detail. The principles implement the reference monitor concept. It is the reference monitor that is formally proven to be correct in those systems that employ formal proofs.

4.7.5. *Specification*

In any complicated system, it is important to specify what is to be built before it actually gets built. This is a well-understood concept in both software engineering and in computer engineering. Without specification, the goal of the implementation will become obscure, and in most instances

[29] ISO Standard 7498 Part 2, "Security Architecture", International Organisation for Standardisation, 1988, page 3.

the result will not work as was wanted, if it works at all. When implementing secure software, specification is doubly important.

The actual functions that the hardware provides must be precisely understood. The desired functions that the software must add to the hardware must be precisely stated. This specification is known in the TCSEC as a Descriptive Top-level Specification if it is written in natural language, or a Formal Top-level Specification if it is written in an artificial specification language.

Artificial specification languages[30] often have precisely known semantics for all their syntactic constructions, and are in consequence excellent bases for abstract models and formal proofs.

4.8. The Six Basic Principles

Normally, underlying all security policies are the following six basic principles. The difference will usually concern the way that these principles are interpreted or applied to a specific circumstance.

4.8.1. Simple Security: READ DOWN

The simple security condition is the basic rule that insists that a subject can only observe information at the same security level, or a lower level. The simple security condition is satisfied when a subject s can have observe access to an object o if and only if $L(s) \rhd L(o)$. The security level of the subject must dominate the security level of the object.

4.8.2. The ∗-Property: WRITE UP

The ∗-property[31] is the basic rule that insists that there be no possible situation in which data could flow from some level in the security lattice to a lower level.

We distinguish between (over)write and append. Because of integrity issues, write should be limited to an object at the same level as the writing subject. A subject may append to a file at a higher level. In practice, both write and append often are limited to the same level. If this restriction is applied, information flows up the security lattice through some trusted mechanism that a lower level process can use to signal a higher level process to read. This mechanism will frequently be a file to which the lower level process writes a journal of its activity, and the higher-level process monitors.

If there is only one object open for observation, the ∗-property is satisfied when a subject s has write access to the object o if and only if

---

30 Two of these specification languages are Gypsy from the University of Texas at Austin, and EVES from I.P.Sharp, Inc. in Ottawa. Both include elaborate automated theorem provers.

31 ∗-property is pronounced "star property".

$L(o) \rhd L(s)$. In practice this is often modified to $L(s) = L(o)$ for the integrity reasons just mentioned.

If there are several objects open for observation, any object that is opened for write must have a level equal to or higher than the least upper bound of the levels of all objects open for observation. Let the objects that are open for observation be $\{o_0, o_1, o_2, \cdots, o_k\}$. Let $o_0 \rhd o_j$ for all $1 \leq j \leq k$. Object o_0 is the lub of the objects that are open for observation. Any object o_w that is open for writing must satisfy the relation $o_w \rhd o_0$.

If all objects open for observation are at the same level, a write can be done securely at that level. If the overwriting of higher-level data with upgraded lower-level data is allowed (implying that this integrity problem is not an issue) then the write can be done securely at a higher level also.

If all objects open for observation are not at the same level, a write can be done securely at a level that is the least upper bound of the levels of the objects open for reading. If the overwriting of higher-level data with upgraded lower-level data is a problem then this write must be restricted to be an append.

The $*$-property ensures that there can be no possibility of downgrading – of observing data at one level and writing it at a lower level.

4.8.3. Tranquillity

The tranquillity rule is simply the rule that insists that the security level of a subject or an object does not change while that subject or object is involved in any activity involving any security rule. For efficiency reasons, as well as correctness reasons, it is obviously not legitimate for the security level of an object to change, for example, while some process is reading it.

In practice, if it is thought that a system will have to support the urgent re-classification of objects, there will need to be a method to revoke a previously approved access. This sudden retraction of an access right can be a difficult implementation issue, and might also be used as a covert path in some strange circumstances. It is easy, however, to limit the bandwidth available on this path.

4.8.4. No Re-use

When a re-usable resource (memory, registers, disk) is allocated to a new user, it must be "empty". This principle is simply the prohibition of covert storage channels.

4.8.5. Accountability

A user (some actual person) must be accountable for every security-related event that happens in the system. In practice this means that all objects must be owned by some authenticated user, and all processes must be executed by authenticated and authorised users or other processes.

4.8.6. Enhanced Privilege

An enhanced privilege is a licence to avoid one or more of the usual limits, rules, and controls that enforce the security policy. Subjects that are in the TCB must be trusted to behave properly because they have enhanced privilege. If they didn't need to be trusted there would be no reason to include them in the TCB.

Any mechanism, including a trusted mechanism in the TCB, should have the least privilege that it needs in order to do its job. A user process has only the privilege that the security policy allows it to have, but the TCB processes will all have enhanced privilege. In practice, this need for extra privilege is what defines the contents of the TCB.

Least privilege is contrasted with awarding to a TCB member all the possible privileges. Least enhanced privilege is an attempt to minimise the special cases that need to be examined when the TCB is examined to determine the level of assurance it satisfies. It is also an attempt to confine the damage that could happen if trusted subjects misbehave or become ill with a virus or other plague.

In practical systems there are about 30 distinct enhanced privileges that a process can have, but not all combinations make sense.

4.9. An Example: Privilege in UNIX[32]

This section follows the work of Guy Grenier and R.C. Holt[33]. The table below describes the type of privileges necessary for a secure UNIX-like system. No attempt has been made to make this table complete. It is an example, and should not be thought to be anything more than an example. It is written in terms of an arbitrary subject *s* that must have the specified privilege to perform the described operation on the data object *o*.

32 UNIX is a trademark of Western Electric.

33 "Policy vs. Mechanism in the Secure Tunis Operating System", *Proc. 1989 IEEE Symp. on Security & Privacy*, Oakland, CA, May 1989, 11 pages.

Privileges that Untrusted Mechanisms Can Have		
No.	Name.	Conditions for s to have the privilege to access o in the described way.
1	Simple Security	For s to observe o, $L(s) \rhd L(o)$.
2	$*-$property	For s to alter o, $L(o) \rhd L(s)$.
3	Discretionary Access	The access permissions of s must allow access to o.
4	Hierarchy	Let o_P be the parent directory of object o. If o is a directory $L(o) \rhd L(o_P)$. Else $L(o) = L(o_P)$.
5	Tranquillity	The privileges of a process must not change after invocation. The level of a process must be static during the lifetime of that process. The level of an object that a process accesses, including the program that the process is an instance of, must not change during the lifetime of that process.
6	Accountability	The same subject must be accountable for both s and o.
7	Passed Accountability	A parent process of s must also be accountable for o.
8	Veto $*-$property	Write allowed even though $L(s) \rhd L(o)$.
9	Writeable Directory	o_1 is in directory o_2. o_1 is writeable by s. o_2 is also writeable by s even if $L(s) \rhd L(o_2)$.
10	Create File	The file o is to be created in directory o_1. If o exists the file is being moved from directory o_2 so both o_1 and o_2 are made writeable by s. If o does not exist, o_1 is made writeable by s and $L(s) = L(o_1)$.
11	Create Directory	The directory o_2 is to be created in directory o_1. o_1 is made writeable by s and $L(o_2) = L(s)$.
12	Open File	o must exist and be observable, or if it does not exist s must have privilege 10.
13	Path Search	Privileges 1 and 3 for the path.
14	Path Compatibility	s references two paths in the file system index to object o_1 and directory o_2. If o_1 is a file, then $L(o_1) \rhd L(o_2)$. If o_1 is a directory, then $L(o_1) = L(o_2)$.

15	Set Security Level	Let the new level of the object o be $L^{NEW}(o)$, and let the old level of object o be $L^{OLD}(o)$. Let o_P be the parent directory of o. If o is a directory $L^{NEW}(o) \rhd L^{OLD}(o) \rhd L(o_P)$ without this privilege and both $L^{NEW}(o) \rhd L(o_P)$ and $L^{OLD}(o) \rhd L(o_P)$ with the privilege. If o is a file $L^{NEW}(o) = L^{OLD}(o) = L(o_P)$ without this privilege and $L^{NEW}(o) \rhd L(o_P)$ with privilege.
16	Access Allowed	Either the owner of s is also the owner of o, or the access is be allowed by this privilege.
17	Link	This privilege permits a directory to be linked. Without this privilege, directory linking is prohibited.
18	User Same	Either the user accountable for s is the same user as the user accountable for the parent of s, transitively back to the shell process of the user, or the action that requires this transitive accountability is allowed by this privilege.
19	Group Same	Either the user's group accountable for s is the same group as the group accountable for the parent of s, transitively back to the shell process of the user, or the action that requires this transitive accountability is allowed by this privilege.
20	Kill	Either s is trying to kill itself, or the parent of s is trying to kill s, or s_1 is trying to kill s and s_1 has this privilege.
21	Setuid	When o is instantiated by s_P, to become s the user accountable for of s will be the user accountable for both s_P and o, which must be the same. If o has its setuid bit <u>true</u> the user accountable for of s will be the user accountable for either s_P or o, which are not necessarily the same.
22	Setgid	When o is instantiated by s_P, to become s the user's group accountable for of s will be the user's group accountable for both s_P and o, which must be same. If o has its setuid bit <u>true</u> the user's group accountable for of s will be the user accountable for either s_P or o, which are not necessarily the same.

BELL-LAPADULA AND SECURE XENIX

5. BELL-LAPADULA AND SECURE XENIX

The Bell-LaPadula (BLP) model is a "state-transition" model of trustworthy systems. The model formally defines system states and rules. The states have little to do with the actual calculation occurring in any program; they correspond to security-related events, such as (requests for) changes in the access matrix. The rules are actions or operations that move the system from state to state. The model includes four axioms that must be preserved in every state. Proofs depend upon these axioms being valid. The proofs demonstrate that any applications of rules to system states can not move the system into an insecure situation if it ever starts in a secure situation.

Secure Xenix is an instance of UNIX that conforms to the BLP model.

5.1. System States

A system state V is an element of set $V = \{B \times M \times F \times H\}$.

B is the set of current accesses and is a subset of the set $S \times O \times A$, where S is the set of subjects, O is the set of objects and A is the set of access privileges defined in the system. The set B defines the access privileges that each currently active subject has to each object.

M is the access matrix. It consists of elements $M_{i,j} \in A$ that define the set of access privileges subject s_i has to object o_j. M encapsulates the Discretionary Access Control (DAC) of the system.

F is a three-component security function that deals with security labels associated with each subject and object:

f_s The first component, f_s, assigns a maximum security level (clearance) to each subject.

f_o The second component, f_o, assigns a security level (classification) to each object.

f_c The third component, f_c, assigns the current security level of each subject.

Note that $f_s \triangleright f_c$. A variation of F also includes a component to assign a minimum-security level f_m to each subject.

H is the current object hierarchy. The current object hierarchy is a collection of rooted directed trees and isolated points.

H is a subset of all functions from objects O to the power-set of objects O subject to the following two restrictions:

(1) For $\{o_i, o_j\} \in O$ then $o_i \neq o_j \Rightarrow H(o_i) \cap H(o_j) = \varnothing$.

(2) There does not exist a set $\{o_1, o_2, \cdots, o_w\}$ of objects such that $o_{r+1} \in H(o_r)$ for $1 \leq r \leq w$, and $o_{w+1} = o_1$.

These two conditions imply that the current object hierarchy is a collection of rooted, directed trees and isolated points. They rule out cycles and objects with multiple parents. If H is a tree structure, then o_r is an object called the root for which $H(o_r) \neq \varnothing$ and $o_i \in H(o_r)$ for any $o_i \in O$. Furthermore, o_i is a superior of o_j if $o_j \in H(o_i)$. For such a tree structure, from a security level viewpoint it will always be the case that $L(o_j) \triangleright L(o_i)$.

H and F encapsulate the Mandatory Access Control (MAC) of the system.

5.2. State Transitions

The system transitions from state-to-state are defined by a set of operations or rules that are requested by subjects on system states. A rule is a function that specifies a decision (output) and a next-state for every state and every request (input).

Let R be the set of request invocations defined, and D be the set $\{\text{yes}, \text{no}, ?, \text{error}\}$ of request outcomes. Thus, a rule ρ is defined as ρ: $R \times V \rightarrow D \times V$, where $R \times V$ is the set of {request, state}-pairs defined in the system for every request and state, and $D \times V$ is the set of {decision, next-state}-pairs defined in the system for every decision and next-state. The decision "yes" ("no") means that the request has (not) been granted; "error" means that the request parameters are meaningless; and the "?" outcome means that some exceptional condition was detected during the consideration of the rule ρ (e.g., table overflow, etc.).

Let $\rho = \{\rho_1, \cdots, \rho_s\}$ be the set of rules. The relation W is the set of state transitions and is defined for any $R_k \in R$, $D_m \in D$, set of states V and set of next-states V^*. by:

(a) $(R_k, D_m, V^*, V) \in W$ if and only if $D_m \neq "?"$ and $D_m \neq "\text{error}"$.

(b) $(D_m, V^*) = \rho_i(R_k, V)$ for a unique i, $1 \le i \le s$.

5.3. Systems, System Appearance, System Actions

Let T be the set of positive integers. X is defined as the set of all request sequences: the set of all functions from T to R; Y is defined as the set of all decision sequences: the set of all functions from T to D; Z is defined as the set of all state sequences: the set of all functions from T to V.

Each of X, Y, and Z represent a set of sequences of successive values of requests, decisions, or states. X, Y, and Z are related to the starting state z_0 in an obvious way: the system starts in state z_0 and X represents all request sequences, Y all decision sequences, and Z all state sequences that can ensue.

Let $x \in X$, $y \in Y$ and $z \in Z$ and let $z_0 \in Z$ be the initial state. A system $\Sigma(R, D, W, z_0)$ is a subset of the cross-product $X \times Y \times Z$ such that $(x, y, z) \in \Sigma(R, D, W, z_0)$ if and only if the three sequences are consistent: starting in state z_0 the inputs x result in the decisions y and a progression through the states z. The system $\Sigma(R, D, W, z_0)$ includes all possible executions that start from state z_0.

A system appearance is defined as each triple (x, y, z) such that $(x, y, z) \in \Sigma(R, D, W, z_0)$ for $x \in X$, $y \in Y$, $z \in Z$. A system appearance is the sequences of states, requests, and decisions that describe one execution of the system.

A system action is defined as each quadruple $(x_t, y_t, z_t, z_{t-1}) \in W$, where x_t, y_t, z_t are respectively the t^{th} request, decision, and state in the sequences $x \in X$, $y \in Y$, $z \in Z$. A system action embodies the move from state z_{t-1} to state z_t as a result of request x_t and decision y_t.

Alternatively, $(R_i, D_j, V^*, V) \in R \times D \times V \times V$ is an action of $\Sigma(R, D, W, z_0)$ if and only if there is an appearance $(x, y, z) \in \Sigma(R, D, W, z_0)$ and some $t \in T$ such that $(R_i, D_j, V^*, V) = (x_t, y_t, z_t, z_{t-1})$.

5.4. Model Axioms

The axioms of the Bell-LaPadula model require the definition of the access privilege set A. In the model, $\mathrm{A} = \{\mathrm{r, w, x, a}\}$. The meaning of these privileges is defined in the model in terms of the ability of a subject to "observe" or "alter" an object as shown below.

Access Privilege	**Can Observe**	**Can Alter**
e**x**ecute	no	no
read	yes	no
append	no	yes
write	yes	yes

The first two of the four axioms to be described below use the above access privilege definitions. A fifth axiom, called the "tranquillity principle" is defined in earlier (1973) versions of the model, but has been removed from later (1976) versions for reasons that will be explained below. These axioms were called properties in the original BLP 1973 work, and this terminology persists.

5.4.1. The Simple Security (SS) Property

A system state $v = (b, M, f, H)$ satisfies the SS-property if and only if, for each element $b \in B$ that has an access privilege of **r** or **w**, the maximum clearance of the subject dominates the classification of the object; or alternatively:

An element $(s,o,\underline{x}) \in B$ satisfies the SS-property relative to the security function F if and only if

(i) $\underline{x} = \mathbf{x}$ and $f_s(s) \lhd f_o(o)$

(ii) $\underline{x} = \mathbf{r}$ and $f_s(s) \rhd f_o(o)$.

The above two conditions restrict subject accesses to objects of the "observation" type. The restriction is based on object classifications and subject maximum clearances. The SS-property also restricts subjects from having direct access to information for which they are not cleared, because in this case the subjects have no access privileges at all to the objects.

5.4.2. The $*-$ Property

Let $\overline{S} \subseteq S$ be the set of untrusted subjects. A system state $v = (b, M, f, H)$ satisfies the *-property relative to the set of subjects $\overline{S}$ if and only if, for each element $(s,o,\underline{x}) \in B$:

(i) $\underline{x} = \mathbf{a} \Rightarrow f_c(s) \lhd f_o(o)$

(ii) $\underline{x} = \mathbf{w} \Rightarrow f_c(s) = f_o(o)$

(iii) $\underline{x} = \mathbf{r} \Rightarrow f_c(s) \rhd f_o(o)$

The above property is intended to prevent unauthorised flow of information from higher security levels to lower ones. In particular, the *-property prevents an untrusted subject from having simultaneously privileges to "observe" information at some level and to "alter" information at a lower level.

Trusted subjects (i.e., subjects not in $\overline{S}$) need not be bound to the *-property in the same way subjects in $\overline{S}$ are.

5.4.3. *Discretionary Security (DS) Property*

A system state $v = (b, M, f, H)$ satisfies the DS-property if and only if, for every element $(s, o, \underline{x}) \in B$, $\underline{x} \in M_{i,j}$.

5.4.4. *Compatibility Property*

The object hierarchy H maintains compatibility if and only if, for any $o_i, o_j \in O$ and $o_j \in H(o_i)$, then $f_o(o_j) \triangleright f_o(o_i)$. This axiom is also called the "non-decreasing level" axiom for the object hierarchy. Descending the object tree structure defined by H is never associated with decreases in security classification.

5.4.5. *(Sometimes) Tranquillity Principle*

The original version of the Bell-LaPadula model (1973) also contained the "tranquillity" principle. This principle (axiom) states that a subject cannot change the security level of active objects. Of course, this is defined for the untrusted subjects $\overline{S}$.

This axiom has been removed from the 1976 version of the Bell-Lapadula model to allow controlled changes of security levels of active objects. The rules that control such changes depend on specific applications (e.g., mail, guards, etc.) and differ from system to system. Surprisingly, the proofs of confidentiality security do not depend on this axiom in any vital way. The existence of f_s as well as f_c is due to the removal of tranquillity. It seems clear, however, that more complex security situations may require tranquillity.

5.4.6. *Activation Axioms*

Object activation or deactivation refers to the creation and destruction of objects. The dynamic creation or destruction of objects in the Bell-LaPadula model would cause the domain of the classification function f_o and the size of the access matrix M to vary dynamically. To avoid this, the entire set of objects ever used are considered extant in either active or inactive form. Furthermore, objects are considered to be MAC labelled in both forms. The use of this artifice requires the specification

(1) of a subject's access to an inactive object,

(2) of the state of newly activated objects,

(3) of the classification of newly-activated objects, and

(4) of the object deactivation rules.

Specification (1) is necessary because active and inactive objects are assumed to coexist in *O*. Since the model defines subjects' access to active objects, it must also define subjects' access, or lack thereof, to inactive objects. If left unspecified, such access may cause security breaches in real implementations.

Specification (2) is necessary because inactive objects have states (since they exist in O). Thus, their activation must specify the relationship between the state of an inactive object and its state at activation.

Similarly, specification (3) is necessary because inactive objects also have a classification in the model, and their classification while inactive might not match the requirements of the requesting subjects. Furthermore, their classification may conflict with the compatibility axiom.

Specification (4) is also necessary because the object deactivation (destruction) rules are security relevant.

The omnipresence of objects, even after destruction, is an awkward thing to deal with, particularly when implementing a secure system. Feiretag, Levitt and Robinson propose two activation axioms that specify only a subject's access to an inactive object and the state of a newly-activated object. Their two activation axioms are:

(i) Non-accessibility of Inactive Objects
 In the sense of the section above discussing the access privilege set, a subject cannot observe the contents of an inactive object.

(ii) Rewriting of Newly Activated Objects
 A newly activated object is given an initial state that is independent of the state of any previous incarnations (activations) of the object.

The two activation axioms can be expressed formally and succinctly as:

(i) Let $O = \hat{O} \cup \overline{O}$ where $\hat{O}$ includes all active objects and $\overline{O}$ includes all inactive objects. Clearly $\hat{O} \cap \overline{O} = \emptyset$. Then

$$[\forall(s,o,\underline{x}) \in B, o \in \overline{O}\,] \Rightarrow [\underline{x} \neq \mathbf{r} \text{ and } \underline{x} \neq \mathbf{w}].$$

(ii) Let $\mathbf{new}(o)$ imply the simultaneous actions $[\overline{O} = \overline{O} - o\,;\ \hat{O} = \hat{O} + o]$ and let $\mathbf{call}[s_i, \mathbf{new}(o)]$ be the invocation of the primitive functional service "**new**" (create object) by s_i; Then, for any function **g** and $\mathbf{state}(o)$,

$$\mathbf{call}[s_i, \mathbf{new}(o)] \Rightarrow \mathbf{state}[\mathbf{new}(o)] \neq \mathbf{g}[\mathbf{state}(o)].$$

5.5. System Security

A secure state is a state that satisfies whatever axioms are deemed necessary and pertinent. This can (and does) change depending on the particular system being considered. For instance, tranquillity may or may not be significant in a secure system, or all objects may be at one level so compatibility is irrelevant. BLP originally defined a secure state as one which satisfied axioms one, two, and three, and five, but in 1976 axiom four was added and five deleted. Later four became an underlying principle that was assumed but did not explicitly appear in the proofs of security (because if the security level of data decreases as the tree-structured directories are traversed, processes at the lower levels can't get past the top of the tree to get at data they otherwise should be able to access). The theorems below assume axiom four, and a secure state is defined as one in which axioms one, two, and three are satisfied.

A state $z \in V$ is a secure state if and only if axioms one, two, and three are satisfied.

A state sequence $Z = (z_1, z_2, \cdots, z_t, \cdots)$ is a secure sequence if and only if z_t, is a secure state for each $t \in T$.

A system appearance $(x, y, z) \in \Sigma(R, D, W, z_0)$ is a secure appearance if and only if Z is secure state sequence.

A system $\Sigma(R, D, W, z_0)$ is a secure system if and only if every appearance $(x, y, z)$is a secure appearance.

Several similar definitions of secure systems can be given. The most straightforward, but not the most useful, is based on proofs that a system satisfies the first three security axioms: the SS-property, the *-property, and the DS-property.

The following three lemmas, along with the Basic Security Theorem prove the security of such a system. The proofs to these three lemmas are not difficult. The first is given and the others are left as an exercise in straightforward reasoning. In each case, the two given conditions state that the accesses added as a result of the request $(b^* - b)$ satisfy the axioms in the new state, and that any access in b that does not satisfy the axioms in the new state is not present in the new state.

Lemma A1.

Let $W = [R_i, D_j, (b^*, M^*, f^*, H^*), (b, M, f, H)]$. For any initial state z_0 that satisfies the SS-property, the system $\Sigma(R, D, W, z_0)$ satisfies the SS-property if and only if W satisfies the following conditions for each action:

(i) each $(s, o, \underline{x}) \in (b^* - b)$ satisfies the SS-property for f^*; and

(ii) each $(s, o, \underline{x}) \in (b)$ that does not satisfy the SS-property for f^* is not in b^*.

Proof:

Pick $(x, y, z) \in \Sigma(R, D, W, z_0)$ and write $z_t = (b_t, M_t, f_t, H_t)$ for each $t \in T$. Then $z_0 = (b_0, M_0, f_0, H_0)$.

[**IF**] The proof proceeds by showing that if z_0 is secure then z_1 must be secure, and then by induction showing that the system is secure.

$(x_1, y_1, z_1, z_0) \in W$. By (i), every $(s, o, \underline{x}) \in (b_1 - b_0)$ satisfies the SS-property for f_1.

Let $\underline{b} = [(s, o, \underline{x}) \in b_0 \mid (s, o, \underline{x})$ does not satisfy the SS – property] for f_1.

By (ii) we have $b_1 \cap \underline{b} = \varnothing$. But if $\underline{b}$ is in both b_0 and b_1 then it is also in $b_0 \cap b_1$. Then $\underline{b} \cap (b_0 \cap b_1) = (\underline{b} \cap b_1) \cap b_0 = \varnothing \cap b_0 = \varnothing$. Hence if $(s, o, \underline{x}) \in (b_1 - b_0)$, then $(s, o, \underline{x}) \notin \underline{b}$ as hypothesised, so that $(s, o, \underline{x})$ satisfies the SS-property for f_1. Since every $(s, o, \underline{x})$ is in either $(b_1 - b_0)$ or $b_0 \cap b_1$ we have shown that z_1 must be a secure state.

By induction on T, z_t is secure so (x, y, z) is a secure appearance. Since (x, y, z) is arbitrary the system $\Sigma(R, D, W, z_0)$ is a secure system.

[**ONLY IF**] Proof by contradiction.

A contradiction of the conclusion of the lemma results in the proposition that:

there is some action (x_t, y_t, z_t, z_{t-1}) such that either

(a) some $(s, o, \underline{x}) \in (b_t - b_{t-1})$ does not satisfy the SS-property for f_t, or

(b) some $(s, o, \underline{x}) \in (b_{t-1})$ does not satisfy the SS-property for f_t but is in b_t.

Suppose (a):

Then there is some $(s, o, \underline{x}) \in (b_t)$ that does not satisfy the SS-property for f_t, since $(b_t - b_{t-1}) \subseteq b_t$.

Suppose (b):

Then there is some $(s, o, \underline{x}) \in (b_t)$ that does not satisfy the SS-property for f_t by the statement of (b).

Therefore the SS-property is not preserved by the stated action, and this contradicts the assumption that the system $\Sigma(R, D, W, z_0)$ is secure.

Lemma A2.

Let $W = [R_i, D_j, (b^*, M^*, f^*, H^*), (b, M, f, H)]$. For any initial state z_0 that satisfies the $*$-property for $\overline{S} \subseteq S$, a system $\Sigma(R, D, W, z_0)$ satisfies the $*$-property for $\overline{S}$ if and only if W satisfies the following conditions for each action:

(i) for each $\overline{S} \subseteq S$, any $(s, o, \underline{x}) \in (b^* - b)$ satisfies the $*$-property for $\overline{S}$; and

(ii) for each $\overline{S} \subseteq S$, if $(s, o, \underline{x}) \in (b)$ does not satisfy the $*$-property for $\overline{S}$, then $(s, o, \underline{x}) \in (b^* - b)$.

Lemma A3.

Let $W = [R_i, D_j, (b^*, M^*, f^*, H^*), (b, M, f, H)]$. For any initial state z_0 that satisfies the DS-property, a system $\Sigma(R, D, W, z_0)$ satisfies the DS-property if and only if W satisfies the following conditions for each action:

(i) if $(s_k, o_1, \underline{x}) \in (b^* - b)$, then $\underline{x} \in M^*_{k,l}$; and

(ii) if $(s_k, o_1, \underline{x}) \in (b)$ and $\underline{x} \in M^*_{k,l}$, then $(s_k, o_1, \underline{x}) \in (b^*)$.

Theorem A1 [Basic Security Theorem].

A system $\Sigma(R, D, W, z_0)$ is a secure system if and only if z_0 is a secure state and W satisfies the conditions of theorems A1, A2, and A3 above.

5.6. Other BLP Theorems

Lemmas A1-A3 focus on properties of the current access sets of B. Lemmas A4-A6 and A7-A9 of the full Bell-LaPadula report represent restatements of lemmas A1-A3 focusing on (A4-A6) properties of sets of system actions of W, and on (A7-A9) properties of individual states of V, respectively. Similarly theorems A2 and A3 are the corresponding restatements of theorem A1.

Lemma 10 restates the results of the lemmas A1-A3, A4-A6, and A7-A9 in terms of property-preserving rules ρ.

The need for the alternate, but equivalent theorems, becomes apparent when one needs to construct proofs of real systems. For example, in systems whose kernel enforces security, it is substantially more convenient to prove lemmas A4-A6 or A10 than Theorems A1-A3 or A7-A9. The reason is system actions or rules can be easily identified with kernel calls and their effects on the system states.

5.7. Assessment of Bell-LaPadula

Bell-LaPadula is not a panacea. It was the first formal model of secure systems to be developed, in the early 1970s. Consequently, it has an edge over other more recent and perhaps better models because it was the underlying model for most of the work that was done for a decade or more. The formal model that the Orange Book designers had in mind was clearly Bell-LaPadula.

5.7.1 Perceived Advantages of Bell-LaPadula

1. The axioms of Bell-LaPadula are modelled on the style of security policy favoured by many national and commercial organisations. Consequently, neither the policy nor the model need be bent out of shape to use Bell-LaPadula in practical situations.

2. Bell-LaPadula is easy to adapt to a fairly broad range of special circumstances. This means that minor changes in the axioms are easily tolerated.

3. In contrast to some more recent models, Bell-LaPadula is perceived to be understandable by people who do not have a doctorate in computer science or in mathematical logic. It has a good intuitive relationship to actual systems that it is used with. If it can be used at all, it is usually fairly easy to apply.

4. The Bell-LaPadula model is well understood and has been extensively used. Designers, implementers, and evaluators are familiar with it and are familiar with its good and bad points.

5.7.2 Perceived Disadvantages of Bell-LaPadula

1. Bell-LaPadula's rather simple access privileges always need interpretation. It is not clear whether this is a strength (flexibility) or a weakness (imprecision).

2. Bell-LaPadula is difficult to adapt to situations in which there are a lot of access privileges that differ from each other only slightly. For example, it is difficult to model the differences between a read access that is a destructive-read-and-restore operation, and one that is a pure observation that does not affect the data value at all.

3. Bell-LaPadula is designed to model the access by subjects to objects. It does not handle easily accesses by subjects to other subjects. The resulting issues of parameter and result passing, and visibility and scope issues, are difficult or messy or both. This means that Bell-LaPadula does not cope well with networks, because protocol exchanges in a network are archetypal examples of subject-to-subject access. Another is the common action of subjects calling subjects with parameters and a result, and this requires some interpretation.

4. Bell-LaPadula does not handle covert channels easily. Nor does any other known model.

5. The *-property is overly restrictive. This is not strictly a criticism of Bell-LaPadula alone since the *-property is not confined to Bell-LaPadula, but appears as policy that must be enforced by any model. Some data-flow models can cope with the *-property better than Bell-LaPadula does.

   Suppose there are two objects o_1 and o_2, and a subject s_3 at different levels of security $L_1 < L_2 < L_3$ respectively. Suppose that it is desired that the subject s_3 copy data o_1 at level L_1 and into data o_2 at level L_2.

   For simplicity, assume the write is just a copy without any observation. Any write is contrary to the *-property because s_3 is prohibited from writing to the lower level L_2. This is the case even though s_3 is fully qualified to observe data at level L_2, but evidently is not qualified to create such data in any circumstance.

   In practice this situation could be handled by s_3 having its current security level lowered to L_2 by a trusted process privileged to do that operation (called downgrading). It could then make the copy within the policy. Unfortunately, the downgrading of s_3 may be prevented by SS-property. This is because there might be other data objects that are open for reading at L_3 and that have nothing to do with the copying. A subject s_3 is prohibited from operating at a level L_2 and having an object at the higher level L_3 open for reading. A better way to cope is to have a TCB-resident subject that is trusted to copy-up. But this adds to the TCB for this simple operation.

   Alternatively, the copy (call it o_3) could be made at L_3 within the policy, and then the copy o_3 could be downgraded to L_2. This is a big deal; the downgrader would need to be convinced that the new o_3 was truly a copy of o_1, because the risk of a covert storage channel is very high. There are also other types of covert channel that could be exploited here, so this type of downgrading is not desirable, although it can sometimes be necessary.

5.8. The Secure Xenix Interpretation of BLP

The interpretation of the Bell-LaPadula model in Secure Xenix consists of a description of the notion of system state, and state transition in Secure Xenix. Furthermore, it includes the definition of the initial state and an argument that explains why the mandatory and discretionary access control of Secure Xenix implies that the axioms of the Bell-LaPadula model are satisfied.

5.8.1. The Interpretation of the System State

The interpretation of the system requires the identification of the state components $B = S \times O \times A$, M, F and H in Secure Xenix.

5.8.1.1. Secure Xenix Subjects **(*S*)**

Processes are the only type of subject in Secure Xenix. A process may create and destroy objects, may activate and deactivate them, may change the discretionary privileges of objects in the access matrix, may change the current access set, and may change the object hierarchy. However, processes may not change the security level of objects. All changes a process makes to the system state are constrained to satisfy compatibility, tranquillity, SS-property, *-property, and DS-property. This is discussed in detail below.

Processes are created at login time or by other processes. A process is identified by a unique process identifier and its user is identified by a non-reusable UID and GID [Gilgor86]. The effective UID and GID of a process are used in all discrete unary access control decisions. Each process contains a security label that is used in mandatory access control decision. Process labelling is discussed below in the section describing the interpretation of the security function in Secure Xenix, and the use of the real and effective UID and GID is discussed in the interpretation of discretionary access control.

5.8.1.2. Secure Xenix Objects **(*O*)**

The user-created objects of Secure Xenix are: files, special files (devices), directories, pipes, message queues, semaphores, shared memory segments, Xenix semaphores, Xenix shared data segments, ACLs, and processes. Secure Xenix also includes system-created and maintained objects such as the special files (devices) that can be opened or closed by user processes. Trusted processes create, maintain, and use similar objects as those of the users.

(1) Files, Special Files, Pipes, Xenix Semaphores, Xenix Data Segments and ACLs

Files are containers of information managed by the Secure Xenix kernel. Files are protected by either ACLs or by protection bits associated with file i-nodes. The security label of each file is represented in its i-node.

The special files are used to represent devices and can be opened or closed by user processes. In the case of special files the object activation and deactivation are equivalent to the opening and closing of a device. In all other aspects the special files function as the user-created files.

The Xenix shared data segments have similar function to that of the files and are represented, protected, and labelled in a similar way. The difference is that the shared data segments allow asynchronous processes to synchronise their read and write accesses to segment data, and that, unlike files that are shared on a per-copy basis, shared data segments are shared on a per-original basis.

Named pipes function as "unbounded" communication buffers and are represented, protected, and labelled in a similar way as the files. The difference between named pipes and shared data segments is that named pipes impose producer-consumer process synchronisation to prevent underflow conditions.

Semaphores are objects that allow the synchronisation between asynchronous processes and have similar representation, protection and labeling to that of files.

Access Control Lists (ACLs) are objects used for the discretionary protection of files [Gilgor86] and are represented as specially-protected files by the kernel. The ACLs are labelled with the same label as that of the files they protect. They are discussed in detail in the section on access matrix representation.

(2) Directories

Directories are containers for files, special files, pipes, Xenix semaphores, Xenix Data Segments, ACLs and other directories. They form the building blocks for the system hierarchy. Directories are maintained and protected by the Secure Xenix kernel and are represented in a similar way to that of files. The directories that contain special files and ACLs are system created or destroyed whereas the rest of the directories are created and destroyed by users. A directory that contains an object is called a parent directory. A special directory called the root is the highest directory in the parent chain. It is its own parent. It has no ACL and always can be searched by all users.

(3) Message queues, Semaphores, Shared Memory Segments and Processes

The objects in this group do not have file system representation. The System V semaphores and shared memory segments have the same function as their Xenix correspondents. The message queues are containers for messages and are used primarily for requests to server processes. Processes are created and destroyed by their parent processes and are identified, labelled, and protected in the same way as that used for their parents.

All objects mentioned above are activated when they are created and deactivated when they are destroyed. Exceptions to this rule are the special files, which activated when they are opened and deactivated when they are closed. Special files (devices) cannot be created or destroyed by users. This is important in the interpretation of the activation axiom (see section 5.8.7).

5.8.1.3. Access Privilege Set of Secure Xenix **(*A*)**

The basic set of access privileges in Secure Xenix consists of the read, execute, write, and null privileges. (An additional privilege, setuid-gid, is defined for executable files. This privilege is discussed in section 5.8.3 below). These privileges are visible to the user and are interpreted by the kernel differently for different objects. Thus, the actual privilege set is substantially larger than the basic set above. In this section we define the access privileges for each type of object of Secure Xenix and its relationship with the access privileges (modes) of the Bell-LaPadula model.

In examining the relationship between the Bell-LaPadula model privileges and the Secure Xenix privileges it should be noted that the e (execute) privilege of the model does not have any correspondent in Secure Xenix (nor in other systems [Bell76, footnote on p.11]). Similarly, the null privilege of Secure Xenix is not explicitly represented in the model. Furthermore, some of the model privileges have no meaning for some of the Secure Xenix objects and have no representation among the privileges define for those objects. These cases are denoted by the phrase "no meaning" in the correspondence table below. Other model privileges that have no meaning for some Secure Xenix objects have representation among the access privileges for those objects. However the access authorisation mechanisms ignore their representation. This means that none of the operations defined on those objects may be authorised by the ignored privileges. (These cases are denoted by the phrase "ignored" in the privilege correspondence tables below.)

(1) File Access Privileges

read (r)	A process granted read access to a file can execute instructions that cause data to be fetched (read) from the file into processor or memory registers that can be manipulated (e.g., copied) by users. The read access of the Bell-LaPadula model maps directly into the Secure Xenix read.
write (w)	A process granted write access to a file can execute instructions that cause data in the file to be modified. This access privilege differs from the write access in the Bell-LaPadula model in the sense that it does not allow any observation of the state of the file being modified. The append (a) privilege of the Bell-LaPadula model maps into the Secure Xenix write privilege. Note that the Secure Xenix write privilege is also necessary for append operations to files. The write (w) privilege of the Bell-LaPadula model maps into the read and write privilege combination of Secure Xenix.

execute (x)	A process granted the "execute" (x) privilege to a file can transfer control to that file and cause portions of the file to be interpreted and executed as instructions. Note that the portions of the file being executed as instructions are not stored in processor nor in memory registers from which they can be copied by users. Thus, the execute privilege differs from the read privilege. Also, this access privilege differs from the e (execute) access of the Bell-LaPadula model in the sense that it allows the observation of the state of the program executing a file, whereas the execute privilege of the Bell-LaPadula model does not. The execute and read combination of the Bell-LaPadula model maps directly into the execute (x) privilege of Secure Xenix.
null (–)	A process with the null privilege for a file cannot access the file in any way. The Bell LaPadula model does not include the null privilege (although the execute privilege semantics comes close to it).
setuid-gid (suid-gid)	Files containing program code have an additional privilege bit that can change the identity (i.e., UID or GID) of the process while executing in that file. This is discussed in the section that describes the discretionary access control in Secure Xenix.

In summary, the Bell-LaPadula privilege corresponds to File Privilege as shown in the following table.

Name	BLP	XENIX
execute	e	–
read	r	r
read & execute	re	x
append	a	w
write	w	rw
no privilege	–	null

(2) Privileges for Special Files, Pipes, Message Queues, Shared Memory Segments, Xenix Shared Data Segments and ACLs

The privileges for these types of objects are the same and have the same meaning as the file privileges. They have the same relationship to the Bell-LaPadula privileges as those of files (discussed above). The only difference between the privileges for this group of objects and file privileges is that the execute privilege (x) has no meaning for this group of objects and, therefore, this field is ignored for all objects in this group.

In summary, the Bell-LaPadula privilege corresponds to this privilege for this group.

BLP	XENIX
e (execute)	–
r (read)	r
re (read & execute)	x (ignored)
a (append)	w
w (write)	rw
–	null

(3) Directory Privileges

read (r)	A process granted read access to a directory can execute instructions that cause directory attributes and contents to be fetched (read) from the directory into processor or memory registers that can be manipulated (e.g., copied) by the users. Note that no information about the objects named by that directory can be retrieved. The relationship of this access to the read access of the Bell-LaPadula model is the same as that of the files.
search (x)	A process granted the search privilege to a directory can execute instructions that match a given string of characters to those of a directory entry. Note that the search privilege is weaker than the read privilege, which could also be used for searching. The read privilege of the Bell-LaPadula model maps into the search privileges with the appropriate restriction; i.e., the read privilege must be restricted to directory-entry reads. Also note that the distinguished Root directory has the search privilege on for all processes in the system.
execute	The execute privilege has no meaning for directories. Thus, the execute and read privilege combination if the Bell-LaPadula model has no meaning either for Secure Xenix directories. Note, however, that the execute privilege bit is reassigned by the access authorisation mechanism to the search operation and thus it denotes the search permission.
add_ entry(w) delete_ entry (w) (rw)	A process granted the add_entry (w) privilege to a directory can execute instructions that cause new entries to be appended to that directory. The append privilege (a) of the Bell-LaPadula model maps directly into this privilege for directories. The Bell LaPadula write (w) access maps directly into the delete_entry privilege (rw) of Secure Xenix.
null (–)	The null privilege has the same interpretation for directories as that for files.

In summary, the Bell-LaPadula privileges correspondence to Directory Privileges as follows.

BLP	XENIX
e (execute)	–
r (read)	r (read)
x (search)	r (restricted read)
re (read & execute)	(x) no meaning
a (append)	w (add entry or delete entry)
w (write)	rw
–	null

(4) Privileges for Semaphores and Xenix Semaphores

The access privileges for System V semaphores are defined in the same was as those for files, and their relationship to the Bell-LaPadula privileges is the same as that of files. The execute (x) privilege has no meaning for semaphores and is ignored by the access authorisation mechanism. The write (w) privilege in isolation has no meaning for System V semaphores. Whenever the write privilege is on but the read privilege is off the write privilege is ignored by the access authorisation mechanisms. Thus, the only non-null accesses defined for System V semaphores are read (r) and read and write (rw).

For Xenix semaphores, the execute (x) privilege has no meaning and is ignored by the access authorisation mechanisms. Although the write privilege has meaning on semaphores in general, the Secure Xenix access authorisation mechanism reassigns that meaning of write to the read privilege and ignores the write privilege. Thus, the read (r) privilege for Xenix semaphores implies both observation and alteration and, therefore, it is equivalent to the write (w) privilege of the Bell-LaPadula model, and to read&write (rw) in Xenix.

In summary, the Bell-LaPadula Privileges correspond to System V Semaphore Privileges as follows.

BLP	System V
e (execute)	–
r (read)	r (read)
re (read & write)	x (ignored)
a (append)	w (ignored whenever read is off)
w (write)	rw (read and write)
–	null

Also, the Bell-LaPadula Privileges correspond to Xenix Semaphore Privileges as follows.

BLP	XENIX
e (execute)	–
r (read)	r (read)
a (append)	w (ignored)
w (write)	r (read and write)
–	null

(5) Privileges for Processes

The only privileges defined for processes (not to be confused with the process code file) are signal, kill, and null. The signal and kill privileges are implemented implicitly for every process and are a stylised form of a "write" to a process body. The null privilege is also implicitly implemented by the kernel through the process isolation mechanism; namely, two isolated processes have null privileges to each other.

5.8.1.4. The Current Access Set in Secure Xenix (***B***)

The current access set B is a subset of $S \times O \times A$. In Secure Xenix, the current access set is represented by a per-process data structure for some types of objects and by a per-type data structure for some other types.

(1) The Per-Process Component:

The per-process component of the current access set consists of a set of descriptors (fd) stored in the u_ofile structure of the per-process u_block. These descriptors point to a file table whose entries contain the current access privileges of semaphores and Xenix shared data segments, and directories. The file-table entries are multiplexed among objects of all processes. Each per-process descriptor points to an entry in the file table. The access privileges of each entry are a subset of the privileges that the process has to the object (discussed in the next section). Note that for semaphores and for shared data segments the current-access-privileges set is the same as the process always has to these objects; i.e., the same as the corresponding access matrix entry.

(2) The Per-Type Component

The per-type component of the current access set consists of special descriptors that contain the access privileges available to each process. These descriptors are semid_ds for System V semaphores, msgid_ds for message queues, and shemid_ds for shared memory segments. The ipc_perm field of these descriptors contains the access privileges a process has to these objects. Here, as for Xenix semaphores and shared data segments, the current access-privilege set is the same as that the process always has to these objects.

5.8.1.5. *The Access Matrix in Secure Xenix* (***M***)

The access matrix M of the system state is interpreted in Secure Xenix through a set of system structures maintained by the kernel. The system structures interpreted for each object as access matrix entries are either access control lists (ACLs) or Xenix (Unix) specifications but not both. These structures represent the storage of the access matrix by column. That is, each object is associated with a list of users that can access the object, each user having a set of access privileges restricting his access. Access control lists and Xenix (Unix) specifications are two different ways of storing the access matrix by column.

An ACL is a set of <principal identifier, access privileges> pairs that is attached to an object. The principal identifier is a non-reusable, two part identifier consisting of a user identifier and a group identifier (UID and GID). The user identifier places each individual user in a separate access control group by himself, uniquely. The group identifier places users in groups whenever such users are related by, or cooperage in, some activity or project. Such groups imply that their members have similar access privileges to a set of objects. A user may belong to several groups; however, at login time he must specify the group in which he wants to be for that login session. If no group is specified at login time, a default group is assigned to the user. Both group-membership and group-default definition on a per user basis are determined by the System Security Administrator (SSA). Default group specifications can be changed by the SSA at the user's request. Note that not all members of a group must be known when the group is formed. Members of a group may be added and deleted by the SSA subsequently.

To simplify principal identifiers, a DON'T CARE (i.e., "wild card") notation has been added [Saltzer 74]. A DON'T CARE in a user or a group field of a principal identifier is denoted by an asterisk (*). For example, the identifier Jones.Networks_FSD puts a user Jones in the Networks_FSD group. By contrast, the identifier Jones.* names a user Jones in any group, whereas the identifier *.Networks_FSD names any user in the Networks_FSD group. The inclusion and exclusion of individual users on ACLs and the review and revocation of privilege mechanisms are presented in [Gligor 86].

Both ACLs and Xenix protection specifications are associated in a one-to-one correspondence with the object they protect. For example, for the objects that have file system representation, the object i-node number is used to identify unambiguously its ACL. The ACL is destroyed upon object (and i-node) destruction. For objects that have file system representation the Xenix protection specification are kept in the i-node itself. For objects that do not have file system representation (i.e., System V semaphores, message queues and shared memory segments), the ACL or the Xenix protection specification is associated with the object through the object's descriptor (i.e., semid_ds, msgid_ds, and shemid_ds). For example, the ACL's i-node number is stored in the descriptor; the Xenix specification themselves are stored directly in that descriptor and used whenever ACLs are not specified.

5.8.1.6. The Security Function ***(F)***

The definition of the security levels as binary encodings, of assignment of print names to binary encodings, and of the (lattice) relationships between security levels is provided in [Gligor 86]. In this section we focus on the definition of the three components of the security function, namely, the assignment of maximum security level (clearance) to each subject, the current security levels (clearance) of each subject, and the assignment of security level (classification) to each object.

The assignment of user clearances in Secure Xenix is performed by the system security administrator (SSA) on an individual and group basis in the user security profile database. The individual user clearance consists of a User Maximum Level (UML), and the group clearance consists of a Group Maximum Level (GML). These values can only be assigned and manipulated by the SSA, and must be in the range $SystemHigh \triangleright UML$, $GML \triangleright SystemLow$ for the System_High and System_Low values defined by the SSA. The subject-maximum-clearance is the greatest lower bound (see [Gligor.86]) of the UML and GML.

The current subject clearance is called the current process level (CPL), and is assigned to that process for its entire lifetime. The CPL is determined at process creation time and must be between the process maximum level (PML) and System_Low. The PML is the greatest lower bound of the UML, the GML, and the terminal[34] maximum level (TML). Note that, because the TML is no greater than the workstation maximum level (WML), the WML is never lower than the PML. The TML and WML are discussed below. The CPL of a process is the user Requested_level at login time, or the user Default_level if no level is requested, if and only if the Requested_level/Default_level is less than or equal to the PML (or equivalently $PML \triangleleft UML$, $PML \triangleleft GML$ and $PML \triangleleft TML$). Therefore, it is clear that the subject maximum clearance always dominates the current subject clearance in Secure Xenix.

Note that a login fault is detected during the computation of the CPL (and PML). The fault occurs whenever the terminal maximum level (TML) is greater than the user maximum level (UML) or the group maximum level (GML). Consequently, an audit record is written. The reason for this action is that the user is likely to try to login from a security area where he does not belong. Also note that a user can always request a level that is lower than both the PML and TML, so long as both that user's GML and PML are no lower than the TML. No login fault occurs in this case.

---

34 This refers to the computer terminal. It is the maximum security level permitted by the physical location of the terminal or workstation.

The assignment of object classifications consists of the assignment of classifications to the workstation components and the assignment of classification to the user-created objects. The assignment of classifications to workstation components is performed by the SSA (during the definition of workstation security profile), whereas the assignment of classifications to user-created objects is done by the Secure Xenix kernel. The current level of the workstation devices is also assigned by the kernel.

The definition of the workstation security profile is performed by the system security administrator, and includes the following classification ranges:

(i) The individual workstation classification range; i.e., workstation maximum security level (WML) and System_Low.

(ii) The classification range of each individual terminal and private devices that are connected to each workstation; i.e., terminal maximum and minimum level (TML, TmL) and the private device maximum and minimum levels (PDML, PDmL).

The assignment of these values to a specific Secure Xenix configuration is performed by the SSA and depends on the operational and the physical security environment. For example, in some operational environments the System_High and System_Low, and all other security levels, may have the same clearance value but different category sets. In such environments, the security levels assigned to individual workstations, devices and file system depend solely on the "Need to Know" basis.

The dependency of the security level ranges on the physical security is equally important. For example, the workstations located to areas accessible to users cleared at low security levels have a lower classification than that assigned to workstations located in areas where all users are cleared at the highest level. Physical security considerations may also require

(1) that the maximum level of a terminal or private device be lower than that of its workstation ($TML \lhd WML$ or $PDML \lhd WML$), and

(2) that the minimum level of a terminal or private device be higher than System_Low ($TmL \rhd SL$ or $PDmL \rhd SL$).

Terminals and other workstation devices may be located in a different physical security area than that of its workstation, and, thus, the TML or PDML may be lower than the WML. Terminals and other private devices are also vulnerable to the additional threat of spoofing, and thus some information contained in the workstation may not be displayed on the terminal or on the private device.

A user can only change the level of a private device or of a terminal to a level that he requests at login time (viz., the computation of the CPL).

The current level of a private device of a terminal can be displayed by the kernel on request. The minimum level of a terminal or of a private device classification may be higher than System_Low because physical security considerations may require that individuals with a lower clearance, or with no need to know, may be denied access to workstations, terminals and private devices located in highly classified areas or in areas with different "need to know". This is done by raising the TmL or PDmL to a correspondingly high security level.

A workstation terminal, or a private device, also has a current classification, called the Current Terminal Level, or the Current Private Device Level (CTL or CPDL). In Secure Xenix, both the CTL and CPDL equal the CPL of the user, system process, or daemon to which they are attached (and that owns or opens them). Note that it is possible to have $CTL \lhd TmL$ because $CTL \lhd CPL$ and CPL equals $Request\_Level \lhd TmL$ of a user whose $UML \lhd TmL$ and $GML \lhd TmL$. For similar reasons, it is possible to have $CPDL \lhd PDmL$.

The determination of the classifications of the user-created (or opened) objects is performed by the Secure Xenix kernel, and consists of the following three groups of rules.

(1) *Classification of Files, Special files, Xenix Semaphores, Xenix Data Segments, and ACLs.*

Objects in this group have a single level for their entire lifetime. Exceptions to this are the special files whose activation level equals the level of the process that activates or opens them. None of the Xenix special files retain any state information in current configuration. Whenever such files retain state information, SSA intervention is required for activation. That is, unless a special trusted process with discretionary access to that object changes the object classification (i.e., downgrade or upgrade), the object classification does not change. The classification of an object in this group is the CPL of the creating process and must be equal to the security level of the directory containing that object.

An object in this group can only be destroyed by a process with the same (CPL) level as that of the object; the object is destroyed only if its reference count equals zero (i.e., it is not shared by any other directory or process). Note that special files are not destroyed; they are only closed.

(2) *Directory Classification*

A directory has a single security level for its entire lifetime, just as in the case or ordinary files. However, unlike ordinary files, the security level of a newly-created directory can be assigned from a range of levels. The lowest level of the range is the CPL of the creating process and must be equal to that of the directory that contains the newly-created directory. The highest level of the range is the WML. If a process creates a new directory but does not request any level for that directory, the default level of the directory is that of the process (i.e., the CPL) and that of the containing directory. The classification of a directory does not change during the lifetime of the directory unless a trusted process with discretionary access to that directory always changes it.

A directory can only be destroyed by a process at the same level (i.e., CPL) as that of the containing (parent) directory. Also, a directory can only be destroyed if it contains no files. This Xenix interface convention introduces a covert channel, discussed in [Gligor 86], because a lower level process can discover whether a higher level process has removed all the files from the higher level directory when it tries to remove them.

(3) *Classification of Processes, System V Semaphores, Message Queues and Shared Memory Segments*

The security levels that are assigned to these objects by the classification rules of the kernel always equal the CPL of the process that created these objects. Similarly, these objects can only be destroyed by the process that created them or by a trusted process at the same level as that of the objects. The classification of those objects does not change during their lifetime unless a trusted process with discretionary access to those objects changes it.

5.8.1.7. Hierarchy ***(H)***

The only Secure Xenix objects that may contain multiple components with different classifications are directories. Thus, the only object hierarchy in the system for the objects that have a file system representation is that provided by the directory hierarchy. All objects in this group (i.e., group (1) above) are classified at the level of the creating process, which must equal that of the directory containing the object.

Objects that do not have file system representation (i.e., objects in group (3) above) are classified at the level of their creator process. This ensures that these objects cannot be at a lower level than that of the processes' current directory. This also maintains the "non-decreasing level" rule for the directory hierarchy. These objects form the isolated points (i.e., the "stumps" in the Bell-LaPadula terminology [Bell76]) of the hierarchy.

The rules for assigning specific classifications to directories in the hierarchy prevent a process from placing a newly-created directory in another directory at a lower level than that process' CPL. However, a process can create an "upgraded" directory that has a higher level than that of the CPL of the creating process and that of its containing directory.

Note that a user process can create links in its current directory to objects that have file system representation. However, links to directories can only be created by trusted processes. User processes can only link (non-directory) objects in the process current directory (i.e., CPL=directory level), and only if the security level of the object being linked equals that of the current directory.

The Secure Xenix hierarchy has a root directory whose level is always System_Low. All processes have the search privilege (x) to this directory.

5.8.2. State Transitions in Secure Xenix

Transitions from state to state are defined by the kernel calls and returns of Secure Xenix. Thus, each rule ρ_i in $\rho: R \times V \rightarrow D \times V$ of the Bell-LaPadula model is represented as follows:

(1) Each request $R_k \in R$ is represented by a specific kernel call or by a trusted process call (these calls are implemented by kernel calls). R is the set of all kernel and trusted process calls.

(2) Each input to R_k comes from the current system state V. That is, both parameters explicitly passed to each call (such as object identifiers, values, pointers, access privileges, and so on) and parameters implicitly passed to each call (such as the system hierarchy, security levels, and so on) belong to the current system state.

(3) Each decision $D_m \in D = \{Yes, No, ?, Error\}$ is represented by a specific return to a kernel call. "Yes" is represented by the successful return parameter. "No" is represented by the error return parameter that corresponds to violations of the access control (e.g., mandatory or discretionary checks). "?" is represented by the error return parameters specifying that the kernel call parameters are faulty (e.g., non-existent file, parameters out of range, attempt to invoke a privileged kernel call, etc.). In general, these error returns are called domain errors. "Error" is represented by error returns that correspond to other exceptional conditions detected during the execution of specific kernel calls (e.g., deletion attempted on a non-empty directory, overflow conditions, etc.). Note that all decisions represent some information from the system state at the time of the kernel call, V, or from the new system state, V^*, entered by the system as a consequence of the call.

(4) Whenever $D_m \neq No$, $D_m \neq ?$ or $D_m \neq Error$, the output of R_k includes a new state V^*, in addition to new hierarchy, or may exclude some objects and access privileges from previous states, and so on.

The D_m's, the characteristics of the expected and of the new state for each R_k are described in the Secure Xenix DTLSs.

5.8.3. Access Control in Secure Xenix

In this section we report the invariant access control checks that are performed in Secure Xenix. This includes the presentation of (1) authorisation checks for mandatory control, (2) authorisation checks for discretionary access control, including the Setuid-gid mechanism, and (3) the computation of the effective access authorisation to objects.

5.8.3.1. Mandatory Access Authorisation

The authorisation rules are divided into three groups depending on the type of object being accessed.

(1) The object is a File, a Special File (Device), a Directory or a Shared Memory Segment or an ACL:

A process may Read (Execute) an object if the CPL of the process dominates the classification of the object.

A process may Write an object if the CPL of the process is equal to the classification of the object.

This rule implies that the data displayed on a private device or on a terminal can be a level that is no higher than that of the CPL and, implicitly, of the CPDL/CTL. Any displayed data that may have to be at a lower level can be labelled separately by a trusted process of the secure application itself. Thus, the application is responsible for providing labels for data fields that would be appropriate for, and that use, the different terminal (i.e., windowing, scrolling, etc.).

(2) The object is a Named Pipe, Semaphore, Message Queue, Xenix Shared Data Segment:

A process may Read/Write (open/close) an object if the CPL of the process equals the classification of the object.

(3) The object is a Process:

A process can signal (kill) another process if the CPL of the latter dominates the CPL of the former.

(4) For all objects, a process has NULL access to an object if the CPL of the process neither dominates nor is dominated by the classification of the object. The two objects are said to be isolated from each other.

The above rules imply that the flow of information in Secure Xenix can only take place from a given level to another level that is no lower than the first.

The mandatory access authorisation rules presented above are invariant for all Secure Xenix kernel calls. That is, depending on whether a kernel call is relevant to a particular type of object, one or several of the above rules apply to that call. These rules are compatible with the SS-property and the $*-$property of the Bell-LaPadula model for the following reasons.

(1) Rules 1 and 2 of Secure Xenix imply conditions (ii) and (iii) of the $*-$property.

(2) Rule 3 of Secure Xenix implies condition (i) of the *-Property.

(3) Since the subject maximum clearance (i.e., the greatest lower bound of UML and GML) always dominates the current subject clearance (i.e., CPL) in Secure Xenix, Rules 1-3 above imply the SS-property.

However, it should be noted that equivalence between the SS-property, the *-property of the Bell-LaPadula model and any system interpretation is impossible in practice. There are two reasons for this.

First, consider the meaning of the execute (e) privilege of the Bell-LaPadula model presented in section 2.4 above. This privilege does not exist in practice because, in any system, the execute privilege implies some observation of the behaviour of the object being executed. Therefore, in practice, the execute (e) privilege must be eliminated from condition (i) of the SS-property and added to condition (ii). Furthermore, it must be also added to condition (iii) of the *-property; otherwise, observation of objects at levels that are higher than those allowed to the user or his process is possible.

Second, consider the implementation of the "append" operation that requires the append privilege (a) of the Bell-LaPadula model. In practice, append operations may require one or more of the following observations of objects or system state:

(1) find the end of the object that is target of the append operation;

(2) find the name of an object in a directory at a higher level than that of the process executing the append operation;

(3) find out whether the object exists;

(4) find out whether the append operation fails due to a storage channel exception.

Consequently, in practice, the append operation implies not only alteration of an object but also observation of the object or of the system state. Therefore, in practice, the append (a) privilege must be eliminated from condition (i) of the SS-property and added to condition (ii) of the $*-$property for the similar reasons to those mentioned for execute (e) above.

With the above two modifications that are required in practice, the SS-property and the *-property would be equivalent to the rules 1-3 of the Secure Xenix implementation. Note, however, that consistency of the Secure Xenix interpretation with the model only requires that Rules 1-3 above imply the SS-property and the *-property.

5.8.3.2. Discretionary Access Control

The discretionary access authorisation rules of Secure Xenix define the Secure Xenix model of discretionary policy. Discretionary policy is characterised by four classes of axioms, namely, (1) authorisation axioms, (2) axioms for distribution of access privileges, (3) axioms for review of access privileges and (4) axioms for the revocation of access privileges. The informal specification of the first three classes of axioms are required explicitly by the [TCSEC 83] in the discretionary access control area of B2-class system. The informal specification of the fourth is required implicitly in the statement that "the enforcement mechanism shall allow users to specify and control sharing for these objects.

(1) Authorisation in Secure Xenix

The specification of the discretionary authorisation mechanisms of any system consists of two parts. First, it must include a specification that relates every (kernel) operation on one or more objects with the privileges required by the operation for those objects. This is necessary because the authorisation mechanism requires different (combinations of) privileges for different operations. Lack of such specification could mean that the wrong privilege may authorise an operation. As seen in section 3.1.3 above, the correspondence between an access privilege to a kernel operation depends on the type of objects and is not entirely obvious.

Second, the discretionary authorisation mechanisms must include a specification of how the current access privileges of subjects are related to the specification of the subjects access to objects by the access matrix. This relationship is defined by the DS-property of the Bell-LaPadula model, and is important because it relates the high-level, human-oriented, discretionary access controls specified by the access matrix with low-level, human-oriented, discretionary access controls specified by the access matrix with the low-level, system-oriented, discretionary controls of the system.

The requests R_k discussed below, namely, CALL, REVOKE, REVIEW, ACCESS, GRANT, EXCLUDE are implemented by kernel calls or sequences of kernel calls that require the reading and writing the ACL and Xenix specifications. ACCESS and CALL are implemented by a single kernel call (i.e., "access" and "exec").

The two general requirements of discretionary authorisation can be expressed by the following two axioms. Let $\rho\colon R \times V \to D \times V^*$ be the set of rules.

For all R_k executed by S_i on some objects O_j with $R_k \neq$ CALL, GRANT, REVOKE, REVIEW, ACCESS, or EXCLUDE, and $\underline{x}$ are the required access privileges for R_k.

(1.1) $D_m = Yes \quad \Rightarrow \quad (s_i, o_j, \underline{x}) \in B$ and

(1.2) $(s_i, o_j, \underline{x}) \in B \quad \Rightarrow \quad \underline{x} \in M_{i,j}$ [Bell-LaPadula 76].

Secure Xenix satisfies both requirements mentioned above. First, the DTLSs of Secure Xenix specify the discretionary privileges for each type of object that are required by each kernel call. Furthermore, the kernel call fail whenever the required privileges are not among the privileges of each object used by the call. Second, each current access of a process to an object is derived from either the objects' ACL or from its Xenix specifications (i.e., i-node, semid-ds, msgid-ds, shemid-ds) when the object is open or created; viz., section 3.1.4 above. Because these data structures represent the access matrix in Secure Xenix (viz., section 3.1.5 above), the DS-property of the Bell-LaPadula model is also satisfied.

(2) Distribution of Access Privileges in Secure Xenix

The policy for the distribution of access privileges must specify how "the access permission to an object by users not already possessing access permission shall only be assigned by authorised users" [TCSEC 83].

In Secure Xenix, the only users that are authorised to distribute object privileges to other users are the owner of those objects. Ownership of an object is a user attribute and not an access privilege. In Xenix, ownership is determined solely by the user identifier (and not by the group identifier GID). Each object has only one owner and only the owner is authorised to modify either the ACL or the Xenix specifications for his objects.

This privilege distribution policy is succinctly stated by the following two axioms.

(2.1) Ownership Axioms:

$$(\forall i \neq j)(\forall s_i, s_j \in S') \quad s_i = Owner(o_i) \quad \Rightarrow \quad s_j \neq Owner(o_i) \text{ and}$$
$$s_i = Owner(o_i) \quad \Rightarrow \quad (for\ \underline{x} \in A)\ \underline{x} \in M_{i,i}$$

(2.2) Privilege Granting Axioms:

Let $\rho: R \times V \rightarrow D \times V^*$ be the set of rules. For all R_k executed by s_i on objects o_j with $R_k = grant(\underline{x}, s_p)$ we have $(D_m = Yes) \Rightarrow s_i = Owner(o_j)$ and $\underline{x} \in M_{p,j}$.

The effects of the privilege granting are equivalent to the inclusion of a user/group identifier on an ACL or in the Xenix specifications. The inclusion of users on ACL's is explained in section 3.1.5 above and on Xenix specifications for an object in [Ritchie 74].

Similarly, the policy for the distribution of access privileges must be able "to specify a list of named individuals and a list of groups of named individuals for which no access to the objects is to be given".

In Secure Xenix this is possible since the owner can either decide not to include a specific user or group in the ACL or Xenix access specification or to exclude a specific user's or group's access as explained in section 3.1.5 above.

(2.3) Privilege Exclusion Axiom:

Let $\rho: R \times V \to D \times V^*$ be the set of rules. For all R_k executed by s_i on $M[s_i, o_i]$ with $R_k = excludes(\{S_j\}, o_i)$ we have $(D_m = Yes) \Rightarrow (\forall j \neq i) \quad s_i = Owner(o_i)$ and $(S_j, o_i, \varnothing) \in B$, where $\{S_j\} \subset S'$.

(3) Review of Access Privileges in Secure Xenix

The policy for review of access privileges "shall be capable of specifying, for each named object, a list of named individuals and a list of groups of named individuals with their respective models of access to that object" [TCSEC 83].

In Secure Xenix, the only user that can perform access review (i.e., reading the ACL or the Xenix specifications) for an object is the owner of that object. However, any user can inquire whether he has access to an object, and what type of access, regardless of the object's ownership. This can be succinctly stated by the following axioms. Let $\rho: R \times V \to D \times V^*$ be the set of rules.

(3.1) For all R_k executed by s_i on $M[s_i, o_i]$ with $R_k = review(o_j)$ we have $(D_m = Yes) \Rightarrow s_i = Owner(o_j)$

(3.2) For all R_k executed by s_i on o_j with $R_k = access(o_j)$ we have $(D_m = Yes) \Rightarrow s_i \in S'$

(4) Revocation of Privileges in Secure Xenix

The policy for revocation of privilege must specify how access privileges for an object can be taken away from users that have these privileges in a selective manner and, possibly, partially.

In Secure Xenix the (selective and partial) revocation of access privilege can be performed only by the owner of an object. The reason is that only the owner of the object may modify ACLs and Xenix specifications. This can be expressed succinctly by the following axiom. Let $\rho: R \times V \to D \times V^*$ be the set of rules.

(4.1) For all R_k executed by s_i on $M[s_i, o_i]$ with $R_k = revoke(\underline{x}, s_p)$ we have $(D_m = Yes) \Rightarrow$ (for $i \neq p$) $\underline{x} \in M_{p,j}$ and $s_i = Owner(o_j)$

(5) The Setuid-gid Mechanism of Secure Xenix

The SETUID protection mode is used to build controlled interfaces to various objects [Ritchie 74]. Whenever a program with the SETUID bit is executed, the invoking process inherits the privileges of the program owner. Every process has both a real and an effective user identifier that are identical except when a SETUID program is executed. Then the effective user identifier is set to that of the program owner. All discretionary access control decisions are based on the effective use identifier and not on the real one. There is a similar mechanism called the SETGID mechanism for changing the effective group identifier.

Although the SETUID feature can be very useful it also poses three types of security risks. First, a poorly designed SETUID program can compromise the program owner's security. Second, the code of the SETUID program may be modified in an unauthorised way. Third, a Trojan Horse in a borrowed program may steal a user's privileges by creating a SETUID program.

The modifications to the SETUID/GID mechanism presented in [Gligor 86] make it impossible for a user to change an existing SETUID program, or for a Trojan Horse to steal user's privileges by creating a SETUID program. However, it is the responsibility of the user to use extreme care in the design of SETUID programs. The operating system cannot protect the user from his own mistakes. Even if the user is careless or malicious, he can only hurt himself by misusing the modified SETUID features mentioned above because he cannot create a SETUID program under a different user's identifier. Note that the mandatory access control (discussed below) remains unaffected by the SETUID mechanism.

The SETUID/GID mechanism of Secure Xenix enforces the separation of privileges between the subject that invokes a SETUID/GID program and the subject that owns the program. This means that a subject invoking a SETUID/GID program may only have indirect access to some of the objects of the SETUID/GID program owner. This can be expressed succinctly by the following axiom:

(5.1) Let $A = \{r, w, x, \text{null}, \text{suid\_gid}\}$.
$[s_j, o_j, indirect(\underline{x})] \in B \quad \Rightarrow \quad call(s_j, o_k)$, $s_i = owner(o_k)$,
$\text{suid\_gid} \in M_{i,k}$ and $(s_i, o_i, \underline{x}) \in B$.

In other words, if a program has the SETUID-GID bit on, the subject s_i executing it has privileges of the owner to objects o_i that may not be directly available to that subject's callers (i.e., s_j).

5.8.3.3. Computation of the Effective Access in Secure Xenix

The effective current access of a subject to an object in Secure Xenix follows two rules. These rules are compatible with the Bell-LaPadula model. They are:

(1) A user process is allowed to access an object in a given mode (i.e., requiring a certain privilege) only if both mandatory and discretionary checks are passed.

(2) The Error value returned for failed discretionary checks must be the same as that returned from failed mandatory checks unless the mandatory checks have passed.

The first rule is necessary because, otherwise, the requirement of the secure states and the Basic Security Theorem of the Bell-LaPadula model would be violated. The second rule is necessary because otherwise leakage of information rule is necessary because otherwise leakage of information from higher levels to lower levels may be possible. That is, whenever the discretionary checks are done first, a higher level subject may revoke or add a privilege for an object to a lower level subject. The lower level subject would then distinguish between denied discretionary access and denied mandatory access errors. Thus, by modulating the discretionary access of the lower level subject to a higher level object, a higher level subject could transfer information to a lower level object. In Secure Xenix, the mandatory access checks are performed before the discretionary checks for every kernel call accessible to a user process. However, this is a stronger requirement than the more general one specified in (2) above.

5.8.4. Initial State z_0

The initial state of any Secure Xenix installation is set by a secure initialisation procedure. The secure initialisation consists of three distinct phases:

(1) System configuration and generation.

(2) System and user profile definition.

(3) Normal start-up or Initial Program Load - (IPL).

The first phase is performed by the Trusted Systems Programmer (TSP) in Maintenance mode. Once this mode is left, the TSP functions are automatically disabled, and only the rest of the administrative users have access to the workstation. The second phase work is performed by the SSA. The third phase work, normally the IP, is performed by anybody with physical access to the power switch of the workstation.

The IP of the Secure Xenix can only take place with input from the fixed disk, whereas in maintenance mode, the IP can only take place with input from the removable media (e.g., diskette) drive. This separation of

IP input is enforced by a special hardware configuration that, when installed by the TSP, prevents user mode IP from using the removable media drive. No cryptographic authentication of the removable medium [Gligor 79] is performed at this time. The TSP is the only administrative use that has access to the internal hardware configuration and he would have to be trusted to configure the system correctly anyway. If the TSP were not trusted, on-site physical surveillance methods would become necessary, cryptographic authentication notwithstanding.

During the Secure Xenix IP, several consistency checks are performed. Xenix already performs file system consistency checks (i.e., through the "fsck" program). In particular, the IP recovers whenever the system is started up after an improper shut-down (after a crash, after power-off during disk I/Os, etc.). This ensures that security label consistency is maintained because each label is written onto the disk with a separate, atomic sector-write operation. In addition to the file system consistency checks, Secure Xenix checks (1) the consistency of the security map, (2) the consistency of the current object label, and (3) the consistency of the overall security level hierarchy (i.e., the non-decreasing security levels for directories). This is done by the "scheck" program.

5.8.5. Compatibility in Secure Xenix

The interpretation of the compatibility axiom in Secure Xenix requires that a directory contains (1) non-directory objects (which have file system representation) only at the same level as that of the directory, and (2) directory objects at the same level as that of the directory or higher. Consequently, if a directory is at a higher security level than that of a subject, all object fields in that directory remain inaccessible to the subject.

The rules for object classification discussed in section 3.1.6 above, and the definition of the Secure Xenix hierarchy discussed in section 3.1.7 above, imply that the compatibility axiom is satisfied.

5.8.6. Tranquillity in Secure Xenix

The section 3.1.6 is specified that both the current process level (clearance) and the classification of objects in Secure Xenix do not change during the lifetime process and of an object, respectively (unless a trusted process with discretionary access to those objects or with root privileges changes those levels). This suggests that the kernel call accesses, and the clearance and classification rules of Secure Xenix satisfy the tranquillity principle of the Bell-LaPadula.

5.8.7. Activation

The design of Secure Xenix satisfies the two activation axioms defined in section 2.4.6 above. First, an active object can become inactive only through destruction. Objects that have file system representation ate inactivated by the destruction of their i-nodes and by the de-allocation of the appropriate table entries (i.e., file descriptor, file table entry). Objects that do not have file system representation are inactivated by the destruction of their descriptors and of their table entries (i.e., shared memory, semaphore and message queue table entries). A process is destroyed by destroying the corresponding process table entry. Consequently, the destruction of all these objects makes them inaccessible to any active process.

Second, whenever an object is created (activated) the state of the object is automatically written by the kernel with zeros. Thus, the previous states of this object, or of any other destroyed object whose storage representation is being reused, are erased before reuse. This is discussed in more detail in a separate document on object reuse. Thus, the state of a newly activated object cannot depend on the state of any previous object incarnation.

Note that the destruction (inactivation) of some objects, such as some special files representing terminals, does not cause the object representation to be "erased". Whenever such objects do not retain state information they can be reactivated and made accessible to different processes (and labelled accordingly; viz. section 3.1.6 above). However, the activation of objects that retain state information after their deactivation requires the intervention of the SSA and of trusted processes (i.e., mount/unmount volumes).

Secure Xenix also satisfies an additional activation axiom that defines the classification of a newly activated (created) object and the object destruction rule.

Consistency with the compatibility axiom and with the $* -$ property requires:

(3) Classification of Newly Activated Objects & the Object Destruction Rule

Let $O = O' \cup O''$, where $O'(O'')$ are the active (inactive) objects, and

$$new(o) \Rightarrow [\{O' = O'' - o\}\{O'' = O' + o\}] \text{ and}$$
$$destroy(o) \Rightarrow [\{O'' = O' - o\}\{O' = O'' + o\}].$$

Let $H^{-1}(o)$ designate the parent of object o.

$$call(s_i, new(o)) \Rightarrow \{H^{-1}(o) \neq \varnothing \Rightarrow f_o(o) \triangleright f_c(s_i) = f_o(H^{-1}(o))\}$$
$$call(s_i, destroy(o)) \Rightarrow \{H^{-1}(o) = \varnothing \Rightarrow f_o(o) = f_c(s_i)\}$$

The interpretation of these axioms in Secure Xenix is discussed in section 3.1.6 above.

5.9. Conclusion

The Bell-LaPadula model is a state-based model of system actions. It is useful in developing assurance that certain properties of a system are obeyed. The TCSEC clearly is dependent upon it. It is easy to see by construction that the access control mechanisms of Secure Xenix satisfy the axioms of the Bell-LaPadula model.

RISK ANALYSIS AND ASSET LABELLING

6. RISK ANALYSIS AND ASSET LABELLING

Computer installations that are to be used with sensitive data assets must undergo an evaluation to gauge their suitability for this purpose. Access to such data assets must be carefully controlled to ensure that only users who have appropriate authorisations can observe or manipulate it. Data labels need to be definitive in their description of the sensitivity of data because the data access controls will depend directly on the values of the data label. The risk that data labels can become corrupted must be carefully controlled so that we can develop any reasonable expectation that the access controls can have a useful purpose.

The worth of data is defined as the relative importance or value or merit of the data in respect of its qualities or in respect of the estimation in which it is held in a particular setting. We will speak almost universally of labels that give the worth of data assets, meaning some measure such as security worth, utility, or some other aspect of the data may in a practical situation be implied by this term. The analysis can easily be extended to situations in which the monetary value of some sort of asset is known. We are not particularly concerned with the measure used to record worth or with how the worth of a data object is deduced. We insist on only three things:

(1) All the data assets that are under discussion have their worth recorded in a label that is associated with the data.
(2) All the data assets that are under discussion have their worth measured according to the same standard, so that the result of a comparison of the worth of any two objects can be meaningful.
(3) A method exists to compare the worth of data assets. We will need to know by inspection of their labels the relative worth of two data objects, or that there is no relation between their worth.

The first of these conditions merely says that if we are to study the evolution of the risk involved in a label, then that label must exist. The second condition asserts that when we compare the worth of two objects the comparison must mean something. We will take the third condition to imply that the possible measures of worth must either be totally ordered or a partially ordered. This arrangement is similar to the worth schemes that appear in the security literature[35].

The users of a system are cleared to access data up to some known worth. By access we mean observation or modification or any other manipulation that pertains to the access policy of interest. It follows that the measure of user clearance is the same as the measure of data worth. Thus, a decision as to whether a user can access a data object can be based upon a comparison of the user's clearance and the data's worth.

Our risk analysis is characterised by the range of data worth and user clearances that are involved in an abstract situation. Each data object is given some intrinsic worth when it is created. In most security-enforcing systems this worth may be changed later only with human intervention, normally done only on an exceptional basis. However, the data will acquire a succession of new values as computation progresses. These new data values occur without user intervention.

In this essay we will derive conditions that, if obeyed, ensure that the risk associated with the correctness of an outcome's data label can not be greater than the risk associated with the data labels on the inputs to the computation. If these conditions are not obeyed it is trivial to set up a sequence of plausible calculations such that the risk associated with the labels on data objects rapidly increases. This is done by recycling data that is the result of a calculation back as an input in a subsequent calculation. The label can rapidly become unacceptably questionable as an indication of the worth of the data it purports to describe.

6.1. Our Risk Measure

The assurance that a data asset is correctly labelled is the essence of our risk analysis. For example, if the worth of some data is described by its label as *Unclassified* and if there is data with a *Secret* worth in the machine, we want to have a measure of the assurance we have that the purportedly *Unclassified* data value is not tainted with *Secret* data[36]. The computer system could not be judged to be capable of enforcing policy if the labelling can deteriorate so that a given data label may bear a poor

[35] The following four papers are perhaps the classical references:
D.Elliot Bell and Leonard J. LaPadula, *Secure Computer Systems: Mathematical Foundations*, MTR-2547, Mitre Corp., Bedford, MA. March 1973.
Dorothy E. Denning, *A Lattice Model of Secure Information Flow*, Comm. ACM, Vol. 19, No.5, pp. 236-243. May 1976.
Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-STD, Library No. S225,711. December 1985.
Canadian Trusted Computer Product Evaluation Criteria, V3.0, Canadian System Security Centre, Communications Security Establishment, January 1993.

[36] The notion that data of relatively lower worth can be tainted with data of higher worth does not serve easily as a metaphor for many other situations. One would not consider lead that is "tainted" with gold to be worth less than lead not so tainted. There is a parallel development, however, that can deal with this inverse circumstance.

relationship to the label that the data should have. The degree of possible imperfection in the relationship between the data and its label in a given situation is the risk that the computer will perform adequately when used in that situation.

The risk associated with an asset is a function of the worth of the asset, the vulnerability of the asset to any relevant attack, and the nature of the threats deployed against the asset. For a function $\Re$ that is rarely if ever known, the function $\text{risk} = \Re(\text{worth, vulnerability, threat})$ captures this relationship. The risk function is not useful as an absolute measure. In most cases an instance of a risk is measured relative to some other instance of risk. A statement that a given risk is, say, 7.8 units, is meaningful only when used as a comparison with greater or lesser risks. We shall follow this relative approach throughout our work.

We desire to perform a risk analysis in order to deduce simple rules that will determine whether or not to permit a known body of users to operate a given computer system

- in a particular environment
- with specific input and system data
- that has a known range of worth.

We will use the term evaluation to refer to this binary decision to permit or prohibit use. The system is evaluated to determine its suitability for an identified use in a particular environment. The objective of an evaluation is to decide whether the clearances of the users, the capabilities of the system, the environment in which it to be used and the properties of the data are consistent with a tolerable risk.

An acceptable evaluation may be granted to an installation in spite of a low assurance level of the system's ability to keep data of different worth separate. The evaluation could be positive if the evaluating authority considers the separation adequate given the proposed operating environment. The same hardware and software might be judged inadequate if there were some users with low clearances, or if different external handling procedures were proposed. The same system in two distinct installations could be approved in one and not approved in another, because in the second the external handling of potentially corrupted data is deficient.

6.1.1. Worth

Data can be stored in a computer, or data can be manipulated by a computer to produce new internal data values or new data results that are output to the users and to the environment of the computer. The hazard is that the label on a data object misrepresents its worth. The only serious misrepresentation is the case where the label alleges a worth that is lower than it should be. Data labelled at too high a level may be difficult to deal with but it does not represent a risk to worth, but rather a risk to the ability to access[37].

The label could have been mislabelled initially with a lower worth than it should have. This represents an error on the part of the user who initially classified the data and produced the label. There is no currently practical way that a computer could correct a mislabelling by a user.

The label also could indicate a lower worth than it should if data of one worth becomes somehow contaminated with data of a second and higher worth without the label changing to represent the presence of the corrupting data. In the security world the corrupting data is at a higher worth than the original data, as in *Confidential* data being corrupted with *Top Secret* data without the label changing to represent the new status of the data. The sense of the corruption is a property of the policy that is being enforced. For our purposes we will limit our analysis to the mislabelling of low level data that has been tainted with data that has a higher worth. The development is adaptable to other policies.

6.1.2. ***Vulnerability***

The vulnerability of a data object's label is a measure of the confidence that the object is correctly labelled and that the computer is capable of preventing a high-level data object from corrupting another data object with a lower worth. There are two extreme situations:

(1) If the label on an object is *Completely Unreliable* the data may be highly vulnerable to being misused even if the access controls work flawlessly, because the access controls rely on the label to describe accurately the data object. It is vulnerable because the label is unreliable.

(2) The label may be *Completely Reliable.* In this case access control can be appropriately managed by effective mechanisms.

6.1.3. **Threat**

Let us focus on some arbitrary data item o_1 with worth W_1. The threat that we have in mind is embodied in the presence in the system of data with worth W_2 that is greater than W_1 and that might have affected the value of o_1 in a way that is not recorded in the label of o_1.

Consider a standalone system that is to contain a mix of *Secret* and *Confidential* material, but which has users who are all cleared to deal with *Secret* data. Assume that the configuration of hardware and software and any security analyses performed on them are such that there is a low level of confidence in the system's ability to keep the data worth *Secret* separate from the data worth *Confidential*. The risk of direct

[37] Data that is labelled overly high may represent a real risk if there is a policy that requires a certain accessibility of data. The analysis that follows is easily adapted to deal with the circumstance.

security problems is low as long as all users are formally cleared to access all data. The remaining very real risk is that *Secret* data and *Confidential* data might combine in a calculation and some of the output might be erroneously labelled *Confidential*. Subsequently, this *Confidential* data is at risk of being mishandled because of its label is too low. This presents a real risk if *Confidential* data is ever removed from the installation. There is not the same risk when *Secret* data is exported. For an acceptable evaluation the external conditions of the installation need to control this risk.

6.2. The Evaluation of Risk

We view the evaluation process as the determination of the level of risk associated with a system. The ultimate goal is to develop conditions for a system such that the risk associated with its outputs is not greater than the risk associated with its inputs.

We compare the worth of data items with the relation $\rhd$, which could be a total order (as in our examples[38]) or a partial order on worth. If $W_1 \rhd W_2$ then W_1 is judged to be of superior worth than W_2. We say that W_1 dominates W_2. The reverse notation $\lhd$ is useful; if $W_1 \lhd W_2$ then W_2 dominates W_1.

We represent the functional dependency implied by the risk function $\Re$ associated with a data label and a worth relation $\rhd$ as $\Re = \langle W, R, X \rangle$ where W is the worth of the data object, the vulnerability R is the reliability with which the worth is known and the threat X is the worth of the most serious contaminant. In this way we describe the influence on a risk function of a vulnerability and a threat.

The actual units of measurement of the reliability R are not important, except that they are required to be able to represent the reliability spectrum from *Completely Unreliable* through to *Completely Reliable*[39] and they must be totally ordered such that any two reliability measures can be meaningfully compared. The decision to require the reliability values to be totally ordered is made for the sake of simplicity; other orderings are possible but they overly complicate the analysis. If the reliability R is less than *Completely Reliable* then there is a greater-than-zero risk that the data has been tainted with some higher-level data. The worth X is the worth of the most serious possible contaminant that may be corrupting the data of interest. This must be some data with a higher worth than that of the data of interest that has coexisted with it in some pertinent environment.

38 We use the classifications *Top Secret > Secret > Confidential > Unclassified*, abbreviated as *TS>S>C>U*, in our examples. This makes explicit the fact that these measures are totally ordered. Everything is directly convertible to partially ordered worth measures.

39 We use *Completely Reliable > Extremely Reliable > Very Reliable > Slightly Reliable> Completely Unreliable*, abbreviated *CR>ER>VR>SR>CU*, in our examples. Any such sequence of totally ordered designations will suffice.

The decision to represent only the single most serious contaminant is made for the sake of simplicity. It is possible that there may be uses for risk measures that include several $\left[R, X\right]$ pairs. Adopting this approach complicates the analysis unduly.

The necessity of having both R and X in the risk measure is apparent when one considers two extreme cases:

Completely Reliable labels:

When the data of interest is labelled with an R value of *Completely Reliable* the X value is superfluous. There can not have been any contamination.

Maximum Worth:

When the maximum possible contaminant is dominated by the worth of the data of interest, $W \triangleright X$, there can be no risk. The R value is irrelevant because no higher worth data was available as a contaminant in the system of interest.

6.3. Risk Factors

When more than one data item contributes to a risk we must adapt our definition of risk function to represent accurately the possibilities. We chose to be pessimistic in our approach; we will identify the riskiest combination of elements and use this worst view of possible outcomes as the risk. We observe that in many cases, the worst case scenario is far less likely than any typical case, but it is always safe.

When we combine several data items and achieve some outcome, the risk involved with that outcome is not a risk function as it was defined above. We will define a risk factor Φ as the risk involved in the combination of several data items.

We define the function **LUB** as follows:

$$Y = \mathbf{LUB}(W_1, W_2, W_3, \cdots) \text{ iff } (\forall k)\; Y \triangleright W_k \text{ and } \neg\exists Z \text{ such that } (\forall k)\; Y \triangleright Z \triangleright W_k .$$

Let several data objects, each with risk $\Re_k = \langle W_k, R_k, X_k \rangle$, be indexed with k and be combined by some process to develop an outcome. Security policy undoubtedly dictates that the nominal worth of the outcome $\overset{out}{W}$ be such that $\overset{out}{W} = \mathbf{LUB}(W_1, W_2, W_3, \cdots)$ because it specifies that $\overset{out}{W}$ be an upper bound on all data worth that participate in the process. The possibility that some contamination of one of the input data objects will taint the output will appear explicitly as a risk of contamination in the output, and consequently will not directly affect $\overset{out}{W}$.

We define the risk factor Φ and the function **MIN** as follows:

$$\Phi = \left\langle \overset{out}{W}, R_\phi, X_\phi \right\rangle = \underset{k}{\mathbf{MIN}}\left(\left\langle \overset{out}{W}, R_k, X_k \right\rangle\right) \text{ iff } (\forall k) R_\phi \le R_k \text{ and } X_\phi = \mathbf{LUB}\big((\forall k)\; X_k\big).$$

The function MIN determines R_ϕ and X_ϕ, where R_ϕ is the least reliability of the labels of the data items that are being combined and X_ϕ is the least upper bound of the contaminants of the data items that are being combined. In practice X_ϕ will often be the worth of the system's highest-level data object.

We have been careful to speak of the outcome of the combination of several data objects. One scenario is that some data being input to a computation results in the generation of a new value for one or more data items. There are other possible outcomes, such as logout, login, or some data or action being passed to the environment that the system occupies. We intend to encompass all these effects, but for simplicity we concentrate on data labels.

6.3.1. Risk and a Reasonable Authority

A meaningful partial order on risks can be defined. Let $\Re_k = \langle W_k, R_k, X_k \rangle$ for all *k* of significance. The partial order on two risks $\Re_A$ and $\Re_B$ is defined as risk $\Re_A$ <u>is at least as good a risk as</u> risk $\Re_B$, written $\Re_A \to \Re_B$, if and only if $W_A \rhd W_B \wedge R_A \geq R_B \wedge X_A \lhd X_B$.

Thus, information labelled $\langle S, VR, TS \rangle$ is at least as good a risk as information labelled $\langle C, VR, TS \rangle$. If everything else is equal the risk associated with manipulating data labelled *Secret* is less than that with data labelled *Confidential* because the handling procedures for *Secret* data are more restrictive and the potential contaminant is closer in level to *Secret* than to *Confidential*.

The second conjunct asserts that increasing the label reliability decreases the risk, and the third conjunct says that lowering the security level of the contaminating data also lowers the risk. These definitions are unlikely to be contentious.

The assurance component *R* of the risks has been defined so vaguely and imprecisely that it is difficult to compare different risks. For two risks $\Re_A$ and $\Re_B$ let $W_A = W_B$ and $X_A \rhd X_B$. The question of which of the risks given by $\Re_A = \langle W_A, VR, X_A \rangle$ and $\Re_B = \langle W_B, SR, X_B \rangle$ is more risky and which is less risky is hard to decide. This is an issue properly dealt with by individual authorities, considering the system environment, the type of asset, the desired requirements on labels, the user clearances, the physical security measures, and other such issues. The relation $\to$ is defined pessimistically because $\Re_A \to \Re_B$ if and only if all three components of $\Re_A$ are at least as risk-free as those of $\Re_B$. This is because $\to$is a partial order. In the case above, and in many other cases, human intervention is needed to decide whether $\Re_A \to \Re_B$ or $\Re_B \to \Re_A$ or, perhaps, $\Re_A = \Re_B$

An evaluating authority, a person or a purpose-designed algorithm, need not be tentative in ordering risks. An authority should possess a more comprehensive definition of risk comparison and be willing to apply it to these enigmatic cases. Different authorities are likely to have different definitions of this relation. The notation $\Re_A \Rightarrow \Re_B$ will be used to denote the notion that the <u>evaluating authority considers risk</u> $\Re_A$ <u>to be at least as good as risk</u> $\Re_B$.

It is expected that $\Re_A \Rightarrow \Re_B$ may hold for many $\Re_A$ and $\Re_B$ where the more pessimistic relation $\Re_A \rightarrow \Re_B$ does not hold. But, the definition of $\rightarrow$ is obvious enough that if it is true that $\Re_A \rightarrow \Re_B$, a reasonable authority will agree that $\Re_A \Rightarrow \Re_B$ as well. We modify the definition of $\Rightarrow$ so that the authority we are dealing with is reasonable.

Thus, for example, it always is true that $\langle C, VR, S \rangle \Rightarrow \langle C, VR, TS \rangle$ because $\langle C, VR, S \rangle \rightarrow \langle C, VR, TS \rangle$. However, $\langle C, VR, S \rangle \Rightarrow \langle C, ER, TS \rangle$ might be true if the authority so decides.

Further discussion will be restricted to installations governed by reasonable authorities whose risk comparison relations are as described.

6.4. Risk in Input Labelling

The task of correctly initialising the worth of data fed into the system is hopefully achieved with high reliability because the sources are human users who are trained to perform this function well. If the situation is otherwise there is not much that can subsequently be done in the system.

There are, however, sources of data that do not inspire this level of assurance; the label on any data that comes to a system from a network may have moot reliability. The reliability of the source computer in labelling its output data becomes a factor in determining the risk inherent in the arriving data. This situation is a reflection of the fact that two computers that are actively exchanging data and are connected by a network form a single system.

External handling procedures for some forms of input, such as magnetic tapes or floppy disks, are a time-delayed version of the network connection. Such input sources may be input by the receiving system only with a human review of the worth of the data. Eventually, such review becomes a chore and in consequence unreliable. In most other cases, policy should require that an authorised user review and properly classify information entered into the system. The intention is that the knowledgeable authorised user decreases the risk in the reliability of the worth labels assigned to the input data he or she reviews.

Input data can come from many sources, not only human users. Consequently, the possibility that labels on received data are unreliable must be considered. The *input risk function* associated with a label on input data o_k is $\overset{in}{\Re}_k = \left\langle \overset{in}{W}_k, \overset{in}{R}_k, \overset{in}{X}_k, \right\rangle$.

The use of $\left[\overset{in}{R}_k, \overset{in}{X}_k\right]$ pairs, providing a distinct $\overset{in}{R}_k$ measure for each possible contaminating situation, could give a more thorough analysis. We choose to ignore this complication far the sake of brevity.

6.5. Risk in Separation

This component describes the system’s ability to maintain the accuracy of security level labels and to control effectively the access to data objects according to those labels within the system. It is concerned with the system’s ability to prevent undetected leakage of higher-level data into lower-level results. When a computing system design is evaluated, as for example when being assigned a national evaluation class[40], it is mostly this component that is being measured.

The internals of computer systems usually differ from other instances of multilevel security in that the mechanisms and procedures that are used to control and separate data at different security levels are equally effective for all data. In contrast to manual security procedures, which might specify escalating security measures as data worth increases, computer systems generally use the same methods for isolating all levels.

The possibility of leakage between levels in a computer, therefore, does not depend on the separation between them. If a system is highly rated to prevent *Secret* information from being contaminated with *Top Secret* information, the same level of assurance should be assumed between, say, *Unclassified* and *Confidential* data. It is not being suggested that the risks are the same. The possible damage usually will vary widely between one case to the next.

For this reason, the risk component due to faulty separation can be categorised by the pair $\left[\overset{sep}{R}, \overset{sep}{X}\right]$. Define $\overset{sep}{X}$ as $(\forall k)\ \overset{sep}{X} \rhd W_k$. $\overset{sep}{X}$ is the highest level of contaminant due to separation with which any data object can be contaminated. Usually $\overset{sep}{X}$ will be the system's highest classified data object. It will be assumed that the level $\overset{sep}{X}$ is unique. In a practical multilevel system, it could easily be the case that there are several high levels of contaminant, $\overset{sep}{X}_1, \overset{sep}{X}_2, \cdots$, that exist. Some of them may not dominate some of the others, because the dominates relation is a partial order. It is possible to find a least upper bound for all these contaminants, but that might be far too high a security level to be representative of reality. Thus, while we will assume that $\overset{sep}{X}$ is unique, it is possible to perform a more detailed analysis that in the extreme uses a pair of the form $\left[\overset{sep}{R}_k, \overset{sep}{X}_k\right]$ for the risk component to data object k due to faulty separation. This approach might give an enhanced model, but to be

40 The classes of the TCSEC, the CTCPEC, or such governmental criteria are being referenced. Recently some effort has been devoted to harmonising these criteria.

brief we chose the simpler approach. We also believe that the simpler approach is adequate and that it is closer to what happens in practice.

The reliability $\overset{sep}{R}$, which is drawn from the same range as the $\overset{in}{R}$ of the input data, measures the reliability of the system's ability to prevent the contamination of labelled data with data labelled with a higher security level. The confidence that $\overset{sep}{R}$ expresses is the level of confidence that the labelled data contains only information with a security level that is dominated by its own security level. The actual security level is not involved here because of the argument that the possibility of leakage between levels does not depend on the separation between them.

Evidently systems can be compared using an obvious analog to $\rightarrow$. If P and Q are two systems, then from the separation viewpoint $\langle VR, S\rangle_P \rightarrow \langle VR, TS\rangle_Q$ or $\langle ER, TS\rangle_P \rightarrow \langle VR, TS\rangle_Q$, and in cases that are not as clear, the evaluating authority may judge that the separation is adequate, as it did in the discussion of the relation $\Rightarrow$. We emphasise, however, that this comparison ignores the environment that these systems occupy; it is only a measure of the strength of their respective separation mechanisms.

6.6. Risk in Output Labels

Even a system with a perfect $\overset{sep}{R}$ can not produce reliably labelled output if the data was incorrectly classified on input. If $\overset{sep}{R}$ is less than perfect contamination in output labels can be caused both by contamination in the inputs and by contamination from internal system data with a level that dominates $\overset{out}{W}$. The objective is to designate the reliability of the risk labels taking into account any unreliability of the input data labels and imperfections in the systems.

6.7. Output Label Risk Factor

An output label risk factor $\overset{out}{\Phi} = \left\langle \overset{out}{W}, \overset{out}{R}, \overset{out}{X} \right\rangle$ can be defined, such that it applies to for the case in which input data may be imperfectly labelled and the separation controls may be imperfect.

We make the reasonable assumption that the system attempts to preserve security. Therefore, it is expected that the system will label each output with the worth $\overset{out}{W} = \mathbf{LUB}(\forall k\, W_k)$ where k ranges over all pertinent data objects. This was mentioned in section 1.2. An output label risk factor $\overset{out}{\Phi}$ can be defined for possibly imperfectly labelled input data and possibly imperfect separation mechanisms.

Possible contaminants must dominate $\overset{out}{W}$ for them to represent a risk. These contaminants can not be the worth of inputs. Because by their nature inputs are pertinent to the calculation, $\overset{out}{W}$ will be at least as high as each their worth. There are two possible sources of contamination.

The first source of possible contamination is the contamination from data objects that may or may not be directly involved in the determination of the output of the combination, but that might affect it due to imperfect separation. Let k range over all system data, excluding inputs, for which $W_k \rhd \overset{out}{W}$ or $X_k \rhd \overset{out}{W}$. These data objects may not affect the output directly. They can, however, contaminate the output due to imperfect separation. This contamination may be mitigated if $\overset{sep}{R}$ is sufficiently high, and vanishes if $\overset{sep}{R}$ is perfect.

The second source of possible contamination is from inputs. We know that for all inputs i_k, $\overset{out}{W} \rhd W_k$, but for any input k it might be the case that $X_k \rhd \overset{out}{W}$ and each such contaminant can contribute to the risk.

Let k range over all data objects in the system, including inputs. Define $\overset{out}{\Phi}$ to be the risk contribution in an output due to contamination from data object k with risk function $\Re_k = \langle W_k, R_k, X_k \rangle$. Then:

$$\overset{out}{\Phi} = \underset{\forall k}{\mathbf{MIN}} \left\langle \overset{out}{W}, \min\left(R_k, \overset{sep}{R} \right), \mathbf{LUB}\left(X_k, \overset{out}{W} \right) \right\rangle,$$

The $\min\left(R_k, \overset{sep}{R} \right)$ function appears because we wish to be pessimistic; a low $\overset{sep}{R}$ or a low R_k will increase the risk because of the likelihood of contamination, so we chose the lowest of the pair of them. Reinforcing our pessimism, the expression $\mathbf{LUB}\left(X_k, \overset{out}{W} \right)$ gives the highest level that dominates $\overset{out}{W}$ that could be a contaminant. Thus, the $\underset{\forall k}{\mathbf{MIN}}$ function will give a risk factor of $\overset{out}{\Phi} = \left\langle \overset{out}{W}, \text{lowest } R, \text{highest } X \right\rangle$, as our pessimistic nature wishes.

A complete analysis would consider the issues already discussed, as well as such issues as the possibility of coupling between the separation risk with perfect inputs and the separation risk from contamination arising from unrelated data. Our analysis, however, is pessimistic in that it chooses the worst case possibility.

6.8. Limiting Cases

The value of $\overset{out}{\Phi}$ in several special cases is interesting, as well as serving to reassure us that we are on the correct track.

6.8.1. *Completely Reliable Separation*

When $\overset{sep}{R}$ is perfect, the result $\overset{out}{\Phi} = \underset{k}{\mathbf{MIN}}\left\langle \overset{out}{W}, \min\left(R_k, \overset{sep}{R} \right), \mathbf{LUB}\left(X_k, \overset{out}{W} \right) \right\rangle$ has several interesting aspects. Clearly, $\min\left(R_k, \overset{sep}{R} \right) = R_k$, and since $\overset{sep}{R}$ is perfect there can be no contamination from data in the system so *k* ranges only over the input data and $\overset{out}{\Phi}_{CRS} = \underset{\forall inputs\ k}{\mathrm{MIN}} \left\langle \overset{out}{W}, R_k, \mathrm{LUB}\left(X_k, \overset{out}{W} \right) \right\rangle$.

6.8.2. *Faultless Input Data*

For faultless input data, $\overset{out}{\Phi} = \underset{k}{\mathbf{MIN}}\left\langle \overset{out}{W}, \min\left(R_k, \overset{sep}{R} \right), \mathbf{LUB}\left(X_k, \overset{out}{W} \right) \right\rangle$ becomes $\overset{out}{\Phi}_{FID} = \underset{\forall \substack{system \\ data\ k}}{\mathrm{MIN}} \left\langle \overset{out}{W}, \overset{sep}{R}, \mathrm{LUB}\left(X_k, \overset{out}{W} \right) \right\rangle$. In this case, *k* ranges over only the system data; the input data is perfect so it cannot result in an output risk.

However, no pessimism is lost if we allow *k* to range over both the system data and the input data. If we do this, it is easy to see that for each input datum i_m it must be the case that $R_m \geq \overset{sep}{R}$. If this were not so, and $R_m < \overset{sep}{R}$ for a particular datum i_m then it would be the case that the output risk factor in this situation $\overset{out}{\Phi} \rightarrow \overset{out}{\Phi}_{FID}$. But we constructed $\overset{out}{\Phi}$ to have the lowest reliability and highest possible contaminant in the general case.

Consequently, we observe that $R_m \geq \overset{sep}{R}$ for all inputs *m*. This is the first example of a control on the external environment, from which the inputs come, that must be true if risk is to be controlled.

6.9. The Environment and Maximal Risk

The evaluating authority must consider a proposed environment and the amount of confidence that can be placed in the output labelling and output routing capabilities of the system. It will make the evaluation decision depending on the reliability of the labels and on the exposure that would result if a worst-case mislabelling occurred. The capability of the environment to deal effectively with classified data can be described by a maximum tolerable risk $\overset{env}{\Re}$ that is a property of the environment.

Different environments will have different $\overset{env}{\Re}$, agreeing with our notion that a system might be approved for one environment but forbidden in another. This decision can be effectively modelled as three constraints.

6.10. The Environment and Maximal Levels

The first constraint is that there is some limit $\overset{env}{\Re} = \left\langle \overset{env}{W}, \overset{env}{R}, \overset{env}{X} \right\rangle$ imposed by the environment on the risk that is permissible. All data risks must be at least as good a risk as $\overset{env}{\Re}$. In most cases there will be a maximum worth that is tolerable in the environment. We take this limit to mean that $\overset{env}{X} = \overset{env}{W}$ since both the worth of the label and the worth of the highest permitted contamination must be dominated by this maximum allowed worth.

It would be overly permissive to forbid worth above $\overset{env}{W}$ but to allow contamination above $\overset{env}{W}$, since $\overset{env}{W}$ is the highest security level for which the proposed environment provides adequate handling procedures. Although $\overset{env}{W}$ will always be an upper bound on the permitted worth, it is not necessarily a least upper bound.

6.11. The Environment and Minimal Reliability

The second constraint concerns the reliability $\overset{env}{R}$ in the maximum risk $\overset{env}{\Re} = \left\langle \overset{env}{W}, \overset{env}{R}, \overset{env}{X} \right\rangle$. We know that $(\forall k)\, \overset{env}{W} \rhd W_k$ and $(\forall k)\, \overset{env}{W} \rhd X_k$. For all data objects o_k we require that $R_k \Rightarrow \overset{env}{R}$.

We define $\overset{sys}{\Phi} = \left\langle \overset{env}{W}, \overset{sys}{R}, \overset{sys}{X} \right\rangle$ with reliability and contamination components that reflect a pessimistic view of all the actual data worth in the system. Thus we have that $(\forall k)\, R_k \geq \overset{sys}{R}$ and $(\forall k)\, \overset{sys}{X} \rhd X_k$. We insist that $\Re_k \Rightarrow \overset{env}{\Re}$ and implies that $\overset{sys}{\Phi} \Rightarrow \overset{env}{\Re}$. Dealing with the relation $\Rightarrow$ is a problem, but in this case we can safely substitute $\overset{sys}{\Phi} \to \overset{env}{\Re}$ because the authority is reasonable. The worth of both $\overset{sys}{\Phi}$ and $\overset{env}{\Re}$ is $\overset{env}{W}$. We know by construction that $(\forall k)\, \overset{sys}{X} \rhd X_k$. If $\overset{sys}{\Phi} \to \overset{env}{\Re}$ it must be the case that $\overset{sys}{R} \geq \overset{env}{R}$. Because $(\forall k)\, R_k \geq \overset{sys}{R}$, we can say that $(\forall k)\, R_k \geq \overset{env}{R}$. This gives an effective lower limit on each R_k; the reliability of the label on any system data must be good enough to ensure that the risk associated with it is at least as good as $\overset{env}{\Re}$.

The new condition that $(\forall k)\, R_k \geq \overset{env}{R}$ is merely an insistence that every label have some minimal level of reliability to satisfy the minimal risk that is tolerable. This does not seem excessively rigid.

If $\overset{env}{\Re} = \langle S, ER, S \rangle$ then a system that includes a data object with a reliability function $\Re = \langle S, SR, S \rangle$ would not be evaluated acceptable because it is not the case that $\langle S,SR,S\rangle \rightarrow \langle S,ER,S\rangle$, and since systems are expected to be reasonable, neither can it be the case that $\langle S,SR,S\rangle \Rightarrow \langle S,ER,S\rangle$.

6.12. Maintaining Minimal Risk

The third constraint is that the system must provide output labels and separation with a risk not exceeding $\overset{env}{\Re} = \left\langle \overset{env}{W}, \overset{env}{R}, \overset{env}{X} \right\rangle$. All outputs of the system must meet or better this permissible risk. The risk $\overset{env}{\Re}$ is a limitation imposed by the security policy on the production of new data values with evolving labels. The system will be evaluated as acceptable only if $\overset{out}{\Phi} \Rightarrow \overset{env}{\Re}$. In general, the value of $\overset{env}{\Re}$ is dependent on the particular system of interest and its proposed environment.

A different maximal output label risk factor for each security level might be a more precise representation. The risk factor $\overset{env}{\Re}$ can pessimistically be thought of as the least upper bound of all these different functions.

It is certain that the risk factor for faultless input data must be such that $\overset{out}{\Phi}_{FID} = \underset{\forall \substack{system \\ data\ k}}{\mathrm{MIN}} \left\langle \overset{out}{W}, \overset{sep}{R}, \mathrm{LUB}\left(X_k, \overset{out}{W} \right) \right\rangle \Rightarrow \overset{env}{\Re}$. We have $\overset{env}{\Re} = \left\langle \overset{env}{W}, \overset{env}{R}, \overset{env}{X} \right\rangle$ and $(\forall k)\, \overset{env}{W} \rhd W_k$ and $(\forall k)\, \overset{env}{W} \rhd X_k$. Thus, $\underset{k}{\mathbf{MIN}} \left\langle \overset{out}{W}, \overset{sep}{R}, LUB\left(X_k, \overset{out}{W} \right) \right\rangle \Rightarrow \left\langle \overset{env}{W}, \overset{env}{R}, \overset{env}{X} \right\rangle$. The relation $\Rightarrow$ is difficult to deal with, but because the authority is reasonable we can substitute the relation $\rightarrow$ without losing our pessimistic approach. The result is that $\underset{k}{\mathbf{MIN}} \left\langle \overset{out}{W}, \overset{sep}{R}, LUB\left(X_k, \overset{out}{W} \right) \right\rangle \rightarrow \left\langle \overset{env}{W}, \overset{env}{R}, \overset{env}{X} \right\rangle$. This states that the separation capability of the system if all input data is labelled completely reliably must be good enough to meet $\overset{env}{\Re}$, so that $\overset{sep}{R} \geq \overset{env}{R}$, surely a reasonable condition. In the general case, we must insist on this condition; in practice we have no way of knowing how reliably the input data is labelled, so we must choose the pessimistic alternative.

If $\overset{env}{\Re} = \langle C,VR,S\rangle$, then a system that includes an output with a reliability factor $\overset{out}{\Phi} = \langle C,SR,S\rangle$ would not be evaluated acceptable because it is not the case that $\overset{out}{\Phi} \rightarrow \overset{env}{\Re}$, and since evaluating authorities are expected to be reasonable, neither can it be the case that $\overset{out}{\Phi} \Rightarrow \overset{env}{\Re}$.

Unfortunately, the requirement $\overset{out}{\Phi} \Rightarrow \overset{env}{\Re}$ can not be used directly as the basis for an assurance policy. One important reason is that $\overset{env}{\Re}$ is not known. All that is known is that $\overset{out}{\Phi} \Rightarrow \overset{out}{\Phi}_{FID} \Rightarrow \overset{env}{\Re}$.

No optimism is introduced by insisting that $\overset{out}{\Phi} \Rightarrow \overset{out}{\Phi}_{FID}$. The implication is that if the inputs are imperfectly labelled the risk should be at least as good as when they are perfectly labelled. When they are perfectly labelled the system must have adequate separation, a condition that we do not find difficult to demand. This does have the possible disadvantage of making the requirement stricter than is really necessary for a standalone configuration. Without further information, there is no way of detecting this potential over-qualification of a single system.

This $\overset{out}{\Phi} \Rightarrow \overset{out}{\Phi}_{FID}$ approximation is still unusable because the precise definition of the $\Rightarrow$ relation is not known. Instead, the $\to$ approximation must be used.

Substituting for $\overset{out}{\Phi}$ and $\overset{out}{\Phi}_{FID}$ in this expression we get

$$\underset{\forall k}{\mathbf{MIN}}\left\langle \overset{out}{W}, \min\left(R_k, \overset{sep}{R}\right), \mathbf{LUB}\left(X_k, \overset{out}{W}\right)\right\rangle \to \underset{\forall \substack{system\\ data\ k}}{\mathbf{MIN}}\left\langle \overset{out}{W}, \overset{sep}{R}, \mathbf{LUB}\left(X_k, \overset{out}{W}\right)\right\rangle.$$

The only problem in dealing with this relation is that the range of k is not the same on both sides of it. However, this difficulty disappears if we extend the range on the right to include the inputs. Effectively, we are requiring that the maximum contaminant on the left, including inputs, is dominated by the maximum contaminant on the right, which excludes inputs. This can be done with safety if we require that $\forall m\ \overset{sep}{X} \rhd \overset{in}{X}_m$. This extends the range of k on the right to be the same as it is on the left. With this change we get:

$$\underset{\forall k}{\mathbf{MIN}}\left\langle \overset{out}{W}, \min\left(R_k, \overset{sep}{R}\right), \mathbf{LUB}\left(X_k, \overset{out}{W}\right)\right\rangle \to \underset{\forall k}{\mathbf{MIN}}\left\langle \overset{out}{W}, \overset{sep}{R}, \mathbf{LUB}\left(X_k, \overset{out}{W}\right)\right\rangle$$

Clearly the worth and contaminant parts of this relation are satisfied because they are identical on both sides of the relation. The relation will hold if $\min\left(R_k, \overset{sep}{R}\right) \geq \overset{sep}{R}$, which effectively requires that for all system data and inputs, $R_k \geq \overset{sep}{R}$. This duplicates the result of section 3.2.1.

Summarising it is seen that for risk to be controlled there are two conditions for all input data i_m:

$$R_k \geq \overset{sep}{R} \qquad \text{(A)}$$

$$\text{and} \quad \overset{in}{X}_m \lhd \overset{sep}{X}. \qquad \text{(B)}$$

Relation (A) asserts that the reliability of the input data labels must not be less than the separation reliability. Relation (B) asserts that the system-high worth level is an upper limit on the worth of any contaminant in the input. We deem both of these conditions reasonable.

Thus the result is that for all data inputs i_m the label of the outcome $\overset{out}{W} \rhd \overset{in}{W}_m$ and that for the input to be acceptable $\overset{in}{R}_m \geq \overset{sep}{R}$ and $\overset{sep}{X} \rhd \overset{in}{X}_m$, and for all system data i_k the reliability of the labels $R_k \geq \overset{sep}{R}$ and for the separation controls to be effective it must be the case that $\overset{sep}{X} \rhd X_k$.

6.13. Discussion of the Results

We have established the following requirements on the data in the system and on the input data to the system:

(1) For all system data objects and inputs o_k, $\overset{env}{W} \rhd W_k$, $\overset{env}{W} \rhd X_k$, $R_k \geq \overset{env}{R}$ and $\overset{sep}{X} \rhd X_k$.

(2) For all data inputs o_m to the system, $\overset{in}{R}_m \geq \overset{sep}{R}$ and $\overset{sep}{X} \rhd \overset{in}{X}_m$.

Requirement (1) establishes that the worth of each of the system's data objects must conform to the limits expressed by $\overset{env}{R}$ and the ability of the system to enforce policy.

$\overset{sep}{R}$ is closely related to the evaluation of the capability of the system with faultlessly classified inputs of separating the system outputs from contamination by system data objects. It is determined by a thorough review of the system and by its operation in a particular type of environment according to established practices. The first condition of requirement (2) asserts that the reliability of the input labels must be at least as good as the reliability $\overset{sep}{R}$. This is not a surprising condition; it is to be expected that with a given $\overset{sep}{R}$ the system is capable of adequately separating data with a reliability not less than some threshold.

We defined $\overset{sep}{X}$ as $(\forall k)\ \overset{sep}{X} \rhd W_k$, so the second condition of requirement (2) that $\overset{sep}{X} \rhd \overset{in}{X}_m$ merely asserts that the worth of the maximum contaminant of a data input must not exceed the system's highest classified data object. It can not introduce a new, more highly classified object into the system as a contaminant.

The system can not ordinarily enforce the risk factor of the input data. This must be assured by external handling procedures that must obtain if the system is to be adjudged sufficiently robust for the intended environment. The intent of these requirements is that a system shall neither include data objects that imply a risk that is greater than that tolerable by the environment, nor produce outcomes that imply more risk than the risk already inherent in whatever inputs the system receives.

6.14. Inter-System Communications

Consider two systems, A and B that may be exchanging data. We perceive them to be in the same environment because they are exchanging data. They both must satisfy the environment's conditions so that $R_k \geq \overset{env}{R}$, $\overset{env}{W} \triangleright W_k$ and $\overset{env}{W} \triangleright X_k$. These systems may or may not both be active at the same time. Computers that are connected to a network are simultaneously active, but for instance a backup or previously recorded magnetic tapes can be one of the systems.

The restriction that input contaminant levels be dominated by the system high level can be rephrased as requiring that the system could be evaluated to handle input data as high as the most highly classified object that it contains. A system A that supplies an input signal to a system B will only be able to satisfy this requirement if its separation $\overset{sep}{R_A}$ is *Completely Reliable* (which is unlikely) or if $R_{signal} \geq \overset{sep}{R_A}$ is at least as high as $\overset{sep}{R_B}$ and if the input signal's maximum levels are such that $\overset{sep}{X_A} \triangleright X_{signal}$ and $\overset{sep}{X_B} \triangleright X_{signal}$ if $\overset{sep}{X_B} \triangleright \overset{sep}{X_A}$. Thus, for communication from A to B to be allowable it must be the case that B's highest security level dominates A's. If the communication is to be bi-directional, B and A must have the same high level.

The other requirement, when applied to our hypothetical case of A sending data to B requires that $\overset{sep}{R_A} \geq \overset{sep}{R_B}$. Bi-directional traffic would require that the reliability of each of the systems be equal. These are difficult requirements to meet, since there is no direct way of measuring the $\overset{sep}{R}$ values except by an arbitrary or authoritarian determination by an evaluation process similar to that used with national evaluation classes to establish that the system can be evaluated to some specific grade. Consequently, these values must be determined as the result of the evaluation process, and maintained while the system is in service.

A system P_1 with outputs ranging from TS to U will undoubtedly be required to maintain label integrity and separation with a higher assurance level than a system P_2 for which the outputs range only from TS to S. However, it may be the case in a specific environment that this very high reliability is only required for the U outputs, and is excessive in the case of the S level outputs. It is an accident of implementation that, as noted earlier, most computer systems will provide the same separation between TS and S as between TS and U data.

If P_1 and P_2 are allowed to intercommunicate, even using floppy disks or other such mechanisms, because of the higher requirements on P_1 the composite system will separate TS and U data more reliably than it does TS and S. However, the standalone output risk factors of P_1 as an approximation of the required reliability were accepted. P_2 may not satisfy the same risk factor criteria. The result of the analysis would not permit this intercommunication unless the reasonable authority judging the relation $\Rightarrow$ was to be exceptionally lenient, or the environment were to be unusual.

Thus, when two otherwise independent systems are connected, if the two systems are very similar, with the same system-high worth, the same reasonable authority, the same separation properties and they are operating in the same environment with the same label reliability, they may exchange data without serious security worry. Otherwise, a reasonable authority must adjudicate all attempts to communicate. In many cases, this degree of involvement from a reasonable authority is probably unacceptable, both because of efficiency considerations and because it would be difficult to develop high assurance in such a busy and broadly responsible authority. It would also be difficult to design such a mechanism because parts of it must be present in both systems. The problem is not unknown in the literature[41].

These results agree with commonly accepted intuition:

(1) A system that can deal with data only up to some limiting worth can not be allowed to deal with any higher level data, and the reliability of the worth given by the label must be up to a standard set by the environment.

(2) Input data can be allowed into a system only if it does not introduce worth that are more unreliable than the system is graded to handle and if it does not introduce higher level contaminants into the system.

The fact that our results agree with our intuition is comfortable. Of course, in many situations throughout the analysis we mentioned potential refinements that may give results that disagree with our inbred intuition in specific circumstances. We have, however, taken the pessimistic side of all modelling decisions, so we do not expect that such disagreements will be common.

6.15. Summary

We have shown that risk can be effectively considered to be an expression of the form $\text{risk} = \Re(\textit{worth},\ \textit{vulnerability},\ \textit{threat})$. The interpretation of each of these parameters is determined by the nature of the system or process that is being analysed. If this is done, and if reasonable but pessimistic choices are adopted, then the result of the analysis can be a useful adjunct to other formal mechanisms that are deployed to improve the confidence that the system controls are likely to be effective. The extended example that is included is not in itself very important, but it

41 See, for example, the appendices of the Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, US National Computer Security Center, document NCSC-TG-005.

does indicate that this sort of risk analysis can be done even in abstract systems with possibilistic properties.

6.16. Acknowledgement

This work is an adaptation of part of the final report of contract W2207-6-AF32/01-ST to the University of Toronto from the Communications Security Establishment of the Government of Canada. The authors of the original report were B. Thomson, E.S. Lee, P.I.P. Boulton, M. Stumm and D.M. Lewis. The principal investigator was E.S. Lee. Much of the work in this essay has benefited enormously from the original work done by Brian Thomson during the contract.

TRUSTWORTHY LAN NETWORKS

7. TRUSTED LAN NETWORKS

This essay describes an architecture for a trusted network intended for a computing environment in which frequent communication occurs amongst computers linked by an open local area network. The network is assumed to be open in the sense that it may be subject to attempts at wiretapping, eavesdropping, and sophisticated interference from unauthorised intruders. This does not imply that the trusted network should be operated in an open environment. We encourage, however, that any network that is to be trusted be operated in a physically secure environment. However, since a trusted network should be designed to withstand possible breaches in security, worst case assumptions should serve as a basis in the design. For example, the trusted network described here is designed to withstand traffic analysis and other attacks in spite of wiretapping and other forms of eavesdropping. Similarly, it ensures that conversations are possible only between subjects connected by legitimate connections and that these conversations are tamper-proof.

The architecture is based on secure data channels, over which only authorised subjects can send, inspect or modify the data stream. This is achieved through the use of a number of mechanisms including encryption, checksums, and nonces[42]. Communication that bypasses these secure data channels is not possible.

Additional security features of the secure data channels include:

- all transmitted packets are of equal length,
- each network node transmits packets at a constant average rate,

[42] A nonce is a one-time identifier that is used to detect playbacks and to make otherwise identical packets appear different when encrypted. In practice it is typically implemented as a 32-bit or longer random number.

- a large proportion of transmitted packets are artificially generated and contain no real data, but are indistinguishable from packets containing real data,
- the transport-level protocols are connection oriented and different keys are used for encryption at the transport level for each connection, and
- each transmitted packet is also encrypted at the link level in its entirety, that is, including the link-level header containing the address fields.

The trusted network is controlled by a single, centralised, highly trusted node, called the Trusted Network Controller (TNC). The TNC arranges for the establishment of all connections and is responsible for authentication and for encryption key distribution. Control of the trusted network is fully centralised, although data channels connect two endpoints directly for high performance.

Much of the network security in the trusted network is based on encryption. We assume the existence of encryption mechanisms that are sufficiently powerful and secure. We do not add anything to the body of knowledge concerning encryption. Nevertheless, the trusted network architecture has a number of features that make it more difficult to compromise keys and which limit the extent of possible damage if there is key compromise. For example, encryption keys are visible only to the TNC and to network entities it trusts, and keys are not passed to higher communication levels or passed to the operating systems of the hosts connected to the trusted network. Additionally, keys are used at multiple levels and must be changed frequently.

The Trusted Network Architecture is described in several steps. Section 8.2 describes the major components of the trusted network and presents an architectural overview and diagram. Section 8.3 informally introduces the specification of the trusted network architecture and presents the reasoning behind many of the architectural choices. It also suggests possible implementations that would meet the specification where it may not be obvious. Finally, in section 8.4 we perform a risk assessment of the trusted network architecture. We analyse the effects of common security attacks and consider the strength of the proposed architecture by analysing the ramifications when individual encryption keys are compromised.

In reading this document, it is important to keep in mind that a good architectural specification is a balance between two forces. On the one hand, it should be sufficiently complete to ensure that all implementations conforming to the specification meet the goals of the architecture. On the other hand, the specification should be minimal, so as not to constrain the use of existing solutions or prevent the introduction of new state-of-the-art mechanisms when they become available.

7.1. Architectural Overview

The Trusted Network Architecture is defined by specifying the hardware base, the various protocols that define the interaction between hardware components, and how encryption keys should be used for communication. We review these hardware components, protocols and encryption in the following subsections.

7.1.1 Hardware

Four hardware components form the basis of the trusted network:

- the physical network,
- the host computers,
- Trusted Network Interface Units (TNIUs), which connect a host to the network, and
- a Trusted Network Controller (TNC).

The sole purpose of the network is to efficiently communicate data between two TNIUs or between the TNC and a TNIU. The only way a host is allowed to connect to the network is through a TNIU. A TNIU contains the network access controller necessary for transmitting and receiving network packets, implements the low-level communication protocols, is responsible for encrypting and decrypting all data, and co-operates with the TNC for several purposes, including the establishment of transport-level connections and key distribution.

The TNC is a highly trusted host in a secure area and centrally manages the network. The primary function of the TNC is that of arbitrating transport-level virtual connections between principals in conformance with a network-wide security policy, that is, between processes running on hosts connected to the network by the TNIUs. This responsibility requires, in turn, that the TNC perform authentication as well as the management and distribution of encryption keys.

The TNC is the only entity that implements a network-wide security policy and therefore has total control over the trusted network. Hence, from a control point of view, the network is completely centralised. However, once a connection is established by the TNC, end-to-end communication occurs directly between subjects without going through the TNC, to allow for a high aggregate communication throughput.

7.1.2 Protocols

In addition to the hardware components, communication protocols must be specified for:

- Host – TNIU communication.
- Link-level communication between two TNIUs or between a TNIU and the TNC.
- Transport-level communication between processes executing on hosts.
- TNIU – TNC transport-level and control communication.

Communication between a host and its TNIU is kept to a minimum and is used only to establish or release transport-level virtual connections and for passing data that is communicated over transport-level connections that terminate at that particular host. The interface between a host and its TNIU will be similar to any standard interlevel interface, as found, for example in one of the ISO OSI architecture definition specifications.

The link-level communication protocol can be similar to standard datagram oriented link-level protocols. As is usually the case, the link level is primarily responsible for link-level addressing and detection of transmission errors (both those caused naturally as well as those caused by intruders). However, a few simple changes or restrictions to standard link-level protocols are necessary before they are suitable for our trusted network. For example, link-level control information must be kept to a minimum to reveal as little information as possible in case link-level keys are compromised. All link-level data is transmitted in encrypted form only. Also, since transport-level data is completely encrypted using keys specific to each virtual connection, it becomes necessary to include a connection identifier field in the link-level header to be able to identify the connection and hence the corresponding appropriate transport-level decryption key.

The transport-level communication protocol must be connection oriented and capable of detecting and suppressing duplicate packets with very long lifetimes. Otherwise, no restrictions are imposed on this protocol level. The transport-level protocol can be executed in either on the host or in the TNIU. The architecture described here does not specify a required execution location. However, we assume it is executed on the host, as depicted in figure 7.3.

Executing the transport-level protocol on the host has the advantage that two hosts can communicate using arbitrary transport protocols, including specialised ones (as long as they adhere to the specification). Also, it makes security verification of the TNIU simpler, since it does not include the verification of the transport-level protocols needed for general interprocess communication. Finally, it allows for a physical separation of plaintext and ciphertext, since encryption is performed in the TNIU.

If the transport level is implemented on the host, then its connections must make use of virtual connections between TNIUs, as established and maintained by the TNIUs themselves. These TNIU – TNIU virtual connection can be viewed as network-level connections, since they connect two end hosts. Each transport-level connection must correspond to a single network-level connection. However, strictly speaking, a network level is not needed in this architecture, since it is restricted to local area networks where no routing takes place.

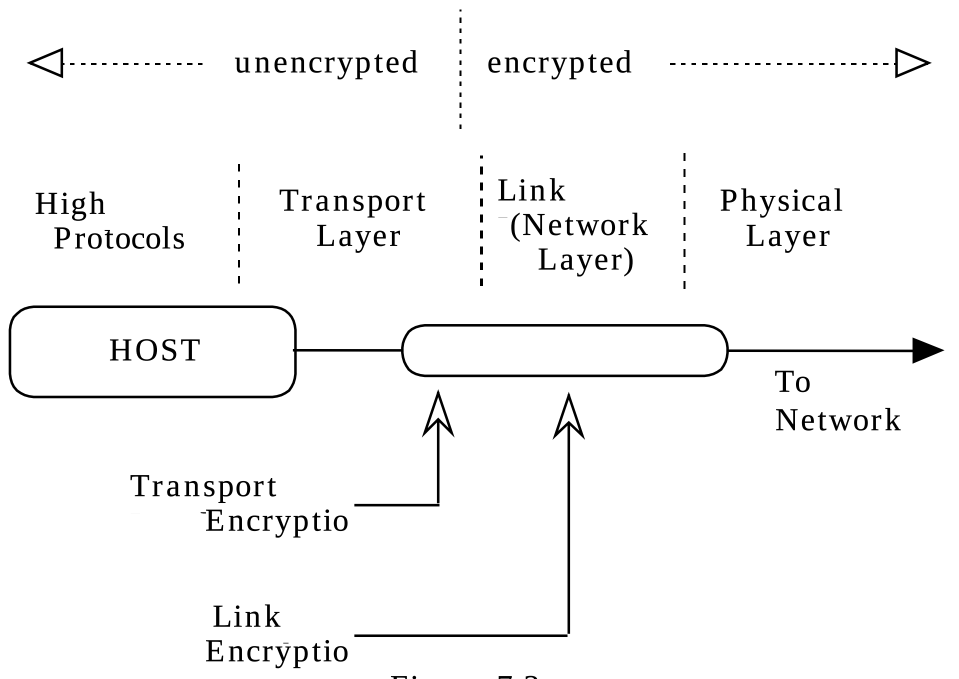


Figure 7.3.
Crypto locations and partitioning of protocol

Therefore, to be precise, one would have to view the implementation of the transport-level protocol as being partitioned between host and TNIU, where the lowest levels of the transport protocol, in particular the transport-level encryption and the multiplexing, are performed in the TNIU. Nevertheless, viewing the TNIU – TNIU connection as a network-level connection has its advantages. It may not be trivial to partition existing transport protocol implementations across two processors (that of the host and the TNIU), but it appears simple to modify existing transport protocol implementations to use a separate network-level connection for each transport-level connection. If the interprocess transport level is implemented on the host side, then the TNIUs will require a separate transport-level protocol implementation to communicate with the TNC, although this can be a simpler, specialised protocol.

Finally, the TNIU and TNC also need higher level protocols for control, network management, key distribution, as well as other functions described in later sections.

7.1.3 Encryption

Encryption is essential to the operation of the trusted network. It is used for authentication purposes as well as for secure communication.

An important consideration is the level in the communication architecture where encryption is to occur. The advantage of link-level encryption over end-to-end, transport-level encryption or vice versa has been discussed extensively in the literature. In the end-to-end approach, the message contents only need to be decrypted once at the destination host. This approach also allows a higher number of identifiable and separately protected entities. But encrypting at the transport level leaves the link (and network) level headers in the clear. On the other hand, link-level encryption allows for the transmission of packets that are completely encrypted, including header information.

In this architecture, encryption occurs at both the link level and the transport level, benefiting from the advantages of both crypto locations.

At the link level, each host has its own link-level key assigned by the TNC that is used to decrypt all received packets. Sending TNIUs obtain the corresponding encryption keys from the TNC, one for each destination TNIU. The encryption algorithms used at the link level do not need to be "heavyweight": it is acceptable to sacrifice some cryptographic strength for performance. The implementation used must allow the timely decryption of each packet header as it arrives from the network, so that the TNIU is not overrun. However, the use of a relatively weak encryption algorithm does not significantly lessen security, since the higher-level portions of the packet are additionally encrypted at the transport level. Encryption at the link level is used mainly to hide traffic patterns and it is conjectured that the utility of any information that can be deduced from traffic pattern analysis will decay rapidly with time.

Separate key sets are used for each transport-level connection and for each direction. The encryption at the transport level needs to be more secure than the link-level encryption. The TNIU is responsible for encrypting transport-level packets in their entirety before network transmission and for their decryption at the receiving end.

Both the link-level and the transport-level encryption algorithms must be stateless. The link-level packets are not reliably delivered, and retransmission may be necessary. The packet may be re-encrypted on each retransmission, and can include a nonce to make the retransmission different. Overall, the transport level is by definition reliable. This reliability is achieved by retransmission originating in the host. The transport-level encryption takes place below this in the TNIU. Because of the possibility of retransmission a stateless encryption algorithm must be used.

7.2. Specification and Discussion

In this section we introduce the specification of the trusted network architecture and present the reasoning behind many of the architectural choices. Where it is not obvious, we also suggest possible implementations that meet the specification.

7.2.1. Hardware Components of the Network

The trusted network contains four major hardware components:

- the local area network itself,
- the hosts connected to the network, with their operating systems and application processes,

- a trusted network controller, and
- trusted network interface units connected to each host.

7.2.1.1 The Network

The network must be a broadcast network, where each transmitted packet is sent to every connected host. This requirement complicates traffic analysis attacks in that it ensures that the level of traffic is equal on each link. Moreover, in a broadcast network routing is unnecessary, which allows us to encrypt all link-level header information, including addresses. Although the network must be a broadcast network, we do not support broadcast or multicast messages, where a single packet is addressed to multiple hosts.

The network must have an aggregate throughput capacity large enough to sustain traffic in a stable manner when all network nodes simultaneously generate traffic at their peak allowable rate. This allows a policy whereby each network node continuously transmits packets at a constant average rate. Whenever a node does not have real data packets to send, it generates artificial, random packets and sends them instead. Since most transmitted packets will be artificial, the packet signal-to-noise ratio on the network will be high. Naturally, artificial packets must be generated such that they appear to be valid encrypted packets.

The network must be structured such that only limited disruption occurs if a path passing through an insecure area is destroyed. Additionally, path failures must be detectable. This can generally be achieved by having the TNC periodically poll each network node. It may be important for each host to know if it is still connected to the rest of the distributed system. If the network is of the type where a locally transmitted packet is also received locally after it has traversed either the entire network (as would be the case in a ring network) or traversed a central point of the network, such as the root of a rooted tree network or the central node in a star network, then each node verifies that it is still connected to the rest of the network on each packet it transmits.

Additionally, it should not be possible for a single network access point (possibly created by an intruder in an insecure area) to disrupt all network traffic. Nevertheless, it is still useful to be able to isolate individual segments of the network in a secure way to isolate, for example, misbehaving or malfunctioning network nodes.

7.2.1.2. The Host and its Operating System

The operating system of the host may contain the communication protocol levels corresponding to the transport level and above. The transport protocol implementation must adhere to the specifications of section 3.3.3.

The host operating system is responsible for supplying to the TNIU the parameters required by the TNC that are needed to decide the permissibility of a proposed connection. The TNC is aware of all parameters concerning security that are relevant to each host.

7.2.1.3. The Trusted Network Controller

The Trusted Network Controller (TNC) is a highly trusted host in a secure area that manages the network. The primary function of the TNC is that of arbitrating transport-level virtual connections between principals, that is, between processes running on hosts connected by the network. It implements the network-wide security policy. The TNC works closely with Trusted Network Interface Units (TNIU) connected to each host.

In order for two processes to communicate, they must first through the TNIU request the TNC to set up a trusted connection for them. The TNC then determines the appropriateness of communication between these two processes, using information provided by the hosts' operating systems, the TNIUs connected to the two hosts, and information held locally. If a connection is permissible, the TNC generates keys for encrypting and decrypting transport-level data and passes them to the two corresponding TNIUs, implicitly granting permission for the two processes to communicate. This protocol is described in more detail in section 8.3.5.

Packets that are transmitted on the network are also encrypted at the link level. The TNC supplies the necessary link-level encryption keys at the same time as it passes the transport-level keys. The TNC can therefore entirely prevent certain hosts from communicating with each other, by refusing to divulge the keys necessary for communication. The TNC will periodically change the link-level key for each host.

The TNC is also used for auditing and administrative purposes. The TNC may want to receive and analyse network traffic directly. It should be able to detect any inconsistent traffic patterns. Also, if the capability to remotely shut down a portion of the network exists, then the TNC may decide to do so, to isolate individual hosts.

7.2.1.4. The Trusted Network Interface Unit

Each host is connected to the network through a Trusted Network Interface Unit (TNIU). This device includes the network access controller necessary for transmitting and receiving packets to and from the network, a sufficiently powerful processor and working memory to be able to implement the lower-level communication protocols, the necessary protocol software, encryption devices, specialised protocols for communicating with the TNC, and a manually installed but changeable set of keys for communicating with the TNC.

All communication from a host to the network must go through a TNIU. If an intruder attempts to transmit packets directly, circumventing the TNIU, then the intended receiver will not recognise the address, since the packet's link-level header will not be properly encrypted.

For additional security, the TNIU can be made tamper-proof to ensure that circuits cannot be modified to bypass components, and so that it is impossible to infer any of the encryption keys it employs by observing the circuits or component interfaces. In a sense, the TNIU can be viewed as a trusted extension of the TNC.

The TNIU has a number of responsibilities. First, it transmits link-level packets of fixed length. If the client wishes to send less transport-level data, then the link-level packet is filled by the TNIU with randomised data to its full length. The random fill data must be chosen so that it is indistinguishable from real data after encryption.

Second, each TNIU always transmits packets at a constant rate. If a host has no data to send, then artificial packets filled with random data are generated and sent to a (non-existent) host with a randomly generated address. If a destination address is chosen randomly, then there is a small probability that the address will be equal to that of one of the TNIUs. This will not be a problem, however, since the packet is also filled with random data and hence will not be recognised as belonging to an existing connection. These artificial packets must also be indistinguishable from real packets after encryption. A timer mechanism is used to decide when to transmit a packet. If a packet with real data is ready for transmission, then it is queued and transmitted instead of an artificial packet the next time the timer expires.

Third, each link-level packet is encrypted in its entirety. A different, randomly chosen, nonce is included in each link-level header to hide link-level retransmissions[43].

Fourth, the TNIU creates virtual connections for the host's processes and obtains encryption keys from the TNC for communication over these connections if the TNC authorises them.

Finally, the TNIU accepts transport-level packets from the host, adds nonces, and encrypts the result using an appropriate transport-level virtual-connection key. Conversely, it decrypts arriving transport-level packets and passes them to the host. The TNC may limit the usage of keys for virtual connections to a certain number of packets or a period of time. The TNC can enforce limits based on time. The TNIU must cooperage with the TNC if limits based on packet count are to be used.

Packets, including noise packets, can be transmitted from each TNIU either with a fixed period or with a constant average period. Transmitting with a constant average period is simple, and allows for the immediate transmission of an occasional high priority packet with no traffic analysis risk.

43 It is not clear that this would benefit security. This is because it is not clear that anything useful is revealed if link level retransmissions are detectable. This conjecture remains unproven either way.

Transmitting with a fixed period also has its advantages. It simplifies the timer mechanism. The bandwidth of the network is effectively partitioned into time division slots, where a TNIU transmits in every n^{th} slot, so the network operates as if it were a slotted network. Slots are assigned to TNIUs dynamically; a TNIU attempts to transmit in each successive slot until its transmission is successful. Thereafter, it transmits a packet every n^{th} slot, where n is determined by the transmission period of the TNIU. A TNIU transmits in a new slot only when a packet that was transmitted in one of its allocated slots did not successfully get through. This can only occur in the case of a transmission error or a retry for other reasons. Hence, if each TNIU transmits with the same period, retries will occur infrequently. Retries appear only when a new TNIU starts up or after a transmission error. If the network load is less than its capacity, then the allocation of slots is stable. Each TNIU will be able to acquire a slot after attempting to transmit in at most n-1 slots.

Operating the network in this fashion allows the network to run at very high loads (above 85% of capacity) without becoming unstable. Moreover, an abnormally high retry rate clearly identifies problems, either regarding the number of transmission errors or, what is more important, concerning unauthorised network transmissions. The disadvantage of transmitting with a fixed period is that it is possible for an intruder to determine that sequences of messages originate at a single host, simply by observing the repetitive traffic pattern. However, this may already be possible to a limited extent if the intruder can monitor specific parts of the physical network or the drop line from a particular TNIU. If the TNC transmits on demand, its transmissions will cause a random rearrangement of the slots, reducing the traffic analysis risk. For performance reasons, we prefer transmission with a fixed period.

7.2.2. Encryption and Keys

In this trusted network architecture, a number of different keys are used to encrypt packets both at the link and transport levels.

It is necessary that each TNIU be capable of decrypting the header of each arriving packet quickly to determine whether the connected host is the intended destination of the packet. The only data visible at the link level are addresses, an identifier indicating which virtual connection the packet belongs to, a nonce that is changed every time the packet is sent to disguise retransmissions, and a checksum. It is necessary to encrypt the entire packet at the link level to hide traffic patterns. Moreover, the encryption algorithm used at the link level must be a stateless block algorithm, so that message losses can be tolerated.

Encryption at the transport level needs to be stronger than that in use at the link level, and must also be stateless.

7.2.2.1. Link-level Keys and Booting

The TNC and every TNIU have link-level keys used for bootstrapping purposes that are periodically manually changed by a security officer. Any TNIU A stores two of these keys, namely ${}_B\lambda^E_{*S}$ for encrypting packets being sent to the TNC S and ${}_B\lambda^D_{SA}$ for decrypting packets received from the TNC. The key ${}_B\lambda^E_{*S}$ is identical on each TNIU. In the notation used here, λ indicates a link-level key, the superscript E denotes Encrypt, and D denotes Decrypt. The prefix B indicates that this key is used for bootstrapping purposes only. The index SA in λ_{SA} means that the key is used in sending packets from S to A. Thus, the key λ_{*A} refers to the key used in sending packets from any host to host A, while ${}_B\lambda^E_{*S}$ is the bootstrapping key used by all hosts for encrypting packets sent to S.

The key ${}_B\lambda^D_{*S}$ is used by S to decrypt received boot packets. For any TNIU A, the key ${}_B\lambda^E_{SA}$ is used to encrypt packets being sent to A by the TNC. The TNC must know all these keys.

These keys, ${}_B\lambda^E_{*S}$, ${}_B\lambda^D_{*S}$, ${}_B\lambda^E_{SA}$, and ${}_B\lambda^D_{SA}$ are used only for bootstrapping and resynchronisation purposes (hence the prefix B), and only for communication between TNIUs and the TNC. They need be changed infrequently.
Link-level communication is encrypted using other link-level keys that are periodically changed on the initiative of the TNC. Each TNIU B is assigned a key λ^D_{*B} for decrypting all arriving link-level packets. The TNC knows the corresponding encryption key λ^E_{*B} for each TNIU.

A TNIU A can obtain from the TNC a link-level encryption key λ^E_{*B} for encrypting link-level packets destined to host B, and host B can decrypt using λ^D_{*B}. Only B can successfully decrypt these packets[44], since only it has the key necessary to do so. Naturally, the TNC may refuse to give a link-level key to A, if it does not wish to allow A to communicate with B.

Finally, each TNIU maintains a link-level key, λ^E_{*S}, used for sending packets to the TNC. This key is also changed periodically, as dictated by the TNC.

7.2.2.2. Transport-level Keys

All communication at the transport level between processes executing on two hosts connected to TNIUs occurs over virtual connections. A pair of keys, κ^E_{AB} and κ^D_{AB}, is assigned by the TNC to each unidirectional connection. Only A knows κ^E_{AB} and only B knows κ^D_{AB}. Additionally, a

[44] Of course, S can also, but S is universally trustworthy.

connection between each TNIU and TNC exists, with two pairs of corresponding keys, $\{\kappa_{AS}^{E}, \kappa_{AS}^{D}\}$ and $\{\kappa_{SA}^{E}, \kappa_{SA}^{D}\}$.

Figure 7.3. Summary of operating encryption and decryption keys.

Key	Level	Description
λ_{*B}^{E}	Link	encrypt packets to host *B*
λ_{*S}^{E}	Link	encrypt packets to the TNC
λ_{*B}^{D}	Link	decrypt packets arriving at host *B*
λ_{*S}^{D}	Link	decrypt packets arriving at the TNC
κ_{AS}^{E}	Transport	encrypt packets sent by host *A* to the TNC
κ_{AB}^{E}	Transport	encrypt packets sent by host *A* to host *B*
κ_{SA}^{E}	Transport	encrypt packets sent by the TNC to host *A*
κ_{AS}^{D}	Transport	decrypt packets from *A* arriving at the TNC
κ_{AB}^{D}	Transport	decrypt packets from *A* arriving at host *B*
κ_{SA}^{D}	Transport	decrypt packets from the TNC arriving at *A*

The TNC may limit the validity of transport-level keys to a specific time interval or limit the number of times they are used. The TNC can enforce time limits by arbitrarily assigning new keys to the connection. It is the responsibility of the TNIUs to request new keys from the TNC for limits based on packet counts.

For bootstrapping and resynchronisation purposes, two transport-level keys, ${}_{B}\kappa_{AS}^{E}$ and ${}_{B}\kappa_{SA}^{D}$, are set manually in the TNIU *A* by a security officer. The TNC will have the two corresponding keys: ${}_{B}\kappa_{SA}^{E}$ and ${}_{B}\kappa_{AS}^{D}$. These keys need be changed infrequently.

7.2.2.3. *Key Distribution*

As described in the previous paragraphs, three mechanisms are used for key distribution. The boot keys are manually set by a security officer. These keys are used during start-up initialisation and, as described later, whenever the TNIU becomes unsynchronised with the TNC in such a way that they do not use corresponding keys.

The TNIU obtains new keys from the TNC by way of normal transport-level communication with the TNC. Each TNIU maintains a virtual circuit with the TNC over which these new keys are obtained. These keys are transmitted in packets encrypted with the link-level and transport keys currently assigned to that virtual circuit. The keys for the TNIU – TNC circuit are changed frequently, possibly as often as each TNIU – TNC connection is negotiated.

Whenever the TNIU recognises that it has lost synchronisation with the TNC for any reason, it then resorts to a recovery procedure. It uses the boot keys to communicate with the TNC, authenticate itself, and obtain new keys. This implies that the TNC be capable of decrypting each arriving packet using two different decryption keys, namely ${}_B\lambda^D_{*S}$ and λ^D_{*S}. This decryption can be done in serial order, if it can be done quickly enough. The data in the recovery packets is encrypted using the transport-level boot keys.

7.2.3. Protocols

Within the trusted network architecture, a number of protocols must be defined, including protocols for:

- Host – TNIU communication,
- TNIU – TNC communication,
- Key distribution,
- Link-level communication, and
- Transport-level communication.

Since this is an architectural definition, we do not specify the implementation for the individual protocols. Rather, we list a set of requirements to which the protocols must adhere, and additionally, present by example a possible implementation.

7.2.3.1. Host – TNIU Communication

Communication between the host and the TNIU should be minimal in the sense that information should be passed through this interface only if necessary for the operation of the interprocess communication protocols. In particular, the TNIU should not divulge reasons why particular connections could not be established. The services that are offered across this interface are:

- virtual connection establishment,
- data transfer, and
- virtual connection release.

Since the TNIU maintains virtual connections for the communication protocol software executing in the host, the host must be capable of referring to specific connections. The TNIU allocates connection identifiers (CIDs) for this purpose. The CIDs should be chosen randomly from a sufficiently large space, so that no information will be divulged by assigning a particular CID. A 32-bit CID space should be large enough. In particular, the CID should not correlate with any actual circuit identifier used internally, nor should it be assigned incrementally.

From a functional point of view, the interface between the host and the TNIU could appear similar to any standard interlevel interface, as found, for example, in the ISO OSI architecture. For instance, the interface could be defined using the following service primitives:

Connect.Request(reqid, process-A, host-A, security-A, process-B, host-B)

Connect.Response(reqid, CID)

Connect.Indication(CID)

Disconnect.Request(CID)

Data.Request(CID, Data)

Data.Indication(CID, Data)

Connect.Request, Disconnect.Request and Data.Request are issued by the host and Connect.Response, Connect.Indication and Data.Indication are issued by the TNIU. The service primitive Data.Indication indicates over which connection the data arrived. It is expected that the transport-level protocols could infer that information from the transport-level control fields, but the explicit inclusion of the CID serves as an added consistency check.

7.2.3.2. Link-level Communication

The link-level protocol header should contain as little data as necessary. The following fields suffice:

- a destination address,
- a nonce,
- a virtual circuit identifier, and
- a checksum.

A source address field is unnecessary, since packets need not be acknowledged at the link level.

The nonce is used to disguise link-level retransmissions. It should be changed on every transmission, including retransmissions. The virtual-circuit identifier is necessary to be able to select the appropriate key for transport-level decryption. The link-level protocol must implement a checksum, to detect modifications to the packet. It is not necessary to distinguish between transmission errors and errors caused by intruders, although the occurrence of abnormally high frequency of errors should be cause for further investigation and therefore reported to the TNC.

7.2.3.3. Transport-level Communication

The transport-level communication protocol must be connection oriented and must be capable of detecting and suppressing duplicate packets. It is expected that whenever an interprocess transport connection is requested, the request is passed on to the TNIU, so that it can set up a secure virtual circuit, as authorised by the TNC. Each transport-level connection must correspond to a single TNIU – TNIU connection. Once a transport-level connection is set up, each transport-level packet is passed from the host to the TNIU (together with the appropriate connection identifier), where it is encrypted using an appropriate transport-level key before being sent across the connection. The reverse path from the TNIU to the host is similar.

7.2.3.4. TNIU – TNC Communication

A TNIU connected to a host *A* communicates with the TNC for four possible reasons:

- To boot. See section 7.2.1.
- To obtain a pair of keys for encrypting and decrypting transport-level data, and the associated link-level key, using a key distribution protocol such as the one described in section 7.3.5.
- To inform the TNC of the closing of a transport connection. See section 7.3.1.
- For the TNC to force a link-level or transport-level key change. See section 7.2.2.

Whenever the TNC and a TNIU communicate, each party must authenticate the other party by recognising that the packets sent are correctly encrypted. However, to prevent delayed playbacks, it is necessary to use nonces and a three-way handshake protocol. This is described in the next section.

7.2.3.5. Key Distribution Protocols

The key distribution protocol must satisfy the following criteria:

- keys are never sent in plain text,
- the protocol survives errors in communication,
- the protocol detects playbacks, and
- keys are passed only to TNIUs.

The following is a possible transport-level key-distribution protocol. This protocol is assumed to execute over the transport-level circuit between the TNIU and the TNC. Consequently, it will be reliable and error-free. Assume a process on host A wishes to establish a connection to a process on host B. A's TNIU requests a connection and associated keys from the TNC, using a three-way handshake protocol:

$$\begin{aligned} \{N_1\}_{\kappa^E_{AS}} &: A \rightarrow \text{TNC} \\ \{N_1, N_2\}_{\kappa^E_{SA}} &: \text{TNC} \rightarrow A \\ \{N_2, connect.\mathrm{Re}quest(parameters)\}_{\kappa^E_{AS}} &: A \rightarrow \text{TNC} \end{aligned}$$

The connection request parameters are described in section 8.3.1. The TNC authenticates A's real identity by recognising that its packets are correctly encrypted, and is certain that it is not a playback because of the nonce N_2. A authenticates the TNC's identity by recognising that the packets sent by the TNC are correctly encrypted, and is certain that it does not receive replayed packets because it recognises its own nonce N_1.

If the TNC permits the establishment of the connection, it returns $f(N_1)$ a well-known function[45] of nonce N_1, a system-wide and unique identifier for the subject virtual circuit, CIRCID, together with the corresponding encryption keys κ^E_{AB} and λ^E_{*B}:

$$\{f(N_1), \text{CIRCID}, \kappa^E_{AB}, \lambda^E_{*B}\}_{\kappa^E_{SA}} : \text{TNC} \rightarrow A$$

45 Suitable functions are simple actions such as adding one to the nonce. The purpose is to indicate to the receiver of the function result that the nonce was operated on before being returned. All TNIUs use the same function.

The TNC must also communicate the corresponding keys to *B*. *A*'s TNIU may then proceed to send a packet to *B*, identifying the packet as belonging to connection CIRCID. It is easy to fully expand this protocol.

7.3. Risk Assessment

In this section we consider the security of a network that adheres to our architectural specification. We do this in two steps. First we consider common security attacks and show what security mechanisms must be compromised for a given attack to be successful. Second, we analyse the extent of damage that occurs when individual encryption keys are compromised. Note that since we do not assume a specific implementation in this analysis, the results are valid for any implementation that adheres to the trusted network architecture specification.

7.3.1. Common Security Attacks

Traffic analysis is not possible without compromising at least some of the link-level keys, since

- each TNIU transmits packets at a constant rate,
- each transmitted packet is received by each TNIU,
- each transmitted packet is of the same fixed length,
- link-level header information is encrypted so that addresses are not visible, and
- retransmissions are not detectable without compromising the link-level encryption key because of the use of nonces.

Each station generates a uniform traffic pattern, independent of the host's communication demands. For traffic analysis to be successful, link-level keys must first be compromised. However, each link-level decryption key that is compromised reveals traffic patterns only of the traffic to a particular host and does not reveal the source of the traffic, since link-level source addresses are not included in the link-level headers.

Passive wiretapping reveals no transport-level data, unless both the link-level keys and transport-level keys are compromised.

Trojan horses and covert channels are not possible, since a host can only communicate through a TNIU and because

- every network packet is of equal length,
- addresses are not visible,
- nonces change the appearance of otherwise identical packets, and the timing between successful transmissions cannot be altered.

A Trojan horse can only be successful by bypassing the TNIU and actively transmitting directly on the network. These transmissions are not intelligible to any TNIUs, since the packets are not properly encrypted. The physical security policy can prevent a host with sensitive data from being in an insecure area that would allow it to bypass its assigned TNIU. The TNC can periodically monitor the load at random times to detect extra traffic on the network.

Packet modification is possible without compromising the link-level encryption and decryption keys, but only in a random sense in that the results of the modification are unpredictable. The effects of any such modification are confined to the particular packet modified, since a block encryption algorithm is used. The link-level checksum or CRC is in the clear. Consequently, the modification can not be detected at the link level. The packet is simply ignored by the destination TNIU, since the decryption fails. Link-level modification is exactly equivalent to denial of service.

Transport-level data can be modified in a specific way only if both the link-level keys and the transport-level keys are compromised. Having the link-level keys, but not the transport-level keys, allows an intruder to modify the transport-level data, but only in a random fashion, in the sense that he cannot predict the outcome of his modifications. Such random modifications can be detected, if a checksum is also employed at the transport level.

If an intruder wishes to delete packets of a connection, then he must compromise the link-level decryption key. An intruder with the link-level encryption and decryption keys, however, has no way of knowing which transport-level connections correspond to which connection identifiers. If he wishes to delete specific packets of a specific connection, then he must also compromise the transport-level keys. Hence, denial of service attacks are possible on a broad basis, but are detectable. Denial of service attacks against a specific (software) server or even for a specific client are only possible if the corresponding transport-level and link-level keys are compromised.

Comparable arguments apply to attempts at delaying individual messages, as well as to playback attacks. Reordering and duplication of packets are detected at the transport level. Again, the intruder must compromise the link-level or transport-level keys to reorder the sequence of packets or duplicate packets of a specific connection or link (by altering transport-level header information).

7.3.2. Damage Extent after Key Compromise

In this section we analyse to what extent security is breached when individual keys are compromised. We show that the compromise of a single key reveals only a very limited amount of information, even if the key is one used to communicate with the TNC. This stability of security is achieved by using separate keys at the transport level for each virtual circuit and in each direction as well as at the link level.

Compromise of a single link-level key λ^{D}_{*B}, used by a TNIU, B, to decrypt all received packets will only expose traffic patterns of the messages encrypted with λ^{E}_{*B}, which are the messages destined for TNIU B. Both the number of virtual circuits as well as the traffic patterns of the individual virtual circuits will become visible. But, an intruder will not be able to read the data portion of the packet since it is encrypted using a transport-level key, and he or she will not be able to determine which virtual circuit is being used by which process. Virtual circuit identifiers used in the packets do not correspond to identifiers used between the host and the TNIU to identify connections. Denial of service attacks against a host, *B*, are possible if λ^{D}_{*B} is known.

Having compromised a link-level decryption key, however, does not allow the intruder to modify a packet other than randomly, a denial of service attack. For this he also needs the corresponding encryption key, λ^{E}_{*B}. Even if λ^{E}_{*B} is also compromised, then the intruder will only be able to successfully modify the header of the link-level packet, or randomly modify the data portion of the link-level packet if he does not also have the transport-level encryption keys. Compromise of a link-level key will not reveal the source of link-level packets.

To observe the traffic between two specific network nodes *A* and *B*, an intruder will need to compromise both corresponding link-level decryption keys, λ^{D}_{*A} and λ^{D}_{*B}, since the key used at the link level is effectively determined by the destination TNIU and not by the sending TNIU.

Compromise of the link-level key used to send to the TNC is more damaging, since it reveals the number of messages being sent to the TNC and hence can indicate of the number of connection requests being sent to the TNC.

Compromise of a transport-level key κ^{D}_{AB} alone is not damaging, since all packets are also encrypted at the link level. If a transport-level key together with the link-level key λ^{D}_{*B} for the destination of the virtual circuit is compromised, then an intruder will be able to observe data being sent over a virtual circuit in one of the directions and potentially be able to selectively remove packets from the network. If the intruder also has the corresponding encryption keys, κ^{E}_{AB} and λ^{E}_{*B}, then he or she will also be able to undetectably modify packets. But not until the intruder has compromised the keys corresponding to the return path, κ^{D}_{BA} and λ^{D}_{*A}, will she or he be able to observe the traffic of a connection in both directions. With these keys, an intruder will successfully be able to modify packets and be undetectable while doing so.

If the transport-level key and the link-level key of the TNC is compromised then an intruder will be able to

- observe the number of connection establishment requests of each TNIU,
- selectively deny service to a particular TNIU, and
- prevent the communication of certain transport-level messages to the TNC, containing, for example, auditing information.

But having compromised these two keys alone will not reveal further keys, since keys are transmitted only in the direction from the TNC to the TNIUs.

If the transport-level key used to decrypt messages sent from the TNC to the host A's TNIU, κ^D_{SA}, is compromised together with the corresponding link-level key, λ^D_{*A}, then all keys the TNC sends to the TNIU will be revealed, including changes to the keys used to communicate with the TNC itself. Depending on the key distribution protocol used, in the worst case this can include the keys κ^E_{Ah}, κ^D_{hA}, κ^E_{*S}, κ^D_{SA}, λ^D_{*A}, λ^E_{*h}, λ^E_{*S} and all other keys that are used for virtual circuits from any particular host A to other hosts h.

BIBA DATA INTEGRITY

8. BIBA DATA INTEGRITY

K.J. Biba, in the abstract of a report sponsored by the U.S. government and dated July 1975, stated[46]:

> The integrity of data and procedures is the cornerstone for the goal of preservation of proper system behaviour. The problem of ensuring integrity is particularly acute for a secure, kernel-based computer utility. This paper investigates the issues of integrity policy for both the kernel itself and for the virtual environment provided by the kernel. While our particular concern is a secure computer utility for the military environment (Multics), the considerations are applicable to a larger community of systems.

Biba quotes Webster as defining integrity[47] as "Integrity – 1a: an unimpaired or unmarred condition; entire correspondence with an original condition; soundness. 1b: an uncompromising adherence to a code of moral, artistic, or other values.". The unabridged, second edition's definition is "Integrity – 1. the quality or state of being complete; wholeness; entireness; unbroken state. 2. the entire unimpaired state or quality of anything; perfect condition; soundness. 3. the quality or state of being of sound moral principle; uprightness, honesty, and sincerity.". Speaking of integrity in general, he goes on to state that "integrity is a static property of a dynamic system. That is, integrity guarantees that a system will perform in the manner determined by its initial, isolated condition. ... A person thought to have the property of integrity is only considered to behave consistently with respect to some standard: no statement (or decision) about the quality of the standard is implied."

46 K.J. Biba, "Integrity Considerations for Secure Computer Systems", Mitre Technical Report ESD-TR-76-372, July 1975. The actual form of words used in the quotation that is given above has been slightly adapted from the original to make it more comprehensible.

47 The shorter Oxford dictionary defines integrity as "integrity 1450 **1.** The condition of having no part or element wanting; unbroken state; material wholeness, completeness, entirety. **2.** Unimpaired or uncorrupted state; original perfect condition; soundness – 1450. **3. †a.** Innocence, sinlessness – 1678. **b.** soundness of moral principle; the character of uncorrupted virtue; uprightness, honesty, sincerity – 1548.

Carrying the analogy to computer systems, Biba defines integrity thus:

> "A system possesses the property of integrity if it can be trusted to adhere to a well-defined code of behaviour. No *a priori* statement as to the properties of this behaviour are relevant."

"The concern of computer system integrity is thus the guarantee of unfailing adherence to a code of behaviour". The concern about the initial condition is dismissed; it is assumed that the system can start with integrity, and the concern is that it retain its integrity by rigidly following a security policy.

The definition of integrity given above is difficult to challenge. The problem with integrity is that subsequently it has acquired a different meaning: many authors knew that the control of access-to-observe was covered by confidentiality policy, and they came to refer to integrity as the control of access-to-modify. Sometimes the control of access-to-invoke has also been included in integrity, but not often. The identification and enforcement of an integrity policy governing proper modifications is essential for the implementation of an effective kernel-based confidentiality policy[48]. It is not, however, a confidentiality policy.

In the remainder of these notes an attempt will be made to use the word integrity as it is used in the trade, whatever definition it has.

8.1. The Elements of a Mandatory Integrity Policy

The integrity policy that is being discussed governs accesses by active subjects to passive objects. Each integrity policy is defined as a set of relations governing allowed accesses by members of the sets of subjects to members of the set of objects. There are three sub-policies to be discussed,

- accesses that imply observation of an object by a subject,
- accesses that imply modification of an object at the behest of a subject, and
- accesses that imply invocation of a subject at the behest of another subject.

There are two important parameters that a policy will use to decide whether to allow or prohibit an access: the clearance of the user responsible for the access request, and the classification of the data.

The user clearance is extended to processes acting on the user's behalf. Every process and every user must have a specific identity that can not be hidden and that is impossible for another user to assume. The user clearance is expressed as an integrity level, just as security levels have been discussed previously. A user's integrity level is a measure of the trustworthiness of that individual to be entitled to modify data, and may involve both a sensitivity measure and a set of allowed

48 Otherwise intruders will modify the kernel.

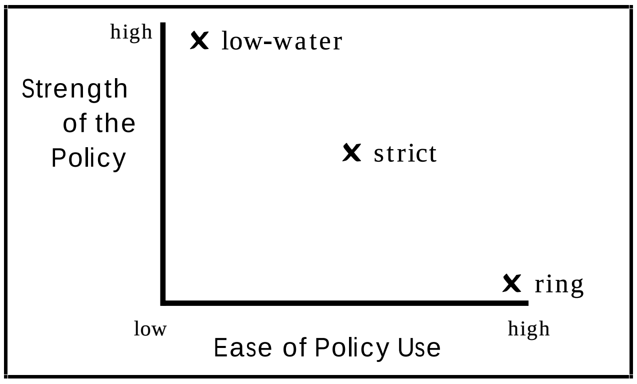
compartments. The same clearance that is used for confidentiality can be used for integrity, but it is convenient to think and speak of it differently. This difference is in spite of the simple principle that a user who is trusted to observe data is not likely to carelessly or maliciously modify it.

Thus, there is a security level and an integrity level. The security level applies to observation in the normal way. The integrity level applies to modification and sometimes to invocation.

The data classification is assigned to data on the basis of possible damage caused by unauthorised disclosure. Like clearance, in general it will involve both a sensitivity and a set of categories. It is possibly the case that the set of security levels that apply to confidentiality will be distinct from those that apply to modification. It is possible that, for instance, a milder form of damage might be done by unauthorised observation of some data, but exceptionally grave damage would result from its unauthorised modification. The former implies a low security level; the latter requires a high integrity level. For the purposes of these notes the possibility of parallel but disjoint classification levels will be assumed. Nobody really knows whether the apparent need for this pair of levels is real or imagined. As Biba says, "It is not very practical to partition the trustworthiness of individuals with respect to disclosure [observation] and sabotage [modification]."

The issue of a separate invocation policy will be left for further study, except for a simplistic view that falls out of the approach to the other policies. There has been little effective work in the field.

Having said all this, and following Biba, the remainder of these notes will present three models of modification policy. Each is consistent with the normal confidentiality policy, and they represent the three points on the curve shown next.



8.1.1 Definitions

The definitions are consistent with those of Bell & LaPadula, so expect no surprises. Each of the three integrity models to be presented has the following basic elements:

S: the set of subjects *s*.

O:	the set of objects o.
I:	the set of integrity levels discussed above.
f:	a function defining the integrity level of each subject and each object; f defines a lattice under the relation dominates.
$\triangleright$:	a relation (a subset of $I \times I$) defining a partial ordering dominates on the set of integrity levels I.
$\triangleright\!\!>$:	an antisymmetric, transitive relation (a subset of $I \times I$) defining a properly dominates relation on I. If $I_A \triangleright\!\!> I_B$ then $I_A \triangleright I_B$ and $I_A \neq I_B$.
$\wedge$:	the meet (greatest lower bound) of its two arguments. If $I_C = I_A \wedge I_B$ then $I_A \triangleright I_C$ and $I_B \triangleright I_C$ and there is no I_D such that $I_A \triangleright I_D$, $I_B \triangleright I_D$, and $I_D \triangleright I_C$.
$\vec{\mathbf{o}}$:	a relation (a subset of $S \times O$) defining the authority of a subject $s \in S$ to observe an object $o \in O$; $s\,\vec{\mathbf{o}}\,o$ iff the subject s is authorised to observe the object o.
$\vec{\mathbf{m}}$:	a relation (a subset of $S \times O$) defining the authority of a subject $s \in S$ to modify an object $o \in O$; $s\,\vec{\mathbf{m}}\,o$ iff the subject s is authorised to modify the object o.
$\vec{\mathbf{i}}$:	a relation (a subset of $S \times S$) defining the authority of a subject $s_1 \in S$ to invoke a subject $s_2 \in S$; $s_1\,\vec{\mathbf{i}}\,s_2$ iff the subject $s_1$is authorised to invoke the subject s_2.

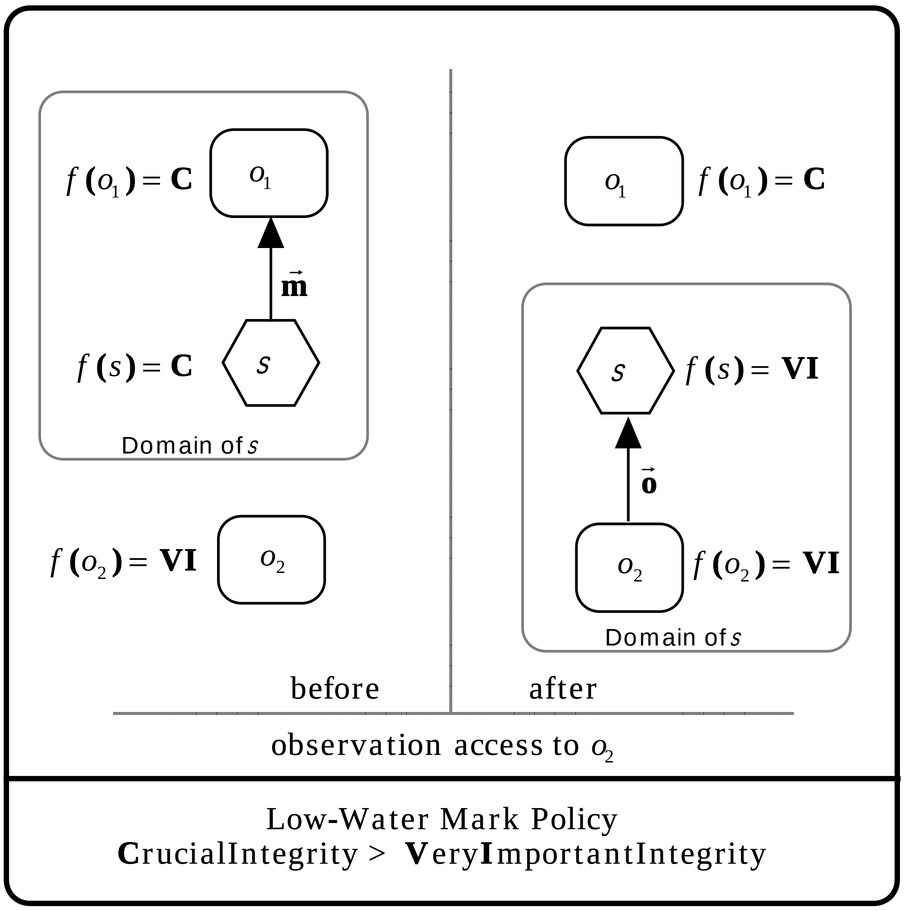
Using these definitions three alternative integrity policies there were alluded to above can be defined. The policies are defined by axioms that constrain the membership of the sets $\vec{\mathbf{o}}$, $\vec{\mathbf{m}}$, and $\vec{\mathbf{i}}$. A number of other policies are conceivable, some of which are of dubious utility.

8.1.2. The Low-Water Mark Policy on Subjects

There is a high-water mark confidentiality policy that requires that a subject's security level always be as high as the security level of any object that has been observed by it. The subject's security level can never decrease; the security level of the subject is either static at the high-water mark, or dynamic and monotone non-decreasing. In the latter case the security level of subjects that are governed by the high-water mark policy is not static but is a function of the previous behaviour of the subject. The high-water policy is based on the notion that if a subject has ever seen an object at the level of, say, secret, its subsequent behaviour could be unalterably changed by the experience. It might not be possible to guarantee that the subject's behaviour would keep the secret data confidential if it were possible for the subject to subsequently operate at a lower level.

A direct analogue to the high-water mark confidentiality policy is the low-water mark integrity policy. The low-water mark integrity policy provides for a dynamic non-increasing value of the integrity level $f(s)$ of subjects. The value of $f(s)$ at any time is the low-water mark determined from the previous behaviour of the subject s. The low-water mark is the greatest lower bound of the integrity level of all objects accessed for observation by the subject s. The subject is constrained to modify only those objects that possess an integrity level that dominates the subject's integrity level.

Clearly time is a factor here. Let the superscript * indicate the "after" alternative, and the absence of the superscript the "before" situation. The following figure illustrates these concepts and the axioms formalise them:



A2.1 $\forall s \in S$ and $o \in O$ the authority $s\,\vec{\mathbf{o}}\,o$ for the subject to observe the object is controlled by the confidentiality policy requiring the subject's security level to dominate the object's security level.

For each observe access of an object o by a subject s, the subject's integrity level $f(s)$, changes immediately after the access. The new value of the integrity level is defined by $f^*(s) = f(s) \wedge f(o)$.

A2.2 $\forall s \in S$ and $o \in O$ the authority $s\,\vec{\mathbf{m}}\,o$ for the subject to modify the object implies that $f(s) \triangleright f(o)$.

This is not the usual write-up condition because its *-property is replaced by this axiom involving integrity levels. For the low-water mark policy the subject's integrity level $f(s)$ is not static. It would be convenient if this axiom could be consistent with the *-property. With careful definition it can be, but only with an important additional restriction. This will be discussed below.

A2.3 $\forall s_1, s_2 \in S$ the authority $s_1 \vec{\mathbf{i}}\, s_2$ for the first subject to invoke the second subject implies that $f(s_1) \triangleright f(s_2)$.

This is not a surprise; a subject can only invoke other subjects that it dominates. In many practical cases both would need to be at the same integrity level if parameters are to be passed (s_2 observes s_1) and results are returned (s_1 observes s_2).

A2.1 ensures that the subject can not benefit from the prior observation of higher integrity level data. By lowering the integrity level of the subject depending on its experience, it is intended to ensure that indirect modifications that are improper can not happen. A2.2 ensures that direct modification is authorised only when the subject is suitably cleared. This is not quite the same as the usual write-up condition. In fact, the integrity level of the subject is pegged at the glb of the integrity levels of all objects yet observed by the subject. Thus, if there are three integrity levels[49], important, very important, and crucial, a subject has seen confidential data the integrity of which is very important, subsequently it can not modify any crucial data. A2.3 ensures that subjects without adequate authority can not achieve some otherwise unauthorised modification by invoking some more privileged subject.

Definition:

D2.1 An information transfer path is a sequence of objects $o_1, \cdots, o_{n+1}$ and a corresponding sequence of subjects $s_1, \cdots, s_n$ such that
$$\forall i \in [1, \cdots, n],\ s_i\, \vec{\mathbf{o}}\, o_i \text{ and } s_i\, \vec{\mathbf{m}}\, o_{i+1}.$$

Theorem:

T2.1 If there exists an information transfer path from object o_1 to object o_{n+1} then enforcement of the low-water mark policy requires that
$$f(o_{n+1}) \triangleright f(o_1).$$

The proof of the theorem is straightforward.

The low-water mark policy has some unfortunate behaviour. The changing integrity level makes programming unpleasantly difficult, since it would seem to be necessary to predict the objects that are to be seen by a subject in advance, and to control this rigidly, if subjects are to avoid ending up at too low a integrity level. Subjects can sabotage their own

49 This example is from Biba *op cit*.

usefulness by making objects that are necessary for proper functioning inaccessible. There is no recovery short of invoking the superuser.

The low-water mark policy is consistent with Biba's hypothesis that "A subject that is trusted not to divulge information is not likely to maliciously modify it". Modern practice realises that, while the possibility of malicious modification persists, the greater danger is carelessness and inattention.

An interesting observation about the integrity lattice and the confidentiality lattice emerges. If one ascribes rigidly to the read-down-and-write-up concept for the (confidentiality) security levels, then it is evident that the confidentiality lattice and the integrity lattice must be duals of one another. This will be discussed again subsequently.

8.1.2.1. The Low-Water Mark Policy on Objects

The Low-Water Mark Policy on subjects explicitly requires that the integrity level of a subject changes. An alternative formulation can be constructed in which the integrity level of an object also changes as the result of a modification. Thus, the axioms would become

A2.1 $\forall s \in S$ and $o \in O$ the authority $s\,\vec{\mathbf{o}}\,o$ for the subject to observe the object is determined by the confidentiality policy. For each observe access of an object *o* by a subject *s*, the subject's integrity level $f(s)$, changes immediately after the access. The new value of the integrity level is defined by $f^{*}(s) = f(s) \wedge f(o)$.

A2.2 $\forall s \in S$ and $o \in O$ the authority $s\,\vec{\mathbf{m}}\,o$ for the subject to modify the object implies that $f(s) \triangleright f(o)$. For each modify access of an object *o* by a subject *s*, the object's integrity level $f(o)$, changes immediately after the access. The new value of the integrity level is defined by $f^{*}(o) = f(s) \wedge f(o)$.

It is evident that the integrity level of any subject or object that is accessed is monotone non-increasing. Ill-considered, malicious, or unplanned behaviour by a subject or group of subjects can result in the integrity level of every subject and object gravitating to the lowest integrity level of any accessed object. In this state all objects are accessible by all subjects.

The modification relation is close to the write-up axiom. However, because the level of a subject can drop until it can modify any object, this policy does not prevent improper modification. Instead it adjusts the level of a modified object to reflect its experience of modification.

8.1.2.2. A Low-Water Mark Corruption Level

Biba identified an unplanned possible outcome of the low-water mark model as the possibility of tracking a risk measure of contamination of a data item with corruption as a result of observation or modification. He defined a <u>current corruption level</u> c as the following:

$c^*(s) = c(s) \wedge c(o)$ for each observe access by a subject s to an object o.
$c^*(o) = c(s) \wedge c(o)$ for each modify access by a subject s to an object o.

Accordingly, for an object the corruption level $c(o)$is the least integrity level of the data that could have been used to modify the object. For a subject the corruption level $c(s)$is the least integrity level of the data that could have been used to modify the object.

An obvious parallel can be drawn with the level of contaminant M that appeared in the risk analysis developed in the previous essay. The policy here is not the same as that in the previous essay. The notion of a contaminant from the previous essay risk analysis is not the same as the idea of corruption defined here. The ideas are, however, similar. A parallel development can develop a measure of the maximum level of contaminant that a given object could be tainted with. This would provide the possibility of a kind of dynamic risk analysis, possibly a significant advance on the static analysis now existing.

8.1.3. The Ring Policy

The low-water mark policy prevents unauthorised direct modifications. Indirect modifications, those that are done later or on behalf of an offending subject, are also prohibited by the lowering of the subjects integrity level.

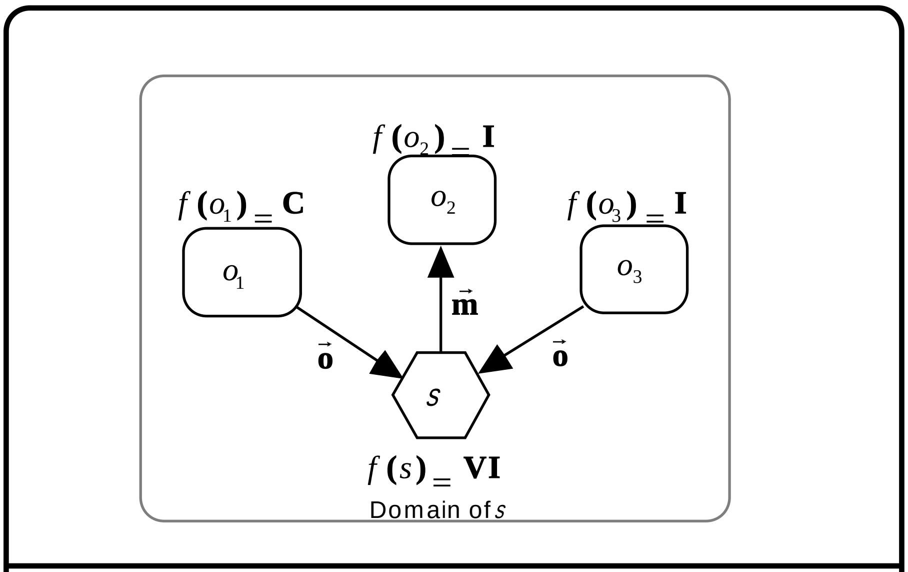
The ring policy described here prevents only direct modifications. It includes no record of a subject's or an object's experience. Thus it includes less substantial assurances of what it defines as integrity, but the flexibility of the system is significantly increased.

The ring integrity policy is defined by two axioms:

A2.4 $\forall s \in S$ and $o \in O$ the authority $s\,\vec{\mathbf{m}}\,o$ for the subject to modify the object implies that $f(s) \rhd f(o)$.

A2.5 $\forall s_1, s_2 \in S$ the authority $s_1\,\vec{\mathbf{i}}\,s_2$ for the first subject to invoke the second subject implies that $f(s_1) \rhd f(s_2)$.

Thus, by construction no object with an integrity level that dominates a subject's integrity level can be modified by the subject. Also, no subject with an integrity level that dominates an invoking subject's integrity level can be properly called by the invoking subject. The uncoupling of the confidentiality and integrity policies allows a subject much more latitude to perform its duties, and implies corresponding responsibility to specify, design, verify and monitor that performance to guard against unwanted effects erupting.



Ring Policy
Crucial > **V**ery**I**mportant > **I**mportant

In the figure the subject s has observe access to both o_1 and o_3. This in spite of the evident fact that the integrity level of o_1 is greater than that of s and conversely for o_3. This illustrates the fact that observe access is controlled by the security (confidentiality) levels. There is no direct required relationship between the levels in this model. In the strict integrity model such a relationship appears.

8.1.3. The Strict Integrity Policy

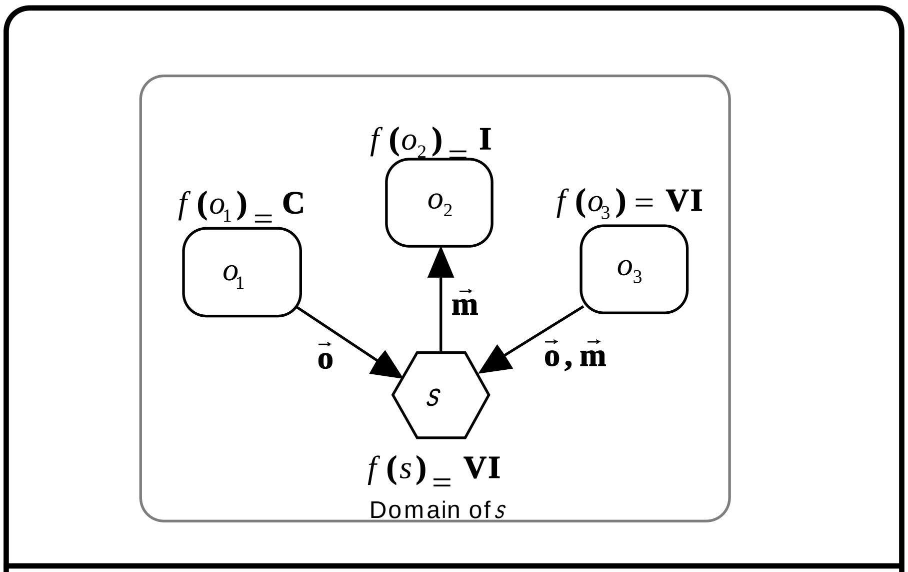
The strict integrity policy is often considered to be the dual of the security (confidentiality) policy. It has three axioms, two of which are directly analogous to the Bell & LaPadula SS property and *-property. The three integrity axioms are:

A2.6 $\forall s \in S$ and $o \in O$ the authority $s\,\vec{\mathbf{o}}\,o$ for the subject to observe the object implies that $f(o) \triangleright f(s)$.

A2.7 $\forall s \in S$ and $o \in O$ the authority $s\,\vec{\mathbf{m}}\,o$ for the subject to modify the object implies that $f(s) \triangleright f(o)$.

A2.8 $\forall s_1, s_2 \in S$ the authority $s_1\,\vec{\mathbf{i}}\,s_2$ for the first subject to invoke the second subject implies that $f(s_1) \triangleright f(s_2)$.

This policy is related to the low-water mark policy as follows: In cases where the low-water mark policy permits a modification and then lowers the subject's integrity level, the strict policy just prohibits the modify access. Thus there can be no improper modifies. In this sense the policy is stricter than the low-water mark policy.



Strict Integrity Policy
Crucial > **V**ery**I**mportant > **I**mportant

Clearly care must still be taken to ensure that the subject is neither careless nor malicious when it modifies an object. This is a common thread that runs through all these policies and that provides the opportunity for the more malevolent security risks to prosper and thrive. A subject that includes a trapdoor or a virus might make arbitrary changes to any object it is legitimately authorised to observe.

A close inspection of the three axioms will lead to the conclusion that the lattice defined by the integrity axioms above is exactly the dual (reverse partial ordering) of the security (confidentiality) lattice. The confidentiality levels could be used as the integrity levels with the integrity dominance relation the converse of the confidentiality dominance relation. This is not normally done because a few inconsistent other responsibilities are associated with each type of level, such as contamination for the integrity levels and the relationship to then user's classification for the confidentiality levels.

Objects with high integrity have low confidentiality, and conversely. This implies that the integrity of a Top Secret object will be dominated by the integrity of a Secret object, et cetera. Unclassified objects with no categories have the highest integrity. The situation is not as absurd as it sounds because of the interpretation that both policies need to be satisfied to overwrite an object. If it can't be observed it can't be overwritten. If only the integrity policy is satisfied appending is allowed but not overwriting. In the append-only case an unauthorised modification becomes something that any subject with observe access can recover from, at least in principle.

It has been observed that objects that are universally accessible for observation, or nearly so, should have the highest integrity because they may be observed by (nearly) any subject, whereas an object with an extremely high confidentiality level is observable by only a tiny community of specialists who should be able to detect and repair poor integrity because they can observe so many things. Against this argument is the notion that an object is accorded a high confidentiality level because it has a high value, or it is highly sensitive, or high damage will result from an unauthorised observation of it. It would be satisfying if an object of high value had high integrity without any special actions on the part of the accessing subject. These are arguments that are not reconcilable. As Prince Henry said[50] when contemplating robbing Sir John Falstaff and some other thieves, "it would be argument for a week, laughter for a month, and a good jest for ever."

Strict integrity is used fairly often. At least one consultant makes a successful practice advising clients on its adoption in restricted application-oriented environments. It has the pleasing property that the security system is not attempting to second-guess users as to the correct value of data. This is an unfortunate property found in many other approaches to data integrity.

50 Shakespeare, HenryIV, Part I, II, ii, 104.

Clark–Wilson Commercial Policy

9. Clark-Wilson Commercial Policy

In 1987 David Clark and David Wilson published an important paper[51] on the subject of commercial versus military security policies. Their thesis is that the important problem in commercial work is the integrity of the data. In the paper they develop an architecture that they devised for the control of data integrity. They then comment on the relationship between their model and others that are favoured more by the military. The Clark-Wilson model is widely referenced. This essay summarises it and discusses its strengths and its weaknesses. The following notes are strongly based on the paper. The material has, however, been reordered and placed in this new context.

9.1. Commercial Security Policy For Integrity

Clearly, control of confidential information is important in both the commercial and military environments. In the military environment the driving policy is confidentiality. However, a major goal of commercial data processing, often the most important goal, is to ensure integrity of data to prevent fraud and errors. No user of the system, even if authorised, should be permitted to modify data items in such a way that assets or accounting records of the company are lost or corrupted. This must be balanced, however, with the need of users to exercise absolute control over the value of the input data that they are providing to the system. As shall be seen, one of the weaknesses of Clark-Wilson is that it attempts to control modifications in such a way as to prohibit those modifications that do not fit with predicted behaviour.

---

[51] David D. Clark and David R. Wilson, "A Comparison of Commercial and Military Computer Security Policies", *Proc. 1987 IEEE Symposium on Security and Privacy*, Oakland California, April 27-29, 1987. pp. 184-194.

Some mechanisms in the system, such as user authentication, are an integral part of enforcing both the commercial and military policies. However, other mechanisms are very different. The high-level mechanisms used to enforce commercial data integrity policies were derived long before computer systems came into existence. In the commercial world there are two mechanisms at the heart of fraud and error control: the well-formed transaction, and separation of duty among employees.

The Clark-Wilson concept of the well-formed transaction is that a user should not manipulate data arbitrarily, but only in constrained ways that preserve or ensure the integrity of the data. A very common mechanism in well-formed transactions is to keep a journal of all data modifications so that actions can be audited later. There have been experiments with computational systems that keep sufficient data in the journal such that time can be reversed and the data base can be restored to a previous state by running the journal backwards. Before the computer, clerks were instructed to write in ink, and in case of error to make correcting entries rather than erase. In this way the books themselves, being write-only with over-writing prohibited, are the journal. Any evidence of erasure was indication of fraud.

Perhaps the most formally structured example of well-formed transactions occurs in accounting systems, which model their transactions on the principles of double entry bookkeeping. Double entry bookkeeping ensures the internal consistency of the system's data items by requiring that any modification of the books comprises two parts, which account for or balance each other. For example, if a cheque is to be written (which implies an entry in the cash account) there must be a matching entry on the accounts payable account. If an entry is not performed properly, so that the parts do not match, this can be detected by an independent test (balancing the books). It is thus possible to detect such simple frauds as the issuing of unauthorised cheques.

The second mechanism to control fraud and error, separation of duty, attempts to ensure the external consistency of the data objects: the correspondence between the data object and the real world object it represents. Because computers do not normally have direct sensors to monitor the real word, computers cannot verify external consistency directly. Rather, the correspondence is ensured indirectly by separating all operations into several subparts and requiring that each subpart be executed by a different person. For example, the process of purchasing some item and paying for it might involve subparts: authorising the purchase order, recording the arrival of the item, recording the arrivals of the invoice, and authorising payment. The last subpart, or step, should not be executed unless the previous three are properly done. If each step is performed by a different person, the external and internal representation should correspond unless some of these people conspire. If one person can execute all of these steps, then a simple form of fraud is possible, in which an order is placed and payment made to a fictitious company without any actual delivery of items. In this case, the books appear to balance; the error is in the correspondence between real and recorded inventory.

Perhaps the most basic separation of duty rule is that any person permitted to create or certify a well-formed transaction should not be permitted to execute it (at least against production data). This rule ensures that at least two people are required to cause a change in the set of well-formed transactions.

The separation of duty method is effective except in the case of collusion among employees. For this reason, a standard auditing disclaimer is that the system is certified correct under the assumption that there has been no collusion. While this might seem a risky assumption, the method has proved very effective in practical control of fraud. Separation of duty can be made very powerful by thoughtful application of the technique, such as random selection of the sets of people to perform some operation, so that any proposed collusion is only safe by chance. Separation of duty is thus a fundamental principle of commercial data integrity control.

Therefore, for a computer system to be used for commercial data processing, specific mechanisms are needed to enforce well-formed transactions and separation of duty. A precondition to a guarantee that data items are manipulated only by means of well-formed transactions is to ensure that the data items can be manipulated only by a specific set of programs. These programs must be inspected for proper specification, design, policy enforcement, construction, and evolutionary maintenance. Controls must be provided on the authority and ability to install and modify these programs so that their continued validity is ensured. One way to enforce the separation of duties is to permit each user to use only certain suites of programs. These must be distinct users, not an individual having several roles. The assignment of people to programs must again be inspected to ensure that the desired controls are actually met. In this way the right of execution access becomes central to commercial data integrity.

9.2. Differences with the Orange Book

The Clark-Wilson integrity mechanisms differ in a number of important ways from the mandatory controls for military security as described in the Orange Book.

With these integrity controls, a data item is not necessarily associated with a particular security level, but rather with a set of programs permitted to modify it. This models quite well the way commercial organisations manage their affairs. However, the use of data networks means that data may be widely accessible. The risk that some other user using some other program (or the same one without authorisation) grows. Putting the controls with the programs (ralph and rebecca can use me) is akin to a capability. It seems more elegant and possibly safer to associate the controls with the objects being protected, the data (ralph and rebecca can modify me). Then if there is any bypass of the security controls it must be an event of considerable significance, involving a bypass of the reference monitor in the TCB.

If Clark-Wilson controls are implemented there must still be a TCB and a reference monitor. Otherwise, careless authorised users, authorised users taking a shortcut, or intruders can bypass the programs that are supposed to govern modifications and can act unilaterally.

Clark-Wilson does not give a user authority to read or write certain data items, but to execute certain programs on certain data items. The distinction between these two mechanisms is fundamental. With the Orange Book controls, a user is constrained by what data items he can read and write. If he is authorised to write a particular data item he may do so in any way he chooses. This effect may speak for the provision of a less permissive interpretation of the Orange book controls. With Clark-Wilson controls, the user is constrained by what programs he can execute, and the manner in which he can read or write data items is implicit in the actions of those programs. Because of separation of duties, it will almost always be the case that a user, even though he is authorised to modify a data item, can do so only by using some of the transactions defined for that data item. Other users, with different duties, may have access to different sets of transactions related to that data.

9.3. Mandatory Commercial Controls

The notion of mandatory control is central to the mechanisms for military security, but the term is not usually applied to commercial systems. That is, commercial systems have not reflected the idea that certain functions, central to the enforcement of policy, are designed as a fundamental characteristic of the system. However, it is important to understand that the mechanisms described in the previous section imply mandatory controls. They are mandatory in that the user of the system should not, by any sequence of operations, be able to modify the list of programs permitted to manipulate a particular data item or to modify the list of users permitted to execute a given program. If the individual user could do so, then there would be no control over the ability of an untrustworthy user to alter the system.

In the commercial integrity environment, the owner of an application and the general controls implemented by the data processing Organization are responsible for ensuring that all programs are well-formed transactions. As in the military environment, there is usually a designated separate staff responsible for assuring that users can execute transactions only in such a way that the separation of duty rule is enforced. The system ensures that the user cannot circumvent these controls. This is a mandatory rather than a discretionary control.

The two mandatory controls, military and commercial, are very different mechanisms. They do not enforce the same policy. The military mandatory control enforces the correct setting of classification levels. The commercial mandatory control enforces the rules that implement the well-formed transaction and separation of duty model. When constructing a computer system to support these mechanisms, significantly different low-level tools are implemented.

Another difference between the two forms of mandatory control is related to its administration. In the Clark-Wilson model there must be considerable discretion left to the security administrator of the system, because the determination of what constitutes proper separation of duty can be done only by a comparison with application-specific criteria. The separation of duty determination can be rather complex, because the decisions for many of the transactions may interact. This greater discretion means that there is also greater scope for error by the security officer, and that the system is nearly unable to prevent the security officer, as opposed to the user, from blundering.

The behaviour, however, of the two mandatory controls is similar. The rules are a fundamental part of each system, and may not be circumvented, only further restricted, by any other discretionary control that exists.

9.4. A Formal Model Of Integrity

This section introduces a more formal presentation for the Clark-Wilson model for data integrity within computer systems. The specific integrity policies associated with accounting practices are used as examples, but the model is believed to be applicable to a wide range of integrity policies.

To begin, those data items within the system to which the integrity model must be applied are identified and labelled as Constrained Data Items, or CDIs. The other data objects in the system are called Unconstrained Data Items, or UDIs.

The particular integrity policy that is desired is defined by two classes of procedures: Integrity Verification Procedures, or IVPs, and Transformation Procedures, or TPs. The purpose of an IVP is to confirm that all of the CDIs in the system conform to the integrity specification at the time the IVP is executed. Thus, what has just been called the integrity specification is a central part of Clark-Wilson. The IVP is supposed to be able to deduce that a given data item is in a state that does not, or may not, conform with reality. This is a requirement without the IVP being able to test reality. In the accounting example, the IVP view of reality corresponds to the audit function, in which the books are balanced and reconciled to the external environment. How the IVP does the latter is not clear.

The TP corresponds to the concept of the well-formed transaction. The purpose of the TPs is to change the set of CDIs from one (hopefully) valid state to another. In the accounting example, a TP would correspond to a double entry transaction.

To maintain the integrity of the CDIs, the system must ensure that only a TP can manipulate the CDIs. Only the TCB can reliably do this. It is this constraint that motivated the term Constrained Data Item. Given this constraint, Clark-Wilson argue that, at any given time, the CDIs meet the integrity requirements. Clark-Wilson calls this condition a "valid state." They assume that at some time in the past the system was in a valid state, because an IVP was executed to verify this. Reasoning forward from this point, they examine the sequence of TPs that have been executed. For the first TP executed, they assert that it left the system in a valid state by the following reasoning.

By definition the TP will take the CDIs into a valid state if they were in a valid state before execution of the TP.[52] But this precondition was ensured by execution of the IVP. For each TP in turn, this necessary step can be repeated to ensure that, at any point after a sequence of TPs, the system is still valid. This proof method resembles the mathematical method of induction, and is valid provided the system ensures that only TPs can manipulate the CDIs.[53]

While the system can ensure that only TPs manipulate CDIs, it cannot ensure that the TP performs a well-formed transformation. The validity of a TP (or an IVP) can be determined only by certifying it with respect to a specific integrity policy[54]. In the case of the bookkeeping example, each TP would be certified to implement transactions that lead to properly segregated double entry accounting. The certification function is usually a manual operation, although some automated aids may be available.

Integrity assurance is thus a two-part process: certification, which is done by the security officer, system owner, and system custodian with respect to an integrity policy; and enforcement, which is done by the system. The Clark-Wilson model to this point can be summarised in the following three rules:

---

52 The CW definition of a TP does not constrain the destination state of a transaction. Rather, it defines which user can modify data. Only if the users modify the data properly will the resulting state be valid. The CW model is murky in this regard, but the intent is clear, so there is no point being pedantic about it. See the next paragraph.

53 There is an additional detail, which the system must enforce, which is to ensure that TPs are executed serially, rather than several at once. During the mid-point of the execution of a TP, there is no requirement that the system be in a valid state. If another TP begins execution at this point, there is no assurance that the final state will be valid. To address this problem, most modern data base systems have mechanisms to ensure that TPs appear to have executed in a strictly serial fashion, even if they were actually executed concurrently for efficiency reasons.

54 And by controlling the data values the users enter. Again CW is deficient in this regard. The reason seems to be that the accounting metaphor was firmly planted in the minds of the authors. Few other transaction-based systems have the ability to require the users to enter data in consistent groups rather than as individual entries. The accounting example will only enforce correct entries if ever entry has an incremental audit associated with it. This may imply more overhead on the system processing power than can be justified.

C1: (IVP Certification) All IVPs must properly ensure that all CDIs are in a valid state at the time the IVP is run.

C2: (Validity) All TPs must be certified to be valid. That is, they must take a CDI to a valid final state, given that it is in a valid state to begin with. For each TP, and each set of CDIs that it may manipulate, the security officer must specify a relation or function that defines that execution. A relation is thus of the form [TP_1, (CDI_a, CDI_b, CDI_c, . . .)], where the list of CDIs defines a particular set of arguments for which TP_1 has been certified.

E1: (Enforcement of Validity) The system must maintain the list of relations specified in rule C2, and must ensure that the only manipulation of any CDI is by a TP, were the TP is operating on the CDI as specified in some relation.

The above rules provide the basic framework to ensure internal consistency of the CDIs. To provide the separation of duty mechanism, additional rules are needed to control which persons can execute which programs on specified CDIs:

E2: (Enforcement of Separation of Duty) The system must maintain a list of relations of the form: [UserID, TP_1, (CDI_a, CDI_b, CDI_c, . . .)], which relates a user, a TP, and the data objects that TP_1 may reference on behalf of that user. It must ensure that only executions described in one of the relations are performed.

C3: (Separation of Duty Certification) The list of relation in E2 must be certified to meet the separation of duty requirement. In effect, they must exactly satisfy policy.

Formally, the relations specified for rule E2 are stronger than those of rule E1 are, so E1 is unnecessary. However, for both philosophical and practical reasons, it is helpful to have both sorts of relations. Philosophically, keeping E1 and E2 separate helps to indicate that there are two basic problems to be solved: internal and external consistency. As a practical matter, the existence of both forms together may permit complicated relations to be expressed with shorter lists, by the use of identifiers within the relations that use "wild card" characters to match classes of TPs or CDIs. On the other hand, mixing a formal definition with an implementation detail is not usually productive. Having only E2 would obviate the worry that E1 and E2 become inconsistent, a matter of considerable concern.

The above relation made use of UserID, an identifier for a user of the system. This implies the need for a rule to define these:

E3: (User Identity) The system must authenticate the identity of each user attempting to execute a TP.

Rule E3 is relevant to both commercial and military systems. However, commercial systems use the identity of the user to enforce very different policies than military systems. The relevant policy in the military context, as described in the Orange Book, is based on level and category of clearance, while the commercial policy is likely to be based on separation of responsibility among two or more users. This difference is exacerbated if in the commercial system users are permitted to change roles, because a different set of authorities and responsibilities may be associated with the user in each of his or her different roles.

There may be other restrictions on the validity of a TP. In each case, this restriction will be manifested as a certification rule and enforcement rule. For example, if a TP is valid only during certain hours of the day, then the system must provide a trustworthy clock (an enforcement rule) and the TP must be certified to read the clock properly, and the clock must present the authentic time in a tamperproof way.

Almost all integrity enforcement systems require that a journal be kept of all TP execution to provide data for a later audit. However, no special enforcement rule is needed to implement this facility; the journal can be modelled as another CDI, with an associated TP that only appends to the existing CDI value. The only rule required is:

C4: (Journal Certification) All TPs must be certified to write to an append-only CDI (the journal) all information necessary to permit the nature of the operation to be reconstructed. In some cases it may be desirable to write sufficient data so a previous state can reliably be restored.

There is only one more critical component to this integrity model. Not all data is constrained data. In addition to CDIs, most systems contain data items not covered by the integrity policy. This unprotected data may be manipulated arbitrarily, subject only to discretionary controls. These Unconstrained Data Items, or UDIs, are relevant because they represent the way new information is fed into the system. For example, information typed by a user at the keyboard may be a UDI; it may have been entered or modified arbitrarily. To deal with this class of data, it is necessary to recognise that certain TPs may take UDIs as input values, and may modify or create CDIs based on this information. This implies a certification rule:

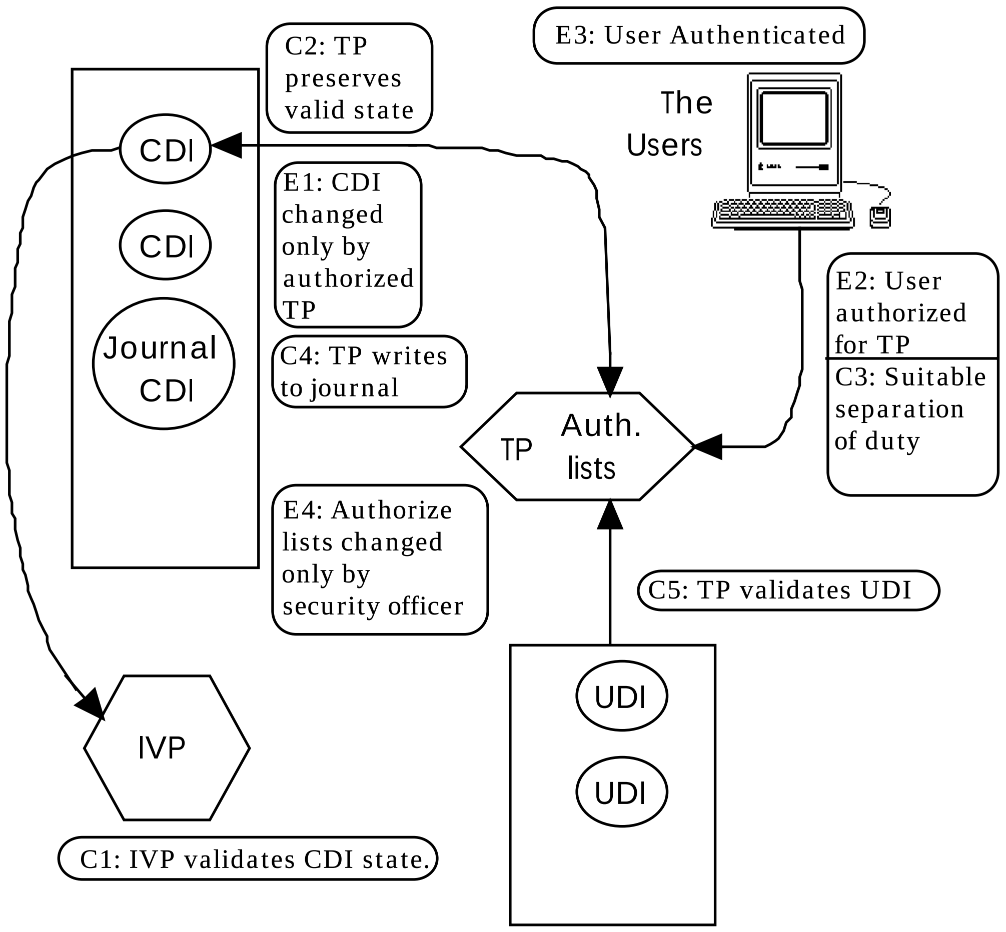
C5: (Transformation Certification) Any TP that takes a UDI as an input value must be certified to perform only valid transformations, or else no transformations, for any possible value of the UDI. The transformation should take the input from a UDI to a CDI, or the UDI is rejected. Typically, this is an edit program.

The only reason for taking such effort to protect the UDI value before transforming it into what amounts to a CDI copy is to make certain that the raw input value from outside the protected envelope gets reliably entered into the journal. In this way a dependable audit can isolate the cause of a suspected integrity problem.

For the Clark-Wilson model to be effective, the various certification rules must not be bypassed. For example, if a user can create and run a new TP without having it certified, the system cannot meet its goals. For this reason, the system must ensure certain additional constraints. Most obviously:

E4: (Initiation) Only the agent permitted to certify entities may change the list of such entities associated with other entities: specifically, the relation associated with a TP. An agent that can certify an entity may not have any execute rights with respect to that entity.

This last rule makes this integrity enforcement mechanism mandatory rather than discretionary. For this structure to work overall, the ability to change permission lists must be coupled to the ability to certify, and not to some other ability, such as the ability to execute a TP. This coupling is the critical feature that ensures that the certification rules govern what actually happens when the system is run.


C2: TP preserves valid state
E3: User Authenticated
The Users
CDI
CDI
Journal CDI
E1: CDI changed only by authorized TP
C4: TP writes to journal
E2: User authorized for TP
C3: Suitable separation of duty
TP
Auth. lists
E4: Authorize lists changed only by security officer
C5: TP validates UDI
IVP
UDI
UDI
C1: IVP validates CDI state.

Together, these nine rules define a system that enforces a consistent integrity policy. The rules are summarised in the figure, which shows the way the rules control the system operation. The figure shows a TP that takes certain CDIs as input and produces new versions of certain CDIs as output. These two sets of CDIs represent two successive valid states of the system. The figure also shows an IVP reading the collected CDIs in the system in order to verify the CDIs validity. Associated with each part of the system are the rules that govern it to ensure integrity.

Central to the Clark-Wilson model is the idea that there are two classes of rules: enforcement rules and certification rules. Enforcement rules correspond to the application-independent security functions, while certification rules permit the application-specific integrity definitions to be incorporated into the model. Enforcement is the process of dependably arranging that the relations that have been certified, whatever they are, actually always hold. It is desirable to minimise certification rules, because the certification process is complex and complicated, may be prone to error, and must be repeated after each program change. In extending the Clark-Wilson model, therefore, an important research goal must be to shift as much of the security burden as possible from certification to enforcement.

For example, a common integrity constraint is that TPs are to be executed in a certain order. In the Clark-Wilson model (and in most systems of today), this idea is usually captured by storing control information in some CDI, and executing explicit program steps in each TP to test this information. The result of this style is that the desired policy is hidden within the program, rather than being stated as an explicit rule that the system can then enforce. Another way to order the TPs is to associate with each an integer, or several integers, and to insist that the integers for TPs about to be executed bear some specific relation to the TP just finished executing. This is not always possible with one associated integer, but with the freedom to use an integer-tuple almost any order can be enforced.

Other variations on the theme of enforcement may exist. Separation of duty might be enforced by analysis of sets of accessible CDIs for each user. Clark and Wilson, in their 1987 paper, stated that they believed that further research on specific aspects of integrity policy might lead to a new generation of tools for integrity control. This has not happened yet.

9.5. Properties of Clark-Wilson

The Clark-Wilson model has two levels of integrity: the lower level UDIs and the higher level CDIs. CDIs are considered higher level because of the control of data modification implied by the TPs and because the data values can be verified using an IVP. In Biba's model, any conversion of a UDI to a CDI could be done only by a security officer or trusted process. Data input is a common system function, and Biba requires that input be done by a trusted component of the TCB. Clearly it can not be done dependably by a mechanism that is outside the security model. Thus, with Biba any input is not a UDI but a trusted CDI right from its inception. The argument in Clark-Wilson that input is a UDI may be based on the idea that the Clark-Wilson model would run on an untrusted platform. This idea is clearly unrealistic.

The Clark-Wilson model permits the security officer to certify the method for integrity upgrade, or in Clark-Wilson terms, those TPs that take UDIs as input values. In this way Clark-Wilson recognises the fundamental role of the TP (which is in a real sense a trusted process). Biba's model lacks any equivalent of rule E1 (CDIs changed only by authorised TP), and thus cannot provide the specific idea of constrained data.

On the other hand, Biba places the onus for the control of new values for data squarely on the authorised user. If the user has appropriate access to data he or she can modify it. Such modification can be a security-relevant event and so be written to the journal of such events that the system must keep. The various programs that the user employs can be evaluated just as rigorously as in Clark-Wilson – in fact Biba is closely associated with the Bell & LaPadula model, which implies that security-significant parts of the system must be modelled and proven safe. So there is a degree of concealed similarity between a Biba implementation and a Clark-Wilson version of the same system.

With Biba there is much finer granularity to the notion of authorisation; this is a result of the flexibility of having the full lattice of security levels available as levels of integrity. In commercial work it is not likely that this flexibility is of much significance.

Following this thought, Lipner[55] has recognised that the category facility of the Bell & LaPadula model can be used to distinguish the general user from the systems programmer or the security officer. Lipner also recognises that data should be manipulated only by certified (production) programs. In attempting to express this in terms of the lattice model, he attaches lists of users to programs and data separately, rather than attaching a list of programs to a data item or a list of data items to a program. His model thus has no direct way to express the Clark-Wilson rule E1, but must rely on the indirect use of categories to accomplish a similar result in a much more unwieldy way. By combining a lattice security model with the Biba integrity model, he more closely approximates the desired model, but still cannot directly express the idea that data may be manipulated only by specified programs (rule E1). On the other hand, the authorised user has control, as he or she should.

The commercial sector would be very interested in a model that would lead to and measure parameters such as: better facilities for end-user authentication; the segregation of duties within the security officer functions, such as the ability to segregate the person who adds and deletes users from those who write a user's rules; the restriction of the security function from user passwords; and the need to provide rule with much better capability to govern the execution of programs and transactions. For the commercial world, these changes would be much more valuable than taking existing applications together with their operating systems and security packages to B or A levels as defined in the Orange Book.

---

[55] Lipner, S.B., "Non-Discretionary Controls for Commercial Applications," Proceedings of the 1982 IEEE Symposium on Security and Privacy, Oakland, CA, April 1982.

INFERENCE CONTROL IN DATABASE SYSTEMS

10. INFERENCE CONTROL IN DATABASE SYSTEMS

Database systems are widely used to store statistical or other such data. The details of the way that the database system operates, the functions and mechanisms it provides, and the way that it organises a search for candidate data records that satisfy some query, are all of little interest to the control of inference[56].

Behind all types of database systems there is a database that is a matrix of records that are identically structured. Let the matrix be called B and let there be N records in it. Each record contains M variables. The set of all N values of each of the variables form the columns of the matrix, called B_j for $1 \le j \le M$. A record is a hopefully consistent collection of data about one individual database entry. In a relational database, for instance, a record is the M-tuple of the relation. Let the field stored at record i, variable j be called x_{ij}. Figure 1 shows the arrangement. The database shown in figure 2 will be used in examples through the major part of these notes.

$$\begin{array}{c|ccccc}
\text{Record} & & & \text{Variable} & & \\
 & B_1 & & B_j & & B_M \\
1 & x_{11} & \cdots & x_{1j} & \cdots & x_{1M} \\
 & \vdots & & \vdots & & \vdots \\
i & x_{i1} & \cdots & x_{ij} & \cdots & x_{iM} \\
 & \vdots & & \vdots & & \vdots \\
N & x_{N1} & \cdots & x_{Nj} & \cdots & x_{NM}
\end{array}$$

Figure 1. A View of a Database

[56] Dorothy E. Denning, "Cryptography and Data Security", Addison-Wesley, Reading, Massachusetts, 1983.

10.2. Knowledge and Inference

There are several terms that must be defined, and assumptions made with respect to them.

Inference: Deducing or concluding some data that is supposed to be unauthorised, from examining the answers to carefully structured queries. The queries can include statistical or summary results.

Internal Knowledge: The data that is stored in the database is the internal knowledge. It is the value (and in some cases the existence) of this internal knowledge that is to be inferred.

External Working Knowledge: The external working knowledge is what can be known about the structure of the database. This includes the number and names of the variables, the range of values they might have, and the number or records in the database. For example there are only two sexes, so this limit may be useful to deduce other data.

External Supplementary Knowledge: The external supplementary knowledge is information not normally released by the database security policy. The security policy for the database might include, for instance, issues of social significance, in addition to the non-disclosure rules.

Figure 2. A Simple View of a Database Called Canada

Province or Territory	Type	Predominant Language	Capital City	Population in 100 000s
Newfoundland	P	English	St. John's	3
Nova Scotia	P	English	Halifax	5
Prince Edward Island	P	English	Charlottetown	1
New Brunswick	P	English & French	Fredericton	6
Quebec	P	French	Quebec	79
Ontario	P	English	Toronto	96
Manitoba	P	English	Winnipeg	8
Saskatchewan	P	English	Regina	12
Alberta	P	English	Edmonton	27
British Columbia	P	English	Victoria	33
Yukon	T	English	Whitehorse	2
MacKenzie	T	English	Yellowknife	2
Keewatin	T	Innuit	Iqualuit	1

10.3. Types of Statistics

Statistics are computed for subgroups of records having attributes in common. The attributes are determined by constructing a proposition. A proposition is a question that admits only the answers true or false. The proposition's value is determined for each record, and those records that satisfy the proposition (result is true) are the result of the statistical query.

In constructing a proposition all the usual arithmetic and relational operators can be used, including the full power of the predicate calculus. In practice, many propositions use only the five operators $\neg$ (not), $\wedge$ (and), $\vee$ (or), =(equal) and $\neq$(not equal). Sometimes + or – are used to indicate set union and set subtraction, and $\wedge$ can also mean set intersection.

The formula describing the proposition is called its characteristic formula C. A characteristic formula C_Q. (Q for query) can be described as, for instance $\left(language \neq english\right) \wedge \left(Population > 500000\right)$ which obviously has the unique result $C_Q = \text{Quebec}$. In these notes, because of the simplicity of the example, we will often abbreviate characteristic formulas when the meaning is obvious, such as in $\left(\neq english\right) \wedge \left(> 500000\right)$

The set of records for which a characteristic formula C is true is called the query set of C. It is convenient to designate both the characteristic formula and its query set by the same symbol, usually C. For example the C of the characteristic formula population ≥ 1000000 is the set

$\left\{\text{Quebec}, \text{Ontario}, \text{Saskatchewan}, \text{Alberta}, \text{British Columbia}\right\}$.
The special characteristic formula *all* is true for every record in the database.

Let variable B_j have v_j possible values, where b_j is a possible value of B_j. Then characteristic formulas of the form $\left(B_1 = b_1\right) \wedge \cdots \wedge \left(B_M = b_M\right)$ can distinguish a known number E of records, where $E = \prod_{j=1}^{M} v_j$. The query set corresponding to a characteristic formula of this form is called an elementary query set because it cannot be specified more precisely. A distinct value b_j is nominated for each variable B_j. There is no way to ask a further question that would subdivide the result. There are E elementary sets in the database, many of which are usually empty. Let γ denote the maximum size of the elementary sets. The quantity γ is the maximum number of records with fields having identical values. In our example $\gamma = 1$. But in practice it is not uncommon for several database records to have corresponding fields with identical values, and then γ is the cardinality of the largest such set.

Statistics are the values returned by a query set *C*.. A general form of a statistic is a finite moment of the form

$$q(C, \theta_1, \cdots, \theta_M) = \sum_{i \in C} x_{i_1}^{\theta_1} \wedge x_{i_2}^{\theta_2} \wedge \cdots \wedge x_{i_M}^{\theta_M} .$$

In this formula, *C* is some characteristic formula with successive conjuncts x_i, each of which is a proposition such as $B = b$ or $B \geq b$. The quantity $x_{i_k}^{\theta_k}$ is the i_k th conduct in *C* raised to the power θ_k. Some examples:

The number or records for which *C* is true:

$$\mathbf{count}(C) = q(C, 0, 0, \cdots, 0)$$
$$\mathbf{count}(\text{english}) = 11$$

With obvious notational inference, the sum of the value of the k th field, which has a numeric value:

$$\mathbf{sum}(C, B_k) = q(C, 0_1, 0_2, \cdots, 1_k, \cdots, 0_M)$$
$$\mathbf{sum}(\text{all}, \text{population}) = 275$$

The average value of some field:

$$\mathbf{avg}(C, B_k) = \frac{\mathbf{sum}(C, B_k)}{\mathbf{count}(C)}$$
$$\mathbf{avg}(\text{all}, \text{population}) = 25$$

The variance:

$$\mathbf{variance}(C, B_k) = \frac{q(C, 0_1, \cdots, 2_k, \cdots, 0_M)}{\mathbf{sum}(C)} - [\mathbf{avg}(C, B_k)]^2$$

The covariance and the correlation are also easily written. This method of describing a statistic is astonishingly general. For descriptive purposes, the form $q(C)$ will be used to mean any statistic of the above form. There are other types of statistics that can not be expressed by the form $q(C)$. For instance, the selection of some specific value from among all the values for B_k, such as the smallest, the median, the largest, the largest that is duplicated, etc., is sometimes wanted.

10.4. Disclosure

Suppose that the data in the column headed Capital City is confidential. The query $\mathbf{CapitalCity}(\text{population} = 9600000)$ is clearly not allowed, because it reveals Toronto. But the query $\mathbf{CapitalCity}(\mathit{all})$ would be acceptable, because it reveals a list of cities, and that is the same as having knowledge of the working external knowledge. Thus, the size of the query set – the number of records that are identified as satisfying the query – is significant when deciding that the result of a query is a disclosure that should be disallowed. All users have some supplementary knowledge, and in combination with the results of a query set of small cardinality, they may be able to infer information that they should not have. One criterion used by some organisations is the n-respondent, k%-dominance rule. This rule disallows revealing the result of a statistic if n or fewer values contribute more than k percent of its total.

Example

Explosives. If a statistic is constructed that includes the explosive power of an atomic bomb and the explosive power of a large number of firecrackers, then for $n \geq 1$ an n-respondent 99.99% rule would disallow revealing the statistic.

A problem arises because the combination of allowed statistics could reveal disallowed information. Suppose that only query sets of size 1 are disallowed. Then it is not hard to arrange that two successive queries reveal information meant to be disallowed. In the example, suppose that the population is considered sensitive data, query sets of size 1 are to be disallowed. Then $\mathbf{sum(}T \wedge \text{english}, \text{population}\mathbf{)}$ has a size 2 and a value 400000. Also $\mathbf{sum(}T, \text{population}\mathbf{)}$ has a size 3 and a value 500000. It takes little wit to realise that $\mathbf{sum(}T \wedge \neg\text{english}, \text{population}\mathbf{)}$ has a value of 100000. From supplementary knowledge, the only predominant language other than English in the Territories is Innuit, and the information is revealed.

This brings up another issue, that of the user's supplementary knowledge. There is no effective way to control the use of supplementary knowledge by a user to infer information that he should not have. This is because there is no way to measure the amount of supplementary knowledge the user might have or does have. As a rule, it is safe to assume that every user remembers information perfectly. A pessimistic policy would need to assume that the user knows all internal knowledge as supplementary knowledge.

The whole enterprise of inferring information from a combination of queries and supplementary knowledge is related to cryptanalysis. For inferring information using queries, there are analogues of the ciphertext-only attacks that are used to try to break encrypted messages. The mechanisms that are used for combined plaintext and ciphertext are analogous to the use of the user's supplementary knowledge.

10.5. Partial Disclosure

The amount of information that is revealed by a query can be measured using information theory. The application of information theory to information release requires a definition of the entropy of information.

10.5.1. Information Entropy

Information theory measures the amount of information in a message by determining the average number of bits needed to encode all possible messages in an optimal encoding. Thus, a message consisting of the Province or Territory field in a database can be encoded in one bit (P or T) if they are both equally likely. Population in 100000s might take 17 bits (assuming no region has more than about 13 billion residents), or for Canada 7 bits.

Formally, the amount of information in a message is its entropy[57]. The entropy of a message X is defined as $H(X) = \sum_{i=1}^{n} p(X_i)\mathbf{log}_2\left(1/p(X_i)\right)$. The sum runs over all values X_i that the message can have. $p(X_i)$ is the probability of the message X having value X_i. As usual, the probabilities must sum to one.

In the example there are two possibilities, P and T, that could be encoded with one bit because the type of each province or territory is permanently assigned as either a P or a T. A similar circumstance arises in a list of people. If the list includes a column describing their sex, that column will use one bit. If one sex is more probable than the other in some circumstance, an optimal encoding would use a fraction of a bit to encode the more probable sex, and rather more than a bit to encode the less probable sex.

The situation with P and T is similar. The probability of P is 10/13 and the probability of P is 3/13. Thus $H(X) = {}^{10}\!/_{13}\,\mathbf{log}_2\,{}^{13}\!/_{10} + {}^{3}\!/_{13}\,\mathbf{log}_2\,{}^{13}\!/_{3} = 0.78$. If P and T were equally probable, $H(X) = {}^{1}\!/_{2}\,\mathbf{log}_2\,{}^{2}\!/_{1} + {}^{1}\!/_{2}\,\mathbf{log}_2\,{}^{2}\!/_{1} = 1$.

Intuitively, each term $\mathbf{log}_2\left(1/p(X_i)\right)$ represents the number of bits needed to encode message X_i with optimal encoding. Following the data in the above example, $\mathbf{log}_2 1.3 = 0.291163$ and $\mathbf{log}_2 13/3 = 2.115477$. If the bit were divisible, a more optimum encoding would use these encodings – a P would be encoded with 0.291163 bits and a T with 2.115477 bits. The field could then be encoded in an average of 0.78 bits. Clearly the bit is not divisible, so that one full bit must be used.

With 1 bit the integers 0 and 1 can be encoded. If there are n messages that are all equally probable, the probability of each message is $p(x_i) = 1/n$ and the entropy is $H(X) = n\left[{}^{1}\!/_{n}\,\mathbf{log}_2\, n\right] = \mathbf{log}_2\, n$. This shows that if n messages are equally probable then $\mathbf{log}_2\, n$ bits are needed to encode the messages so that each has a distinct encoding. If $n = 1$ then $p(X_1) = 1$ and $H(X) = \mathbf{log}_2 1 = 0$ — there is no information because there is no choice of message. Reminiscent of Saturday morning television.

10.5.2. Disclosure

Let K denote a user's supplementary external knowledge. Let R denote a set of statistics released to a user. A Statistical Disclosure about some

[57] Entropy also is used in thermodynamics as a measure the unavailability of thermal energy for conversion into mechanical energy. It also appears elsewhere as a measure of the disorganisation of the universe. Generally, that which is organised can be transformed into another form, while that which is disorganised is unavailable. The use in information theory has only a tenuous connection with the other uses.

statistic q occurs whenever the entropy of the combination of K and R is less than the entropy of K alone. This is expressed as $H_{K,R}(q) < H_K(q)$.

If the disclosure uses the supplementary knowledge associated with a user, a <u>compromise</u> has taken place. The degree of compromise is measured by the reduction in entropy $H_K(q) - H_{K,R}(q)$. <u>Complete compromise</u> occurs when $H_K(q) - H_{K,R}(q) = H_K(q)$ in which evidently $H_{K,R}(q) = 0$ because there is no residual information left after the complete compromise.

If disclosure occurs without the use of supplementary knowledge, it is called <u>resultant disclosure</u>. If it is assumed that the totality of everything the user *might* know would take an infinite number of bits to encode it, then any query that returns a result can make a resultant disclosure. However, given that some resultant disclosure has been tolerated, any further resultant disclosure can be measured. Notice that entropy always decreases unless the model includes a way to simulate the possibility that the user will forget information. This is not done in practice, because it does not represent a pessimistic approach to security.

If a compromise is not complete, it is a <u>Partial Compromise</u>. In this case the information *q* is not determined exactly. There are at least three ways that this might occur. Assume that there is a lower bound and an upper bound on the value that *q* can be expected to have, so that $GLB \leq q \leq LUB$. This is equivalent to a user being aware of the working knowledge of the database.

(1) *Narrowing bounds*. If the query results in either *GLB* increasing or *LUB* decreasing, a partial compromise has occurred. When $GLB = LUB$ a complete compromise has occurred.

(2) *Negative*. It is revealed that $q \neq q'$ for some q' in $GLB \leq q' \leq LUB$. A sufficient number of these results, together with a working knowledge of the database, can gradually result in a complete compromise, or if the values of q' are clustered near the *GLB* or the *LUB*, a partial compromise of type (1) immediately above happens.

(3) *Probabilistic*. If some new bounds GLB' or LUB' can be established, with probabilities that they are the actual bounds, then the interval $[GLB', LUB']$ is called the confidence interval that $GLB' \leq q \leq LUB'$, and $p(GLB' \leq q \leq LUB')$ is the confidence level corresponding to the confidence interval.

Suppose an estimate q' of the value *q* of a random variable is made, with known standard deviation[58] $\sigma_{q'}$. Then the 99% confidence interval is given by $p(q - 2.575\sigma_{q'} \leq q' \leq q + 2.575\sigma_{q'}) = 0.99$. It is 99% certain that

[58] The standard deviation is the square root of the variance developed in section 3.

q' lies in the interval given in this equation. Conditions for various levels of confidence are given by:

$$p(q - 2.575\,\sigma_{q'} \le q' \le q + 2.575\,\sigma_{q'}) = 0.99$$
$$p(q - 1.645\,\sigma_{q'} \le q' \le q + 1.645\sigma_{q'}) = 0.90$$
$$p(q - 1.960\,\sigma_{q'} \le q' \le q + 1.960\,\sigma_{q'}) = 0.95$$

The general equation that these are specific cases of is beyond the scope of this course[59]. The numbers assume that there are a large number of measurements that contribute to the estimate q'. If a small number of measurements are involved the interval widens slowly.

10.5.3. Protection

There is no perfect scheme of database security. It is only possible to attempt to prevent unauthorised users from inferring close approximations of sensitive values. It is said that a sensitive statistic q is protected with confidence $\overline{p}$ within interval length $L = LUB - GLB$ and $p(q \in L) \ge \overline{p}$. Working with this arrangement is in practice quite difficult. A number of approximations, many of doubtful provenance, seem to be necessary to apply confidence intervals to security.

The number of statistics needed to reduce the uncertainty about a statistic q to an unacceptably low figure can be used to represent the difficulty in compromising q. If a very large number of statistics are required to infer q with a high probability, then q might be considered to be fairly well protected. This situation is analogous to attacking a cipher that requires a large amount of cipher text to give a high probability that the cipher can be broken. In this situation there will be a corresponding high processing cost – perhaps beyond the range of that which it is practical to compute.

10.6. Control Mechanisms

There are a number of mechanisms that can be built into the database system to make the inference of sensitive information difficult. The most obvious is to control the type of statistics that are released.

10.6.1. Macrostatistics

Macrostatistics are collections of related statistics, usually supplied as a summary view of the database in which each row of the macrostatistics represents a (fairly large) number of rows of the actual database, and where the columns are often combined as well. This can make the inference of individual data very difficult, even if supplementary knowledge is deployed. Often, additional artificial records are added to the database to conceal the number of microstatistics participating in a macrostatistic.

[59] See Alberto Leon-Garcia, "Probability and Random Processes for Electrical Engineering", Addison Wesley, 1989, Section 5.4, pages 291-296.

The whole area is applicable only to situations like the output of census bureaux, where there are likely to be a huge number of records that are all very similar.

10.6.2. Query Set Size Control

Modern general purpose database systems include language features to help the user formulate his questions, and to help reduce the effort required to use a database. It is natural that these systems try to enforce a security policy. The database needs a mechanism that will both prevent the result $q(C)$ of a query based on the characteristic formula C from being composed of too few records, or from excluding too few records from the reply. If either of these are allowed to happen it is too easy to deduce another query set that will result in an inference that reveals too much information. See section 5.2 for an introduction to the measurement of how much is too much.

Let the query set size of the statistic $q(C)$ be k. In general, a statistic $q(C)$ is permitted only if $n \leq k \leq N - n$. The quantity $n > 0$ is a parameter of the security policy as it applies to the database. If $n \geq N/2$ no statistic will ever be released, because $n \leq k \leq N - n$ can not be true. Consequently, $n < N/2$. In practice, n must be quite a bit less that $N/2$ if the database system is to be usable. Otherwise, queries will be required to always contain close to half the records, and they are too hard to construct and don't represent a usable tool.

As a special case, $q(all)$ is always permitted. This is because $q(all) = q(C) + q(\neg C)$ and it will be easy to find some statistic S with query set size σ such that $n \leq \sigma \leq N - n$. Note that the query set size of $\neg S$ is $N - \sigma$, and $n \leq N - \sigma \leq N - n$ as well, so $q(all)$ can be computed trivially.

10.6.2. Tracker Attacks

A tracker is a pair of related characteristic formulas that can be used to infer information even in the presence of query set size control. Let user I be the one person for whom $q(C)$ is true. It is desired to find out if characteristic formula D is also true for user I. If C can be divided into two parts then the user may be able to compute $\mathbf{count}(C \wedge D)$ from use of the parts.

A tracker attack starts with finding a decomposition of C into $C_1 \wedge C_2$, such that $\mathbf{count}(C_1)$ and $\mathbf{count}(C_1 \wedge \neg C_2)$ both satisfy the query set size control. The query set size of $\mathbf{count}(C_1 \wedge \neg C_2)$ can not be larger than the query set size of $\mathbf{count}(C_1)$ because $\mathbf{count}(C_1)$ clearly identifies more records than $\mathbf{count}(C_1 \wedge \neg C_2)$ identifies.

$$n \leq \mathbf{count}(C_1 \wedge \neg C_2) \leq \mathbf{count}(C_1) \leq N - n$$

The pair of formulas C_1 and $C_1 \wedge \neg C_2$ are called the tracker. The compromise is as follows.

$C_1 \wedge C_2$ uniquely determines I because of the way C_1 and C_2 have been built. Let $T = C_1 \wedge \neg C_2$. The statistics **count**(T) and **count**$(T + C_1 \wedge D)$, both of which are permitted by the query set size control, can be used to compute

$$\mathbf{count}(C \wedge D) = \mathbf{count}(T + C_1 \wedge D) - \mathbf{count}(T).$$

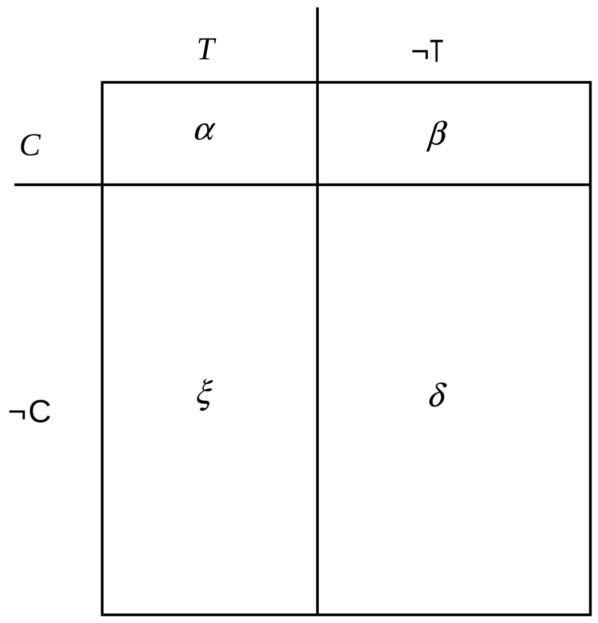
10.6.3. General Trackers

A general tracker is a characteristic formula T with query set size τ such that $2n \le \tau \le N - 2n$. The statistic $q(T)$ is always answerable because τ is smaller than the query set size control k. It is evident that $n \le N/4$ or the (single) general tracker can not exist. It has been shown that if the maximum size of the elementary set $\gamma \le N - 4n$ then the database must have at least one general tracker.

Let the query set size of $q(C)$ be k. The compromise works as follows. It is obvious that $q(all) = q(T) + q(\neg T)$. If $k < n$ then compute $q(C)$ from $q(C) = q(C + T) + q(C + \neg T) - q(all)$, while if $k > N - n$ compute $q(C)$ from $q(C) = 2q(all) - q(\neg C + T) + q(\neg C + \neg T)$.

For example, in the following figure the Greek letters represent statistics that are true in their area of the plane. The statistics α are true in the upper left corner of the plane, and so on. It is desired to deduce $q(C)$ using the tracker T as a vehicle to avoid query set size control. The assumption is that the number of statistics in $q(C)$ would fail query set size control; the figure is drawn to crudely imply this state of affairs. The way that the tracker works is shown after the figure.

$$\begin{aligned} q(C) &= q(C+T) + q(C+\neg T) - q(all) \\ &= (\alpha+\beta+\xi) + (\alpha+\beta+\delta) - (\alpha+\beta+\xi+\delta) \\ &= \alpha+\beta \end{aligned}$$

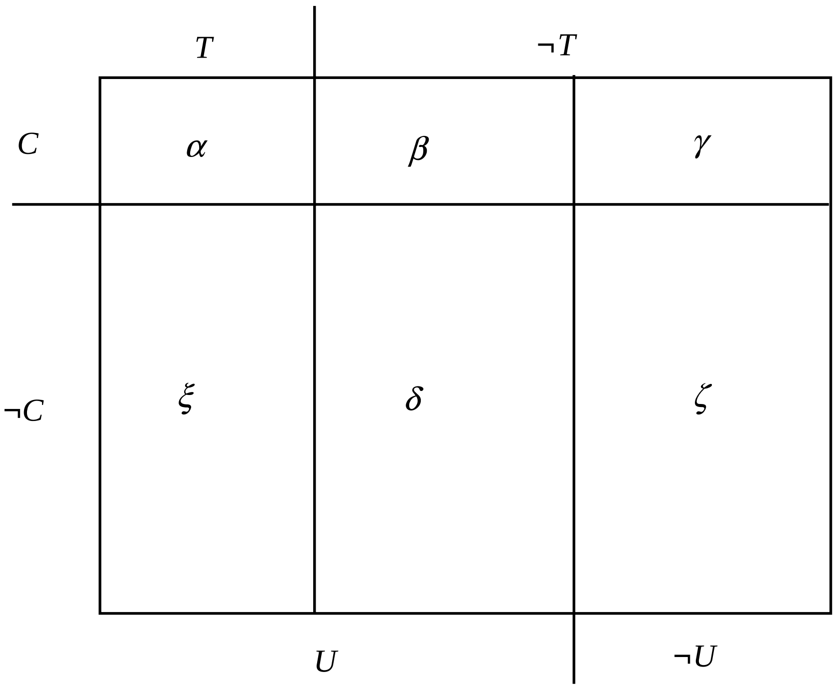


A double tracker can be built from two characteristic formulas, T with query set size τ and U with query set size υ such that $T \subseteq U$ and $n \le \tau \le N - 2n$ and $2n \le \upsilon \le N - n$. The compromise proceeds as follows. If $k < n$, $q(C) = q(U) + q(C + T) - q(T) - q(U \wedge (\neg(C \wedge T)))$ while if $k > N - n$, $q(C) = q(\neg U) - q(\neg C + T) + q(T) - q(U \wedge (\neg(\neg C \wedge T)))$.

For a double tracker, $n < N/3$, a slight improvement over the single variety. Obviously higher levels of tracker can be conceived, although it gets increasingly difficult to find them.

$$\begin{aligned} q(C) &= q(U) + q(C + T) - q(T) - q(U \wedge (\neg(C \wedge T))) \\ &= (\alpha + \beta + \xi + \delta) + (\alpha + \beta + \gamma + \xi) - (\alpha + \xi) - (\beta + \xi + \zeta) \\ &= \alpha + \beta + \gamma \end{aligned}$$

The quantity $U \wedge (\neg(C \wedge T))$ implies those areas of the figure in which U is true and C and T are not true.



10.6.3. Linear Equations and Other Attacks

Let $\mathbf{Q} = q_1, q_2, \cdots, q_m$ be a set of statistics, all of which have been released to a potential attacker. A linear system $\mathbf{HX} = \mathbf{Q}$ can be solved, in which $\mathbf{H}$ is a matrix such $\mathbf{H}_{ij} = 1$ if $q_j \in q(C_i)$ and $\mathbf{H}_{ij} = 0$ otherwise. Solution of this linear system, together with the working knowledge, can now supply other statistics that can be inferred from the set $\mathbf{Q}$. Trackers are a specialised form of this linear equation attack. The members of q_k can be formed from operations, such as intersection or union, on available statistics. For instance, using the usual identities, it is easy to show that $q(\neg C \wedge D) = q(D) - q(C \wedge D)$. Such identities can be used to select the members of $\mathbf{Q}$ so that the linear system is specialised to solve for the desired statistic.

It has been pointed out that one type of query asks for a value for B_k, such as the smallest, the median, the largest, the largest that is duplicated, etc. All these can be the basis of an attack, demonstrating that query set size control is not a good enough control.

For example, using the median, an attack can be constructed as follows. Let the following be statistics:

median(PorT, population**)** = 6

median(PorT using other than English, population**)** = 6

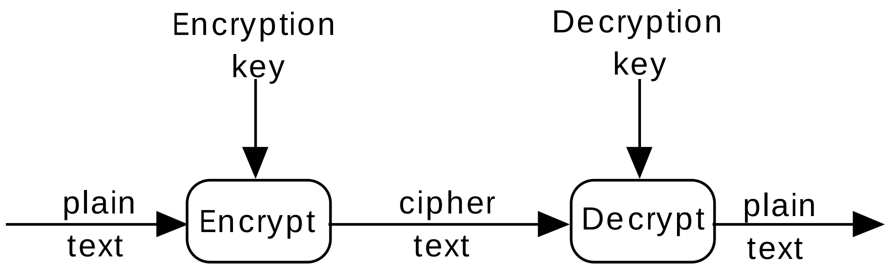
The example is trite, but if it is known from supplementary knowledge that New Brunswick is the smallest province in which French is an official language, and there are two, and that the Keewatin territory uses Innuit and has a tiny population, then it is straightforward to deduce that the population of New Brunswick is 6. The technique depends on the attacker's ability to devise the two (or more, if you want to get complicated) median questions.
Another attack is based on repeated insertion of a record, asking a query, deleting the record, and asking the query again.

If a database is dynamic in the sense that users can insert and delete entries in it, a careful monitoring of the database can show the values that are being inserted and deleted. It may be possible to deduce some unaltered data from a contemplation of summary values, if the entries are cleverly chosen, but it is not thought that this is generally the case.

ELEMENTARY CRYPTOGRAPHY

11. ELEMENTARY CRYPTOGRAPHY

Cryptography is the process of encrypting a plaintext with an algorithm the action of which is moderated by an encryption key, and the corresponding process of decryption. The encryption–decryption algorithm is generally not secret. It is the modern practice that the algorithm should be available to any observer, but that knowledge of the algorithm and a supply of messages should not in practice enable an intruder to deduce the key.



Cryptanalysis is the process of deducing the message by either deducing the key and running the (known) algorithm, or by knowing the algorithm and ciphertext and recovering the plaintext. There are several types of attacks:

ciphertext-only:
: the plaintext is recovered from knowledge of a sufficient number of ciphertext messages.

known plaintext:
: the plaintexts corresponding to some known ciphertexts is available. This is used to try to deduce the key. In some cases the algorithm may not be known, and it is then used to attempt to deduce both the algorithm and the key.

11.1. A Simplistic Introduction to Cryptography

There are two mechanisms used in encryption and in the corresponding decryption. Both mechanisms use the key to govern their actions. Some techniques use a sequential combination of both mechanisms.

Transposition:

In a transposition, the bits, letters, words, ... that are the components of a message are rearranged in a different order. The recipient must know how to undo the transposition and recover the original message. The rearrangement is determined by the key in some unique way, so that if both the sender and the receiver know the key they can determine how to transpose or its reverse.

Substitution:

A substitution encryption replaces the components of a message (bits, letters, words, ...) with a substitute. The recipient must know how to undo the substitution and recover the original message. The substitution is determined by the key in some unique way, so that if both the sender and the receiver know the key they can deduce the correct output from their input.

In the following discussion the following text :

```
IN THE BEGINNING GOD MADE THE HEAVEN AND THE EARTH,
```

from the opening sentence of a well-known book, will be used throughout unless an example with special properties is required. The elements of the message to be transposed or substituted for are the letters in the message. For convenience, only upper case letters will be used.

11.2. Transposition Ciphers

Transposition ciphers rearrange the letters according to some regular pattern. For instance, the letters of the message could be written in a matrix by rows, and then read out of the message by columns or with the entries in a row jumbled. In order to show a blank it has been replaced with the symbol ø. Normally in a transposition cipher, the blanks are left out and the recipient is left to conclude where they should be. Because natural language is so redundant this is rarely a problem. For example:

```
INTHEBEGINNINGGODMADETHEHEAVENANDTHEEARTH
```

is not hard to read.

Consider the matrix

I	N	Ø	T	H	E	Ø
B	E	G	I	N	N	I
N	G	Ø	G	O	D	Ø
M	A	D	E	Ø	T	H
E	Ø	H	E	A	V	E
N	Ø	A	N	D	Ø	T
H	E	Ø	E	A	R	T
H	Ø	Ø	Ø	Ø	Ø	Ø

as a transposition matrix. In this arrangement, the plaintext has been written into a matrix with seven columns, and enough rows to hold it. One encryption is obtained by reading down the columns from left-to-right. This reading gives the ciphertext

```
IBNMENHHNEGAØØEØØGØDHAØØTIGEENEØHNOØADAØENDTVØRØØIØHETTØ.
```

as an encryption of the original message.

This ciphertext is easy to decode. An elementary analysis will notice that there are 56 letters. It is natural to try a 7×8 arrangement of the letters, and the message is immediately evident.

There is a more general form of such encryptions.

Let c be the number of columns in the matrix, and let p be a permutation of the digits $1, 2, 3, \cdots, c$. If $c = 7$ then 1234567 is a permutation, and so is 3721654. Such transpositions are countable. Each permutation can be assigned a unique numerical identifier in the sequence $1, 2, 3, \cdots, c!$. In this way, the transposition can be represented by a number. The pair $\langle c, p \rangle$ can define two different transpositions.

COLUMNWISE: The c specifies how many columns there are, and the p specifies the order (permutation) that they are to be read from top to bottom. The example above corresponds to $\langle 7,1234567 \rangle$. The permutation $\langle 7,3721654 \rangle$ corresponds to the ciphertext

```
øGøDHAøøøIøHETTøNEGAøøEøIBNMENHHENDTVøRøHNOøADAøTIGEENEø.
```

ROWWISE: The c specifies how many columns there are, and the p specifies the order (permutation) that the columns are to be read one-by-one from left to right. The permutation $\langle 7,1234567 \rangle$ returns the original text, while $\langle 7,3721654 \rangle$ results in

```
øøNIEHTGIEBNNIøøGNDOGDHAMTøEHEøEVAEATøNøNDøTEHRAEøøøHøøø.
```

The latter method is sometimes preferable because the matrix can be dealt with a row at a time, with no necessity to write the whole thing down before producing the output.

To decipher a transposition message, the matrix is reconstructed. This is elementary if $\langle c, p \rangle$ are known. The message is written back into a matrix with the proper number of rows, according to the order that it was read out of the matrix. The original message is then evident in the matrix.

Transposition ciphers have the same relative frequencies of the appearance of the letters as does the natural language. This, used together with frequency-of-appearance tables of the common digrams (2-letter sequences) and trigrams (3-letter sequences) make transposition ciphers of little primary value. They are used, however, in conjunction with substitution ciphers to concoct some very secure mechanisms.

11.3. Substitution Ciphers

There are four types of substitution ciphers.

SIMPLE: Simple substitution ciphers replace on a one-for-one basis each character of a plaintext with the corresponding character of a ciphertext. This is done throughout the message.

HOMOPHONIC: Homophonic substitution ciphers replace each character of a plaintext with one of the corresponding characters of a ciphertext. The mapping is, in general, one plaintext character to many ciphertext characters.

POLYALPHABETIC: Polyalphabetic substitution ciphers replace each character of a plaintext with the corresponding character of a ciphertext – a simple substitution. They change the substitution frequently, usually after each substitution. Thus, the first letter might be encrypted with one simple substitution, the second with another, and so on.

POLYGRAM: Polygram substitution ciphers are more general than the other ciphers. In general, they replace blocks of characters of a plaintext with the corresponding block of a ciphertext.

11.3.1. Simple Substitution Ciphers

The simplest simple substitution cipher is called a keyword mixed alphabet. A ciphertext is written down, letter by letter, beneath the plaintext, with duplicate letters in the ciphertext omitted. As an aid to remembering the ciphertext, it is common to use a simple phrase; for example `GENESIS CHAPTER ONE VERSE ONE`. The ciphertext is then completed by writing down the rest of the letters (the ones that do not appear in the phrase) in some arbitrary but known way. For example:

A	B	C	D	E	F	G	H	I	J	K	L	M	N
G	E	N	S	I	ø	C	H	A	P	T	R	O	V

O	P	Q	R	S	T	U	V	W	X	Y	Z	ø
B	D	F	J	K	L	M	Q	U	W	X	Y	Z

Our example encrypts as
`AVZLHIZEICAVVAVCZCBSZOGSIZLHIZHIGQIVZGVSZLHIIGJLH`.

Ciphers that just shift the alphabet to the right k positions modulo the cardinality of the alphabet are known as shifted alphabets. If the cardinality of the alphabet is n, then for $0 \leq j \leq n-1$, $\text{ciphertext}_j = \text{plaintext}_{(j+k)\,\mathbf{mod}\,n}$. The case where $k = 3$ is known as a Caesar Cipher because Julius Caesar is supposed to have used it that way. Augustus Caesar favoured $k = 4$.

If k and n are relatively prime, then $\text{ciphertext}_j = \text{plaintext}_{(j \times k)\,\mathbf{mod}\,n}$ will produce a workable cipher alphabet, because, as j goes from 0 to n-1, the subscript on the right-hand side will become a permutation of the integers from 0 to n-1. More elaborate functions can also be used, for instance polynomials, but they are not better than the elementary methods because there is a strong mathematical relationship between the elements of the cipher alphabet.

Lots of other symbols have been used for the cipher alphabet. Among others, there have been simple substitution ciphers using mystical diagrams[60] and musical notes[61].

Given enough text, these techniques are all fairly easy to decipher with a ciphertext-only attack. The use of single-letter frequency-of-use distributions is sufficient. If the attacker has some knowledge of the possible contents of the message then the process is helped considerably. For example, the message might begin with the word CONFIDENTIAL. If this proves to be true then 10 of the 26 ciphertext letters are immediately known. Knowledge of that many letters usually leads to very rapid deduction of the rest by simple inspection.

11.3.2. Homophonic Substitution Ciphers

The idea of a homophonic substitution cipher is to have several or many ciphertext equivalents of each plaintext letter. Each ciphertext symbol is, however, the equivalent of only one plaintext letter. Letters that are used frequently will have more equivalents than those that are seldom used. The letter frequencies have been evaluated as follows (for English):

Relative frequency of English characters from a large text.			
Letter	**Frequency**	**Letter**	**Frequency**
ø	12.17	N	5.44
A	6.09	O	6.00
B	1.05	P	1.95
C	2.84	Q	0.24
D	2.92	R	4.95
E	11.36	S	5.68
F	1.79	T	8.03
G	1.38	U	2.43
H	3.41	V	0.97
I	5.44	W	1.38
J	0.24	X	0.24
K	0.41	Y	1.30
L	2.92	Z	0.03
M	2.76	others	6.57

Suppose that homophones are developed with the relative frequencies of occurrence suggested in the table. Let each homophone be a three digit number between 000 and 999, chosen for instance at random. There would be 122 3-digit numbers representing ø, 114 representing E, and so on. The 65 three digit codes for others are used to represent punctuation, or are left unused. The construction of a suitable example is easy but tedious.

[60] For example, Conan-Doyle used some semaphore symbols in a Sherlock Holmes story.

[61] J.S. Bach encrypted his name into several of his works.

In practice three digit homophones would rarely be used. Two digit ones are sufficient, except for extremely long messages. At the extreme, if there are enough homophones so that they need not be reused in any reasonable number of messages, this technique is extremely difficult to penetrate. Its disadvantage is that the message grows in length, because the number of characters needed to encode a homophone can be more than the number needed to encode a letter.

There are a number of ways to chose homophones other than that shown above. For instance, a somewhat infamous cipher known as B2 (Beale number 2) uses the United States Declaration of Independence (USDoI) to arrive at a homophone. The words of the USDoI are numbered in sequence from 1 up. Beale enciphered each letter of the plaintext message by substituting the word-number of some word in the USDoI that began with that letter. Thus word 6 is the word *human*, so the homophone 006 could be used to represent an H in the plaintext. Mechanisms like this are popular among people who want to travel widely and be able to access their homophonic substitution table. Every good library could be expected to have a copy of some particular printing of a defined and reasonably popular book of reference, and so that could be used as a way to build homophones without actually carrying a text while travelling buy specifying the page number to begin with. The method is well-known among the espionage community. All that need be remembered by the encypherer is the name of the book.

11.3.3. Polyalphabetic Substitution Ciphers

Simple substitution ciphers use a single mapping from plaintext to ciphertext. This is a one-to-one mapping. Homophonic substitutions use a one-to-many mapping. Polyalphabetic substitution ciphers conceal the mapping by using several successive simple substitutions and varying each of the simple substitutions according to some planned pattern, letter by letter.

Using successive simple substitutions is a just a linear combination of simple substitutions; without any additional variation it is identical to some (other) simple substitution. The method gets its strength from changing one or more of the successive simple substitution ciphers at each plaintext letter.

11.3.4. Vigenère Cipher

A representative technique, used by Vigenère in the 16th century, employs a table like that given below. The encipherment proceeds as follows. Some key is chosen, a phrase such as

```
GENESIS CHAPTER ONE VERSE ONE.
```

The message is written down, and the key is written under it.

```
IN THE BEGINNING GOD MADE THE HEAVEN AND THE EARTH
GENESIS CHAPTER ONE VERSE ONEGENESIS CHAPTER ONE V
```

A Vigenère Table	
	plaintext
	ABCDEFGHIJKLMNOPQRSTUVWXYZø
Key	**ciphertext**
A	ABCDEFGHIJKLMNOPQRSTUVWXYZø
B	BCDEFGHIJKLMNOPQRSTUVWXYZøA
C	CDEFGHIJKLMNOPQRSTUVWXYZøAB
D	DEFGHIJKLMNOPQRSTUVWXYZøABC
E	EFGHIJKLMNOPQRSTUVWXYZøABCD
F	FGHIJKLMNOPQRSTUVWXYZøABCDE
G	GHIJKLMNOPQRSTUVWXYZøABCDEF
H	HIJKLMNOPQRSTUVWXYZøABCDEFG
I	IJKLMNOPQRSTUVWXYZøABCDEFGH
J	JKLMNOPQRSTUVWXYZøABCDEFGHI
K	KLMNOPQRSTUVWXYZøABCDEFGHIJ
L	LMNOPQRSTUVWXYZøABCDEFGHIJK
M	MNOPQRSTUVWXYZøABCDEFGHIJKL
N	NOPQRSTUVWXYZøABCDEFGHIJKLM
O	OPQRSTUVWXYZøABCDEFGHIJKLMN
P	PQRSTUVWXYZøABCDEFGHIJKLMNO
Q	QRSTUVWXYZøABCDEFGHIJKLMNOP
R	RSTUVWXYZøABCDEFGHIJKLMNOPQ
S	STUVWXYZøABCDEFGHIJKLMNOPQR
T	TUVWXYZøABCDEFGHIJKLMNOPQRS
U	UVWXYZøABCDEFGHIJKLMNOPQRST
V	VWXYZøABCDEFGHIJKLMNOPQRSTU
W	WXYZøABCDEFGHIJKLMNOPQRSTUV
X	XYZøABCDEFGHIJKLMNOPQRSTUVW
Y	YZøABCDEFGHIJKLMNOPQRSTUVWX
Z	ZøABCDEFGHIJKLMNOPQRSTUVWXY
ø	øABCDEFGHIJKLMNOPQRSTUVWXYø

Letter by letter, the key letter selects the row of the table above, and the message letter selects the column. In this way, the ciphered message becomes

ORMXZ … .

11.3.5. Rotor Machines

Rotor machines are just a mechanical way to mechanise the Vigenère cipher, with the possibility of some considerably more elaborate enciphering.

In principle, they operate as follows:

- There is a collection of co-axial rotors (or the electronic equivalent).
- Each rotor has two sides, and 26 (or some such number) contacts around the circumference of each side.
- The segments on the left side of any rotor are connected to the segments on its right side such that no back-to-back segments are interconnected. Each of the rotors that is used has a different connectivity.
- The segments on the right side of rotor j connect to the segments on the left side of rotor $j+1$.

- Signals are fed into the mechanism at the left side of the leftmost rotor. Since the rotor has 26 contacts, there is a one-to-one relationship between a contact and the letter being enciphered. Encrypting an A is done by feeding an electrical current into the A contact at the left side of the leftmost rotor. The signal proceeds through the rotors, moving from a contact on the leftmost rotor to a different contact on its right side, and thence to the next rotor in the chain.
- The right end rotor *m* is special, in that it has contacts only on its left side. The contacts that would be on its right side are connected directly to different contacts that feed the signal back to a new contact on its left side. The signal proceeds back to the left through a chain of contacts until it emerges at the left rotor.
- When the signal emerges at the left rotor the letter corresponding to the contact that it appears at is the encrypted version of the letter being enciphered.
- Each of the rotors has a mechanism to rotate it on its axis by one or more segments, dependant on the motion of neighbouring rotors. These mechanisms are adjustable with insertable pins and a drive pickup from the rotor to the left. Typically the leftmost rotor jogs one position for each encrypted character, the second jogs one position as a function of the rotation of the leftmost rotor, the rotation of the third from the left depends on the rotation of the second from the left in a different way, and so on.

For the purpose of description, suppose that there are 3 movable rotors, called the leftmost, the middle, and the rightmost. In practice there are always at least three rotors, but rarely more than five. To use the machine, the rotors are put in a starting position. Around their edge they have printed the alphabet. The rotors are moved by hand until each displays the proper edge letter against an index used for that purpose. Also, the pins that govern the progression of the rotation of the rotors are set up according to prior agreement. This set-up operation is in reality a method of setting the key into the machine.

There is a keyboard that causes an electrical signal to appear at one of 26 permanent segments that are in contact with the left side of the leftmost rotor when the corresponding key is activated. Typing a key causes an electrical signal to appear on exactly one of the input contacts, and this signal will move through the connections in the leftmost rotor to one of the segments in the right side of the leftmost rotor. This rotor does, in effect, one simple substitution encipherment.

The right side of the leftmost rotor is connected to the left side of the middle rotor. The middle rotor does another simple encipherment.

The right side of the middle rotor is connected to the left side of the rightmost rotor. The rightmost rotor does another simple encipherment. The signal connects through the end rotor, entering at one contact and leaving at another. It flows back through the rightmost, middle, and leftmost rotors, not interfering with the incoming signal because of the way the rotors are wired. The signal eventually emerges from one of the 25 other segments at the left side of the left rotor. The segment that it appears at indicates the ciphertext letter corresponding to the input plaintext letter.

When the output appears, the leftmost rotor is jogged one contact position around its axis. This may result in the rotation of the other rotors, depending on the pin placement on the rotors. Thus, the next letter typed will use a different simple substitution cipher on the left rotor, and maybe a different cipher on one or more of the other rotors.

A little thought will reveal the fact that if the rotors start in the proper position, the ciphertext can be fed into the rotor machine at the connection it came out of when it was encrypted, the plaintext will appear where it was input. It is a self-inverting mechanism.

The rotor machine does generate apparently long key sequences because of the constant movement of the rotors, and might be expected to be correspondingly secure. However, there are two problems. First, no letter can encrypt as itself. Second, most of the rotor motion will be embodied in the leftmost rotor, the one that moves once per letter. And that fact, as well as the limitations of the pin-placements, means that if a body of ciphertext is available, it is not too difficult to deduce some of the wiring of the rotors. Also, the placement of some of the pins can be deduced.

Once some wiring is known, the rest becomes easier to find. Rotor machines are not sufficiently secure for use today, because computers can try millions of possible arrangements in minutes. However, up until the 1970s they were extensively used, with a 4 or 5 rotor machine the norm. They were reasonably secure provided the key distribution was secure, and the rotor wiring was physically protected.

The breaking of the rotor machines used by the German navy in the Second World War was first done by deducing the key. This was done by the Polish before hostilities broke out in 1939. The Germans distributed the key (the initial rotor setting) as a duplicated group of characters that were prepended to each message – not very clever. All that was needed was to steal a rotor machine. Polish intelligence delivered two to the British in 1939.

Later in the war the key distribution was more sophisticated. However, since there were only a limited number of different rotors shipped with each machine, and all machines had to have the same rotor set, once some information about one machine was deduced, it applied to all. The starting position of the rotors was changed daily. Once the wiring of the rotors was uncovered, it was a fairly simple matter to deduce the starting position of the rotors for some given day.

Even if the rotors were changed around, and the pins adjusted, the availability of the wiring of the rotors meant that there were a small enough number of combinations of settings that only a small amount of ciphertext was required to break the key for that day. Sometimes a single message of some hundreds of characters was enough.

11.3.6. Polygram Substitution Ciphers

The idea of a polygram substitution cipher is to encipher blocks of letters, and so lessen the effect of any statistical analysis that might be attempted and that is based on letter frequencies.

11.3.7. The Playfair Cipher

The Playfair cipher is a polygram substitution cipher. It was used by the British in World War I. It depended on a key not dissimilar to that used in the keyword mixed alphabet. Some key is chosen, a phrase such as GENESIS CHAPTER ONE VERSE ONE. The British eliminated J from their alphabet. Also, blanks were ignored. Using our example, the key phrase is written in a 5×5 square, filling in the square with the letters that do not appear in the key phrase in their alphabetic order.

G	E	N	S	I
C	H	A	P	T
R	O	V	B	D
F	K	L	M	Q
U	W	X	Y	Z

This table is used in the cipher as follows. The plaintext is written down according to the following rules:

- All instances of J are deleted or replaced.
- All blanks are deleted. In order to make the message unambiguous, it was necessary to do this carefully. Blanks were sometimes written as X to overcome problems.
- All repeated letters (TT in aTTack, or GG in biG Gun) have a filler put between them – usually X was used for this filler, but the other characters that appear infrequently were also used for this purpose.
- If the message had an odd number of characters, it is padded, usually with an X, so it has an even number of characters.

The Playfair key table is quite easy to use, particularly manually. Let an arbitrary pair of letters of the (adjusted) message be called mn and let the corresponding ciphers be $\alpha\beta$. In the message, these will be the 1st and 2nd, or the 3rd and 4th, or in general the $2i^{\text{th}}$ and $2i+1^{\text{st}}$ message letters, for $i \geq 0$. The mn is encrypted according to the following rules:

- If m and n are in the same row, α is the letter in that row to the right of m, and β is the letter in that row to the right of n. The leftmost letter in the row is considered to be to the right of the rightmost letter in the row (wraparound).
- If m and n are in the same column, α is the letter in that column below m, and β is the letter in that column below n. The top letter in the column is considered to be below the bottom letter in the column (wraparound).

- If m and n are in different rows and columns, α and β are the letters at the other two corners of the rectangle having m and n at its corners, α in m's row, and β in n's row.

Again using our favourite example, the message after adjustment and enciphering becomes as shown below.

```
INTHEBEGINXNINGXGODMADETHEHEAVENANDTHEXEARTH
GSCASONEGSNAGSNUERBQTVIHOHOHVLNSVAQDOHWNCVCA
```

This cipher is stronger than it looks. A modern version might use a larger rectangular array to include lower case, or punctuation, or numerals. It is not necessary that the array be square. The method can be adapted to rectangular arrays. Col. Abel of the Soviet Union used just this kind of cipher in his espionage forays into the United States in the 1960s and 1970s, with the added wrinkle that the enciphered message was re-encyphered a second time with a different Playfair table based on a different key phrase that the first Playfair table. The reason for the popularity of this method is that no props are needed to use it and a few easily-remembered phrases and a few simple rules embody the whole method. "orange juice is red" might be used, with the table as follows, where the table has been filled out with numerals.

o	r	a	n	g	e
j	u	i	c	s	r
d	b	f	h	k	l
m	p	q	t	v	w
x	y	z	1	2	3

11.3.8. Block Codes

There are many block codes that are not ciphers, including whole books of words and equivalent 5- or 7-digit codes that can be used to code the word. several different codes can be used for each word. The number depends on the frequency of occurrence of the word in typical text. In 1940 in Vancouver a team of German agents were using one of these ciphers that was based on a key extracted from a catalogue of commercial dyes. They were sending to Germany their observations of naval and air activity in the Puget Sound and Esquimalt area. They were particularly interested in the airframe factories in Seattle because of the Boeing factories.

11.3.9. Product Ciphers

A product cipher is a succession of (usually simple) ciphers intended to produce a much more difficult cipher. The modern Data Encryption Standard (DES) is an example of a product cipher; it is by far the most important encryption method used with computers at this time. It will probably soon be replaced by the Advanced Encryption Standard in the next year or so. Because of its importance DES will be described in detail in the next section.

11.3.10. The Data Encryption Standard

The DES is an example of an encryption method know as a Feistel cypher. It encrypts data in a 64-bit block, using a 56-bit key. It is intended to be used in computers, and special purpose chips are available that implement it and that are capable of very high datarate (now >2 Gb/s). This US standard is composed of

- an initial permutation that transforms the incoming 64-bit block of data according to a fixed initial permutation and then divides it into two 32-bit blocks,
- a sequence of 16 combined transposition and substitution steps, and
- an final permutation that transforms the outgoing 64-bit block of data according to a fixed final permutation .

The permutations are not particularly interesting. A table for the initial permutation is given in the accompanying table IP. The permutation table is really a list, but it is displayed as an array for reasons of space. The list index of any entry in the table shown is obtained by adding the row and column indexes that are shown. If the input data is in a Boolean array $input[1..64]$, for each bit *j* of the input, $1 \le j \le 64$, the result of the permutation is $permuted\_input[j] = input[ip[j]]$. Thus, $d_1 d_2 d_3 \cdots d_{64}$ permutes to the sequence $d_{58} d_{50} d_{42} \cdots d_7$.

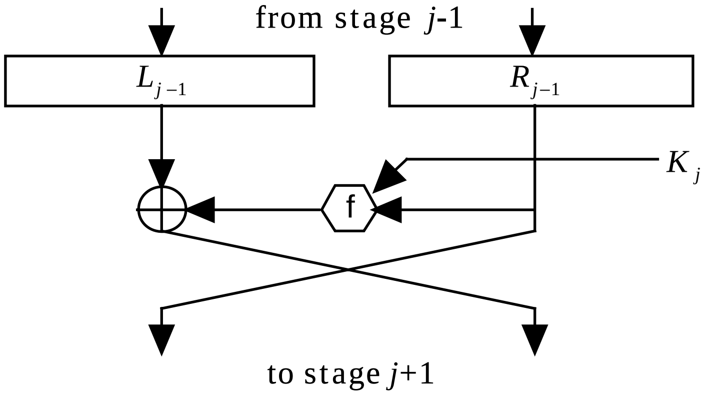
ip	1	2	3	4	5	6	7	8
0	58	50	42	34	26	18	10	2
8	60	52	44	36	28	20	12	4
16	62	54	46	38	30	22	14	6
24	64	56	48	40	32	24	16	8
32	57	49	41	33	25	17	9	1
40	59	51	43	35	27	19	11	3
48	61	53	45	37	29	21	13	5
56	63	55	47	39	31	23	15	7

And the final permutation is given in the accompanying table OP. It is true that $op = ip^{-1}$. If the output data is in a Boolean array $raw\_output[1..64]$, for each bit *j*, $1 \le j \le 64$, of the raw output of the 16th interior stage the result of the permutation is $output[j] = raw.output[op[j]]$. Under this permutation $a_1 a_2 a_3 \cdots a_{64}$ permutes to the sequence $a_{40} a_8 a_{48} \cdots a_{25}$.

op	1	2	3	4	5	6	7	8
0	40	8	48	16	56	24	64	32
8	39	7	47	15	55	23	63	31
16	38	6	46	14	54	22	62	30
24	37	5	45	13	53	21	61	29
32	36	4	44	12	52	20	60	28
40	35	3	43	11	51	19	59	27

48	34	2	42	10	50	18	58	26
56	33	1	41	9	49	17	57	25

Stage j of the 16 interior stages is depicted in the following figure. Each of the stages is identical, except that the last stage omits the crossing of the L and R outputs.



Each of the 16 interior steps is identical. Let T_j represent the output of the j^{th} step, and let L_j and R_j be the left and right halves of T_j. Thus $T_j = L_j R_j$ for $L_j = t_1 t_2 t_3 \cdots t_{32}$ and for $R_j = t_{33} t_{34} t_{35} \cdots t_{64}$. L_j is computed from $L_j = R_{j-1}$ and R_j is the result of $R_j = L_{j-1} \oplus f\left(R_{j-1}, K_j\right)$ where *f* is a complicated and intricate function that does an expand-shift-substitute-permute operation, *K* is derived from the key, and ⊕ is the Boolean exclusive-OR function.

After the 16^{th} step, the left and right halves are not exchanged. Instead the block $R_{16} L_{16}$ in that order is the input to the final permutation.

11.3.11. The DES Function f

A schematic view of the function f is given in the figure given below and entitled Figure Showing the Function *f*. The function *f* expands the 32-bit R_{j-1} to 48 bits using a special bit selection table called E. This table just gives some (apparently *ad hoc*) rules for selecting the 48 bits from the 32-bits of R_{j-1} by selecting some bits twice. The bit selection table is:

E	1	2	3	4	5	6
0	32	1	2	3	4	5
6	4	5	6	7	8	9
12	8	9	10	11	12	13
18	12	13	14	15	16	17
24	16	17	18	19	20	21
30	20	21	22	23	24	25
36	24	25	26	27	28	29
42	28	29	30	31	32	1

This table is used in the same way that the *ip* table is used; the 32 bit sequence $a_1 a_2 a_3 \cdots a_{32}$ expands to the 48 bit sequence $a_{32} a_1 a_2 a_3 a_4 a_5 a_4 a_5 a_6 \cdots a_{32} a_1$.

These 48 bits are exclusive-ORed with 48 bits chosen from the key. The key that is used at each stage is derived from the input key by an arcane selection method. The input key is 64 bits, with bits $8, 16, 24, \cdots, 64$ being parity bits for the previous 7 bits. The parity bits are discarded, leaving an effective key of 56 bits. The parity bits are discarded and the bits of the 64 bit key are permuted according to permutation & choice PC-1 shown next:

PC-1	1	2	3	4	5	6	7
0	57	49	41	33	25	17	9
7	1	58	50	42	34	26	18
14	10	2	59	51	43	35	27
21	19	11	3	60	52	44	36
28	63	55	47	39	30	23	15
35	7	62	54	46	38	30	22
42	14	6	61	53	45	37	29
49	21	13	5	28	20	12	4

The result is a 56 bit field, that is split into two halves called C and D of 28 bits each. The formation of the successive values of K_j proceeds as shown in the figure below. In this figure, the LS-*j* boxes do a left circular shift of one or two bits, as given in this table:

key stage	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Finally, the 56 bit field formed by concatenating the two 28 bit fields $C_j D_j$ is converted to a 48 bit field according to permutation & choice PC-2 shown next:

PC-2	1	2	3	4	5	6
0	14	17	11	24	1	5
6	3	28	15	6	21	10
12	23	19	12	4	26	8
18	16	7	27	20	13	2
24	41	52	31	37	47	55
30	30	40	51	45	33	48
36	44	49	39	56	34	53
42	46	42	50	36	29	32

After all this, the successive values of the key K_j all have 48 bits that are derived from the 56 bits of key that was input.

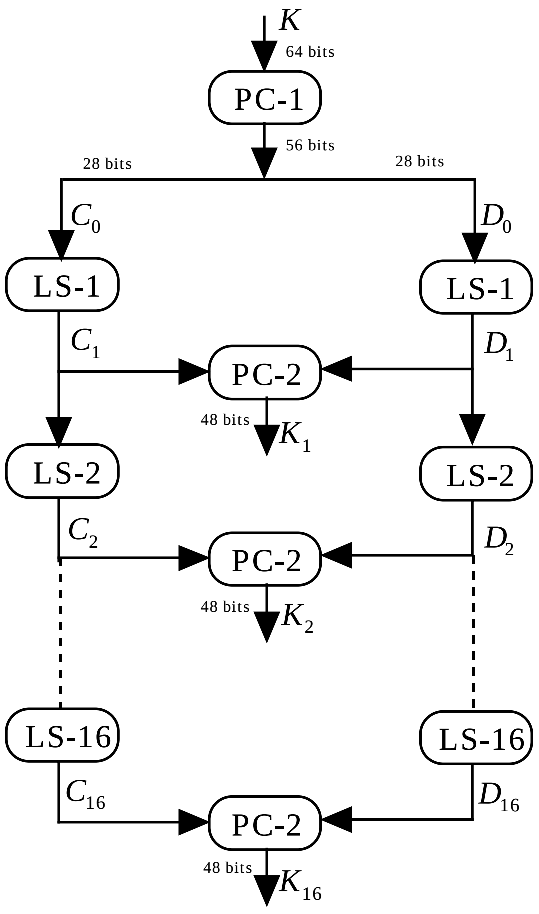


Figure showing the Key Calculation

The 48 bit result of the exclusive-OR of the key and the output of the bit selection E is split into eight 6-bit blocks. Each 6-bit block is input into a substitution function (called an S-box) that returns a 4-bit block, and these are assembled into a 32-bit output to form the input to another permutation P. The result of this last permutation is $f\left(R_{j-1}, K_j\right)$. The permutation P is:

P	1	2	3	4
0	16	7	20	21
4	29	12	28	17
8	1	15	23	26
12	5	18	31	10
16	2	8	24	14
20	32	27	3	9
24	19	13	30	6
28	22	11	4	25

The eight selection functions can be examined at leisure. There has been no rationale published for the particular selection functions that are used. It is not known publicly if the mechanism is a member of a family of similar mechanisms, or if it is unique.

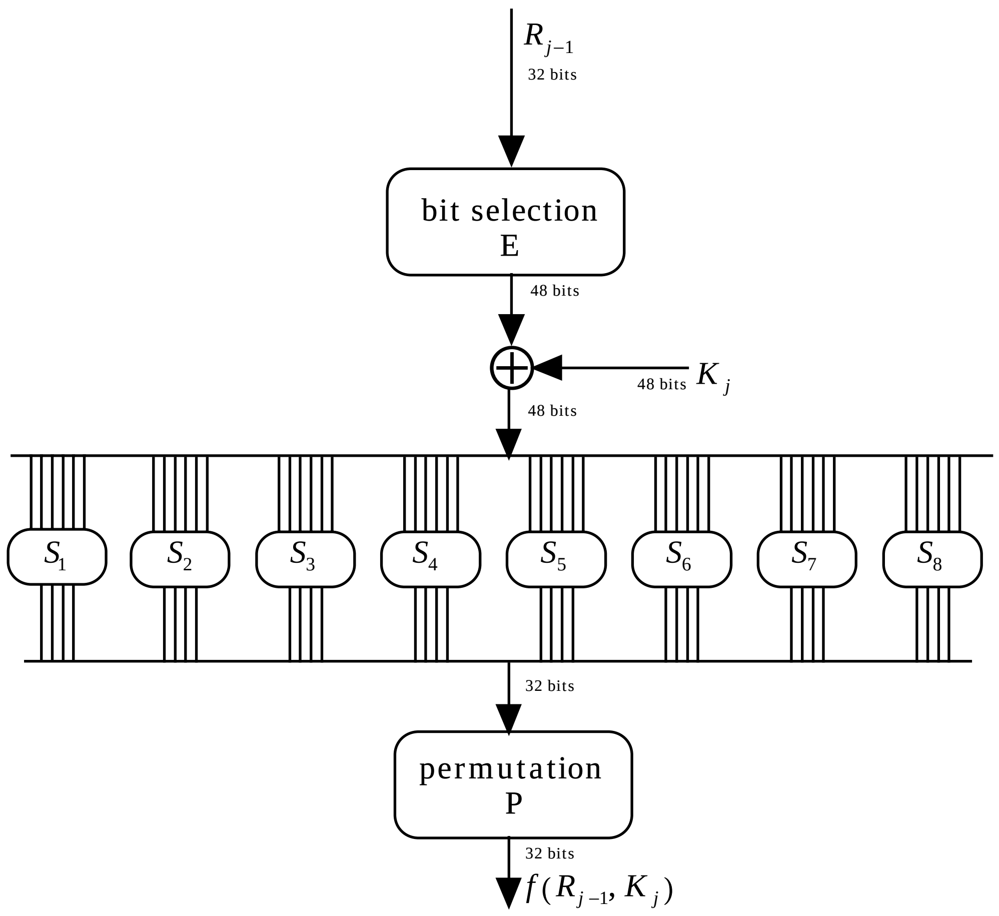


Figure Showing the Function *f*.

Let the input to S-box *j* be $B_j = b_1b_2b_3b_4b_5b_6$. Form an index $\rho = b_1b_6$ and convert it to a decimal integer. Form an index $\chi = b_2b_3b_4b_5$ and convert it to a decimal integer. The indexes ρ and χ are used to select an integer from the following table. The four-bit output of the S-box is the binary representation of that decimal integer.

For example, if $B_3 = b_1b_2b_3b_4b_5b_6 = 011101$, the row index is $01_2 = 1_{10}$, the column index is $1110_2 = 14_{10}$, so the output of S3 is $11_{10} = 1011_2$. Of course, in an actual DES circuit or program, there is no need or advantage of the decimal representation; we are using it here for ease of typing.

Each row in the S-box table is a permutation of the 16 integers from 0 to 15. There are $16! = 2.09 \times 10^{13}$ such permutations. Just why these 32 are preferred over any of the other permutations is not known. The actual analysis of DES along with its design decisions remains a mystery. Some considerable effort has gone into trying to reconstruct the reasons why these choices were made, with no visible success yet.

		Column index															
	Row	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S_1	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
S_2	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_5	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
S_6	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

As far as is known, DES does not contain any design blunders of any great moment. Some patterns have emerged in the results of the 16 stages, including the possibility that some of them can, in specific peculiar circumstances, regenerate the input to a box earlier in the sequence. As far as is known, nobody has been able to use this observation to break the cipher.

There was a great controversy that raged about the rather short key, because the original proposal suggested a 112 bit key. Some researchers with suspicious minds or active imaginations felt that the method may have been deliberately weakened. It is equally likely that the key was shortened so that the original implementation of DES could attain some target encryption datarate. The debate continues.

DES does, however, have known weaknesses. In order to discuss DES sensibly, let $c = \mathbf{DES}[p, k]$ indicate that, using key k and DES, some plaintext p is being ciphered to produce a ciphertext c. Many weaknesses or alleged weaknesses have been published; a few of the better known ones are:

complements:	Let the ones complement of x be denoted $\overline{x}$. If $c = \mathbf{DES}[p, k]$ then it will also be true that $\bar{c} = \mathbf{DES}[\overline{p}, \overline{k}]$. This effect is probably not serious unless a run of keys includes a known sequence of a key and its complement, and even then it may be of illusory value to an intruder.
weak keys:	The key is split into two halves, and the effect of the mechanisms is that each half is shifted circularly and independently. If the value being shifted is all zeroes or all ones, the same key will be used at each stage. Since the effect is known these keys can be avoided, but their existence does not fill one with confidence. In hexadecimal, example weak keys are 01010101 01010101, FEFEFEFE FEFEFEFE, and 1F1F1F1F 1F1F1F1F.
paired keys:	There are known pairs of keys that produce the same ciphertext for any plaintext. This means that if $[k_1, k_2]$ is such a pair, $c = \mathbf{DES}[p, k_1]$ and also $c = \mathbf{DES}[p, k_2]$. Several of the known pairs are shown in the following table.

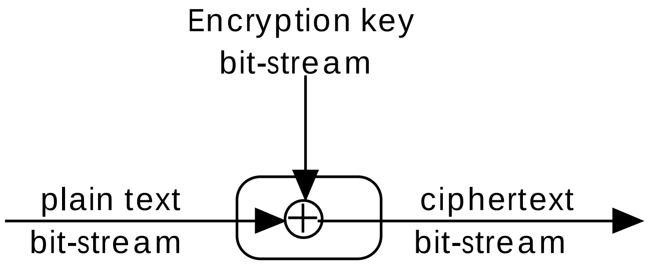
k_1	k_2
01FE 01FE 01FE 01FE	FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1	E01F E01F F10E F10E
01E0 01E0 01F1 01F1	E001 E001 F101 F101
1FFE 1FFE 0EFE 0EFE	FE1F FE1F FE0E FE0E
011F 011F 010E 010E	1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE	FEE0 FEE0 FEF1 FEF1

Even with these remarks about its deficiencies, DES remains highly respected. There are an enormous number of computer users who believe it to be impregnable. We shall see. DES is certainly slower than other complicated mechanisms that seem to be equally strong. These competitors, however, were not blessed by the Agency, perhaps because they were developed without its help, or perhaps because it can not decipher messages that have used them, or more likely because certifying cipher techniques to be used by business is not its job.

In late 1998 and early 1999, a competition was initiated by the US government to develop the Advanced Encryption Standard (AES). The key length and the block length will be longer. The ability to encrypt and decrypt with software that runs quickly on machines of different wordlengths, 8 bit to 64 bit, is central to the choice.

11.5. One-Time Pads

The one-time pad is the *non plus ultra* of encryption schemes. The figure shows a data stream being exclusive-ORed with a key stream.



The one-time pad works exactly in this way. Decryption is precisely the same operation. If c is the ciphertext bit-stream, p is the plaintext bit-stream, and k is the encryption key bit-stream, then $c = p \oplus k$, Because of the properties of exclusive-OR, it is also the case that $p = c \oplus k$. This is easily seen by operating on both sides of $c = p \oplus k$ as follows:
$c \oplus k = p \oplus k \oplus k$ but $k \oplus k = 0$ and $p \oplus 0 = p$ from the elementary properties of exclusive-OR, so $c \oplus k = p$ as predicted.

The strength of the one-time pad is entirely contained in the properties of the encryption key bit-stream. It must have the following properties:

- The encryption key bit-stream must be available to both encrypter and decrypter, but must be kept highly confidential otherwise.
- The bits of the key must be independent and uncorrelated in the following senses: each bit must equally probably by either 0 or 1; each successive pair of bits must equally probably be 00, 01, 10, or 11; all eight triples of bits must be equally probable, and so on. High-strength one-time-pad systems will ensure that these probabilities are as they should be up to at least 16 bits, and possibly further.
- There must be zero probability of two plaintexts being enciphered with the same key. This is because if $c_1 = p_1 \oplus k$ and $c_2 = p_2 \oplus k$ then if the exclusive-OR of the two ciphertexts is calculated,
  $$c_1 \oplus c_2 = p_1 \oplus k \oplus p_2 \oplus k = p_1 \oplus p_2 \oplus k \oplus k = p_1 \oplus p_2.$$
  The exclusive-OR of the two messages is a big help to an intruder.

Provided that precautions are taken to see that these conditions hold, the one-time-pad is the strongest known encryption mechanism. Because of the encryption key bit-stream statistics (the second condition above), the ciphertext will have similar statistics as a long-term average.

11.5.1. Developing the Key Stream

The hard part is the first condition. How can two players chose a key bit-stream without exchanging enough data so that any intruder could also deduce it? The answer is that both parties must have the same collection of data that they keep highly confidential. When they need a new key they use the next one available from their data collection. In pen-and-paper days the collection of data was printed as a pad of paper, and exactly two copies of each pad existed. The two parties both had a copy of the same pad. The top page on the pad was used by each to encrypt and to decrypt, taking care to never re-use the page. It was ripped off the pad and destroyed as soon as it had been used. The only plaintext messages that were needed were occasional re-synchronising of the number of the top page. The only major problem is keeping the pad pages in synchronism.

A bit stream with the desired statistical properties can be developed by using mathematical objects called "primitive polynomials modulo 2" or, more prosaically, linear recurring sequences or feedback shift registers. For example, the polynomial $x^{41} + x^3 + x^0$ defines a recurrence relation for obtaining a new random bit from the 41 preceding bits. It is guaranteed that this process will not repeat until $2^{41} - 1$ bits have been generated. The only difficulty is to avoid setting all 41 bits to zero. The forty-one zeroes sequence is not part of the linear recurring sequence; if the 41 bits are started at the initial pattern all-zero the sequence is stuck at that value.

These linear recurring sequences are rather special. Suppose a linear recurring sequence (LRS) of order n is being used (41 in the example above). The calculation can be done as follows: Let the bits be identified as b_1 for the most recently generated bit to b_n for the bit generated n time units ago, where n is the order of the LRS being used. In the above example the bits would be identified as $b_1, b_2, b_3, b_4, \cdots, b_{41}$. The next bit will be called b_0 and it is not difficult to deduce that, for the example, $b_0 = b_3 \oplus b_{41}$. This is exceptionally easy to compute, so these sequences are often used to generate streams of random bits. Watson[62] lists LRS sequences up to order 100. A few examples are given in the following table. The coefficients are always one, so all that need be given is the nonzero powers of x – the example can be written [41,3,0] Many others are known.

[96,7,6,4,3,2,0]
[64,4,3,1,0]
[48,7,5,4,2,1,0]
[32,7,5,3,2,1,0]
[16,5,3,2,0]

[62] E.J. Watson, "Primitive Polynomials (Mod 2)", Math. of Computation, Vol 16, p 368.

The first of these examples will generate a sequence of binary digits with satisfactory statistical properties from any of $2^{96}-1$ starting points. This is about 7.923×10^{28} possible starting points.

LRS sequences do have the problem that if the polynomial that is being used is known then concealing the starting point is vital. They are not desirable because of this simple fact. Also, the process they use is linear so even if the polynomial is not known some number of examples of key bit-stream will suffice to deduce it. These could be recovered from, for example, encrypting known plaintext. Better key bit-stream generators are known but they all suffer from the necessity of concealing the starting point.

11.6. Rivest-Shamir-Adelman

The Rivest-Shamir-Adelman (RSA) scheme owes its effectiveness to the difficulty of factoring large numbers. Let a plaintext be denoted as M and the corresponding cryptotext be C. Let n be a (very) large integer with exactly two prime factors, p and q , so that $n = pq$. These two factors are not known, and because the integer is so large it is not practical to compute them. The method depends on this difficulty in factoring large numbers, and on some results from doing arithmetic with the *mod* function.

Define the quantities e and d such that $ed \bmod ((p-1)(q-1)) = 1$. To compute these, choose d relatively prime to $(p-1)(q-1)$. The easy way to do this is to make d prime. An efficient algorithm to compute e given n and d exists[63]. It is not necessary to know p or q to find e.

The idea behind RSA is to use the two equations

$$C = M^e \bmod n \qquad (1)$$
$$M = C^d \bmod n \qquad (2)$$

The presence of the mod function makes the computation much quicker that it otherwise might be. This arises because

$$\prod M^e \bmod n = \left((M \bmod n)_1 \times (M \bmod n)_2 \times \cdots \times (M \bmod n)_e\right) \bmod n$$

The mod function can be used on all partial products to contain the build-up of the size of the product.

The strength of RSA is partly derived from its strong encryption, and partly from the fact that it can be easily used for public key encryption. The key e can be stored in public in a file. The key d is kept secret by one of the computers that can receive encrypted messages. To send a message to that computer, the public key e is looked up and the message is encrypted using equation (1) above. Only the intended

63 See D.E. Denning, "Cryptography and Data Security", A-W 1983. Figure 1.22 on page 44.

recipient knows the private key d. It is not computationally feasible to compute d given e and n.

It has been estimated that the fastest known factoring algorithm for an n-digit number that does not use parallel computation takes $T(n) = \mathrm{O}\left(e^{\sqrt{(\ln n)\ln(\ln n)}}\right)$. If $n = \mathrm{O}\left(10^{129}\right)$, then $T(10^{129}) = \mathrm{O}\left(7.25 \times 10^{17}\right)$ and at a rate of one computation every nanosecond, factoring the 129 digit number would take 23 years. The actual number that is used is the following:

					114	381	625	757	888	867	669
235	779	976	146	612	010	218	296	721	242	362	562
561	842	935	706	935	245	733	897	830	597	123	563
958	705	058	989	075	147	599	290	026	879	543	541

An effort to factor this number by using numerous workstations on the internet all operating in parallel according to an organised arrangement was successful a few years ago. A 140-digit number has been factored in 1999. It is now being recommended that n have well over 200 digits. Finding numbers that are that large and that have exactly two large prime factors is difficult.