INFORMATION TO USERS

This material was produced from a microfilm copy of the original document. While the most advanced technological means to photograph and reproduce this document have been used, the quality is heavily dependent upon the quality of the original submitted.

The following explanation of techniques is provided to help you understand markings or patterns which may appear on this reproduction.

1. The sign or "target" for pages apparently lacking from the document photographed is "Missing Page(s)". If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting thru an image and duplicating adjacent pages to insure you complete continuity.
2. When an image on the film is obliterated with a large round black mark, it is an indication that the photographer suspected that the copy may have moved during exposure and thus cause a blurred image. You will find a good image of the page in the adjacent frame.
3. When a map, drawing or chart, etc., was part of the material being photographed the photographer followed a definite method in "sectioning" the material. It is customary to begin photoing at the upper left hand corner of a large sheet and to continue photoing from left to right in equal sections with a small overlap. If necessary, sectioning is continued again — beginning below the first row and continuing on until complete.
4. The majority of users indicate that the textual content is of greatest value, however, a somewhat higher quality reproduction could be made from "photographs" if essential to the understanding of the dissertation. Silver prints of "photographs" may be ordered at additional charge by writing the Order Department, giving the catalog number, title, author and specific pages you wish reproduced.
5. PLEASE NOTE: Some pages may have indistinct print. Filmed as received.

Xerox University Microfilms

300 North Zeeb Road
Ann Arbor, Michigan 48106

77-2406

GUDES, Ehud, 1945-
THE APPLICATION OF CRYPTOGRAPHY TO DATA
BASE SECURITY.

The Ohio State University, Ph.D., 1976
Computer Science

Xerox University Microfilms, Ann Arbor, Michigan 48106

© Copyright by

Ehud Gudes

1976

THE APPLICATION OF CRYPTOGRAPHY
TO DATA BASE SECURITY

DISSERTATION

Presented in Partial Fulfillment of the Requirements for
the Degree Doctor of Philosophy in the Graduate
School of The Ohio State University

By
Ehud Gudes

The Ohio State University

1976

Reading Committee:

Prof. Jerome Rothstein
Prof. Harvey S. Koch
Prof. Stuart H. Zweben
Prof. Douglas S. Kerr

Approved By

Adviser
Department of Computer
and Information Science

ACKNOWLEDGMENTS

I would like to express my gratitude to my major adviser, Professor Jerome Rothstein for accepting me as his advisee in the middle of this research, and for many contributions to this work especially in its later parts. The most important ones were first, his observation that Shannon's unicity distance is inappropriate for data base cryptography and that the work factor approach was needed, and second, his idea of using the "random key generation" method as an easy way to achieve a high work factor. Professor Rothstein's constructive criticism and his tireless efforts to improve the presentation of this work are greatly appreciated. I would like to thank my former major adviser, Professor Fred Stahl of Columbia University, for introducing me to the subject of cryptography, for his help in formulating the basic problems and for directing the first half of this research.

I would like to thank Professor Harvey Koch for his help and encouragement throughout this research and for his contributions to the data base model. Professor Koch's advice in difficult moments was instrumental for the completion of this work. I want to thank Professors S. Zweben and D. Kerr for serving on my reading committee and for their helpful comments and suggestions which improved the presentation of the results significantly.

Thanks also go to Mark Ebersole from IRCC for providing the file used in the experiment, and to Steve Miller for his help in implementing the NBS block cipher. I also would like to thank Professor M. Yovits and the Department of Computer and Information Science for their financial support in the last three and a half years.

This work is dedicated to Amiella, my wife, for her help, encouragement, understanding and endurance throughout this long and demanding road.

VITA

August 2, 1945 . . .	Born - Haifa, Israel
1967	B.Sc., Technion - IIT, Haifa, Israel
1970-1972	System Programmer, Technion Computer Center, Haifa, Israel
1973	M.Sc., Technion - IIT, Haifa, Israel
1973-1976	Research and Teaching Associate, The Ohio State University, Columbus, Ohio

PUBLICATIONS

With A. Reiter, "On Evaluating Boolean Expressions", Software Practices and Experience, December, 1973.

With F. Stahl and H. Koch, "The Application of Cryptographic Transformations to Data Base Security", NCC Proceedings, 1976.

FIELDS OF STUDY

Major Field: Computer and Information Science

Studies in System Programming. Professors J. Rothstein,
S. Zweben and D. Kerr

Studies in Information Storage and Retrieval.
Professor H. Koch

Studies in Automata Theory. Professors J. Rothstein
and H. W. Buttelmann

TABLE OF CONTENTS

	Page
ACKNOWLEDGMENTS	ii
VITA.	iii
LIST OF TABLES.	vii
LIST OF FIGURES	viii
 Chapter	
I. INTRODUCTION AND MOTIVATION	1
1.1 Computer Data Security	1
1.2 Data Security Risks and Their Countermeasures.	3
1.3 The Role of Cryptography in Data Security	7
1.4 The Objectives and the Organization of this Dissertation	9
II. REVIEW OF CRYPTOGRAPHY	13
2.1 Traditional Cryptography	13
2.2 Measures for Cryptographic Security.	17
2.3 Applying Cryptography to a Computer System.	19
2.4 Cryptography and Data Bases.	26
2.5 Summary.	27
III. REVIEW OF DATA BASE SECURITY.	28
3.1 General Data Security Models	28
3.2 Data Base Security vs. Operating System Security.	31
3.3 Basic Concepts in Data Base Security	33
3.4 Analysis of Data Base Models	35
3.5 Cryptography and Data Bases.	39
3.6 Summary.	41
IV. A MULTI-LEVEL STRUCTURED MODEL OF DATA BASE	42
4.1 Introduction	42

	Page
4.2 Four Levels Model.	43
4.3 Notation and Definitions	51
4.4 Examples of Standard Structures in the Level Notation.	56
4.5 Discussion	62
4.6 Summary.	62
V. CRYPTOGRAPHIC TRANSFORMATIONS IN THE MULTI-LEVEL MODEL	64
5.1 Basic Definitions.	64
5.2 Transformations Between Physical Levels	65
5.3 Using Cryptographic Transformations.	74
5.4 Summary.	80
VI. DESIGN OF A SECURE FILE SYSTEM BASED ON USER CONTROLLED CRYPTOGRAPHIC TRANSFORMATIONS	81
6.1 General System Structure	81
6.2 Compartmentalized, Data Independent Protection Specifications.	88
6.3 Hierarchical Protection Specifications.	102
6.4 Data Dependent Protection Specifications	111
6.5 Summary.	116
VII. EVALUATING THE SECURITY OF FILE ENCRYPTING	118
7.1 Introduction	118
7.2 Review of Shannon's Measures	119
7.3 The File as a Message Source	122
7.4 Combinatorial Approach to the Work Factor Measure	128
7.5 Definition of the Work Factor.	146
7.6 Other Factors.	148
7.7 Summary.	150
VIII. EXPERIMENTS WITH FILE ENCRYPTING	152
8.1 Introduction	152
8.2 General Description of the Experiment.	153
8.3 Detailed Description of the Experiment	158
8.4 The Results.	164
8.5 Analysis of the Results.	176
8.6 Summary.	182

	Page
IX. SUMMARY AND SUGGESTIONS FOR FUTURE RESEARCH	183
9.1 Summary and Main Contributions.	183
9.2 Points for Future Research.	185
BIBLIOGRAPHY	187
APPENDIX A	194

LIST OF TABLES

		Page
Table		
1	CLEAR.	165
2	CAESAR	166
3	VERNAM	167
4	NBS.	168
5	PRVERNAM	169
6	FHOMOPHON 1.	170
7	FHOMOPHON 2.	171

LIST OF FIGURES

Figure		Page
1	Technical Safeguard and Data Security Risks.	1
2	A Portion of an Access Matrix	30
3	The Levels Approach	32
4	Multiple Physical Levels.	44
5	Spreading Protection Specifications and Mechanisms.	49
6	The Connection Between the Logical Data Base Level, the Physical Data Base Level, the Logical Records and the Physical Records.	55
7	A "Set" Structure	57
8	Access Paths.	59
9	Record Splitting.	61
10	Types of Cryptographic Transformations.	73
11	Transformation Between Two Physical Levels.	77
12	Using Communication Keys.	78
13	General Structure of the File System.	84
14	General Description of the Access Mechanism	85
15	Access Matrix	88
16	A Validation Record	92
17	The "User Profile" Scheme	93
18	The "Keys Record" Scheme.	95

	Page
19 The "Key Inversion" Problem	97
20 A Keys Record	98
21 A Tree Directory.	102
22 Access Hierarchy.	104
23 Keys Records for Independent Access Hierarchy	104
24 An Inverse Directory.	109
25 The Stack Structure	110
26 Security Atoms.	112
27 DDAO Specifications	115
28 DDAO "Keys Records"	116
29 A Statistic S	127
30 The Combinatoric Problems	131
31 A "Retrievable" Cipher.	142
32 A "Partially Retrievable" Cipher.	144
33 The Layout of the File.	154
34 Part of the Field Statistic	155
35 Part (b) of the NBS Cipher.	160
36 Relative Cost of Ciphers.	178
37 Comparison Between the Two Fields	179
38 Retrieval Overhead.	180
39 The Security of File Ciphers.	181

CHAPTER ONE

Introduction and Motivation

1.1 Computer Data Security

Computer data security has become a subject of great importance and concern in recent years. This concern stems from the increase in seriousness of problems in three areas. First there is the problem of reliable and continuous operation of the computer system. "Bugs" in software or hardware are increasingly costly for organizations which rely heavily on computer systems for daily operations. Long "down times" are very undesirable and costly, whether caused by software or hardware failure, or by natural disaster. Reliability, correctness and adequate recovery procedures are very important aspects of computer data security and they are gaining significant attention from both users and manufacturers of computer systems.

The second area is the increasing number of illegal penetrations into computer systems. The number of reported computer crimes has been on the rise in recent years and losses caused by these crimes are very large. (See a survey of computer crimes by Parker [PAR73].) For example, in the Equity Funding Corporation case [McL73], the company greatly inflated its financial report by listing fictitious assets and by using its computerized accounting system to mislead the auditors. The loss to stockholders was many millions of dollars. The more common crimes involve people in key positions (e.g. operators, system programmers) who utilize the computer system for their own benefit. Preventing such crimes by providing more secure and better controlled computer systems is an important aspect of computer data security. The need for better data security is also reflected in the following citation from Westin [WES72].

With more sensitive information being added to the computerized segment of organizational record systems, the value to unauthorized persons of penetrating security clearly becomes greater. Also, the fact that some organizations will be

storing their confidential files off their premises, in time-sharing and service bureau computer centers, means that security must be provided at those sites and in data transmission, as well as at the terminal and for the files of the primary organization.

The issue of privacy, i.e., the unauthorized use of information, is the third area of concern. It may be the most important one from the point of view of the public. With the use of advanced computer systems and the centralization of government, a great deal of information about individuals is stored in computer systems. Unauthorized disclosure or possible misuse of this information is of great concern to many citizens in most of the free world. Several countries have enacted laws or regulations to safeguard the privacy of individuals. Among them are Great Britain [HAN74], Sweden [SWE73] and the United States [US74]. To enforce these laws and to preserve privacy we need more secure computer systems. The need for better data security in order to fulfill the privacy laws was recognized by the National Bureau of Standards (NBS). In 1975 NBS published a report "Computer Security Guidelines for Implementing the Privacy Act of 1974" [NBS75] which states:

Although the Act sets up legislative prohibitions against abuses, technical and related procedural safeguards are required in order to establish a reasonable confidence that compliance is indeed achieved. It is thus necessary to provide a reasonable degree of protection against unauthorized disclosure, destruction or modification of personal data, whether intentionally caused or resulting from accident or carelessness.

We see then the strong connection between the privacy problem and the data security problem. Since privacy and security are often confused, we stress the differences between them with the following quotation from Conway et. al. [CON72].

Information Privacy involves issues of law, ethics and judgment. Whether or not a particular individual should have access to a specific piece of information is a question of information privacy. . . . Information Security involves questions of means - procedures to ensure that privacy decisions are in fact enforceable and enforced.

We have now seen the reasons for concern and therefore the importance of computer data security. It is obvious that the area of data security is very broad and we cannot cover

all aspects of it in this dissertation. There are several good surveys (Hoffman [HOF69], Weiss [WEI74]), bibliographies (Bergart et. al. [BER72]), and books (Van Tassel [VAN72], Martin [MAR73], Hoffman [HOF73]) on the subject. While we recognize the importance of "reliability", "physical security", and "administrative security", we restrict our attention to the technical problems associated with penetrations of computer systems. To motivate our specific approach we first review the risks to security which exist in computer systems and the safeguards available to protect against these risks.

1.2 Data Security Risks and Their Countermeasures

Threats to computer security and their countermeasures are discussed in a classical article by Petersen and Turn [PET67]. They distinguish between "accidental disclosures" that come from such things as hardware failures or partially debugged programs, and between "deliberate infiltration." In the second category they distinguish between "passive infiltration" such as wiretapping or electromagnetic pickup, and "active infiltration." The latter is the most common and dangerous method of compromising security (particularly if computer systems are becoming more reliable). To the category of active infiltration belong such things as:

- Browsing - using legitimate access to the system to "browse" through unauthorized files. Browsing can also occur if the "standard" access control mechanism is "by-passed."
- Masquerading - forging the identification of a legitimate user.
- Inside-
personnel - having access to the system by virtue of a position with the information center.
- "Piggy back"
and "Be-
tween lines"
entry - both employing a special terminal tapped to the communication network. "Piggy back" entry permits selective interception and false answering of messages between user and processor. "Between lines" entry, means unauthorized entry while the legitimate user is inactive but still holds the communication channel open.

Petersen and Turn suggested several countermeasures against these threats to computer security. The main ones are access management and processing restrictions (which are generally called "access control"), threat monitoring (i.e., recording of all unsuccessful attempts for access), and privacy transformations - i.e., enciphering data for the purpose of protection. They showed that none of the above methods can protect against all threats. It is true that some of these threats may require quite a sophisticated "enemy," but others are quite common and particularly acute since they cannot be countered by the generally used access control methods. For example, access control cannot protect against a system programmer who "dumps" part of the content of a disk onto the printer in a "stand-alone" mode. The last threat can be overcome by the use of privacy (cryptographic) transformations (*). The threats against which cryptographic transformations are particularly helpful are: system error, entry by system personnel, physical acquisition of removable files, wiretapping and "piggy back" or "between lines" entry. The first three threats are quite common and do not require special equipment or a sophisticated enemy. Petersen and Turn stressed the importance of cryptographic transformations as an effective protection tool. They also noted that none of the mentioned countermeasures can alone protect against all threats, and suggested that in any secure computer system these methods should be applied in combination. Even though Petersen and Turn's paper appeared in 1967, not much (public) research has been done on cryptography and most of the research in data security has concentrated in the access control area. (For example, see Graham and Denning [GRA72], Lampson [LAM69], Jones [JON73], Hsiao [HSI73].)

Recently, cryptography has received renewed recognition as an important protection tool by the National Bureau of Standards [NBS75]. Also, some computer companies have started to publish new research on cryptography, for example, IBM [FEI75], and HONEYWELL [BAR74]. NBS, in their recent report [NBS75] suggest the use of data encryption as a safeguard against several security risks. In particular they encourage its use in network oriented computer systems. The importance of data encryption, in the NBS view, is depicted in Figure 1 ([NBS75] pp. 10-11). As can be seen from this figure, data encryption is found effective against five of

(*) We prefer the use of "cryptographic transformations" over "privacy transformations" to avoid confusion with the term "privacy."

the seven categories of risks described. As a result, NBS has published a data encryption algorithm [FED75] and has proposed it as a Federal Information Processing Standard (FIPS).

Figure 1 - Technical Safeguards and Data Security Risks.

Figure 1 - Continued

We have discussed in length the justification for applying cryptography for computer data security. However, there are some people who doubt the practicality of this important protection tool. The most common objections are that either enciphering is very expensive or it is hard to control in a complex shared information system (e.g. see Hartson [HAR75] p. 6). As will be shown in this dissertation, these difficulties have been exaggerated.

From the discussion above, our interest in investigating the ways to apply cryptography for data security is justified. In order to understand further our specific approach we need to discuss the role of cryptography in computer data security and its relationship to other protection mechanisms commonly used in computer systems.

1.3 The Role of Cryptography in Data Security

Cryptography antedates modern day computers by centuries. The history of cryptography is given in a fascinating book by Kahn [KAH67]. In the past, cryptography was mainly used to protect information transferred through communication lines. Since communication lines are an integral part of computer systems today, this aspect of cryptography is very important and is currently the main application of cryptography in computer systems. As an example, one of the most popular uses of "communication cryptography" is in "cash-dispensing" systems. In these systems the information supplied by the user and the "affirmative" message sent by the computer are both enciphered. This is done to prevent some "clever" users from "emptying" somebody else's account by "tapping" the communication line. However, even this, close to traditional, use of cryptography is now significantly different from what it was in the pre-computer era. As stated by Martin ([MAR73] p. 204):

The subject, however, has drastically changed its nature with the advent of computers. When a computer or computer-like logic can be used to do the coding, a much more complex form of enciphering can be used. On the other hand, the computer will now be used to aid in the decrypting and to search at high speed through very large numbers of possible transformations.

The subject of "communication cryptography" is well established and several research papers have been published in this area. (For example, see Baran [BAR64], Feistel et al. [FEI75].) Also, almost all of the patents which have been issued on cryptography deal with "communication cryptography" (a list of patents dealing with cryptography was compiled by Gudes and Stahl [GUD75]). However, the more challenging role of cryptography, that of protecting data within the computer system, e.g. protecting data on secondary storage files, or protecting against "scrutinous" system personnel, is not yet widespread. Here the problem is more complicated because of the various physical media concerned (main memory, CPU, etc.), the processing which needs to be done on the data, and the problem of sharing the same data between different users. Some researchers have addressed various aspects of the problem (Van Tassel [VAN69], Skatrud [SKA69], Carrol and McLellen [CAR70], Turn [TUR73] and Stahl [STA74]). However, the main problem is how to integrate the use of cryptography with other protection mechanisms, thus achieving a more secure computer system. We believe that this problem has been a major obstacle to expanding the use of cryptography for computer data security.

In this dissertation we deal with this problem of integration, with the "place" of cryptography in a data secure system and its relationship to other protection mechanisms. However, we do not discuss all parts of a computer system. We set a less ambitious goal, and we deal only with one very important part of the computer system, the data base system. One reason for this choice is the increasing importance of the data base system in any information processing organization, and the fact that the most sensitive data will probably be part of the data base system. Also, the complex and dynamic structure of data base systems and therefore the great amount of research and development in the area of data bases, justify concentrating in data base security problems. We therefore limit our research to the application of cryptography to the data base area. Other aspects of applying cryptography in computer systems, such as hardware and network problems, were addressed recently by Muftic [MUF76].

The justification for pursuing the subject of our research - the application of cryptography to data base security, has now been established. The specific directions of our research are discussed in the next section.

1.4 The Objectives and the Organization of this Dissertation

In this dissertation we are interested in the combined areas of cryptography and data base security. We review both of these subjects in the next two chapters. We stress that our main interest is data base security problems. We shall not deal with other security areas, such as operating system security. We seek ways to enhance the security of a data base system by using cryptographic transformations as an integral part of the data base design. We investigate the effect of these transformations on the data base structure, representation, and operation, and we show how to use them effectively in order to enhance the security of the data base. Of particular interest is the complementary relationship between cryptographic transformations and other protection mechanisms generally used in data base systems. We shall see that cryptography helps in areas where access control mechanisms are weak.

The last statement can be demonstrated by the following examples. Most access control mechanisms assume correct operation of physical security and computer hardware. If the hardware fails and unauthorized data is disclosed, a security compromise occurs. However, if the data is enciphered, the loss caused by hardware failure is reduced. A similar situation occurs with false identification. For most protection mechanisms correct identification is prerequisite to the correct execution of the protection mechanism (e.g. "capability" based mechanism, see Lampson [LAM69]). As stated by Hartson ([HAR75] p. 5):

User identification is a keystone to every access control system. If an interceptor can successfully masquerade as an authorized user by forging an identification or stealing a password, the rest of the protection system is surely obviated.

In this dissertation (Chapter 6) we shall show several examples in which this constraint of correct identification is relaxed and still the security of the system is not compromised. Again, this is done by the use of cryptographic transformations.

We see then that cryptography helps especially in the areas usually ignored by access control mechanisms (sometimes called the "gray areas" of protection). On the other hand, as we shall see later, cryptography

alone cannot solve all security problems. For example, there is no easy way, using cryptography only, to protect against WRITE access (i.e. against somebody who wants to write on or change the data). Another problem is that another protection mechanism is sometimes needed to protect the cryptographic keys (see Chapter 5). An important conclusion of this research is that cryptography should not replace but rather complement the "standard" access control mechanisms. In practice, cryptographic transformations and access control mechanisms should be designed to complement and reinforce each other, so that each will protect the "weak" areas of the other. Explicit characterization of this "complementary" relation is a major objective of this dissertation. We also show that effective application of cryptography cannot be done after the system is designed, but rather it has to be an integral part of the total system design. Examples of such designs will be given in this dissertation.

There are several more specific objectives to this work:

- a) A Conceptual/Structural Model. The first objective is to show how cryptographic transformations can be applied as an integral part of a data base system. A serious obstacle to achieving this goal is lack of a suitable data base model. Developing such a model is an important objective of this dissertation. The model developed is shown to be general, flexible and a useful framework for investigating cryptographic problems.
- b) Design of Cryptographic Schemes. The second objective of this work is to recommend general schemes for implementing cryptography in data base systems. These schemes should provide high levels of security and sufficient flexibility to match different protection requirements. They are developed in Chapter 6. We also point out the advantages and disadvantages of each scheme, thus giving the designer or the data base administrator the information he needs to base his design decisions on his own priorities.
- c) Analyzing Cost-Effectiveness. The third objective of this work is to evaluate the

cost-effectiveness of cryptographic transformations applied to data base systems. It is important to note here that very little research has been done on the evaluation of security in general (some evaluation was done by Baum [BAU75]). Some "cost" measurements of several ciphers were done by Friedman and Hoffman [FRI74], but a general cost-effectiveness analysis in relation to the security of ciphers applied to data bases has not been done. This objective is too ambitious to be fulfilled completely in one dissertation. We partially achieve it by first defining a measure for evaluating the security of file enciphering, and second by conducting an experiment which measures the overhead in using cryptographic transformations in a file system.

- d) Insight into Implementation Problems. The fourth objective of this dissertation, which is very important in applied research, such as this, is to specify the practical considerations involved in applying cryptography in a data base system. Some important design questions are not apparent unless at least a small scale system is implemented, or alternatively a simulation of the system is conducted.

In future chapters we show how the objectives of this dissertation are reached. Although the work centers around one topic, namely "the application of cryptography to data base security", the problem is attacked from several directions. This explains both the titles and the contents of the different chapters. The research is conducted on several levels. The conceptual/structural model is reported in Chapters 4 and 5. The design of part of this model is reported in Chapter 6. The implementation of part of the design is reported in Chapter 8. The cost-effectiveness analysis is done in Chapters 7 and 8. A more detailed description of the various chapters follows.

Chapters 2 and 3 together with Chapter 1 compose a complete introduction to this dissertation. This longer than usual introduction is needed since we are dealing with the interaction of two quite different areas: cryptography and data base security, and we need basic concepts from both areas. Most of the material in

Chapters 2 and 3 is not new, however some basic definitions and original examples are given. Also, the analysis of current data base models given in Chapter 3 motivates the development of our multi-level model of a data base. Therefore, even the knowledgeable reader is advised to read Chapters 2 and 3 in order to understand the rest of this dissertation. Chapter 4 presents the multi-level model of a data base and explains its relation to security. Chapter 5 describes the way in which cryptographic transformations are integrated into the multi-level model and identifies the different types of transformations which can exist in this model. Chapters 4 and 5 fulfill our first objective. In Chapter 6 the problem of designing a secure data base system based on cryptographic transformations is investigated. Several schemes are suggested for different types of protection policies (e.g. compartmentalized, hierarchical). The advantages and disadvantages of each scheme are shown and the implications for the data base designer are discussed. Chapter 6 fulfills our second objective. Chapter 7 describes our measure for evaluating the security of ciphers applied to files and data bases. Chapter 8 describes the experiment in which different ciphers are applied in a small scale file system, and the overhead associated with the use of these ciphers is measured. Chapters 7 and 8 fulfill our third and fourth objectives. Chapter 9 discusses several problems for future research. In particular, more research can be done in the "security measure" area. The Appendix contains some of the computer printouts which resulted from the experiment and are used in Chapter 8.

In the following chapters we try to give clear justification, clear definitions and enough examples to permit good understanding of the issues treated. We have avoided abstractions and "heavy formalisms" where they are not needed. The real life, practical implications of the issues discussed are of central importance.

CHAPTER TWO

Review of Cryptography

In this chapter we review the basic cryptographic concepts, describe several types of ciphers, and discuss their properties. We then review the measures suggested by Shannon for evaluating the security of ciphers [SHA49], and describe the changes in cryptography as a result of computer technology. In particular, the differences between "communication cryptography" and "data base cryptography" are discussed. An example of file enciphering is given which shows the strong connection between the data base structure and cryptographic transformations operating on that data. Finally, the need for a data base model for further investigation of this problem is shown, and motivation for the later development of such a model is given.

2.1 Traditional Cryptography

There are not many references about traditional cryptography, mainly because of the secrecy which wraps the entire area. Kahn [KAH67] gives a good history of the subject. Gaines [GAI56] describes the main pre-computer cryptographic and cryptanalysis techniques. Although Gaines does not use a technical language and stresses human intuition in cryptanalysis, she mainly uses letter frequency, common words and phrases and basic statistical parameters of language. These are still the main tools of cryptanalysis today. Some preliminary definitions from Kahn and Gaines will make subsequent discussion more intelligible. The "plain text" (p.t.) is the message to be put into secret form. The methods of "steganography" conceal the very existence of the message (also called a "concealment cipher"). The methods of "cryptography" do not conceal the presence of the message but render it unintelligible to outsiders. The unintelligible form of the plain text is called the "cryptogram" or ciphered text (c.t.). The transformation of the plain text into

a cryptogram is called "encipherment". The inverse transformation of the cryptogram into plain text is called "decipherment". The system which generates cryptograms from plain text is called the "cipher". Transforming a cryptogram to the plain text with no pre-knowledge of the cipher system is called "decryptment" (to distinguish it from decipherment). The science of "solving" ciphers in this way is called "cryptanalysis". "Cryptography" includes both cryptography and cryptanalysis.

Two basic types of ciphers are: "Transposition ciphers"--where the order of the letters (words) in the plain text is changed; and "Substitution ciphers"--where every character or group of characters is replaced by another character or group.

One distinguishes between "cipher" and "code". The latter is substitutional in nature and more semantically oriented. A "code book" is a book with many words or phrases and their corresponding codewords. In computer systems binary codes are the way we represent letters or numbers. A "cipher alphabet" is the list of equivalences used to transform the plain text into the secret form in a substitution cipher. When only one "cipher alphabet" is used we have a "monoalphabetic substitution". When more than one cipher alphabet is used in the same cryptogram we have a "polyalphabetic substitution". When a group of characters is replaced by another group of characters we call it "polygram substitution". Every cipher employs a "key" as an essential part of the system. This key determines the order of the letters in a "transposition cipher" or which cipher alphabets to use in a "polyalphabetic substitution". For the same cipher procedure and the same plain text, we get different cryptograms with different keys. Finding the key (rather than finding the procedure) is the main problem of the cryptanalyst.

We will now describe some of the traditional ciphers and their corresponding cryptanalytic methods.

One of the first monoalphabetic ciphers is the CAESAR cipher. In this cipher every letter is replaced by another letter according to a constant shift.

example:

(Plain text)	p.t.	IBMSYSTEM	
(Cipher text)	c.t.	HALRXRSDL	shift -1

Even if we don't use constant shift but a general mono-alphabetic substitution, this is a simple cipher to "break". Using frequency analysis we first guess what is the substitution for "e" and "t". Then, using "digram and "trigram" analysis (Gaines [GAI56]) we find the substitution for "h" (digram "th") and for "r" (digrams "ER", "RE"). Using least occurring letters we find the substitution for letters such as: Q, X, J, Z and knowing that neighbors to these letters may be only vowels we find the substitution for other vowels. So this cipher is easy to break because it preserves all the underlying language statistics. A harder cipher to break is the "polyalphabetic" cipher. The most famous one is the "Vignere table". In this case we have 26 possible cipher alphabets and the key determines which alphabet we use every time.

example:

key	BEDB EDBEDBED BE DBEDBED
message	SEND SUPPLIES TO MORLEYS
cipher	TIQE WXQTOJIV US PPVOFCV

We use here periodically 3 different alphabets. To decrypt this cipher, frequency analysis is not enough because "E" for example is replaced once by "I" and another time by "F". The main drawback of this cipher is the periodicity of the key, because we use the same key over and over. We can first find its length and then, if we have enough ciphered data and using language analysis tools, we can convert the problem to one of monoalphabetic substitution. Intuitively it is clear that the longer the key (relative to the length of the message) the harder it will be to "break" the cipher.

In the simple transposition cipher we write the message in rows and read it by columns according to some key.

example: (columns are read off in the order of the digits above them and presented as five letters groups)

Key	3412576		
	EXAMPLE		
	FORATRA		
	NSPOSIT		
	IONXXXX		
Message:	ARPNM	AOXEF	NIXOS
	OPTSX	EATXL	RIX

Since the message is of length 28 there are virtually 28! possible permutations. But someone who knows the "route" (the fact that we write in rows and read in columns) has only 7! (length of key) possibilities. However, in general we don't know the route, so such a simple transposition is quite effective. The tools to break it are the same language analysis tools, mainly digram and trigram analysis, since they allow us to reorder the columns (rows) in the right order.

Not surprisingly, a combination of simple substitution and simple transposition (called "combination cipher") is very effective and not at all simple to solve (Van Tassel [VAN69]).

A recent cipher is the "generalized homophonic cipher" (Stahl [STA74]). In this cipher every letter of the source alphabet can be replaced by any one of several letters from the cipher alphabet, according to its original frequency. Stahl shows how to choose the cipher alphabet, and the number of substitutions per letter so that the resultant single frequency curve or even the digram or the trigram frequency curves are close to "flat". The main advantage of this cipher is that it is designed to nullify the basic tools of the "enemy" cryptanalyst, the language analysis tools, because it is designed to hide (or flatten) all the known statistical properties of the language.

Among the polyalphabetic substitution ciphers the most important one is the Vernam cipher. In this cipher we have only two letters in the alphabet: [0,1] and two cipher alphabets: [0,1] and [1,0].

The operation of the system is based on the XOR (exclusive OR) operation and therefore it is well adapted to digital hardware.

example:

key	01010	
p.t	11111	+encipher
c.t	10101	
key	01010	
c.t	10101	+decipher
p.t	11111	

The Vernam cipher has the advantage of being involutory, i.e. the same procedure is used for enciphering and deciphering (in this case XORing with the right key). This makes the implementation of a system based on such ciphers very simple.

Shannon [SHA49] proved that if we use a truly random key with length equal to the length of the message then the VERNAM cipher would be completely secure and "unbreakable". Even before Shannon, this fact was known, but the problem was always how to produce and transfer a truly random key. Practical solutions were: long tapes or key memories (Skatrud [SKA69]) or "good" algorithms to produce random keys from relatively short seeds (Carroll and Mclelland [CAR70]). The problem with these methods was that with a large amount of data we would get repetition of the key in the first case, or enough information to find the "seed" of the "random key" in the second case. Therefore, in reality it was very hard to create and maintain a completely secure system.

Recently, a new type of cipher has been suggested. This is the block cipher. In this case, a block of characters (bits) is enciphered as one unit by a very complicated transformation, so that most of the relations which exist between characters in the "clear" block are destroyed.

This cipher is usually involutory to permit the same algorithm (or hardware) to be used for enciphering and deciphering. Such a cipher was recently published by NBS [FED75]. It is based on a cipher suggested by Feistel [FEI73].

We have seen then some of the different types of ciphers. We need some way to evaluate the security of these ciphers. This is discussed in the next section.

2.2 Measures for Cryptographic Security

The most important paper on evaluating the security of ciphers was published by Shannon in 1949 [SHA49]. This is a mathematical, information theory oriented paper which is still the basis for most theoretical research in cryptography. We postpone the mathematical discussion of Shannon's measures to Chapter 7. Here we just want to review the main concepts.

Shannon defines two important concepts for such a measure: the "unicity distance" and the "work factor". The unicity distance is the theoretical minimum amount of ciphered data that the "enemy" has to acquire in order to break the cipher. The unicity distance depends on the type of cipher used and on the redundancy of the "source"

language. For example, for English text and monoalphabetic cipher, Shannon computed a unicity distance of 27 characters. In reality, however, "the enemy" will acquire more data than the "unicity distance" and maybe even some "clear text". In this case, it is important to know the "work factor", that is, how much effort and resources (time, money) the enemy has to invest in order to break the cipher, and to compare it to the "value" of the information. If the value of this information is very high, then ciphers with high "work factors" are needed.

A designer would like, of course, to make the "work factor" as high as possible. Shannon suggests a way to achieve a high "work factor" by using mixed transformations or a combination cipher. If the design is correct, then the statistical properties of the language are "confused" and "diffused" over very long pieces of data, and this makes the work of the cryptanalyst extremely difficult. The concept of "work factor" is extremely important in the case of file or data base cryptography. This is because the large amount of data in files forces the assumption that the "enemy" has more data than the "unicity distance". Therefore only a high work factor gives an adequate protection. This will be discussed further in Chapter 7.

The amount of security is not the only measure which is important in a cryptographic system. Shannon suggests five criteria to evaluate a cryptographic system: a high degree of security, smallness of the key, simplicity of enciphering, low propagation of errors, and little expansion of the message size. Not all these criteria can be achieved simultaneously and every designer has to choose his tradeoff. However, Shannon showed that if any one of these criteria is removed, then the other four can be satisfied. Since 1949, computers and advanced technology have changed the relevance of some of Shannon's criteria. A discussion of these criteria in the computer age can be found in Martin [MAR73] (P.209). For example, simplicity of enciphering which was important for a human operation, is generally not so for the computer or special purpose digital hardware. Other criteria which are important in file or data base systems will be discussed in Chapter 6. One general principle is true even in the computer age. This is the notion that the security of a cryptographic transformation should reside with the key. As stated by Skatrud [SKA69]: "A key must withstand the operational strains of heavy traffic. It must be assumed

that the enemy has the general system, therefore the security of the system must rest with the key".

This general principle is strengthened by the recent public disclosure of the NBS encryption algorithm [FED75]. It is fundamental for any secure system based on cryptography, and it will also be our general principle throughout this thesis.

2.3 Applying Cryptography to a Computer System

In this section we show the differences between applying cryptography to messages in a communication system and applying cryptography to data stored in a computer system. We also give an example of enciphering a file subject to constraints typical of the computer environment.

2.3.1 Cryptography and the Computer Environment

The goal of most of the past work and research done in cryptography was to protect the transfer of information through communications lines. Clearly, in a time sharing system or a computer network environment the problem of secure communication is very important, but no less important is the security of the common data base or the file system. However, there are significant differences between applying cryptography to a communication system and to a file system. Turn [TUR73] discusses some of these differences. (We will use the "+" to indicate an advantage in case of files and "-" to indicate an advantage to a communication system.)

- + . In communication systems the encoding and decoding operations are done at two different locations and two copies of the key are required, while in file applications these operations are performed at the same location and only one copy of the key is needed.
- . A specific communication usually involves one user, while a file may be shared between many users with different access and processing authorizations.
- . In communication links, the message remains transformed for a very short time interval since encoding and decoding operations are performed almost simultaneously. In file applications this time interval may be days or months.

- . The transformed records in files may be subject to selective changes at unpredictable time intervals and at unpredictable frequencies, while the message in a communication link is negligibly changed in transit.

- . A change of the privacy transformation keys in a communication application is a simple replacement of the old key with the new one. In data files this entails reprocessing the entire file of records, or maintaining an archival file of previously used keys and associated indices.

+ . A common-carrier communication link normally uses certain signal patterns for internal switching control (e.g. control characters). These should not appear in the cipher text form of messages. There is no such problem in files.

+ . Communication links often have high error rates while errors in the file system are amenable to detection and control.

+ . There is much less processing capability available at terminals than in the central processor.

- . In communication systems totally random keys can be used once and discarded, but in file systems they must be stored or means provided for their generation later, thus reducing the level of security of file systems.

- . Most file systems require "selective retrieval", that is the ability to retrieve specific records or fields. It is desirable to have these records enciphered without a dependency on other records. This limits the possible application of very long pseudo-random keys.

We see then that there are significant differences between "communication cryptography" and "file cryptography". The last one needs further investigation. One of the first on-line, file oriented systems which has used cryptography is the LUCIFER system.

2.3.2 The LUCIFER System

An example of a system which tries to protect both the communication and the file system can be found in an IBM system [SMI72] which uses the IBM LUCIFER. The cipher used is described by Feistel [FEI73]. It is a block cipher

which is a combination of transposition and nonlinear substitution in several layers, and has a very high "work factor". This system has several interesting points in its implementation.

- a) It is an on-line communication system. The user uses it through a remote terminal and the system tries to validate the legal user and to protect data in files, in the main processor and in communication lines.
- b) Data is enciphered and deciphered in the terminal by special hardware called LUCIFER and by software or hardware in the main processor. So, information in communication lines is protected. LUCIFER is simple to use either in ciphered or non-ciphered mode. The hardware is the same for different terminals and only the key changes.
- c) Special authentication procedures are used on both sides of the line to assure the computer that there is a legal user on the other side and to assure the user that there is the computer (and not the "enemy") on the other side.
- d) There is a special key for reading (updating) a confidential field. This key is not available at all in the computer and the data flows in its ciphered form from the file up to the terminal.

Some of the problems with this system are:

- a) Inability to "process" ciphered text.
- b) Most of the information (beside the confidential field) is in plain text in the processor main core.
- c) No arrangements for file and data sharing are made.
- d) There is a problem of controlling the use of the "special key" which is not known to the system at all.

2.3.3 The Processing Problem

The main disadvantage of the LUCIFER system is that we have to decipher the information in order to process it, and we may have to do it frequently. The enciphering/deciphering process might entail excessive overhead. We would then like to "process" ciphered data for two reasons:

- 1) If only the ciphered data is in memory, then the system is more secure.
- 2) We save the processing time of enciphering and deciphering the information.

The word "process" may mean different operations such as: updating, retrieval, sorting and merging, numerical processing, etc. Let us look first at the retrieval operation. Suppose we want an answer to a query such as:

NAME? CITY = 'COLUMBUS'

which means: give all the names of people who live in Columbus. If we don't have the information directly (for example, by an inverted file) we have to search the file. But if we don't want to decipher the whole file, then we need (for the compare operation) all the ciphered forms of COLUMBUS in all records to be identical. That is, we have the condition that equal plain text will result in equal ciphered text. This immediately eliminates the application of the important polyalphabetic substitution ciphers (unless every time we cipher 'COLUMBUS' we start from the same point in the key). We call such a cipher for which:

$$Pt_1 = Pt_2 \iff Ct_1 = Ct_2$$

(Pt - plain text, Ct - ciphered text)

a "retrievable" cipher.

Every monoalphabetic cipher is clearly a retrievable cipher, but as we already know, monoalphabetic ciphers are not very secure.

For a sorting operation we would like to be able to use the existing efficient SORT packages for sorting a ciphered file. For this we must have:

either $Pt_1 > Pt_2 \iff Ct_1 > Ct_2$

or its dual $Pt_1 > Pt_2 \iff Ct_2 > Ct_1$

and we call this cipher a "sortable" cipher. (*)

The hardest problem is the "processing" of numerical data. In many cases numerical information needs the most security, but is also used in calculations, including logical and arithmetic operations. Also, it is feasible that the "enemy" will know the range of these numbers, which makes the problem harder.

If we request, for example, that

$$Pt_1 + Pt_2 = Pt_3 \iff Ct_1 + Ct_2 = Ct_3$$

then a possible transformation is:

$$Ct = a \cdot Pt,$$

where 'a' is a number. However, this is a very easy cipher to break.

In algebraic terms, if we consider the numbers as a groupoid then we would like to find a multiplication preserving relation between the plain text numbers and the ciphered numbers.

It is intuitively clear that there is a tradeoff between a very secure cipher and a cipher with "processable" properties. However, as we see in the next section it is possible to design "processable" ciphers and still not lose too much security.

(*) A sortable cipher is retrievable.

proof: Assume $Pt_1 = Pt_2$, if $Ct_1 > Ct_2$ (i.e. $Ct_1 \neq Ct_2$)

then $Pt_1 \neq Pt_2$. Therefore $Ct_1 = Ct_2$

2.3.4 File Enciphering

In order to explain our ideas of "processable" ciphers and enciphering files we give an example of enciphering a typical file.

Assume we have a sequential or direct file with n fixed length records, where each record has the following structure:

id f_1 f_2 - - - - f_m

id is a unique identifier for each record and f_i is a field of fixed length l_i . Some of the fields may be strings of characters, others may be numerical.

We would like to suggest a secure cryptographic transformation for the entire file which has the following characteristics:

- a) Accessing - Retrieving record i will not require accessing another record j (for reasons of efficiency). This dictates that a ciphered record $p(i)$ will contain all and only the fields of the "clear" record i .
- b) Processing - All the fields must be "retrievable." Some fields may be "sortable." Other requirements might exist on numeric fields.
- c) Simplicity - The same enciphering algorithm must be applied to each record in the file.

Clearly, in order to fulfill b) the field structure must be preserved. A "retrievable" substitutional cipher applied to each field may not be secure enough. On the other hand a combination of transposition and substitution ciphers will be quite secure (Van Tassell, [VAN69], Shannon, [SHA49]). So permutation of the fields in the record is useful for providing security.

Consider the following scheme: To each record we add a transposition key, and to the whole file we add one record of substitution keys -- we call it record S.

The record structure is now:

id d_1 d_2 - - - - d_m f_1 f_2 - - - - f_m

d_1 d_2 - - - - d_m

Transposition Key

The transposition key is composed of m binary numbers d_i , where d_i points to the displacement of field f_i from the beginning of the record. Even before the enciphering we had to know this displacement in order to access the field. Now this displacement may change, and we get it from the key in the record.

The additional record S has the following structure:

id	k_0	k_1	k_2	- - -	k_m
----	-------	-------	-------	-------	-------

$k_0, k_1, \dots, k_m$ are substitution keys, where k_0 is used to encipher the transposition key. The simplest case is where these keys are used as Vernam keys and each k_i has the same length as the field f_i .

The enciphering/deciphering algorithm is as follows:

enciphering:

- a) choose permutation of the fields
- b) compute the transposition key
- c) encipher each field, including the transposition key, with its own substitution key

This system is very secure. "Breaking" one record alone is theoretically impossible, because the Vernam keys have the same length as the field to which they apply.

To break the whole file we first need to break the transposition key. This key has $m!$ possible values and usually $m!$ is much greater than the number of records in the file (e.g. $m = 10$ and $n = 10,000; m! \sim 3,700,800$). If we choose these transpositions at random, then the enemy has no knowledge of the statistics of the keys. It follows that any substitution cipher applied to this field will be secure.

Does this scheme fulfill our requirements?

Almost. Requirements (a) (selective retrieval) and (c) (simplicity) are fulfilled. In order to retrieve field f_i in its ciphered form we must first decipher the transposition key. So it is a compromise to requirement (b), but a small one, since the overhead of deciphering the transposition key is small and then each field can be processed in its ciphered form (because the Vernam cipher is "retrievable").

The Vernam cipher is not good for "sortable" fields. For this we have to use another substitution cipher (e.g. CAESAR). There are several comments to be made about the scheme described above:

- 1) The transposition is important if we want to be able to process ciphered fields and still provide high security.
- 2) $m!$ must be much greater than the number of records in the file in order to provide good security. Otherwise, we have to add some "null" fields (or split others).
- 3) The extra storage needed for the transposition key is not always wasteful. If the fields are of variable length we must include their length as part of the record. The transposition key will save us specifying this length. (We can compute it as difference of two displacements.)
- 4) Techniques other than substitution may be applied to each field, for example a "concealment" cipher.
- 5) The system seems simple and secure but more research is needed to evaluate its security and efficiency.

A variation of this scheme of encipherment for a sequential file, as well as other schemes, will be discussed (as part of the experiment) in Chapter 8.

2.4 Cryptography and Data Bases

A major conclusion from the example above for file enciphering is that there is a strong connection between the file structure, the data representation, and the cryptographic transformations operating on this data. This connection is even "stronger" in the case of data base systems. The reason, of course, is the complexity of data base systems. Here, we need to answer questions such as: what is the effect of these transformations on the physical structure of the data base? and on its logical structure? What control over these transformations should the data base administrator have? What is the relation of cryptographic transformations to other access control mechanisms?

To answer these questions we need a framework - a data base model, in which such questions can be analyzed. Since we first need to review data base security models, we postpone further discussion of this question until Chapter 3.

2.5 Summary

In this chapter we have briefly reviewed the topic of cryptography. We first introduced the basic concepts and then discussed some of the well-known ciphers. We then discussed problems of cryptography in the computer environment concentrating on the area of cryptography for file or data base systems. Finally, we motivated the need for developing a data base model for further investigation in this area.

CHAPTER THREE

Review of Data Base Security

In this chapter we review the subject of data base security, starting with the well-known "access matrix" data security model. We then show that data base security should be dealt with separately from Operating Systems security and then define the basic concepts of data base security. Using these concepts we analyze current data base models and show their disadvantages in relation to security. Finally, we discuss cryptography and data bases and motivate the development of a data base model for further investigation in this area.

3.1 General Data Security Models

Research in data security and research in data base systems have only been combined recently. In most current research on data security, the whole system is treated, and no special considerations are given to any part of the computer system. We call this approach the unified approach, since all the components of the computer system are treated "uniformly". The most famous model of data security, of the unified approach type, is the access matrix model of Graham and Denning [GRA72]. In this model a protection system is composed of 3 parts:

- 1) Set of objects - an object being an entity to which access must be controlled. An object can be a program, a file, a device, etc.
- 2) Set of subjects - a subject being an active entity whose access to objects must be controlled. A subject may be regarded as a pair (process, domain), in which a process is a program in execution, and a domain is the protection environment (context) in which the process operates. The connection between users and processes is done through the procedure of user identification.

3) The rules which govern accessing of objects by subjects.

The type of access each subject has to each object can be represented by an access matrix. A portion of an access matrix is shown in Figure 2. For example, since $A(S_1, F_1) = \text{READ}$ then subject S_1 has a READ access to object F_1 (notice that subjects can also serve as objects). The enforcement of the access matrix specifications is done by the use of Monitors, where each type of object has its own monitor. A major problem here is the way in which the access matrix entries can be changed. The third component of the model, the rules, are used for this purpose. Graham and Denning suggest several types of rules, such as: transfer, grant, or delete which are used to modify the access matrix. They discuss at length the problem of "untrustworthy" subjects and how to prevent such subjects from transferring access rights to unauthorized subjects. For that they introduce the '*' notation. For example in Figure 2 since S_2 has SEEK^* access to D_2 , then S_2 can transfer the access "SEEK to D_2 " to other subjects.

The access matrix model has become very popular and a basis for much research in the area of data security. A major problem with the access matrix model is the representation in storage of the matrix. Since the number of objects can be large, and since the matrix is probably sparse, it is impractical to represent it as a two-dimensional matrix. Two main approaches exist. In the first approach the access matrix is stored by rows, i.e. each subject is associated with a list of objects to which it has access (For subject S the list is:

$$\{(X_1, A[S, X_1]), (X_2, A[S, X_2]) \dots\}$$

This list is usually called a capability list or C-list (see Lampson [LAM69]). The advantage of this method is that only the C-lists of active subjects need to be resident in core, so much overhead is saved. One of its disadvantages is that it is difficult to change access rights which relate to a single object (e.g. deleting an object) because of the need to search the reference to this object in all possible C-lists. A variation of the C-list approach called "authority item" was suggested by Hsiao [HSI73]. The concept of capability has been generalized to something similar to an address. That is, any access to an object is allowed only if the right capability exists in the C-list (environment) of the subject. The capability approach has become very popular

OBJECTS

	SUBJECTS			FILES		DEVICES	
	S ₁	S ₂	S ₃	F ₁	F ₂	D ₁	D ₂
S ₁	CONTROL	BLOCK WAKEUP		READ WRITE*		SEEK	OWNER
S ₂		CONTROL	STOP	OWNER	UPDATE	OWNER	SEEK*
S ₃			CONTROL	DELETE	OWNER EXECUTE		

Figure 2 - A Portion of an Access Matrix

and several research projects are based on this approach (e.g. Fabry [FAB74], Jones [JON73]). Present computer systems which use variations of the capability approach are the HYDRA system [WUL74], and System 250 [COS72].

The second approach for access matrix representation is by columns. This means that each object has associated with it the list of subjects which have access to this object, and their access rights. (e.g. for object Y the list is:

$$\{(S_1, A[S_1, Y]), (S_2, A[S_2, Y]) \dots\}$$

This approach is called the access list approach. It is used, for example, in the MULTICS system for the protection of one type of objects called segments. The major disadvantage of this method is the difficulty of making changes in the access rights of individual subjects (e.g. "deleting" a user from the system).

The models described above are not the only data security models which have been developed. A good survey of the other models is given in Popek [POP74b]. For the work we will need the concepts of C-lists and Access lists which will be used in relation to the design of cryptographic schemes in Chapter 6. In general, as we will see in the next section, the unified approach is not always adequate, and some part of the computer system, such as the data base system, may require special considerations.

3.2 Data Base Security vs. Operating System Security

The main shortcoming of the "unified approach" to data security is that looking at each possible element of the data base (e.g. the pair field-value) as an object requires an enormous access matrix. However, there are additional drawbacks.

Because data base systems are much more complex than file systems, their security problems are more complicated. In a data base, protection may be required from file and record levels down to the field level. User protection requirements in a data base are more complex than protection of a full file (or segment) and protection specifications based on complex boolean expressions may be needed. Access paths to data have to be protected, and sensitive parts of the data base system, such as the Data Definition Language (DDL) require special protection.

The unified approach mentioned above is therefore not well adapted to handling problems in data base systems. We believe that the security problems of data base systems should be separated from those of operating systems. (A similar view is held by Minsky [MIN74] and by Tsichritzis [TSI74]). However, since data base systems use the operating system services, the two problems cannot be completely disconnected. This is also reflected in the following statement from Tsichritzis [TSI74].

A secure data base can obviously be implemented on a secure operating system. However, a secure operating system does not imply a secure data base.

We therefore favor a levels approach, in which the operating system is in an inner level of the data base system, and the hardware/firmware is in an inner level of the operating system. This is shown schematically in Figure 3.

Figure 3 - The Levels Approach

Since the outer levels use the services of the inner levels, each level of the system must be correct and secure in order to assure the security of the next outer level. (A similar approach was used by Dijkstra [DIJ68] in the design of the THE system.) Since we are interested in data base systems, our assumption is that the hardware and the operating system are correct and secure. This is not as a strong assumption it might seem at first. Since problems of files and data bases have been removed from the operating system domain, the operating system is smaller and easier to verify, but the problem of data base protection is more complex.

In this thesis we concentrate on the security problems of data base systems. We therefore need first to define the basic concepts of this subject. This is done in the next section.

3.3 Basic Concepts in Data Base Security

Looking at the literature on data security, we find some degree of confusion about the basic concepts. We will use the terms security, protection, and access control interchangeably and assign them the following meaning by McCauley [McC75], "The process of determining the authorized users of the data base and of determining which access may be permitted and which would be denied."

We first need to distinguish between the protection specification and the protection mechanism in a computer system. The protection specifications (denoted by PS) are the translation of management privacy views into exact specifications. The protection mechanism (denoted by PM) is the mechanism to execute correctly the protection specifications and to assure that any protection violation of these specifications will be detected. We define a "security flaw" as a protection violation which is not detected by the protection mechanism. A "security flaw" can be defined either inside or outside the domain of the standard protection mechanism. Both failure of the protection hardware and wiretapping of information can cause a "security flaw". In the design of protection mechanisms we must always have to consider the chance of a "security flaw". A very important question is: who is able to change the protection specifications and what are the rules and mechanisms for these changes? Graham & Denning [GRA72] showed the complexity of this question. We view the rules to change the protection specifications as part of the protection specifications themselves and the mechanisms to change them as part of the protection mechanism.

Different systems use different protection mechanisms. Most mechanisms have two parts: the protection procedure and the protection data. The protection data, not to be confused with the protection specification, is data internal to the protection mechanism. Examples are passwords in case of the passwords protection mechanism, or tags in Friedman's model [FRI70]. The protection procedure is analogous to programs or procedures in programming systems and is the coded form of the protection mechanism algorithm. An important example of such a two-part protection mechanism is cryptographic transformation. In this case the transformation algorithm is the protection procedure while the cryptographic keys are the protection data. The analogy to programming systems can be carried further as follows:

Programming Systems

Procedure
Temporary variables
Input data
Output data

Data Security

Protection procedure
Protection data
Protection specifications
Protection decision
(grant or deny access)

An important problem in programming systems is that of binding, i.e. when do we bind the program to its input data? The issues of flexibility vs. efficiency in program binding are well known and we do not discuss them here (e.g. see Freeman [FRE73] or Elson [ELS73]). However a similar situation exists in data security, i.e.: when should the protection specifications be combined with the protection mechanism? We define protection binding as the process in which the protection specifications and the protection mechanism are integrated. Protection binding is discussed extensively by Conway, Maxwell and Morgan [CON72] (without using that term). They distinguish between two levels of binding: at compilation time for data independent protection and at run-time for data dependent protection. Additional levels of binding can be defined. Jones [JON73] distinguishes between three levels of binding (in operating systems). Conway, et. al. also discussed the questions of efficiency vs. flexibility. However one problem, related to binding, which exists in data security and not in programming systems is the problem of a "security flaw". Early binding is more efficient and less flexible but it is also less secure since once the protection binding is done, any change to the data after binding (e.g. to the object code after protection binding at compilation time) can cause a "security flaw". While such changes are prohibited by the protection mechanism, the probability of

failure, which is never zero, increases. An example of a late binding in which the chance for a "security flaw" is small, is Friedman's authorization model [FRI70]. To summarize, there are three issues involved with protection binding: efficiency, flexibility and the probability of a "security flaw".

As an example of the use of the concepts above consider the protection mechanism in the OS/360 file system [IBM72]. It is a password protection mechanism where a file can be accessed if and only if the right password is given by the user during OPEN. The protection procedure then is part of OPEN. The protection data is the list of passwords. The protection specification is the distribution of passwords between users according to the privacy decisions: which user has access to which file. The passwords here have a double role: both as the protection data and as the way to express the protection specifications.

To summarize this section, we have defined some basic concepts in data security which we will use in the rest of this thesis. In the next section we will use these concepts in the analysis of current data base models.

3.4 Analysis of Data Base Models

Now we analyze several known data base models from the security point of view, using the concepts that we have defined in the last section. We can distinguish between two types of data base models. The first type consists of general data base models in which security is not the central issue. Examples for this type are: the relational model [COD70], the CODASYL model [COD71] and the entity-set model [AST72]. The second type consists of less general, security oriented models. Examples of the second type are: Hoffman formulary model [HOF71], Hsiao & McCauley attribute based model [McC75], and Friedman authorization model [FRI70]. Among models of the first type, the best known is the CODASYL model which also addresses the security problem directly by providing several protection tools. It has been adopted as the basis for several commercial data base systems (e.g. UNIVAC's PMS [UNI73]). We therefore analyze this model first and in more detail than the others.

3.4.1 The CODASYL Model

The CODASYL model uses several protection tools.

- a) The main one is through the Data Definition Language (DDL). The DDL provides protection by privacy locks which are specified in the Schema (or Sub-Schema), and privacy keys which must be provided by a run-unit which seeks to access the data protected by privacy locks. Privacy locks can be defined on each of the following data types: Schema, area, record, data item, sets. In this case we can say that the protection specifications are equivalent to the distribution of privacy keys between users (similar to OS/360 passwords method) while the protection mechanism and the protection data are part of the DDL (Schema or Sub-Schema).
- b) The Sub-Schema DDL can be used to describe only part of the data base accessible to a group of users. In this case, gross protection specifications are implemented as part of the Sub-Schema DDL.
- c) On the data-item level the DDL provides the ENCODING/DECODING clause which can be used to implement cryptographic transformations.
- d) The data base procedures can be used for protection purposes, but since no detailed description of these procedures is given, we will not discuss them further. However, they may be used for implementing some of the ideas presented later.

Although the availability of these tools can greatly enhance the security of a data base, we find several disadvantages with the CODASYL scheme. The first one is philosophical and is common to other data base systems. The others are more technical in nature.

The first disadvantage is excessive centralization of protection specifications, mechanisms and control in the hands of the Data Base Administrator (DBA) and physically in the DDL part of the data base. In the CODASYL model, the DBA has complete control over the security of the data base. The same thing is suggested by Share & Guide [SHA70], "the DBA via the DDL must have the facility

to define the security requirements for each unit of data he defines." While, in some instances this kind of centralization may be required, cases are conceivable in which such centralization is a disadvantage. If the DBA or the DDL are compromised the security of the entire system is compromised. From the operational point of view, the DBA should not have to know every protection specification in the system, (e.g. some delegation process as in [GRA72] is usually needed) nor should he have to know the content of every file or record in the data base. We believe that the concept of the DBA was created mainly for the efficiency of centralized control on the structure and the performance of the data base. However we think that total knowledge by the DBA of the structure of the data base, which is essential for the correct operation of the data base system, does not imply total knowledge of the content of each field or the exact coding of each field. (This is essentially the "need to know" concept.) Furthermore, the DBA should not be concerned with individual access rights (e.g. students) but rather with gross protection specifications and should concentrate instead on the maintenance and performance of the data base as a whole. Even if, from an administrative point of view centralized control is needed, there is no reason to centralize the protection mechanism. Currently, if the schema DDL (which is a physical part of the data base accessed by all users) is compromised, the security of the whole system is compromised. Spreading the protection mechanisms throughout the system, or providing several protection mechanisms which complement and check each other will decrease the danger of such global compromise. We do not advocate decentralization over centralization but we claim that the ability to decentralize must be present in secure data base systems. Others have advocated decentralization too. In the design of MULTICS one functional objective was to provide for decentralization of protection specifications [SAL74]. Saltzer also mentions the danger of an overprivileged System Administrator. Decentralization was also found desirable in studies made by IBM and TRW as reported by Hoffman [HOF75].

The technical problems of the CODASYL model arise because the main protection mechanism, i.e. the privacy keys/locks mechanism is too simple and not general enough. There is no easy way to protect individual records (vs. record types). There is no easy way to specify and implement value or data dependent protection or to specify and implement complex, boolean expression type, protection specifications (see [McC75]). No tools are given for implementing context protection or history dependent protection

(see Nee [NEE73]). Another disadvantage of this mechanism relates to the time of the protection binding. Since the latest protection binding is done at the schema DDL level, no further protection checks are made, and the danger of a "security flaw" during accessing of the physical data base exist.

A very effective protection mechanism, namely cryptographic transformations, is exploited very little in the CODASYL model. The data item enciphering which can be implemented using the ENCODING/DECODING clause is usually not secure enough, and more complex transformations are needed. Also no consideration is given to the problem of "processable" ciphers.

The next chapter describes a model which overcomes most of the disadvantages mentioned above.

3.4.2 Other Models

Another model is the INGRES relational data base system of Stonebraker & Wong [STO74]. They suggest a pre-censorship protection mechanism. It consists of modifying queries according to protection specifications. Suppose, for example, that a user is allowed to see the SALARY fields of employees in department X, only if the value of this field is less than \$10,000. Suppose this user issues a query: NAME, SALARY? (DEPT = X); which means: give the names and salaries of all people who are working in department X. This query will be modified in the following way: NAME, SALARY? (DEPT = X) & (SALARY < \$10,000) so that only the allowed records will be retrieved. There are two disadvantages to this method:

- a) only protection specifications which can be expressed via the query language are allowed, and
- b) the protection binding is done at the query translation time and no further protection checking is done.

In models of the second type we see that the major difference between them is the level in which protection binding is done. In Hoffman's Formulary model protection binding is done by the ACCESS procedure after the translation of the user queries, but before accessing the physical data. (This model allows also for a SCRAMBLE/UNSCRAMBLE procedure but no further details on their implementation is

given). In the Hsiao/McCauley [McC75] model the protection binding is done during the time of accessing. The protection specifications are part of the directory entries and a user is allowed to access data through specific directory entries only. That is, if the user is not allowed to see a record with a specific keyword X, then he does not have access to the directory entry containing this keyword. (Other aspects of the Hsiao/McCauley model are discussed in Chapter 6.) In the Friedman authorization model protection binding is done during access to the physical data, by checking protection "tags" which are part of the physical data.

We see that in the models above, different protection mechanisms are implemented in different levels of a data base. The existence of several levels of a data base is well known but its relation to security has not been yet recognized. The main theme of the model in Chapter 4 is the existence of several data base levels and the possibility of spreading the protection specifications, and mechanisms throughout these levels.

To summarize, in this section we have analyzed some data base models from the security point of view. We saw that the CODASYL model suffers from excessive centralization and from lack of generalized protection mechanisms. The more specialized security oriented models implement protection on one data base level only, while other levels of the data base are ignored. The model presented in the next chapter addresses these shortcomings.

3.5 Cryptography and Data Bases

Although the possibility of enciphering is mentioned both by the CODASYL model [COD71] and by Hoffman's formula model [HOF71], a complete model which includes cryptographic transformations as an integral part of the data base system is not presented. This motivates our development of such a model.

We repeat here (see Section 2.4) the major differences between "communication cryptography" and "data base cryptography":

- a) The problem of selective retrieval - because files are usually organized so that selective retrieval of records can be achieved, it is very desirable that enciphering (deciphering) of record i will not depend on another record j.

This constraint prevents use of the popular Vernam Cipher using pseudorandom number generators of a very large period. Such a generator will be usually used for enciphering large quantities of data (e.g. the whole file) and will have to include more than one record in the enciphering process.

- b) The long "life" of the data - data in data bases usually resides there for relatively long periods. Therefore the very popular method of changing the cryptographic keys often, cannot be applied since it may require a complete reprocessing of the data base or a large part of it.
- c) The processing problem - data in files and data bases is stored for processing purposes. It would be very desirable if we could process the ciphered data in the same way we process the "clear" data. The reasons for this are that the system is more secure if only ciphered data is processed and the overhead of enciphering/deciphering every time we access the data is saved. We would like therefore to design "processable" ciphers. An example of a file enciphering which provides some degree of "processability" was given in Section 2.3.4. It was also shown that in the case of a "processable" cipher, applying the cryptographic transformation on the data item level only may not be secure enough.

Given the constraints above, it is clear that the subject of data base cryptography is strongly connected to the subject of data base organization, representation and accessing. None of the current models of data base systems address this problem directly. In the CODASYL model, for example, we are faced with the following questions:

- a) To which level should the cryptographic transformations belong? To the physical structure, to the logical structure, or to the mapping between them?
- b) Should the Data Base Administrator have complete control of the cryptographic transformations? Similarly should the keys for these transformations be part of the system (e.g. in its data definition language) or should users have only some of the cryptographic keys?

- c) Should cryptographic transformations preserve or destroy the structure of the data base and what are the advantages and disadvantages in each case?
- d) What is the relation to other protection mechanisms? Should they complement each other and how should it be done?

One of the objectives of our research is to answer these questions. However, in order to answer them we need a framework--a data base model in which the security problems and their relation to the data base structure are clearly identified. Such a model will be presented in Chapter 4.

3.6 Summary

In this chapter we showed why security problems in data base systems should be discussed separately from those in operating systems. We defined some basic concepts such as: protection specifications, protection mechanism and protection binding. Using them we analyzed current models of data base systems and pointed out some serious disadvantages related to security in general, and to the problem of centralization in particular. We then examined cryptography in relation to data base systems and showed the need for a data base model in which the associated problems can be clearly identified and discussed. This justifies the development of such a model in Chapters 4 and 5.

CHAPTER FOUR

A Multi-level Structured Model of a Data Base

4.1 Introduction

The discussion in Chapter 3 showed two important facts, namely that data bases have a multiple-level structure and that protection mechanisms and specifications need not be implemented in one level only. In this chapter we carry these ideas further and develop a model which allows spreading of the protection mechanisms and specifications in all these levels, and in which protection binding can be performed in each of these levels.

The existence of several data base levels is well known. Usually a distinction is made between the logical level (structure) and the physical level (structure). Sibley & Taylor [SIB73] give a good discussion of these levels and the mapping between them. In the CODASYL model we can distinguish three levels: the Sub-Schema and Schema levels which define the logical structure and the storage level which defines the physical structure. A four level model known as the entity-set model was suggested by Senko, et. al. [AST72]. Another four level model, similar in concept to the one we suggest here, was suggested by Sibley [SIB74]. Research has been done on the formalization of these levels and the transformation between them, directed mainly toward data translation [YAM74] or data base simulation [SCH74]. Though we could have developed most of our notions in the framework of one of these models, we prefer to develop our own terminology and structure for two major reasons. First, it is better adapted to presenting the security point of view in which we are interested. Second, our model differs from others by its explicit recognition of the existence of more than one physical level in a data base and its use thereof. In most data base models only one physical level is recognized and this is usually the secondary storage structure. However in most conventional systems (even with virtual memory), data exists physically in more than one medium. This fact is very important from the security and cryptographic points of view. The main idea in our model is that a data base is composed of several logical or abstract levels which describe data

that physically resides in one or more physical media and therefore has one or more physical structures. The existence of these physical media was recognized in CODASYL ([COD71] p. 15). In CODASYL the three media identified are:

1. The user-working areas
2. The system buffers
3. The secondary storage

Therefore we should have logical schema which describe the data structure in each of these media. Actually in the CODASYL model the Sub-Schema describes the data in physical level 1 while the Schema describes the data in physical levels 2 and 3. Another example for the existence of several physical levels is a data base which resides on an hierarchy of memories, part of it on a drum, part of it on a disk and part of it on a magnetic tape.

4.2 Four Levels Model

In our model we have four logical - abstract levels:

1. User-logical level - UL
2. System-logical level - SL
3. Access level - AL
4. Storage level (Also called structured storage level) - STL

Each of these levels can be composed of several sub-levels. Corresponding to each logical level there is a physical level which is connected to a physical medium. The exact relation between logical and physical levels is explained in the next section. In this section we describe the logical levels and their relation to security.

The user-logical level corresponds to the way a user or a user group sees the data base. It is very similar to CODASYL's Sub-Schema with the exception that we do not require the user-logical level to be a subset of the system logical level. Indeed, it can be useful to have complex transformations between the two levels. Usually there are several user-logical level structures in a data base. The system-logical level describes the whole logical structure of the data base. It may correspond to CODASYL's Schema with the difference that indexes, directories, and access paths are not part of the system-logical level (they are part of the Schema in CODASYL).

The access level describes the directories, indexes and all possible access paths in the data base. The storage level describes the way data and access paths are represented on a particular physical secondary storage device(s) and describes characteristics which are special for these devices. To each physical level corresponds one or more logical levels, since several logical levels might map data on one physical media. An example is shown in Figure 4.

Figure 4 - Multiple Physical Levels

Referring to the above figure, the user-logical level describes the data as it appears at the user site. The system logical level interprets data which appears in memory. The access and storage levels interpret data which reside on the secondary storage devices. It is conceivable that in the future secondary storage hardware will do all access calculations so that no access information needs to be in memory or described in the system logical level. The main idea we are stressing is the existence of more than one physical level corresponding to more than one physical medium. In Section 4.3 we present a formalism to describe these physical levels. In this section we discuss the abstract levels only and the implications of this structure.

4.2.1 Languages

We distinguish between two types of languages operating within this structure: description languages and access languages. A description language is declarative in nature, similar to a data definition language, and describes the data in one of these levels. We therefore have four description languages, or four different "programs" in the same language - DL(1), DL(2), DL(3), DL(4), which describe the logical structure of the data in the four levels.

An access language is parametric or procedural in nature and it represents the different forms that a user query has in the different levels of the data base. Again we may have four languages:

- AL(1) - a user-oriented query language for manipulating the user-logical structure; should be natural language oriented.
- AL(2) - a system-oriented query language, mostly parametric but may contain some procedures to manipulate logical relations.
- AL(3) - a procedural data manipulation language possibly similar to the CODASYL DML, with primitives to describe different access paths.
- AL(4) - the primitives of the data management system (e.g. ALLOCATE overflow area).

A query in AL(1) is translated to a query in AL(2), which is translated to AL(3), etc.

4.2.2 Data Independence

This issue is discussed extensively in other data base models. Clearly queries in AL(1) have the highest degree of data independence while those in AL(4) have the lowest degree of data independence. However queries in AL(1) have the lowest degree of efficiency since they have to be translated (interpreted) by three levels of translators (interpreters).

4.2.3 Security

4.2.3.1 Spreading Protection Specifications

Now, with the levels structured model, we can show how the protection specifications and mechanisms can be spread out in the different levels of the data base. On level 1, each user group has its own user-logical structure. The protection specification on this level is very simple: by definition of the user-logical structure UL(i) of user group i, we specify the data to which user group i has access. The protection mechanism is also very simple. Access to data which is not defined in UL(i) is not allowed! Of course there may be variations

on this scheme, depending on the description language DL(1), but this is the most natural way. It is important to note that from the security point of view there is one more level, we call it level 0, before the user logical level. This is the identification and authentication level. The right identity of a user must be assured in order for some of the protection mechanisms to work correctly. Such an identification process was described recently by [EVA74] and [PUR74] using one way cryptographic transformations.

On the system-logical level, all of the data for all user groups is described. On this level finer protection specifications than those specified in the user-logical are possible. For allowing different access by different groups to the same data field, a password protection mechanism can be used. For example, if, for reading field X password A is needed, and for writing on it password B is needed, then putting password A in the user-logical level of the user-group which can read field X, and password B in the user-logical level of user group which can write on field X completes this protection specification. Alternatively, these passwords may not be part of the system at all, but may have to be supplied by the users at the terminals every time they want to access field X. Of course, password protection mechanisms can also be implemented in level 1, however it seems more efficient to have the protection specifications corresponding to a field which is shared by many users in the system-logical level. The system logical level is the natural place to specify data independent protection. For example, at this level one can use the password mechanism to protect all occurrences of a record type or field type (similar to privacy locks in CODASYL). However, when we try to implement data or value dependent protection, the password mechanism is insufficient. Two types of solutions are possible. The first is to specify and implement data dependent protection in higher levels of the data base. The second is to specify data dependent protection on the system logical level but to use other protection mechanisms such as pre-censorship and post-censorship. The main difference between pre- and post-censorship is the time of the protection binding. In the pre-censorship case the protection binding is done before accessing and the modified query is used in higher levels of the data base. This mechanism has the advantage that unauthorized data is probably not transferred to

main memory. (*) In the post-censorship case, protection binding is done after accessing. In this case unauthorized data may appear in main memory. However, more complex protection specifications can be implemented. Other forms of data dependent protection such as context protection [NEE73] can also be implemented on the system logical level, provided that the right history records are kept in this level.

On the access level both data dependent and access dependent protection can be implemented. For example, suppose we have the following protection specification (which is data dependent). A user can access a record only if the SALARY field in the record is less than \$10,000, and the DEPT field in the record is equal to Y. Suppose we have a directory with keyword entries corresponding to the fields SALARY and DEPT. If the system must go through the directory during accessing, then it is easy to verify that a user will not be able to use unauthorized directory entries corresponding to the unauthorized keywords. Such a scheme, where the protection specifications and mechanism are connected with the directory, was suggested by McCauley [McC75].

The main disadvantage of this method is that it is access dependent since protection exists so long as one uses the legal access path of: directory → file. However, no protection for illegal access paths such as "sequential search" through the file is given (The assumption is that these illegal access paths are impossible by means of physical security or hardware.). (**) Usually most protection specifications, even data dependent ones, are access independent. However, if for some reason access dependent specifications are needed, (e.g. some users may not be allowed to use some indexes) the appropriate level to implement them is the access level.

On the storage level it is even more natural to implement data dependent protection. This can be done by attaching to each data item a protection "tag" (Friedman [FRI70]) or "key" (Reiter [REI72]). This is

*This also depends on the implementation of the access mechanism which may use memory for its purposes.

**Actually, McCauley's model implements protection also on the storage level by using "file partitioning" and the "fire wall" idea. See Chapter 6.

wasteful in terms of secondary storage space, but it has the advantage of allowing the most general specifications and late protection binding which is more secure. Another mechanism which can be used on this level is an extension of the "ring" mechanism [GRA68] used in the MULTICS system. Suppose that each process belongs to a "ring" and each physical record has a "security ring" associated with it. A process is allowed to access a record only if its "ring" subsumes the record's ring. Specific occurrences of records are easily protected by this method. Finally, on the storage level it is possible to implement protection specifications which cannot be implemented on other levels. For example, one can implement protection of a physical area such as "cylinder" or "track" for error recovery purposes (see also CODASYL [COD71] p. 27). One can also specify protection which is dependent on physical device characteristics such as their addresses.

To summarize this section we see that it makes sense to spread protection specifications and mechanisms through the different levels of the data base. This is also shown in Figure 5. An important motivation for it is that some of these specifications are more appropriate for one level than for the other, as was shown above.

4.2.3.2 Auditing

Auditing is recognized to be a very important tool in data security. However, the data gathered by security auditors can be used for performance evaluation. The division of the data base into several levels allows us to distinguish between these two purposes of auditing. If the protection specifications are mainly defined in levels 1 and 2, then this is where the security auditing should be done. Performance auditing should be done on levels 3 and 4 where counts on the use of indices and directories can be maintained, without preserving the identity of individual accesses, and data needed for determining reorganization points and data migration times can be gathered. So the multi-level structured model helps the DBA to decide where to implement auditing functions and what data to gather at each level.

4.2.3.3 Cryptographic Transformations

Cryptographic transformations are a very effective protection mechanism. How do they fit in the multi-level

PS(i) - Protection specifications on level i.

PM(i) - Protection mechanism on level i.

Figure 5 - Spreading Protection Specifications and Mechanisms

structured model? Very simple, they are a subset of the possible transformations between the physical levels of the data base! Since there exist several physical levels of a data base, there exist transformations between these levels. If these transformations are used as a protection tool, they are called cryptographic transformations!. We now see the strong connection between the way a data base is structured and the use of cryptographic transformations for protection. In Chapter 5, we discuss this subject in more detail.

4.2.4 Discussion

The goal of presenting the four levels model is not to suggest a new more secure model of a data base. However, this model has two important properties:

- 1) It enables us to understand the relationship between several, seemingly unconnected data security models. It shows that in essence the different models implement the protection binding in different levels of the data base and therefore can be viewed as special cases of the four levels model (of course we simplified the subtle aspects of each of these models).
- 2) It shows that protection specifications and mechanisms can be spread throughout different levels of a data base, and that more than one protection mechanism can be used. Another important aspect of this model is that it can be used for investigating the security engineering problem. For example we can ask questions such as: which protection mechanisms are best at each level? How should we spread the protection specifications between the levels? Cost estimates, security measures and "trade-off's" are needed for answering such questions. More research can be done in this area of security engineering.

The presentation until now has concentrated on the logical structure of the multi-level model. In order to understand the physical structure in its relationship to cryptography we need appropriate notation. In the next section we develop a formalism to describe the physical structure of a data base.

4.3 Notation and Definitions

When one looks at a data base as it is represented on secondary storage one sees a sequence of 0's and 1's. These binary digits make sense only when one knows the right structure, coding and interpretation of the data. One starts with the simple division to data items and then builds more complex blocks of the structure (repeating groups, records, files). The basic concept then for describing a data base is the data item. A data item is a set of bits conceived of as a unit, which has characteristic properties such as the following:

attribute or interpretation - what the data item
is and what it is
used for

value

length - number of characters

coding

address

We denote data items as d_i

A data item often represents one or more properties of other data items. We give two examples:

<u>Example 1</u>	<u>d_1</u>	<u>d_2</u>
interpretation:	length of d_2	name
value:	4	JOHN

In this case d_1 has the interpretation: length of d_2 .

<u>Example 2</u>	<u>d_1</u>	<u>d_2</u>
interpretation:	units	distance
value:	0 or 1	100

If $d_1 = 0$ then d_2 is in miles

If $d_1 = 1$ then d_2 is in kilometers

Here d_1 contains part of the interpretation of d_2 . We need the interpretation of d_1 in order to know what d_2 represents. The interpretation of d_1 may be documented in a manual which describes the system. We see then that a set of data items may have several levels of interpretation, some of which are in the data base itself and some of which are only implicit.

Two data items are called similar if they have the same interpretation. A field is an abstract concept representing a set of similar data items. A field has no value but usually has a unique name or identifier. A data item occurrence of a field is a data item which belongs to the set of similar data items represented by that field.

The concept of a field is important since its name contains part of the interpretation for the corresponding set of data items. For example, the field "AGE" gives part of the interpretation for the data items occurrences '9' '10' '15'.

The notation $d_{kh} \sim F_k$ means d_{kh} is the h -th data item occurrence of field F_k .

We can now define the concepts of logical records and physical records. Recall (Section 4.2) that our model identifies several logical and physical levels. A level (either logical or physical) is denoted by a superscript i .

A logical record is a set of fields with a unique name or identifier. The order of fields in a logical record is immaterial because each field can be identified by its name. We denote a logical record on level i as LR_j^i . We denote a field which is an element of that logical record as F_{jk}^i . We therefore have

$$LR_j^i = \{ F_{jk}^i \mid k = 1, \dots, m \}$$

where m is the number of fields in logical record LR_j^i .

The concept of logical record gives further interpretation to the corresponding set of fields. For example the record's name 'PERSONNEL' gives part of the interpretation to fields 'AGE' and 'ADDRESS'. The identifiers 'AGE', 'ADDRESS', 'PERSONNEL' appear in the description language (DL) for a particular level. That is, the interpretation of data items is done by fields and logical records through the DL (in a similar way a DECLARE statement in PI/I gives some interpretation to variables which map into locations in core).

The logical data base on level i is a set of logical records and their interpretation which is expressed in the corresponding description language (plus some implicit interpretation).

The logical data base on level i is denoted by LDB(i)

$$\text{LDB}(i) = \{ \text{LR}_j^i \mid j = 1, \dots, s_i \} + \text{interpretation}$$

where s_i is the number of logical records on level i.

A physical record is an ordered set of data items, each of which is an occurrence of a field, such that all these fields belong to the same logical record.

More formally, given logical record,

$$\text{LR}_j^i = \{ F_{j1}^i, F_{j2}^i, \dots, F_{jk}^i, \dots, F_{jm}^i \}$$

a physical record PR_{jh}^i is an n-tuple of data items

$$\text{PR}_{jh}^i = (d_{j1h}^i, d_{j2h}^i, \dots, d_{jh}^i, \dots, d_{jnh}^i)$$

such that $(\forall p \exists k) (d_{jph}^i \sim F_{jk}^i) \quad (*)$

We call physical record PR_{jh}^i an occurrence of logical record LR_j^i and denote the relation between them by

$$\text{PR}_{jh}^i \sim \text{LR}_j^i$$

A logical record then represents a set of physical records similar to the way a field represents a set of data items.

Physical records are the basic units in which data is stored on secondary storage devices such as disks or magnetic tapes. For convenience we consider only "continuous" physical records. That is, physical "gaps" are not considered part of a physical record. Each physical record has a unique address which allows the access mechanism to locate it. The address of a data item within a physical record is determined by its position relative to the beginning of the record and the lengths of all data items before it. The order of data items within a physical record is thus important for finding their addresses.

(*) An example where $n \neq m$ called a "repeating group" is shown later.

The physical data base on level i is the set of all physical records on that level.

$$PDB(i) = \{PR_{jh}^i \mid j = 1, \dots, s_i, h = 1, \dots, m_{s_i}\}$$

The connection between logical data base level, physical data base level, logical records and physical records is seen in Figure 6.

Figure 6 - The Connection between the Logical Data Base Level, the Physical Data Base Level, the Logical Records and the Physical Records.

4.4 Examples of Standard Structures in the Level Notation

To illustrate the above notation and concepts we apply them to five common data base structures. In all examples the level index is suppressed.

Example 1: repeating group

Defined in CODASYL as: "a collection of data that occurs an arbitrary number of times within a record occurrence". Using our notation we can define it as a set of similar data items which occurs within one physical record.

Suppose we have a repeating group which gives information on a person's children. The logical record is:

$$LR = (F_1, F_2, F_3)$$

The following interpretation will appear as part of the DL for this record.

LR - information on children

F_1 - no. of children

F_2 - name

F_3 - age

Physical records which are occurrences of this logical record are:

$$PR_1 = (d_1, d_2, d_3, d_4, d_5)$$

where $d_1 \sim F_1$, $d_2 \sim F_2$, $d_4 \sim F_2$, $d_3 \sim F_3$, $d_5 \sim F_3$

and value $(d_1) = V(d_1) = 2$, $V(d_2) = 'AMY'$, $V(d_3) = 25$,

$V(d_4) = 'SUSAN'$, $V(d_5) = 22$ and

$$PR_2 = (d_1, d_2, d_3, d_4, d_5, d_6, d_7)$$

where $d_1 \sim F_1$; $d_2, d_4, d_6 \sim F_2$; $d_3, d_5, d_7 \sim F_3$

and $V(d_1) = 3$, $V(d_2) = 'JOHN'$, $V(d_3) = 29$, etc.

Example 2: "SETs" as in CODASYL

A "SET" is defined in CODASYL as a named collection of record types with one record type declared as OWNER and the other record types declared as "MEMBER".

CODASYL's "record type" is very similar to our "logical record". We can define a "SET" as a named one to many relation between a logical record called "OWNER" and logical records called "MEMBER".

As an example we describe a structure composed of two "SETs" A and B shown in Figure 7.

Figure 7 - A "SET" structure

LRX, LRY, and LRZ are names of logical records. LRX and LRY are "OWNERS" of "SETs" A and B respectively, and LRZ is "MEMBER" of each of the two "SETs".

$$LRX = (F_{x1}, F_{x2}, \dots, F_{xm}, F_{x,m+1})$$

$$LRY = (F_{y1}, F_{y2}, \dots, F_{yn}, F_{y,n+1})$$

$$LRZ = (F_{z1}, F_{z2}, \dots, F_{zk}, F_{z,k+1}, F_{z,k+2})$$

All the first m , n or k fields are "standard" application-oriented fields. The other fields have the following interpretations:

- $F_{x, m+1}$ - address of a physical record occurrence of LRZ which is a MEMBER of "SET" A
- $F_{y, n+1}$ - address of an occurrence of LRZ which is a MEMBER of "SET" B.
- $F_{z, k+1}$ - address of an occurrence of LRX which is an OWNER of "SET" A.
- $F_{z, k+2}$ - address of an occurrence of LRY which is an OWNER of "SET" B.

The "repeating group" example is an instance of an intra-record data structure, while "SET" is an instance of an inter-record structure. Clearly any structure whether inter- or intra-record can be conveniently represented in our notation. We stress that structure is part of the logical description in contrast to physical records which are simply ordered tuples of data items.

We slightly modify the notation replacing the "level" index by a descriptive letter. This permits easy introduction of distinctions within a level, e.g. between users. The user logical level LDB (1) is now denoted by:

$$LDB(1) = \{U_1(DB), U_2(DB), \dots, U_N(DB)\}$$

Since there are N different users (or user groups) we denote the j'th logical record belonging to user i as UR_{ij} . The part of the user logical level associated with user i now becomes

$$U_i(DB) = \{UR_{i1}, UR_{i2}, \dots, UR_{in}\}$$

We denote the j'th logical record on the system logical level as LR_j . The system logical level is, in the new notation,

$$LDB(2) = \{LR_1, LR_2, \dots, LR_S\}$$

In the CODASYL model a user logical record is a subset of some system logical record. We allow more complex relations between records in these levels. Specifically they are used for cryptographic protection in the next chapter.

In analogous fashion we introduce the term access record which is simply another name for a logical record on the access level, denoting the i'th access record as AR_i . The access level now becomes

$$LDB(3) = \{AR_1, AR_2, \dots, AR_m\}$$

The relations obtaining between the system logical and the access levels are very important. We now give some examples.

Example 3: Access paths

Suppose we have a file (a file is represented in our notation as a logical record on the system logical level)

$LR = \{FR_1, FR_2, \dots, FR_n\}$ and two indices. The first

index specifies for each data item occurrence of field FR_1 the record(s) which contain this data item. The second index does the same for field FR_2 . In our notation the logical record LR (in the system logical level) is related ("translated") to three access records: AR_1, AR_2, AR_3 .

Their field structures are:

$$AR_1 = \{F_{11}, F_{12}\}$$

$$AR_2 = \{F_{21}, F_{22}\}$$

$$AR_3 = \{F_{31}, F_{32}, \dots, F_{3n}\}$$

The interpretations are

$$F_{11} = F_{R1}$$

F_{12} = address of record containing a specific data item occurrence of F_{R1} . (If there is more than one such record we need another field to specify the number of these records.)

$$F_{21} = F_{R2}$$

F_{22} = address of record containing a specific data item occurrence of F_{R2} .

$$F_{31} = F_{R1}, F_{32} = F_{R2}, \dots, F_{3n} = F_{Rn}$$

We therefore have three access paths (AP), using the notation $\langle AR_1, AR_2, \dots, AR_n \rangle$ to mean an access path

from AR_1 to AR_n through the successive AR_i . We have

$$AP_1 = \text{access path 1} = \langle AR_1, AR_3 \rangle$$

$$AP_2 = \langle AR_2, AR_3 \rangle$$

$AP_3 = \langle AR_3 \rangle$ represents a sequential search through the original file.

The access paths are shown in Figure 8.

Figure 8 - Access Paths

With the concepts of access records and access paths, we can represent any directory or index structure.

Example 4 Hsiao/Harary generalized file structure [HSI70]

The Hsiao/Harary generalized file structure consists of a file and a directory. They define the concept of keyword which is an attribute-value pair, or name-value pair in our terminology. The file is divided into cells. In each cell, records which contain the same keyword are linked by a pointer (linked list). The directory contains

- 1) a list of all keywords
- 2) for each keyword, a list of all cells in which records containing this keyword reside
- 3) for each cell, the address of the first record in the cell containing that keyword.

In our notation the generalized file structure can be written as a logical record

$$LR = \{FR_1, FR_2, \dots, FR_n\}$$

which is "translated" into two access records: AR_1 (directory), AR_2 (file).

$$AR_1 = \{F_{11}, F_{12}, F_{13}, F_{14}, F_{15}, F_{16}\}$$

has the following interpretation:

$$\text{keyword} \begin{cases} F_{11} = \text{name of field } F_{Ri} \\ F_{12} = \text{value of data item occurrence of field } F_{Ri} \\ F_{13} = \text{no. of records containing the corresponding keyword} \\ F_{14} = \text{no. of cells containing such records} \\ F_{15} = \text{address of first record in a cell} \\ F_{16} = \text{no. of records in that cell} \end{cases}$$

Note that F_{14} , F_{15} , and F_{16} constitute a repeating group. In AR_2 we introduce A_{2i} to denote a field interpreted as a "pointer".

$$AR_2 = \{F_{21}, A_{21}, F_{22}, A_{22}, \dots, F_{2n}, A_{2n}\}$$

where

F_{2i} = a "standard" F_{Ri} and

A_{2i} = the address of next record in cell containing
the keyword associated with F_{2i} .

The possible access paths are:

$AP_1 = \langle AR_1, AR_2 \rangle$ - which represents accessing the file
through its directory

$AP_2 = \langle AR_2 \rangle$ - which represents a sequential search.

Example 5: Record Splitting.

A file of records is split into two files, each containing half of each record. In our notation we are given the logical record

$$LR = \{F_{R1}, F_{R2}, \dots, F_{Rn}\}$$

LR is "translated" into two access records.

$$AR_1 = \{F_{11}, F_{12}, \dots, F_{1k}, A_{1k}\}$$

$$AR_2 = \{F_{2,k+1}, F_{2,k+2}, \dots, F_{2n}\}$$

where

$$F_{ij} = F_{Rj}, \quad i = 1 \text{ and } j = 1, \dots, k \text{ or } i = 2 \text{ and } j = k+1, \dots, n$$

A_{1k} = the address of the occurrence of the second
half of the original physical record

The only possible access path is:

$$AP = \langle AR_1, AR_2 \rangle$$

This situation is shown in Figure 9.

Figure 9: Record Splitting

We will use this example in the next chapter.

The concepts of access record and access path are useful since they allow the description of different access structures with a uniform notation.

We denote the logical records in level 4 (storage level) as SR_1 . The storage level is then

$$LDB(4) = \{SR_1, SR_2, \dots, SR_p\}$$

In some cases we might want to introduce an additional storage level $LDB(5)$ which can represent "back up" physical media (e.g. magnetic tape). We call level 5 the unstructured storage level for reasons to be discussed in the next chapter.

4.5 Discussion

The main ideas presented through the multi-level model are the separation between physical and logical levels and the identification of several physical levels in a data base. The examples given in the last section have further shown the usefulness and generality of this model. Traditionally, a data base was viewed as consisting of one physical level and several logical levels. Since our model allows any number of physical levels, it includes the traditional view as a special case. Furthermore, the identification of several physical levels in a data base has several advantages. First, it is a better approximation to reality, since data in data base systems usually exists in several physical media at any time. Second, it permits the identification of transformations on the data between different physical levels, thus it enables us to define the important protection mechanism called cryptographic transformations (see Chapter 5 for details). Third, the concept of multiple physical levels can be used to model the recently developed distributed data bases in computer network. In this case the data base at some node of the network will be considered as a pair of logical and physical levels. This shows the generality and extendability of the model. More research can be done on this extension of the multi-level model to the "distributed data bases" case.

4.6 Summary

In this chapter we presented a multi-level structured model of a data base. Its main advantage in relation to

security is the possibility for decentralization. The protection specifications and mechanisms can be spread through all levels of the data base. The model also generalizes the physical structure by allowing multiple physical levels for a data base. In the next chapter we use this physical structure to define cryptographic transformations and to show their possible application between different levels of the data base.

CHAPTER FIVE

Cryptographic Transformations in the Multi-level Model

In this chapter we discuss the connection between the physical structure of a data base and the cryptographic transformations applied to it with special reference to the multi-level model. We identify several types of cryptographic transformations which can be applied between different levels of the data base, show different methods of using and controlling these transformations, and discuss the advantages and disadvantages of each method. In particular we discuss the problems associated with the user controlled transformations which are used later (Chapter 6) as a basis for the design of different authorization systems.

5.1 Basic Definitions

In a dynamic data base system, data which at some time t is found in physical level i , may also be found, possibly at another time, in physical level j . Since the data representation and structure may be different in the two levels, we can talk about transformations on the data "between" the two levels.

We can now define cryptographic transformations in a data base.

Definition: Any transformation on the data between two physical levels used as a protection mechanism is called a cryptographic transformation.

We can use the notation developed in Chapter 4 and define the transformations between two consecutive physical levels i and $i+1$ (the transformation between any two levels is a composition of transformations between consecutive levels). If we denote LR_j^i , LR_p^{i+1} as logical records on levels i and $i+1$, and

PR_{jh}^i - physical record on level i , where $PR_{jh}^i - LR_j^i$

PR_{pg}^{i+1} - physical record on level $i+1$, where $PR_{pg}^{i+1} - LR_p^{i+1}$

then the general cryptographic transformation is:

$$PR_{jh}^i = f(PR_{p_1g_1}^{i+1}, PR_{p_2g_2}^{i+1}, \dots, PR_{p_mg_m}^{i+1})$$

In other words, f is an algorithm which gets as input m sets of bits (m physical records) and outputs one set of bits as a result. Notice that the physical records can correspond to different logical records.

We usually distinguish between two classes of transformations: Structure preserving and Structure destroying transformations. In the case of structure preserving the logical structure in the two levels is identical. We write $LR_j^i = LR_j^{i+1}$ if both LR_j^i and LR_j^{i+1} have the same set of fields. We also require that a physical record at level i will be the result of transformation on only one physical record at level $i+1$ (i.e. $m=1$). More explicitly, if conditions

- 1) $LR_j^i = LR_j^{i+1}$ and
- 2) $PR_{jh}^i = f(PR_{jg}^{i+1})$ where $PR_{jh}^i \sim LR_j^i$
 $PR_{jg}^{i+1} \sim LR_j^{i+1}$ exist

then f is structure preserving. In this case we can define the inverse transformation f^{-1} .

$$PR_{jg}^{i+1} = f^{-1}(PR_{jh}^i)$$

If either of the above conditions are not met then we have a structure destroying transformation. Examples for both kinds of transformations are given below.

The main advantage of structure preserving transformations is that no overhead is associated with restructuring while accessing the data. We now identify the possible transformations between the different levels in the multi-level model.

5.2 Transformations Between Physical Levels

5.2.1 User Logical <----> System Logical

We start with the transformation of data between the user-logical and system-logical levels. Clearly, the transformation between the corresponding physical levels depends on the transformation between logical levels. In general, these transformations can be very complex. However, if we

want to use system services to process and query data items and also want the ability to share data among other users, the common data item structure has to be preserved. We identify three types of transformations between physical levels 1 and 2.

In all three types of transformations we view f as composed of a set of transformations f_k on individual data items. That is, if

$$PR_{jh}^i = (d_{jh,1}^i, d_{jh,2}^i, \dots, d_{jh,k}^i \dots d_{jh,n}^i)$$

$$\text{and } PR_{jg}^{i+1} = (d_{jg,1}^{i+1}, d_{jg,2}^{i+1}, d_{jg,k}^{i+1} \dots d_{jg,n}^{i+1})$$

then $PR_{jh}^i = f(PR_{jg}^{i+1})$ means that $d_{jh,k}^i$ is a result of a transformation f_k on one or more data items $d_{jg,k_1}^{i+1} \dots d_{jg,k_r}^{i+1}$.

Type 1: Data item substitution.

$$d_{jh,k}^1 = f_k(d_{jh,k}^2)$$

$$d_{jh,k}^2 = f_k^{-1}(d_{jh,k}^1)$$

This is the common substitution transformation which does not destroy the data item structure. Its advantage is its simplicity and the fact that some processing, such as queiring can be done on data items even in their ciphered form. Its disadvantage is that it may not be secure enough (because some statistical properties useful for the "enemy" are not hidden).

Type 2: Data item expansion

$$d_{jh,k}^2 = f_k(d_{jh,k_1}^1, d_{jh,k_2}^1 \dots d_{jh,k_r}^1)$$

A data item in the system logical level is expanded to several data items and a finer structure on the user logical level. As an example suppose we have the fields: NAME, AGE, SEX on the user logical level, while on the system logical level we have one field: PERSONAL DATA, whose occurrences are a scrambled form of occurrences of the three fields on the user logical level. This transformation

can be made more secure than the former one but it has the following major disadvantage: since the fine structure on the user logical level is destroyed, a user cannot ask queries which require processing on the system logical level about individual fields, such as: NAME, AGE, SEX, (these fields are not defined in the system logical level which means

$$LR_j^1 \neq LR_j^2).$$

Type 3: Data item contraction.

$$d_{jh,k}^1 = f_i(d_{jh_1,k}^2, d_{jh_2,k}^2 \dots d_{jh_m,k}^2)$$

In this case data items from different physical records participate in the transformation. One interesting case of the application of this transformation is the statistical data base. If all the data items on level 2 which participate in the transformation are similar, i.e.

$$d_{jh_1,k}^2, d_{jh_2,k}^2 \dots d_{jh_m,k}^2 \sim F_{jk}^2 \text{ then}$$

$d_{jh,k}^1$ may represent some statistical property of these data items. For example, a user may be able to see only the average but not individual data items!

The last two types of transformations are of the "structure destroying" class. Type 2 violates the first condition. Type 3 violates the second condition. Clearly more complex transformations can be defined between level 1 and 2.

One comment is appropriate at this point. There may be some cryptographic transformations between the two levels of which we are not aware. For example, if level 1 and level 2 are far apart and are connected by telephone lines, some enciphering device may be connected to these lines. Currently, we are not interested in this kind of enciphering which has nothing to do with the data base structure and as far as the representation of the data in the two levels is concerned does not have any effect. The same comment is true in the case of transformations between other physical levels of the data base.

5.2.2 System Logical <----> Access <----> Storage

We discuss the transformations between these three levels together since as explained at the end of Chapter 4, we consider the case of one physical medium serving both levels 3 and 4. We identify three types of transformations, substitution oriented, transposition oriented, and access oriented. For convenience we eliminate here the indexes j, h for logical record j, physical record h.

Type 1: Substitution oriented transformations

This was called data item substitution in the previous section, but now we stress that levels 2 and 4 are involved.

$$\begin{aligned}d^2_i &= f_i (d^4_i) \\ d^4_i &= f_i^{-1} (d^2_i)\end{aligned}$$

The importance of this transformation is that it allows us to define processable ciphers. Most of the processing of data items is done on the system logical level (in main memory). So our definition of processable ciphers must refer to data in level 2. The advantages of being able to process data in its ciphered form were discussed in Chapter 2. Informally, a processable cipher is a transformation from the system logical level to the storage level which preserves the data item structure and allows the type of operations which are done on the "clear" data items to be done on the ciphered data items. Formally, if

$$d^2_1 = f (d^4_1) \text{ and } d^2_2 = f (d^4_2) \text{ and } G \text{ is a defined}$$

operation on data items in level 2 and G^1 is a defined operation on data items on level 4, then f is processable if and only if:

$$\exists G, G^1 \text{ s.t. } G (d^2_1, d^2_2) = f (G^1 (d^4_1, d^4_2))$$

Usually we also require that $G = G^1$. Examples of the operation G such as "COMPARE" and "ADD" were shown in Chapter 2.

Type 2: Transposition oriented transformations

The order of data items in a physical record is important in order to find their addresses. However, this order may change from one physical record to another. For example,

suppose both the logical records on level 2 and level 4 are

$$LR = SR = \{F_{R1}, F_{R2}, F_{R3}\}$$

Following are the occurrences of physical records on level 4:

$$PR_1 = (d_{11}, d_{12}, d_{13})$$

$$PR_2 = (d_{21}, d_{22}, d_{23})$$

$$PR_3 = (d_{31}, d_{32}, d_{33})$$

where

$$d_{11} \sim F_{R1}, d_{12} \sim F_{R2}, d_{13} \sim F_{R3}$$

$$d_{21} \sim F_{R2}, d_{22} \sim F_{R1}, d_{23} \sim F_{R3}$$

$$d_{31} \sim F_{R3}, d_{32} \sim F_{R2}, d_{33} \sim F_{R1}$$

That is, the order of data items in a physical record on level 4 changes from one record to another. On the other hand, this order is fixed in the occurrences of physical records on level 2. This is defined as transposition transformation. Such a transformation is very effective against "browsing" since the starting address of data items is not known to the illegal "browser". Its disadvantages are that it increases the access time of finding a data item, and that another data item has to be added which contains information about the specific order in each record (see also Section 2.3.4).

Type 3: Access oriented transformations

With this transformation we encipher only the data items which allow the transfer from one access record to another in a specific access path. For example, suppose we use the "splitting" example (Chapter 4, Example 5) and

$LR = \{NAME, F_{R2}, SALARY, F_{R4}\}$ is translated into two access records.

$$AR_1 = \{F_{11}, F_{12}, A\} \quad \text{where } F_{11} = NAME, F_{12} = F_{R2}$$

$$AR_2 = \{F_{21}, F_{22}\} \quad \text{where } F_{21} = SALARY, F_{22} = F_{R4}$$

and A = address of second half of an occurrence of LR . If we encipher the field A , then matching the salary to its name without deciphering the field is very difficult. The reason is that we have used the notion of access path for enciphering. In this case we enciphered the only existing access path! This type of transformation can be very effective, since address fields usually have random statistical properties.

5.2.3 Combined Transformations

Shannon [SHA49] has shown the value of combining cryptographic transformations. A combined transformation based on substitution and transposition which provides some processing capabilities was shown in Chapter 2. Below are two examples of substitution/access combination.

Example 1: Hashing

Suppose we have a "hashing file". Each record in such a file is placed in a specific address using a hashing function on one specific data item, usually called the record "key". In our notation, we have one logical record

$$LR = \{F_{R1}, F_{R2}, \dots, F_{Rn}\}$$

and two access records

$$AR_1 = \{F_{11}\} \quad \text{where } F_{11} = F_{R1} \text{ is the hashing field.}$$

$$AR_2 = \{F_{21}, F_{22}, \dots, F_{2n}\} \quad \text{where } F_{2i} = F_{Ri}$$

We have two access paths. The legal access path $\langle AR_1, AR_2 \rangle$ and the illegal one $\langle AR_2 \rangle$ which is equivalent to a sequential search. Suppose we use the following transformation:

(K+d) hashing (address + K')

where $d \wedge F_{R1}$ is the record's key, and K is the cryptographic key. The meaning of this transformation is that as a result of the hashing process we get an address and a key K'. Key K' which is the "residue" of the hashing transformation, is used for enciphering data item d, or the full record (e.g. using substitution). The enemy now has much difficulty in following the illegal path of $\langle AR_2 \rangle$ since in order to guess what the substitution is, he needs to go through the legal access path, using the algorithm and the cryptographic key, K, which, of course, he does not know.

Example 2: Inverted File

Suppose we have part of the "file" described in Example 3 of Chapter 4. That is

$LR = \{ F_{R1}, F_{R2}, F_{R3} \}$ "translated" into -

$AR_1 = \{ F_{11}, F_{12} \}$

$AR_2 = \{ F_{21}, F_{22} \}$

$AR_3 = \{ F_{31}, F_{32}, F_{33} \}$

The three possible access paths were $\langle AR_1, AR_3 \rangle$,

$\langle AR_2, AR_3 \rangle$ and $\langle AR_3 \rangle$. We add to access records AR_1 and AR_2 the key fields K_1 and K_2 respectively. That is

$AR_1 = \{ F_{11}, F_{12}, K_1 \}$

$AR_2 = \{ F_{21}, F_{22}, K_2 \}$

All records which contain a data item occurrence of field F_{R1} (=keyword) are enciphered by specific occurrence of field K_1 . Similarly we use K_2 for records which are accessed by AR_2 . Therefore all records which have the same keyword are enciphered by the same substitution key. Again the illegal path $\langle AR_3 \rangle$ is protected by the fact that for

deciphering we need to know which records contain a particular keyword, but this information is contained in the directory only. This forces the use of the legal access path! (*) Another type of access-substitution transformation was suggested by Bayer and Metzger [BAY75]. They discuss the enciphering of a specific directory organization called B-trees. The methods above can be generalized to any hierarchical structure in which the higher level nodes contain the keys to decipher the lower level nodes. This is also used in Chapter 6 in discussing hierarchical protection specifications. Many variations of the combined transformations technique are possible.

5.2.4 Storage <-----> Unstructured Storage

The unstructured storage level defined in Chapter 4 is often used for backup or data migration purposes (the corresponding physical medium is typically a magnetic tape). The transformation between the storage and the unstructured storage levels can in general be structure destroying, since "back-up" volumes are seldom accessed. An example of such a transformation is to use a pseudo-random number generator to encipher a complete "file" or a large part of a data base. This type of transformation can be made very secure. Its major disadvantage is that complete deciphering is necessary before any access is possible.

We summarize this section in Figure 10. It shows the major types of cryptographic transformations which can be applied between the various physical levels. The next section discusses how to use and control these transformations.

(*) The case where a record contains two keywords or can be accessed by more than one access path can be further researched based on McCauley [McC75] (see Chapter 6).

Figure 10
Types of Cryptographic Transformations

5.3 Using Cryptographic Transformations

Cryptographic transformations can be used in two major ways:

- a) As part of the standard access control, i.e., serving as a protection mechanism for implementing the protection specifications as they are defined in each level. In this case the user must know about the existence of these transformations since he probably holds some of the cryptographic keys.
 - b) As a system tool for protecting against illegal access paths such as browsing or wiretapping. In this case the cryptographic transformations are completely controlled by the system. They are not connected to a specific protection specification and the user does not have to know about their existence.
- Of course, the combination of these two methods can be used. We call the first type user controlled transformations, the second system controlled transformations.

Actual implementation of the two methods depends on the data base structure and on the way the protection specifications and mechanism are spread through the different levels of the data base.

5.3.1 System Controlled Transformations

The following transformations are examples of system controlled transformations (method 2).

- 1) Transformations between unstructured storage and structured storage levels are completely system controlled.
- 2) Combined transformations such as substitution - transposition or substitution - access can be system controlled for protecting against the illegal access path of browsing (e.g. in case of "stealing" a physical secondary storage device). In this case the cryptographic keys for these transformations have to be internal to the data management or access primitives. However the only way to protect these keys is by standard access control methods.
- 3) Substitution oriented transformations, including processable ciphers under system control, can be used to protect against wiretapping or against dumping of main memory. Again the keys for these transformations are either

internal to the processing primitives of the query language (probably AL(2)) or are part of the DDL for Level 2, but they must be protected by standard access control methods. An example of the implementation of system controlled transformations is to have an enciphering/deciphering device between the I/O unit and main memory. This device enciphers/deciphers every piece of data passing through it. Its operation is independent of the protection specifications and the cryptographic keys must reside permanently in the system (probably in the device itself). This is therefore a case of system controlled transformations.

We see then that system controlled transformations are easy to implement and quite simple to control. They can be controlled by the DBA and he can change the cryptographic keys at any time. They are very effective against illegal access paths such as "browsing" or "wiretapping" but they must be complemented by standard access control methods for protection of the cryptographic keys. This method, of using both system controlled cryptographic transformations and standard protection mechanisms (e.g. passwords) such that each mechanism protects the "weak" areas of the other, can be very effective provided that adequate administrative security exists so that the probability of compromising both mechanisms at once is low. A system which is based on the cooperation between cryptographic transformations and access control was suggested recently by Muftic [MUF76].

5.3.2 User Controlled Transformations

The situation with user controlled transformations is more complex, and the main reason is the existence of sharing and overlapping protection specifications. Clearly, if every user has access to a unique part of the data base, then this part can be enciphered by a unique key and only the appropriate user has the right key. But this case is unrealistic, since one of the main purposes of a data base is sharing data. McCauley [McC75] has suggested a way to partition a data base using security atoms. The security atoms are data base dependent and not protection specification dependent, and therefore do not change if the protection specifications change. Every user has access to a set of security atoms according to the protection specifications. A way to use cryptography in this case will be to encipher each security atom with a unique key and distribute the keys between users according to the protection specification. This will be discussed further in Chapter 6.

Another possible use of user controlled cryptography is in the post-censorship mechanism. The only "clear" data which is sent to the user is the data that he has legal access to.

The main advantage of user controlled transformations versus system controlled transformations is in the "place of the key" problem. In the system controlled case, the cryptographic keys must reside at all times in the system and therefore must be protected by the standard protection mechanisms. In the user controlled case there is the possibility that only the users will have some of the cryptographic keys to their authorized data. These keys do not reside in the system except at the actual time of the user accessing his data. In this case even the DBA cannot "decipher" the user's data since he does not have the right key. This is certainly in accordance with our earlier statement that the DBA does not have to know the content of every data item in the data base. Such a system, where only the authorized user has the cryptographic key was implemented by IBM [SMI72] using the LUCIFER system to protect a "secure field". This method has several problems. The first is the possibility of penetration of illegal cryptographic keys. Since the right key is not stored in the system, some checking algorithm (e.g. check digit) must be used. The probability that an illegal key will not be detected by the checking algorithm is not zero and its results in the case of updating can be disastrous. The second problem is that it is very hard to control the "user controlled transformations" case. For example, close cooperation between users who share the same data is needed if the cryptographic keys are to be changed. Also, close cooperation is needed between users and the DBA in some cases of error recovery and data base reorganization. Possible solutions to these problems will be described in Chapter 6. The third problem called the "key transfer" problem is discussed in the next section.

5.3.3 The "Key Transfer" problem

In Section 5.1 we defined cryptographic transformations as transformations between two physical levels. This is shown schematically in Figure 11.

Figure 11 Transformation between Two Physical Levels

If the data in level $i+1$ is the ciphered data and the data in the level i is the clear data then we need only one copy of the cryptographic key at any time. In the case of the system controlled transformations this key must reside in the system (e.g. in the box f). However in case of user controlled transformations this key must be transmitted from the user site (e.g. a terminal) and a major problem is the protection of these keys, while they are "moving" through the system. This is called: the key transfer problem.

Sometimes, we "adjust" f to one physical level. For example, if f is "adjusted" to level 1, that is, f is applied only at the terminal, then we don't have a "key transfer" problem. Data is enciphered and deciphered at the terminal and it remains in a ciphered form in all subsequent physical levels (communication lines, main memory, secondary storage, etc.) This is exactly the case with the "secure field" in the LUCIFER system [SMI72]. However this means that any processing which needs to be done, has to be done on the ciphered data. This limits either the type of ciphers applied (processable only) or the processing which can be done. On the other hand, if all processing can be done at the terminal (e.g. editing, see Stahl [STA74]) then no key transfer is necessary. This case where f is implemented in level 1 only, can be used either as system controlled or as user controlled. We see then that there is no "key transfer" problem if we use processable ciphers or if all processing is done at the terminal.

The more common application of cryptography will be to data which need to be processed in clear form. (For example, a user wants to execute his own program which resides in an enciphered form in secondary storage.) If this is the case

and we use user controlled transformations, then the appropriate key must be transferred to the system and we are faced with the "key transfer" problem. It is important to note, however, that these keys exist in the system only for the short time required for accessing and then they are discarded. We consider the danger to "permanently resident" keys such as in the system controlled case more serious than the short time danger to the "moving" user controlled keys. Even so, we would like to discuss some possible solutions to this problem.

We actually have two problems:

- a) Protecting the key which is transferred through the communication line.
- b) Protecting the key while it is "moving" through main memory and the CPU.

The first problem can be solved by "standard" cryptographic means using system controlled transformations. For example, the terminal and the Communication Control Unit (CCU) can employ the same transformation with the same key. Thus the key which is transferred from the terminal to the system is enciphered at the terminal and deciphered at the CCU. This is shown schematically in Figure 12.

Figure 12: Using Communication Keys

The "communication key" can be either specific to a user as in the LUCIFER system or specific to a terminal. The terminal oriented keys might be easier to implement and to control, and they will probably be hard-wired in the two units. In this case we need to employ a cipher which makes the mapping

(clear, cipher) $\longrightarrow$ key
 very difficult to perform. The reason for this requirement is the following. If the above transformation is simple

(e.g. in case of VERNAM cipher) then a user can login, "tap" the communication line, and then by having both his clear and ciphered data, find the communication key. He then can use this key to decipher the keys of other users. A cipher with the property of very difficult access to the key from both the clear and ciphered data is the NBS Block cipher (see Chapter 8 for details). Such a cipher should then be used in this case. Clearly, the communication keys have to be protected physically and possibly changed periodically by a security officer.

The second problem is more difficult to solve. In general it is solved by protection mechanisms which exist as part of the operating system. For example, in a process oriented system a process will be created for each requested access to the data base. When the operating system creates this process it can store the cryptographic key as part of the Process Control Block (PCB) which is in a protected area of the system (or as part of the APT entry in MULTICS, see Organick [ORG72] p. 287). The cryptographic key disappears, of course, when the request is completed and the process is destroyed. In a capability oriented system the key will be stored as part of the capability for the segment(s) of data which we want to access. (*)

We again see the need for cooperation between access control and cryptographic transformations. Here, access control is needed to protect the cryptographic keys which are "moving" through the system.

A different approach is employed by Muftic [MUF76]. Muftic suggests keeping the data in a ciphered form in main memory and having an enciphering/deciphering unit between memory and CPU. He uses a combination of user and system controlled transformations plus an access control mechanism. His hardware, however, differs from current systems architectures, and with the more "standard" architectures the "key transfer" problem does arise.

More research can be done on the "key transfer" problem. However, as was noted above, we consider the danger to permanently resident keys more serious than the danger to the "moving" keys. We therefore assume that the "key transfer"

(*) One variation of this scheme would be to store these keys in the operating system as long as the user is "active" or "logged in" so that any new access would not require retransfer of the key, and as a result problem a) is greatly reduced.

problem is solved, for example by the methods described above, and we will investigate the design of a secure data base system based on user controlled cryptographic transformations. This is done in Chapter 6.

5.4 Summary

In this chapter we saw how cryptographic transformations "fit" into the multi-level model of a data base. We showed how the existence of several physical levels can be used to frame a useful definition of a cryptographic transformation and identified the different types of transformations which can be applied between the various physical levels. We discussed how to use and control these transformations and defined two main classes: User controlled transformations and System controlled transformations. We explained the advantages and disadvantages of each method. User controlled transformations present several interesting problems (e.g. the problem of "sharing") which are further discussed in Chapter 6.

CHAPTER SIX

Design of a Secure File System Based on User Controlled Cryptographic Transformations

In this chapter we suggest different schemes for designing a secure file system which is based on user controlled cryptographic transformations. In Chapter 5 we identified two types of cryptographic transformations: System Controlled Cryptographic (SCC) Transformations and User Controlled Cryptographic (UCC) Transformations. The former are completely controlled by the system, and do not depend on specific protection specifications. A disadvantage of SCC transformations is that all cryptographic keys are under system control, and therefore are physically within the system at all times. On the other hand, UCC transformations are directly related to the protection specifications, and have the advantage that cryptographic keys need to reside in the system only temporarily. UCC transformations have the disadvantage that users must know about their existence, since they have to supply (and possibly to remember) the cryptographic keys at the appropriate times. UCC transformations may also have the "key transfer" problem discussed in Section 5.3.3.

In this chapter we investigate the following question: How can UCC transformations be used to enforce different protection specifications? This is important because if one wants to use UCC transformations he must understand their capabilities and their limitations. Initially, we discuss this problem for a simple file system. The generalization to the multi-level model is treated in Section 6.1.3.

6.1 General System Structure

6.1.1 Protection Specifications for a Simple File System

Assume we have a set of files: $F=\{F_1, F_2, \dots, F_n\}$
and a set of users: $U=\{U_1, U_2, \dots, U_m\}$

A user may have access to one or more files. There can be different types of access, such as: READ, WRITE, EXECUTE. There are also different types of protection specifications,

or different protection policies. In the following description of protection policies, we do not distinguish between different types of access. The most common protection policies are:

- a) Compartmentalization ("need to know") - Each user has access to a group of files and to this group only. All files are on the same level. That is, there is no hierarchy between files. All users are on the same level, i.e. there is no hierarchy between users.
- b) Hierarchical - There is some partial ordering between files or between users, or both. For example, $F_1 > F_2$ means that if user U_i has access to F_1 , then he also has access to F_2 . Similarly, if $U_1 > U_2$ then if U_2 has access to F_i , then U_1 also has access to F_i . An example of file hierarchy is the MULTICS system [ORG72] (directories and segments are used instead of files). An example of user hierarchy is the Military classification system (using a hierarchy of degrees of clearance: restricted, secret, top secret. . .).
- d) Data Dependent and Data Independent - The distinction between these two types is discussed by Conway, Maxwell and Morgan [CON72]. Data independent protection specifications mean that access to a file is independent of its changing content, or of the value of some of its fields. Also, all records and fields in the file are subject to the same access. Data dependent protection specifications mean that access to records in the file depends on their content. Different records are allowed different access, and changing the content of the file might change the access to it. Data dependent protection specifications were discussed extensively by McCauley [McC75].

In the following sections we shall deal with several cases:

- 1) CDI - Compartmentalized and Data Independent.
- 2) HI - Hierarchical and Data Independent.
- 3) DD - Data Dependent.

We shall not deal with more complex cases, which can be viewed as combinations of these three cases and whose solutions are based on similar principles to the ones discussed here.

6.1.2 Basic Design Principles

We want to use UCC transformations to enforce the above protection specifications. We shall do it by showing several schemes in which files are enciphered and the cryptographic keys are distributed to the appropriate users. The main idea is that "non-existence" of access here means primarily inability to decipher, and therefore to understand the information. For example, if user U_1 does not have the cryptographic key with which file F_1 is enciphered, then this is equivalent to saying that U_1 does not have access to F_1 . This does not mean that we forbid the use of access control. On the contrary, cryptography permits security to be maintained even if a user has "bypassed" the file access control mechanism and has got access to the ciphered file. It should be clear that cryptography alone cannot protect against someone who just wants to destroy information.

The general structure of a system based on UCC transformations is shown in Figure 13. It is composed of three major parts: Files, an Access Mechanism which we generally call the "system", and Users. The system gets a request for access to a particular file from a specific user. It then requests the appropriate cryptographic key needed to access the file from the user. That is, it is assumed that "accessing" involves "processing" and the data in the file must first be deciphered. If no processing is needed then only ciphered information is transferred to the user and no "key transfer" is needed. In that case a user can use a key "inside" a terminal to decipher the data, as was suggested in the LUCIFER system [SMI72]. A general description of the operation of the access mechanism is shown in Figure 14.

From Figures 13 and 14 we see that two types of dangers to security exist in this system. The internal danger exists when "failure" of the access mechanism occurs. This can cause exposure of cryptographic keys and/or exposure of clear data. This problem was discussed in Section 5.3.3 as part of the "key transfer" problem. There, we assumed it was solved by the appropriate protection mechanism in the operating system, usually by access control mechanisms. Clearly the "internal" danger disappears if no processing needs to be done on the data. The first "external" danger is the danger to clear keys or data during transfer through communication lines. It can be overcome by using system controlled cryptographic transformations on both sides of the communication line, as described in Section 5.3.3. If processing is not needed this danger is non-existent. The second "external" danger is that of an unauthorized user being able to decipher the information in the files outside of the system domain (e.g. by stealing a volume and using it on another computer). Here our system is particularly strong, since data in the files is enciphered

Figure 13 - General Structure of the File System

Figure 14 - General Description of the Access Mechanism

and even the DBA or system programmers are not supposed to know the right cryptographic keys needed to decipher this data.

To summarize, the system is always protected against "external" dangers. If no processing is needed then there is no "internal" danger. If processing is needed then we need some access control mechanism in the operating system to protect against "internal" dangers. In the following discussions we assume processing is needed and that some form of operating system access control mechanism is present. We repeat here the statement in Chapter 1, that "standard" access control mechanisms do not protect at all against the "external" dangers mentioned above.

Having presented the general structure of the system we now state several design principles whose goal is to increase the security of the system. They are:

- a) No clear cryptographic keys reside permanently in the system. This to avoid compromising the security, either by the system administrator or by a user who has bypassed the protection mechanism which protects these keys. The users themselves have to know the clear keys since they have to supply them to the system during access. (Again, we assume here that during the "short time" of access, the mechanism to protect these keys works correctly.)
- b) Authentication of users is not required. Most protection mechanisms require the identification and authentication of users (for example, capability lists mechanism). Not relying on identification adds another level of protection since even if the identification process is bypassed, the security of the system is not compromised, provided that the cryptographic keys are not compromised.
- c) All ciphers are assumed known to all users. The security of the system rests with the cryptographic keys and not with the ciphers. This is the usual assumption in cryptography (see Shannon [SHA49], Van Tassel [VAN69] and NBS [FED75]).
- d) The cryptographic schemes described below are designed to protect against READ access. Usually other protection mechanisms are needed to protect against other types of access. However, we will show at least one scheme which can be used to protect against WRITE.

The above design principles assure that the security of the system will be maintained under severe conditions such as compromise of the identification procedure or compromise of the file access control mechanism. Before describing the different cryptographic schemes, we want first to show the connection between the simple file system and the multi-level model discussed in Chapters 4 and 5.

6.1.3 The Connection to the Multi-level Model of a Data Base

In Chapter 5 we defined cryptographic transformations as transformations between two physical levels of a data base. Since physical records are instances of logical records, the transformations between two physical levels depends in general on the structure of the corresponding logical levels. We then defined two types of transformations: "structure preserving" and "structure destroying". In the case of "structure preserving" transformations between physical level i and $(i+1)$ the corresponding logical levels have the same structure. That is,

$$\forall_j \quad LR^i_j = LR^{i+1}_j$$

In this case the only difference which can exist between the corresponding physical levels is in data representation or coding.

The case of "structure preserving" transformations is important since it is analogous to the file system case. LR^1_j corresponds to a clear file, LR^{i+1}_j corresponds to the ciphered file, and the protection specifications at level i which define the type of access allowed to LR^1_j correspond to the protection specifications defined for file F_j in the file system.

This is exactly equivalent to a file system case, in which every file is enciphered as a complete unit, no one file is split into several files, and no two files are merged into one file. In other words, the logical structure of the file system is not affected by the enciphering process. It is clear that "structure preserving" transformations can occur at any level of the data base and that protection specifications can also be defined at any level. Therefore the analogy to the file system exists at any level.

Clearly "structure preserving" transformations are the simplest ones. The use of more complex transformations changing the logical structure will not be considered here. They can be viewed as a composition of two simpler transformations, one changing logical structure and not data coding,

the other changing data coding but not logical structure. The first type is usually not related to the protection specifications. We will therefore be interested only in the latter type, which is "structure preserving".

We have now seen the desired generalization using the multi-level model. The use of cryptography in a file system is analogous to the use of "structure preserving" transformations between two physical levels of a data base. Therefore, all the schemes presented for a file system are also relevant for a general data base system.

In the following sections we go back to the file system, but the connection to the multi-level model should be kept in mind.

6.2 Compartmentalized, Data Independent Protection Specifications

6.2.1 Introduction

In this section we show how to enforce Compartmentalized, Data Independent (CDI) protection specifications using UCC transformations. We first assume only one type of access - READ.

Assume a set of files: $\{F_1, F_2, \dots, F_n\}$

and a set of users: $\{U_1, U_2, \dots, U_m\}$

Since only READ access is allowed, the protection specifications can be represented as a boolean access matrix A. In this matrix users are "rows" and files are "columns", and $A(i,j) = 0$ means U_i is denied access to F_j . For example, in Figure 15 U_1 is allowed to access files F_1 and F_3 only.

	F_1	F_2	F_3
U_1	1	0	1
U_2	0	0	1
U_3	0	1	1

Figure 15 - Access Matrix

Let $AU_i = \sum_{j=1}^n A(i,j)$. AU_i is equal to the number of elements in row i which are different from zero and is the number of files user U_i is allowed to access. Similarly we define $F_jA = \sum_{i=1}^m A(i,j)$. F_jA is equal to the number of users who are allowed to access file F_j .

We make the following simplifying assumptions about this matrix without essential loss in generality:

- 1) No two rows are equal. This means that if two users have exactly the same access rights then they are considered to be one user (sometimes we say "belong to the same user group").
- 2) No two columns are equal. Files are viewed as different only if users differ in their access to them. The distinction between two files which can be accessed by the same group of users is not interesting here and is irrelevant to the file security problem.
- 3) No zero rows or zero columns are allowed: they are "purged" users or files.
- 4) Since we deal with CDI protection specifications we have to encipher the whole file as a complete unit. It is simpler to assume that only one cipher, C , is used throughout the system, and that different files are enciphered with the same type of cipher but by different cryptographic keys. It is of course assumed that all users can know which type of cipher is used(*).

With these assumptions we suggest several schemes which can enforce CDI protection specifications. All schemes have the following obvious objectives.

Objective 1

Any suggested scheme must be complete. "Complete" means that if $A(i,j)=1$ then user U_i should be able to access file F_j using the scheme.

(*) Not much is gained by having more than one cipher. Theoretically, multiple ciphers are equivalent to a single cipher of higher complexity.

Objective 2

Any suggested scheme must be secure. "Secure" means that if $A(i,j) = \emptyset$ then user U_i should not be allowed to access file F_j by using the scheme.

A correct scheme is both complete and secure. We can now prove a simple theorem.

Theorem

Under assumptions 1,2,3,4 and objectives 1 and 2, every file has to be enciphered with a different key.

Proof: Assume two files F_i, F_j $i \neq j$, are enciphered by the same key K . By assumption 2, columns i and j are different. Suppose that they differ for row p , and assume $A(p,i)=1$ and $A(p,j)=\emptyset$. Since user U_p is allowed to access file F_i then by objective 1 he must be given key - K . But then user U_p will be able to access file F_j , which contradicts objective 2! Our first assumption then that the two files are enciphered with the same key is wrong. QED.

We therefore have n different cryptographic keys - $K_1, K_2, \dots, K_n$ which are used to encipher the n different files. (K_1 for F_1, K_2 for F_2 , etc.).

We now describe different schemes to implement the CDI protection specifications.

6.2.2 Scheme 1 = The "Simple" Scheme

The simplest scheme is to give each user the keys to all the files he is allowed to access. User U_i when he wants to access file F_j will have to supply the key K_j . The system will then use this key to decipher the data (i.e. allow access). This scheme has several undesirable properties from the human engineering point of view.

6.2.2.1 The Human Engineering Problem

Two problems arise in the scheme above. The first problem is that a single user U_i has to remember (store) AU_i keys. AU_i might be very large, making it inconvenient for the user. This is because a user has to maintain and store a large number of keys, and has to remember exactly which key to use for which file. We can define the concept of user convenience and claim that "user convenience" decreases

when AU_i increases. That is, the more keys the user has to remember, the less "convenience" he has.

The second problem is that there is a non-zero probability for a user to "lose" a key. A reasonable assumption is that "loss probability" increases with AU_i . A "loss" of a key may cause a user to "lose" authorized access. However, if this key is "seized" by another user, it may provide him with unauthorized access. Therefore, a key which is "lost" can compromise the security of one or more files. In scheme 1, a key affects exactly one file. This, as will be shown later, is not always the case. We call the number of files which are affected (i.e. can be deciphered) by a "loss" of a key - the key domain. We denote "key domain" of key K_j as DK_j . We define system risk as the expected value of the number of files compromised by the "loss" of keys, divided by the total number of files in the system. That is,

$$\text{system risk} = \frac{\sum_{j=1}^{\text{no. of keys}} \text{PL}(K_j) * DK_j}{n}$$

where $\text{PL}(K_j)$ is the probability of losing key K_j . (*)

We would like to design schemes with maximum "user convenience" and minimum "system risk". However, there is clearly a tradeoff between the two goals. One of the main purposes of showing several schemes for enforcing CDI protection specifications is to show this tradeoff and to provide the designer with several alternatives.

6.2.2.2 The Validation Problem

Since in a scheme based on cryptography every user can try to access every file, we need a method to validate the key used in accessing a file. If we assume only READ access, an invalid key does not cause any harm because the user who uses it will get "garbage" in return. However, files which are kept in a ciphered form also need to be updated. The assumption is that data added to the file is enciphered by the system, using the key supplied by the user, to avoid integrity problems. In the case of update the validation of the key is a must, since "penetration" of an invalid key (by mistake or on purpose) can be disastrous. We therefore present a validation scheme to be used during WRITE, which may or may not be used during READ.

(*) The above concepts can be defined more precisely and more research can be done on the human engineering problem. However, the definitions as stated above will be sufficient for us.

We add at the beginning of each file a "validation record". A simple validation record for file F_i is a ciphered form of its key K_i , say K'_i . This is shown in Figure 16.

Figure 16 - A Validation Record

The cipher which is used for validation is a one way cipher $K'_i = f(K_i)$. Such ciphers have two properties:

1) several "clear" keys can be enciphered to the same ciphered key. 2) it is very difficult to invert the cipher and to get any clear key from the ciphered key. So even if the "enemy" knows the ciphered key, and though there are many keys which are enciphered to this form, it is extremely difficult for the enemy to find out any of these clear keys. Such ciphers can be used for authentication purposes. They are described, for example, by Evans et. al. [EVA74] and by Purdy [PUR74]. The important point is that it is very difficult to invert K'_i and get K_i from it, while the transformation $K'_i = f(K_i)$ is very easy to perform. In the rest of this chapter f , g , h are used to denote one way ciphers.

The one way cipher is used by the system for validation as follows. Upon getting a request for access to file F_i the system requests key K_i . It then performs $f(K_i)$ and allows access only if $f(K_i)$ is actually the key in the validation record of file F_i .

6.2.3 Scheme 2 = The "User Profile" Scheme

In the next two schemes we store in the system a ciphered form of the cryptographic keys. We use a cipher C which is parameterized by a user key. For example, $K'_j = C(KU_i, K_j)$ means: K'_j is the ciphered form of K_j using the parameter KU_i . The deciphering transformation is denoted as C^{-1} . That is,

$$K_j = C^{-1}(KU_i, K'_j)$$

In scheme 2 each user has a user profile which contains a list of the AU_i keys he is going to use. This list is enciphered using a special user key - KU_i . The "user profile" can be organized as a sequence of file identifications and their corresponding ciphered keys. This is shown in Figure 17.

id. of file 1	K'_{i1}
⋮	⋮
id. of file j	K'_{ij}
⋮	⋮
id. of file AU_i	K'_{i, AU_i}

Figure 17 - The "User Profile" Scheme

Where $K'_{ij} = C(KU_i, K_j)$

$$K_j = C^{-1}(KU_i, K'_{ij}).$$

and K_j is the key used to access file F_j .

This scheme operates as follows: Upon receiving an access request to file F_j , the system finds the right entry in the "user profile" (using the file id.). Then the system retrieves the second half of this entry and deciphers from it key K_j , using KU_j supplied by the user. The key K_j is then used to access the file F_j (and validated if needed).

The scheme as described above suffers from the following problem. According to our assumptions user U_i may be able to read the "user profile" of another user, say U_j (e.g. he "bypasses" access control). If the two users have access to a common file, say F_k , then user U_i may be able to deduce the user key KU_j by using the common K 's entry in their "user profiles". U_i then may be able to use the "deduced" key to perform unauthorized access. This problem is called the key inversion problem and it is described in more detail in the next section. Here we just note that the solutions for this problem suggested in Section 6.2.4.1 are also appropriate here, for the "user profile" scheme.

The "user profile" scheme has several advantages and disadvantages. The main advantage of this scheme is that a user has only one key to remember, "user convenience" is higher than in scheme 1 and "loss probability" is lower. But now, one key, the user key, affects AU_j files. Therefore, "key domain" increases over scheme 1, and the effect on "system risk" is not clear. "System risk" may be lower or higher depending on the actual implementation.

The main disadvantage of this scheme is that it requires an authentication process. As the system must know which "user profile" to use upon each request, user identification is needed. This scheme is thus similar to a capability list mechanism which also relies on an identification process. However, our scheme has the advantage that even if the identification process is "bypassed", the correct cryptographic key (KU_j) is still needed for accessing. Another disadvantage of this scheme, (which also exists in "capability lists"), is related to changes in the protection specifications. Deletion of a file, for example, will necessitate the reprocessing or reenciphering of several user profiles.

Key validation in this scheme can be done as in scheme 1. User key validation is not needed because using an invalid user key will cause a validation error for the file key.

To summarize, this section presents a second scheme called the "user profile" scheme. It is similar to a capability list scheme, it increases "user convenience" over scheme 1 but requires an authentication process.

6.2.4 Scheme 3 = The "Keys Record" Scheme

This scheme is similar to the "access list" mechanism mentioned in Chapter 3. At the beginning of each file we add a "keys record" in addition to the validation record. This is shown in Figure 18.

K'_{j1}	K'_{j2}	...	K'_{jn_j}
validation record - K'_j			
File F_j			

Figure 18 - The "Keys Record" Scheme

As before, K_j is used to encipher file F_j . K'_j is used for validation, i.e. $K'_j = f(K_j)$. K'_{ji} is a new key and it is a ciphered form of key K_j .

$$K'_{ji} = C(KU_i, K_j)$$

$$\text{and } K_j = C^{-1}(KU_i, K'_{ji})$$

Where KU_i is a unique key for user U_i .

Using this scheme, the system operates as follows. User U_i who wants to access file F_j must supply his key KU_i . The system retrieves the "keys record" for file F_j . It finds the right entry in the "keys record" for user U_i and retrieves

K'_{ji} (the way to find the right entry will be discussed later). It then deciphers K'_{ji} , using KU_i , to get K_j .

$$K_j = C^{-1}(KU_i, K'_{ji})$$

Finally, it validates K_j and uses it for accessing F_j .

This scheme seems to have the following advantages:
 1) No need for authentication, which is an advantage over Scheme 2. 2) Higher "user convenience" since each user has to remember only one key. However, the scheme, as described above, suffers from a serious problem: it is not secure! The reason for this "breach" of security, which is called the "key inversion" problem, is explained next.

6.2.4.1 The "Key Inversion" Problem

Suppose we have two users U_1 and U_2 , and two files F_1 and F_2 with the access matrix shown in Figure 19. We show how user U_2 who should not have access to F_2 , can gain such access. Scheme 3 for this situation is shown in the bottom half of Figure 19.

It is important to remember that the assumption is that "keys records" can be read by both users and that both users know the cipher C . Now, since user U_2 has access to F_1 he can get K_1 by executing $K_1 = C^{-1}(KU_2, K_{12}')$. However, if U_2 knows K_1 and he can read K_{11}' , then he can execute another transformation called C^* to get KU_1 from it. That is,

$$KU_1 = C^*(K_{11}', K_1)$$

The transformation C^* is very interesting. Its meaning is: get the key (KU_1) from the clear message (K_1) and the cryptogram (K_{11}'). Now, once user U_2 get access to KU_1 he can use it to get access to file F_2 through K_{21}' . So we have a "breach" in security!

There are several ways to overcome this problem.

a) Use different user keys for different files. That is, instead of using one key per user - KU_i , use as many as AU_i keys. In that case, KU_{ij} means - the key used by user U_i to access file F_j . In the "keys record" we have

$$K_{ji}' = C(KU_{ij}, K_j)$$

Clearly the transformation C^* will not help in this case, since different keys are used for different files. The major disadvantage of this method is that it decreases "user convenience" and it does not seem better than scheme 1!...

b) Choose cipher C such that the transformation C^* is very difficult to perform. We will call such cipher a "key

	F_1	F_2
U_1	1	1
U_2	1	0

Access Matrix

K_{11}'	K_{12}'
K_1'	
File F_1	

K_{21}'	0
K_2'	
File F_2	

Corresponding "Keys Record" Scheme

Figure 19 - The "Key inversion" Problem

non-invertible" cipher. An example of such a cipher is the NBS Block Cipher [FED 75]. (It is described in detail in Chapter 8.) Even if C and C^{-1} are known, it is very difficult to get C^* . That is, it is very difficult to get the key from both the clear message and the cryptogram. If it is impossible to perform C^* , the above "breach" in security disappears.

In general, we recommend the second solution, i.e. using "key non-invertible" ciphers, for solving the "key inversion" problem.

6.2.4.2 Searching the "Keys Record"

Until now we did not explain how the system finds the right entry in the "keys record". The question is: Given a "keys record" such as shown in Figure 20

K'_{j1}	. . .	K'_{ji}	. . .	K'_{jn_j}
-----------	-------	-----------	-------	-------------

Figure 20 - A Keys Record

how does the system find the right entry K'_{ji} for user U_i ? Since we do not assume an authentication process, we cannot keep "user id" in the "keys record". It is also not secure to have the keys in a constant order in each file, since it may help the "enemy" to find the desired entry. One possible solution is to change this order from one file to another, then to use a sequential search and decipher all possible entries until the right entry is found. How does the system know that it found the right entry? By using the validation process. Only one entry will not result with a validation error and this will be the right entry.

A more efficient method is based on hashing (e.g. see Knuth [KNU75]). Assume we have a hashing function h which operates on the user key KU_i to get the appropriate address in the "keys record". If A_{ji} stands for the address of

K'_{ji} in the "keys record" of file F_j , then

$$A_{ji} = h(KU_i)$$

The system now works as follows. When user U_i requests access to file F_j , the system executes $h(KU_i)$ to find

K'_{ji} . It then uses KU_i again to decipher K'_{ji} and to get K_j . Finally it validates K_j and uses it for accessing F_j . An immediate question is what to do in case of collisions. One solution is to use the "chained overflow" method (the pointers do not have to be enciphered, although they might be). The system now operates as follows. Upon getting KU_i , it executes $h(KU_i)$, finds A_{ji} , deciphers K'_{ji}

and validates K_j . If a validation error occurs, the system first assumes collision and starts to go through the overflow chain. If there are no more items on the chain and there is still a validation error, we call it a user validation error. It may indicate an attempt at illegal access, and is not allowed.

The "user validation error" can help to increase security in the following way. Usually the "keys record" is large enough to contain many "empty" entries, beside the "valid" entries. These entries can contain "garbage" and there is no way that a penetrator can distinguish (just by looking at the contents) between "garbage" and "valid" entries. Suppose now that the enemy tries to use transformation C^* . His problem is that, since he does not know KU_i , he does not know the right entry. He can try all entries or choose an entry at random. There is a high probability for him to use a "garbage" entry, thus gets a wrong user key and get "user validation error" and therefore get caught! So for the enemy the chance of successful penetration is low, while the risk of "being caught" is high.

One comment is appropriate here. Cryptographic schemes assume that the protection is done through cryptography and that "standard" access paths may be "bypassed". However the hashing method just described is access path dependent, since a user must go through the "keys record" and the validation process. If one physically "steals" the two files, then he can try all possible entries in the "keys record" without the danger of being caught. The ultimate security in that case is either to use different keys for different files, or to use a "key non-invertible" cipher.

Another variation of the "keys record" scheme is the following. Some users with "good memory" can use different keys for different files, while users with "poor memory" and "light" security requirements can use the same key for all files. In that case, the enemy who does not know this fact (that he is faced with a "good memory" user), can try hard to execute C^* and overcome the "hashing" process and still get a "useless" key.

To summarize, the "keys record" scheme has several advantages but may suffer from the "key inversion" problem unless "key non-invertible" ciphers are used. It can provide the ability to catch the enemy by the special use of hashing and validation processes. Variations of the scheme provide more flexibility for different users. Some will use the same key for all files, and some will use different keys for different files. We then match the number of keys a user has to know to his ability to remember and to his security requirements. The "keys record" scheme is therefore a very flexible scheme which can easily be designed to meet different security requirements.

6.2.5 Different Access Types

Although cryptographic transformations are mainly effective against READ, they can be used to protect other access types by the use of the validation process. We will show now the protection of access other than READ for the simplest case, Scheme 1.

The simplest method is to use different keys for different access types. But it is not immediately clear how to do it, since each file is enciphered with one key only. We suggest a scheme in which we can distinguish between two access types: READ or READ, WRITE. That is, either one has only READ access, or he has both READ and WRITE access (or no access at all). The files are structured as before with a validation record, where K_j is used to encipher file F_j , and K'_j is used for validation, i.e. $K'_j = f(K_j)$. We now choose K_j as a result of another "one way" function f_1 on another key, KW_j (this function is similar to f and might actually be f). We define

$$K_j = f_1(KW_j)$$

and set $KR_j = K_j$

There are now two keys per file and they are used as follows. Upon READ request the system takes the key supplied by the user (KR_j) and validates it. Upon WRITE request the system takes the key supplied by the user (KW_j), executes $f_1(KW_j)$, and validates the result. The system knows whether to operate on the key with f_1 or not from the type of access requested by the user.

We have several comments on this scheme. 1) a user with a WRITE access (has KW_j) has also a READ access since he can always execute $KR_j = f_1(KW_j)$ himself. (As usual, the assumption is that f_1 is known to all users.) 2) A user with a READ access only, cannot gain a WRITE access, since he cannot invert the function f_1 . 3) It is important to note that the protection against WRITE is access path dependent, since the user must use the "standard" user/system interface, in order for the protection to work.

Similar modifications to the one described above for scheme 1, in order to provide protection for other than READ access, can be made also to the "user profile" scheme and to the "keys record" scheme. However, all schemes will suffer from the same problem, that the protection for other than READ access is access path dependent and does not exist outside the "standard" system (e.g. a system programmer might be able to destroy a file even if he is not able to read it). In the following sections we therefore discuss protection of one type of access only - READ access.

6.2.6 Further Security Problems

In relation to the schemes described in Sections 6.2.2 - 6.2.5, there are two more important problems.

The first is how to protect the sensitive parts of the system such as "user profile" or "keys" records. We need only to protect against WRITE since even if these areas are exposed to unauthorized users, there is not much they can get out of them. In fact, most of the discussion before was to show that even if these "sensitive areas" are exposed to READ, the security of the system is not compromised! This is basically why we insisted that no "clear" cryptographic keys reside permanently in the system. The protection of these areas against WRITE can be done best by other protection mechanisms. Very few people should be authorized to write on these areas (e.g. only a security officer). This exemplifies the fact that cryptographic schemes must be complemented by other protection mechanisms to provide complete security. Conversely, most protection mechanisms should be complemented by cryptographic schemes to yield a higher level of security.

The second problem is related to the effects of changes in the protection specifications on the different schemes. In scheme 1, changing the access matrix element $A(i,j)$ from $\emptyset$ to 1, meaning giving key K_i to user U_j . Changing $A(i,j)$ from 1 to $\emptyset$ however, involves reenciphering file F_j and giving the

new key to all F_i A users who are allowed to access file F_i . Otherwise, the former key can be used by user U_i to illegally access file F_i . For the other schemes we have similar problems. Adding access is usually simple, while removing authorized access is usually complex (we will not give details here). The difficulty of changing the protection specifications is one of the trade-off parameters for the designer of cryptographic protection schemes.

6.2.7 Summary

In Sections 6.2.1 - 6.2.6 we discussed the advantages and disadvantages of several schemes for implementing CDI protection specifications. This presents the designer with different alternatives and several trade-off parameters. Among them are: degree of security, user convenience, system risk, and the ease with which changes in protection specifications can be made. Using these trade-offs the designer of a secure file system based on UCC transformations can then make his design decisions in a rational, systematic way.

In the following sections we discuss (in less detail) two more types of protection specifications: Hierarchical and Data Dependent.

6.3 Hierarchical Protection Specifications

6.3.1 Introduction

In "hierarchical protection specifications" we refer to any partial ordering, relative to access, either between files or between users. The most common one is the tree hierarchy between files (segments), commonly expressed as a tree directory (see, for example, MULTICS - Organic [ORG72] p. 134). An example of such a tree is shown Figure 21.

Figure 21 - A Tree Directory

To access C_1 a user must first access directory A, and then access directory B_1 , because in a tree directory the address of any node is in its immediate parent. Therefore a user who wants to access C_1 will need some information from higher level nodes. This means that a READ access to C_1 requires also READ access to A and B_1 . The partial ordering is then $- C_1 > B_1 > A$. We see then that the READ access has a reverse hierarchy to that of the tree directory. A similar observation was made by Walter et. al. [WAL73] "Every directory has an equal or lower classification than any file it eventually contains". We call this case the reverse hierarchy case.

The situation is different for a WRITE access. A WRITE access to C_1 does not mean a WRITE access to B_1 or A. On the contrary, directory B_1 contains the pointer to C_1 , so that a WRITE access to B_1 derives also a WRITE access to C_1 . The partial ordering is then $- A > B_1 > C_1$, and the access hierarchy is equal to the directory hierarchy. We call this case the standard hierarchy case.

In general there may be a complex relation between access hierarchy and directory structure. As a first case we discuss the access hierarchy alone, independent of directory structure (that is, we deal with the access hierarchy only and ignore the directory structure), calling it the independent access hierarchy case.

6.3.2 Independent Access Hierarchy

Assume the access tree shown in Figure 22. (It does not have to be a tree, but the discussion for trees will be relevant to other partial orderings.) Clearly $A > B_1 > C_1$, $A > B_1 > C_2$ etc. Suppose user U_1 has access to C_1 and user U_2 has access to C_2 . How do we enforce these specifications using UCC transformations? The simplest method is a variation of the "keys record" method. Let key KN encipher the file named by node N, e.g. KC_2 for file C_2 , KA for file A, etc. Files A and B_1 have the structure shown in Figure 23, where α and β are standard fields in the "keys record" scheme (see section 6.2.4). The other fields are new and have the following meaning.

Figure 22 - Access Hierarchy

keys
records →

α	KAB_1'	KAB_2'
KA'		
File A		

β	KBC_1'	KBC_2'
KB_1'		
File B ₁		

Figure 23 - Keys Records for Independent Access Hierarchy

$$\begin{aligned}
KAB'_1 &= C(KA, KB_1) & KB_1 &= C^{-1}(KAB'_1, KA) \\
KAB'_2 &= C(KA, KB_2) & KB_2 &= C^{-1}(KAB'_2, KA) \\
KBC'_1 &= C(KB_1, KC_1) & KC_1 &= C^{-1}(KBC'_1, KB_1) \\
KBC'_2 &= C(KB_1, KC_2) & KC_2 &= C^{-1}(KBC'_2, KB_1)
\end{aligned}$$

The system now operates as follows. On request for file C_1 , the system first demands key KA . (It is assumed that the system knows the access hierarchy.) If the user supplies KA (i.e. has access to A), the system accesses the "keys record" of file A and deciphers key KB_1 . Similarly it deciphers KC_1 from the "keys record" of file B_1 . Finally it validates KC_1 and uses it to access file C_1 . If the user does not have access to A , the system asks for KB_1 , and if the user does not have access to B_1 the system asks for KC_1 . The access hierarchy is therefore enforced. The number of keys a user must "remember" depends on the hierarchical level of the files to which he has access. If he has access to C_1 and to C_2 but not to B_1 , he needs to remember both KC_1 and KC_2 . So "user convenience" is a complex function of the protection specifications and the access hierarchy. (*)

The disadvantage of this scheme is the overhead involved in that, in order to access lower level nodes, the system must first access higher level nodes, but this can be expected in a hierarchical scheme.

Returning to the discussion in Section 6.3.1, it is clear that the "standard" hierarchy (e.g. for WRITE access) can be implemented using the scheme just described for "independent access hierarchy". In the next section we discuss the important case of "reverse hierarchy".

6.3.3 READ Access = "Reverse" Hierarchy

In this case the access hierarchy is exactly the opposite of the directory hierarchy. Suppose we are given the directory structure shown in Figure 21. The corresponding access hierarchy is:

-
- (*) In a "pure" keys record scheme, key KA will be deciphered using the user key KU_i which is the only key the user has to remember.

$$\begin{array}{l} C_1 > B_1 > A \\ C_2 > B_1 > A \end{array}$$

$$\begin{array}{l} C_3 > B_2 > A \\ C_4 > B_2 > A \end{array}$$

$$\begin{array}{l} D_1 > C_4 \\ D_2 > C_4 \end{array}$$

Assume as before that file N is enciphered by KN. Suppose also that we have the following protection specifications. User U_1 has access to C_1 , and user U_2 has access to C_3 . Then user U_1 must also have access to A and to B_1 , while U_2 must also have access to A and B_2 . How do we enforce such protection specifications?

A simple method is to give a user all the relevant keys, i.e. giving KA, KB_1 , KC_1 to U_1 , and KA, KB_2 , KC_3 to U_2 . Such a scheme is not convenient because a user will have to remember too many keys.

We therefore seek a method for the system to generate the appropriate keys using the known access paths in the directory tree. We will show two methods to achieve this. The "collapsing" method and the "inverse directory" method.

6.3.3.1 The "Collapsing" Method

Suppose we have a "one way" cipher function - g, and a set of keys with the following relations (R).

$$\begin{array}{ll} KB_1 = g(KC_1) & KB_1 = g(KC_2) \\ KB_2 = g(KC_3) & KB_2 = g(KC_4) \quad (R) \\ KA = g(KB_1) & KA = g(KB_2) \end{array}$$

Let us postpone, for a moment, the discussion of how we can construct such relations and assume we have obtained the above set of keys with relations (R). The system will use these relations to generate the right keys. Assume, as before, that in order to traverse from A to B_1 , key KA is needed, and in order to traverse from B_1 to C_1 , key KB_1 is needed. The system now operates as follows. On a request for access to C_1 , the system asks for KC_1 . The system then generates keys KB_1 and KA and stacks them. It does this by using $KB_1 = g(KC_1)$ and $KA = g(KB_1)$.

After the appropriate stack of keys is built the system pops it up while traversing down the directory tree for accessing C_1 . A similar action will occur upon requesting access to C_3 .

This scheme has the advantage of high "user convenience". For example, a user who has access to C_1 and therefore to B_1 and A , needs to remember only one key, KC_1 . Its security depends on how hard it is to derive a key from its "brother" key. Two keys are called brothers if they both "hash" into the same key. For example, keys KB_1 and KB_2 are "brothers". If it is difficult to find one "brother" from another, then having access to C_1 will not render access to C_2 or other unauthorized files. As we see below, this task of finding the "brother" of a given key can be made very difficult.

The problem is then how to find such functions g and such a set of keys which will fulfill relations (R). One way to do this is by using a function with a large "collapse" or "degeneracy" (i.e. many keys are mapped by g into the same key). Purdy [PUR74] has investigated such functions. The set that he suggests, $g(x)$, is a set of polynomials, $P(x)$, modulo a very large prime number P . That is

$$g(x) \equiv P(x) \text{ modulo } (P).$$

The degeneracy of this system, i.e. the number of keys which are "hashed" into the same key is at most n , where n is the degree of the polynomial $P(x)$. n can be very large (e.g. 2^{24}) so the degeneracy of the system can be very large. However, the security of the system depends upon the size of P . For large P , (e.g. $2^{64} - 59$) even though the degeneracy is large, it is almost impossible to invert g . Trial and error is shown by Purdy to be impractical. So we have a function with large degeneracy with the further property that given a specific key it is very hard to find a "brother" for that key. How then does the designer exploit this to find a set of keys which satisfies the relations (R)?

The following idea was suggested by Prof. Rothstein. The designer can produce a large number L of random keys. If L is large enough he is able to get many keys with "brothers" in the set and which will satisfy relations (R) (see Section 6.3.3.2 for an explanation). Now, the designer has to do this only once, since he can produce enough keys to take care of additions and changes in the tree directory. The enemy's task is still very difficult since he needs to find the "brother" of a specific key and cannot use the "random generation" method used by the designer.

6.3.3.2 The Enemy's Task Compared to the Designer's Task

Assume Q keys in the keys space, and that the degeneracy of g is n , giving $m = Q/n$ different groups of keys. Suppose the designer produced L keys at random ($L \ll m$). What is the probability that they will come from distinct groups?

The number of ways to choose any L keys out of the m groups is $m^{**}L$. The number of ways to choose L keys from m distinct groups (where order is important) is $m!/(m-L)!$. Therefore, the probability that L keys come from distinct groups is

$$\text{Prob} = m! / (m^{**}L * (m-L)!)$$

Since m and L are very large, we can use the Stirling approximation and get

$$\text{Prob} = (m / (m-L))^{**}(m-L+1/2) * e^{**(-L)}$$

Taking logarithm and neglecting the $1/2$ we get

$$\text{LnProb} = -L + (m-L) * \text{Ln}(1/(1-L/m))$$

Since $L \ll m$, if we let $x = L/m$ we can approximate $\text{Ln}(1-x)$ by $-x$.

$$\text{LnProb} = -L - (m-L) * (-L/m)$$

$$\text{LnProb} = -(L^{**}2)/m$$

$$\text{Therefore } L \approx (-m \text{LnProb})^{**}1/2$$

We are interested in how large L should be in order that the probability of not getting brothers is very small. As an example, take this probability as $e^{**(-10^{**}0.5 * 10^{**}4)}$. Assume also $Q = 2^{**}54$ (a 54 bits key) and $n = 2^{**}24$. Then $m = 2^{**}30 \sim 10^{**}9$. Then

$$L = (10^{**}9 * 10^{**}4)^{**}1/2 = 10^{**}5$$

This means that in order to be certain to get some keys with "brothers" in the set, the designer will have to produce and check $10^{**}5$ keys, which is a reasonable number.

How hard is it for the enemy to find the brother of a specific key? The enemy searches all the keys in the key space sequentially, but their belonging to a specific group is almost random. Therefore the probability that he will hit the desired group is $a = 1/m$ (actually $(n-1)/Q$). The expected number of trials

before success is shown by Purdy to be $1/a$, which is equal to m here.

For the example above the enemy will have to check on the average $10^{**}9$ keys before successful penetration. This is much larger than the $10^{**}5$ keys the designer needs to check. The odds against the enemy can be vastly increased with ease (large Q) with small penalty to the designer.

6.3.3.3 The "Inverse directory" Method

This method is connected with the way in which the system finds the right path to a file. For example, how does the system find the right path $A \rightarrow B \rightarrow C$, as a result of an accessing request to C ? One method using an "inverse directory" was described by Reiter and Gudes (see Reiter et.al. [REI71]). There the "inverse directory" is called File Attribute Dossier - FAD). In an "inverse directory" every node in the "standard" directory is stored with its corresponding access path from the directory's root. As an example, the "inverse directory" shown in Figure 24 corresponds to the directory described in Figure 21. (*)

Figure 24 - An Inverse Directory

The search for the right path is done as follows. Assume a request for C_1 . The system finds C_1 in the "inverse directory", then it traverses down until it comes to a terminal node. Along the way it stacks all nodes which it passes. After the stack has been built the system goes through the real directory while popping up the stack.

(*) It could also be represented as an "inverse" tree, but then the structure of each node would be more complicated.

We now add one extension to this scheme. Each node of the "inverse directory" contains, beside its name, also a "keys record" with the corresponding keys. For example, in the path for C_1 , nodes A and B_1 above have the following structure:

A	KB_1A'
B_1	KCB_1'

where $KB_1 = C^{-1}(KC_1, KCB_1)$ and $KA = C^{-1}(KB_1, KB_1A')$. KC_1 is the key given by the user and it is used for deciphering the other two keys, KA and KB_1 . The stack is now built as shown in Figure 25.

A	KA
B_1	KB_1
C_1	KC_1

Figure 25 - The Stack Structure

The system can now use this stack to access the real directory as in the "reverse hierarchy" case.

6.3.4 Summary

This section discusses schemes to implement hierarchical protection specifications using UCC transformations, with emphasis on file hierarchy. We showed that the access hierarchy does not have to be the same as the directory hierarchy, and in particular we investigated the "inverse hierarchy" case. We showed an interesting application of "one way" ciphers in the "inverse hierarchy" case. We did not discuss at all the "user hierarchy" case (i.e. $U_1 > U_2 > U_3 \dots$ etc.) If the "user hierarchy" is stored in the system, then a scheme similar to the "standard" hierarchy scheme can be used. More research can be done on the combined case of both user and file hierarchy.

6.4 Data Dependent Protection Specifications

6.4.1 Introduction

There are many forms for Data Dependent (DD) protection specifications. A recent dissertation by McCauley [McC75] deals with this subject extensively. The schemes which we describe in this section are based mainly on McCauley's dissertation.

McCauley's work is based on concepts presented in two earlier papers. The "generalized file structure" by Hsiao and Harary [HSI70], and the idea of "file atoms" by Wong and Chiang [WON71]. For the generalized file structure Hsiao and Harary defined the concepts of keywords and records (see also Chapter 4, Example 4), and they showed an efficient algorithm for the retrieval of queries composed of boolean expression of keywords. Wong and Chiang showed that the existence of keywords and records defines a boolean algebra over the set of records. Suppose there are n keywords: $k_1, k_2, \dots, k_n$. There are 2^n possible minterms of the form $k_1 k_2 \dots k_n$, where k_i is either k_i or its negation $\bar{k}_i$. Each record fulfills one and only one of these minterms. The collection of all records which fulfill exactly one minterm is an atom. Usually there are much fewer atoms than the maximum number of 2^n , since an atom is defined as non-empty. The important property of this organization is that any boolean expression can be viewed as the union of minterms. Therefore the response to a query, which is expressed as a boolean expression of keywords, is composed of the union of atoms.

McCauley applies this boolean algebra to security. He defines the concept of security keywords. Security keywords are those keywords which are used for specifying protection. These security keywords induce a partitioning of the file to security atoms. A security atom corresponds to a minterm of security keywords.

It is useful to look at this model using a bi-partite graph. Assume the security keywords are on one side of the graph and the records on the other. An example is shown in Figure 26.

Figure 26 - Security Atoms

A line from K_i to R_j means - record R_j contains security keyword K_i . We have here three atoms: Atoms $[R_1]$, atom $[R_2]$, and atom $[R_3, R_4]$. The reason that the third atom contains two records is that records R_3 and R_4 contain exactly the same security keywords. Further examples can be found in [McC75].

McCauley defines several types of protection specifications. The most important one, from our point of view, is TYPE.5 protection specifications. They are defined as:

TYPE.5 (U , B , (permit/deny))

where U is the set of users, and B is a boolean expression of security keywords. (Actually, B also includes the type of access required, but we can eliminate it because we are interested only in READ access.) The meaning of TYPE.5 specifications is that all records which fulfill the boolean expression B, i.e. contain the keywords for which B is true, are permitted (denied) to the particular user. This is a general type of DD protection specifications and therefore powerful.

Referring back to the concept of security atoms, it is easy to see that the relationship between a query and an atom in Wong and Chiang is similar to the relationship between TYPE.5 specification and a security atom in McCauley. TYPE.5 protection specification will in effect define the set of atoms permitted (denied) to a user. The important point here is that security atoms are complete units in relation to access. Therefore all records in a security atom are allowed the same access and the security atom is now the unit of access in analogy to the file in the file system case. The partitioning of the file to security atoms does not change even if the protection

specifications: change, since this partitioning depends only on the records' content. This also justifies looking on a security atom as a complete unit in relation to access.

In TYPE.5 specifications one record is the smallest possible atom. That is, even though the access to a record depends on the value of its individual fields, the access to all fields of a record is the same, and all fields in a single record are protected in the same way. We call this type of protection specification: Data Dependent Record Oriented (DDRO).

We would like to introduce a more general type of protection specification. Assume that the record is logically divided into two parts as shown below.

S_1	S_2	...	S_m	A_1	A_2	...	A_n
-------	-------	-----	-------	-------	-------	-----	-------

$S_1, S_2, \dots, S_m$ are the security attributes, and $A_1, A_2, \dots, A_n$ are non-security attributes (i.e. do not produce keywords which participate in B). We define TYPE.6 protection specification as follows.

TYPE.6 (U , B , permit/deny , A)

A is a set of non-security attributes $A_1, A_2, \dots, A_n$. This specification means that if a record fulfills the Boolean expression B then only the attributes in A are permitted (denied) for access. TYPE.6 becomes TYPE.5 when A includes all the attributes in the record. TYPE.6 specifications allow us to define the following example specification: A medical clerk can see only the medical fields (non-security attributes) of employees who are not head of a department (security keyword). We call this type of protection specification: Data Dependent Attribute Oriented (DDAO). In the following sections we will show schemes to enforce DDRO or DDAO protection specifications.

6.4.2 DDRO Protection Specifications

As was explained above the security atoms are completely analogous to files in the file system. Therefore we would think that schemes which were used for the file system can also be used here. This is in general true but we need some modifications. If we look on the security atoms as "columns" in the access matrix, then similar assumptions on this matrix, to those made in the file system, can also be made here (see Section 6.2). The main one is that each atom is unique in its access specifications or that all

atoms which are allowed the same access are "grouped" into one atom. Using the notation from Section 6.2, given n atoms $A_1, A_2, \dots, A_n$, we need n different keys $K_1, K_2, \dots, K_n$ to encipher them.

Now we can use any one of the three schemes described for the file system. We would like to make some comments on each of them.

The "simple" scheme is to give each user the exact set of keys which he needs for accessing his permitted set of atoms. This scheme has two major disadvantages. The first also existed in the file system case and it is the low "user convenience". But the situation here is even worse, since the number of atoms may be very large. The second disadvantage is unique to the DD case. The problem is that when a user issues a query he does not know ahead of time which atoms will be retrieved in response to his query. Therefore he does not know which cryptographic keys he will need. This causes a serious problem in the user/system interface. The next two schemes overcome this problem.

The second scheme is very similar to the "user profile" scheme. In McCauley's dissertation it was suggested that each user will have a protection cluster list (a "cluster" can be viewed as equivalent to an atom here.) We extend this idea and put the cryptographic keys in the protection cluster list. This list will now look like:

$$\{ \text{User id, (Atom 1, } K_1), (\text{Atom 2, } K_2), \dots \}$$

The protection cluster list is, of course, enciphered by the user key KU_i . This scheme has similar advantages and disadvantages to those of the "user profile" scheme, i.e. high "user convenience" on the one hand, and need for authentication and sensitivity to changes in the protection specifications on the other hand. The important point is that it overcomes the problem described before for the "simple" scheme. The system, in response to a query, uses the protection cluster list and intersects it with the list of clusters which can answer the query (the exact algorithm is described in [McC75]). It then uses the user key KU_i to decipher the appropriate cryptographic keys and the user is free from worrying which atoms/keys will be used during the query execution process.

Scheme 3 is completely equivalent to the "keys record" scheme of the file system. Here, we add to each atom a

"keys record" as was done for the files. All the advantages and disadvantages of scheme 3 in case of the file system, including the "key inversion" problem, exist here too. However, in the DD case, in contrast to the file case, it is not expected that one user will have different keys for different atoms. As was explained above, this is because the user is not expected to know which atoms will be retrieved as a result of his query. A "key non-invertible" cipher is therefore essential here.

6.4.3 DDAO Protection Specifications

In this section we show one scheme to implement DDAO (TYPE.6) protection specifications. Since DDAO specifications are also based on boolean expression of security keywords, the atom structure is not affected. That is, we have the same list of atoms as before. However, for different users there will be different non-security attributes permitted (denied). This is because the set A in TYPE.6 specifications is different for different users. The situation may be even more complex than allowed in TYPE.6 specifications. It can be that even for the same user, for different atoms, different attributes are permitted (denied). An example for the last case is shown in Figure 27.

How can we enforce such protection specifications? The simplest way is to use a variation of the "keys record" scheme. This is shown in Figure 28.

	User U ₁	User U ₂
Atom 1	Atr ₁ , Atr ₂ - YES	Atr ₃ , Atr ₄ - YES
	Atr ₃ , Atr ₄ - NO	Atr ₁ , Atr ₂ - NO
Atom 2	Atr ₁ , Atr ₃ - YES	Atr ₁ , Atr ₂ , Atr ₃ - YES
	Atr ₂ , Atr ₄ - NO	Atr ₄ - NO

Figure 27 - DDAO Specifications

Corresponding "keys record" scheme

Figure 28. - DDAO "Keys" Records.

KAT_{ij} is used to encipher attribute A_j in atom i . This is because we need different keys for the same attribute in different atoms.

KAT'_{ij} is used for validation purposes.

KA_{ijk} is the ciphered form of KAT_{ij} for user U_k . That

is, $KAT_{ij} = C^{-1}(KU_k, KA'_{ijk})$

where KU_k is the "user key" for user U_k .

Beside the complication for different attributes, this scheme is similar to the "keys record" scheme and has all its advantages and disadvantages.

More research can be done on other very general protection specifications.

6.5 Summary

In this chapter we discussed the methods of designing a secure file system and of enforcing different types of protection specifications by using user controlled cryptographic transformations. We described schemes to implement

Compartmentalized, Hierarchical and Data Dependent protection specifications. We showed several schemes to implement each of these types and discussed the advantages and disadvantages of each scheme. We stated the basic design principles of these cryptographic schemes which make them a general and powerful tool for protection. We discussed the security which each scheme provides and we defined other criteria such as : "user convenience" and "system risk" to evaluate their effectiveness. We showed that the different schemes provide different degrees of "user convenience" and "system risk". The designer of a secure file system based on user controlled cryptographic transformations can now use these criteria in order to choose the scheme which best satisfies his objectives.

CHAPTER SEVEN

Evaluating the Security of File Enciphering

7.1 Introduction

In this chapter we suggest measures for evaluating the security of file enciphering. For any system which claims to be secure, one would like to have a measure that expresses its security either on an absolute scale or relative to other systems. The security of a system based on cryptography is cipher dependent. In general, the harder it is to break a cipher, the more secure is the system. For systems based on access control security is generally "all or nothing". One usually looks at access control as a complete separate unit, ignoring environmental factors (e.g. the probability of a hardware failure). One tries to prove that the access control mechanism is correct, which means that the system is secure (see [McC75]). The main advantage of cryptography based systems is that the environment is not ignored. It is assumed that the enemy has bypassed access control and has gotten a significant amount of the ciphered data. It is assumed that the enemy has some knowledge about the characteristics of the clear data and knows the kind of cipher used. Under these "worst case" assumptions, we want to know how much security the system provides.

In general, we will not "prove" that the system is completely secure, since this involves perfect ciphers which are very hard to control or synchronize (see [SHA49]). Generally, the cipher will not be perfect and the system is not 100% secure. However, it may be that the amount of work required to "break" the cipher is so huge that the system can safely be considered secure. Our main goal will be to develop a measure for the amount of work required to break a cipher in case of file enciphering. As in Chapter 6, we consider files rather than general data bases, because it simplifies the discussion and can be generalized.

Most of this chapter will be devoted to developing our measure for the security of file enciphering. This measure is called: "The work factor of a statistic", and it measures the difficulty of "breaking" a particular cipher using a particular statistic. Higher values for this measure mean more secure ciphers and therefore more secure systems. The security of a cipher is not the only factor which determines the security of cryptography based systems. It was shown,

for example, in Chapter 6 that some ciphers are better than others for a particular cryptographic scheme (see Section 6.2.4 - the "key inversion" problem). This and other factors will be discussed in Section 7.6.

Before we can develop our measure we review Shannon's work on evaluating the security of cryptographic systems [SHA49]. It is still the theoretical basis for most current work in cryptography (we, however, depart from Shannon's work as will be seen later). It is based on Information theory and the reader of the next section is therefore assumed to know the basic concepts of information theory.

7.2 Review of Shannon's Measures

We cannot review here all the points discussed by Shannon [SHA49]. We concentrate on the points relevant to later discussion.

A cipher system is a function $E = f(M, K)$

where

E	-	cryptogram
M	-	message
K	-	key

It is preferable to think about this function as a family of functions parameterized by key. That is

$$E = T_i(M)$$

where T_i means the transformation using key K_i . For all practical purposes T_i has a unique inverse. (Not true for "one way" ciphers). That is

$$M = T_i^{-1}(E).$$

Using these definitions Shannon discusses the algebraic properties of cipher systems.

In order to evaluate the security of cipher systems Shannon uses concepts from Information theory. He assumes messages are coming from an ergodic source with given a priori probabilities $P(M_i)$. Similarly he defines a priori probabilities $P(K_i)$ of choosing a particular key from a key "source", and the a priori probability of getting a particular cryptogram is $P(E_i)$. He then defines the following conditional probabilities:

$P_M(E)$ - the probability of getting cryptogram E given the message M . It is the sum of the probabilities of all keys which produce E from M .

$P_E(M)$ - the probability that given the cryptogram E the message was M.

The cryptanalyst intercepts E and therefore likes to maximize $P_E(M)$, i.e. to find that M for which $P_E(M) = 1$ while $P_E(M^1) = 0$ for $M \neq M^1$. Shannon then defines the concept of a perfect cipher. A necessary and sufficient condition for perfect secrecy (i.e. perfect cipher) is:

$$P_M(E) = P(E)$$

which by Bayes theorem means

$$P_E(M) = P(M)$$

That is, the a posteriori probability of the clear message being M is not greater than its a priori probability. In other words, having the message in ciphered form gives the enemy no more information about the original message than he has before intercepting the ciphered message. As an example, he proves that a VERNAM cipher with random key is a perfect cipher.

The discussion above applies to discrete spaces M, K, and E. However, in practice, E_i , M_j and K_k can be of any length and the security of a cipher system will depend on these lengths. Shannon shows by an example how the a posteriori probabilities increase as the length of the intercepted text increases. For further understanding of this phenomenon Shannon introduces the concept of equivocation, which measures the remaining uncertainty about M after interception of the cryptogram. The key equivocation is defined as:

$$H_E(K) = \sum_{E,K} P(E,K) \log P_E(K)$$

and the message equivocation is defined as:

$$H_E(M) = - \sum_{E,M} P(E,M) \log P_E(M)$$

These equivocations are defined for messages (keys) of fixed length N. In general the key equivocation depends on N and Shannon denotes it as $H_E(K,N)$. He shows that the key equivocation decreases as N increases. If it is equal to zero, the a posteriori probability of one key is one, and that of all the other keys is zero. That is, we know the unique key and so have broken the cipher. If the key equivocation is "close" to zero, the cipher is effectively broken since one key has become very probable.

Shannon then investigates how the key equivocation approaches zero, as message length increases, for an ergodic

source and a random cipher. In this very common case the equivocation curve is fitted by two parts. The first part decreases linearly with N .

$$H_E(K, N) = H(K) - D * N$$

The second part which starts at the point $N = H(K)/D$ decreases exponentially with N . $H(K)$ is the entropy of the key source. D is the redundancy of the language. D can be computed as D_N/N . D_N is the redundancy per letter and it is computed by $D_N = \log G - H(M)$ where G is the total number of possible messages of length N and $H(M)$ is the entropy of the message source.

The point $H(K)/D$ is then a crucial point since beyond it the key equivocation approaches zero. Shannon calls this point the Unicity Distance. This is his main measure for the security of a cipher system. If N , the length of intercepted text is less than the unicity distance, then there is no unique solution and the cipher is essentially secure. If N is greater than the unicity distance, then there is a unique solution, though in practice it may be hard to get it. As an example Shannon computes the unicity distance in the case of English source and monoalphabetic cipher and gets the value of 27. That is, after 27 characters of ciphered text are intercepted we are virtually guaranteed to get a unique solution.

In practice, the enemy may intercept a message much longer than the unicity distance, and still not be able to "break" the cipher in a reasonable amount of time. Shannon defines a second measure - the work characteristic (work factor) of a cipher - $W(N)$. "Good" ciphers should have $W(N)$ which remains high even when N exceeds the unicity distance. The way to estimate $W(N)$ is by investigating the tools which are used by the cryptanalyst. These are usually statistical tools (of course the cryptanalyst can use enumeration of all possible keys since there is a unique solution, but this method is usually too expensive). The cryptanalyst measures the statistical properties of the ciphered text and by comparing them to the statistical properties of the source (which are assumed "roughly" known), he deduces the more probable keys. Increasing the work factor means "hiding" these statistical properties. To increase the work factor Shannon suggests the following principle:

A considerable amount of key should be used in enciphering each small element of the message.

Shannon also suggests two methods to counter statistical tools. The first is the method of Diffusion in which statistical properties of small segments of the message (e.g.

digrams) are "dissipated" into very long statistics. These make $W(N)$ decrease only for extremely large N . The second is the method of Confusion in which there is a very complex relationship between the statistic of the clear and ciphered data. (For instance, there is no simple relationship between the single character frequency of the clear and ciphered text in case of VERNAM cipher.) This also increases $W(N)$.

Even with the realization of the importance and practicality of the "work factor" Shannon does not show how to compute it. Other researches have concentrated mainly on the unicity distance. Tuckerman [TUK70] computed it for the VIGNERE cipher. Stahl [STA74] and Matyas [MAT74] computed it for the Homophonic cipher. As we shall see in the next section, for a file system, the important measure is the work factor, and not the unicity distance, mainly because of the large amount of ciphered data assumed intercepted in such a system.

To summarize, in this section we have reviewed some of Shannon's work. We introduced the two important concepts of "unicity distance" and "work factor" which we shall use in the next sections.

7.3 The File as a Message Source

7.3.1 A "Non-Ergodic" Source

A major assumption of Shannon's work is that messages come from an ergodic source. This means all long messages have similar statistical properties, and the longer the message, the "better" its statistical properties "match" the source's statistical properties. This is why long intercepted cryptograms give more information to the cryptanalyst than short cryptograms. This fact also serves as the basis for the definition of "unicity distance".

The situation in the case of enciphering files is drastically different. First, a file can be considered as either one message or a set of messages. In each case the length of each message is finite. This contrasts with the ergodic source in which messages of any length are possible. Second, a file, even if it was created by "natural" processes, does not exhibit "consistent" statistical properties(*). Part of a file does not necessarily exhibit similar properties to the whole file (in contrast with the ergodic source). Also the whole file exhibits "strange" statistical

(*) We are actually concerned with one field throughout the file, but the discussion is similar for all fields.

properties because of the many repetitions of one value. This is apparent from the file statistics which are used in our experiment described in Chapter 8. (see also Figure 30). Third, in Shannon's analysis the assumption is that the enemy knows the statistical properties of the source (e.g. English) and therefore uses them in the cryptanalysis process. The validity of these assumptions is questionable in the file case. Finally, our assumption throughout this thesis is that the enemy has access to all of the ciphered file, and therefore there is no point in talking about short or long cryptograms.

Because of the above differences between a file source and an ergodic message source, much of Shannon's analysis is not relevant to the file case. However, the concepts of "unicity distance" and "work factor" are still very useful, particularly the latter.

Since we assume that the whole ciphered file is intercepted then we have a constant length cryptogram (or a set of cryptograms) of length N . According to the enemy's knowledge about the source file and the type of cipher used, there may or may not be a unique solution. If the enemy knows little about the source file, then a very simple cipher can be secure. We then have two main cases to consider:

- 1) There is no unique solution. This is analogous to having $N < \text{unicity distance}$, and can happen if the enemy knows very little about the statistical properties of the source file.
- 2) There is a unique solution which can be achieved by enumeration. This can happen only if the enemy can compare a possible solution to his knowledge about the source file and find that only one solution is possible.

In order to distinguish between these two cases we need more precise assumptions about the enemy's knowledge. This is discussed in Section 7.3.3.

In this chapter we deal with case 2 - unique solution. This is realistic because files usually contain large quantities of data, usually short keys are used and usually the enemy knows something about the file. The problem of the enemy is how to achieve the unique solution in a reasonable amount of time. The problem of the designer is to devise ciphers making the "work factor" so high that the enemy will be deterred.

Our goal in the following sections will be to develop a measure for the "work factor" of a file cipher - a cipher

which is applied to a file. This measure can be computed even when we are not sure whether we have case 1 or case 2, since a very high work factor is practically equivalent to case 1. This measure will depend on the statistical properties of the file, on the type of cipher used, and on the assumed knowledge of the enemy.

In order to develop this measure we need some more definitions.

7.3.2 Definitions

In Section 7.2 we defined a cipher as a function. It will prove useful later to view a cipher as a relation C

where $C = (E, M, K)$

That is, if $E_i \in E$, $M_j \in M$, $K_n \in K$ and $(E_i, M_j, K_n) \in C$

then $E_i = f(M_j, K_n)$

We can now introduce the following relational definitions:

Definition 1 A cipher is uniquely decipherable (UD) iff.

$\forall i, j, r, s$ if $(E_r, M_j, K_s) \in C$ and $(E_r, M_i, K_s) \in C$
then $M_i = M_j$

That is, given a unique key and cryptogram, we get a unique message.

Definition 2 A cipher is uniquely encipherable (UE) iff

$\forall i, j, r, s$ if $(E_i, M_r, K_s) \in C$ and $(E_j, M_r, K_s) \in C$
then $E_i = E_j$

That is, given a unique key and message, we get a unique cryptogram.

Every cipher in Shannon's work is both UD and UE. We will have use for greater generality in the case of file ciphers.

It is possible to view the clear file as one message M , and the ciphered file as one cryptogram E . However, sometimes it is convenient to partition a file into fields or records and to view M as the concatenation of several short messages $m_1, m_2 \dots m_n$. Similarly the ciphered file E

can be partitioned into $e_1, e_2 \dots e_r$. With no loss of generality we can take $r = n$ as we can adjoin nulls to the partition with the smaller number of elements. It is still convenient to have one key for the whole file K . (Even if different keys are used for different records, they can be viewed as "parts" of one key.)

There are many ways to partition M into $m_1, m_2 \dots m_n$. Some of these partitions correspond to the statistical analysis we would like to do on the file. For example, if $\text{length}(m_i) = 1$ then we deal with the single character statistic. If $\text{length}(m_i) = 2$ then we deal with the digram statistic. If $\text{length}(m_i) = \text{length}(\text{field})$ then we deal with the field statistic. Let us assume that for a particular statistic the same partition is applied to M and to E (i.e. $\text{length}(m_i) = \text{length}(e_i)$). We now have two ordered sets:

$$M = (m_1, m_2, \dots, m_n)$$

$$E = (e_1, e_2, \dots, e_n)$$

where e_i is the ciphered form of m_i . (*) There is certainly a relationship between the two sets. We call this relationship the statistic cipher, since it is related to a particular partition and therefore to a particular statistic. We can now have the following definitions:

Definition 3 A statistic cipher C_S is a relation $C_S = (e_i, m_i, K)$ where $e_i \in E, m_i \in M$ and $(E, M, K) \in C$, where C is the file cipher. In other words e_i is the ciphered form of m_i .

Definition 4 A statistic cipher is uniquely decipherable iff:

$$\begin{aligned} \forall i, j \text{ such that } (e_i, m_i, K) \in C_S \text{ and } (e_j, m_j, K) \in C_S \\ \text{if } e_i = e_j \text{ then } m_i = m_j. \end{aligned}$$

Definition 5 A statistic cipher is uniquely encipherable (UE) iff:

$$\begin{aligned} \forall i, j \text{ such that } (e_i, m_i, K) \in C \text{ and } (e_j, m_j, K) \in C_S \\ \text{if } m_i = m_j \text{ then } e_i = e_j. \end{aligned}$$

All our file ciphers will be both UE and UD. However, their different partitions may or may not result in UD or

(*) This is not a restriction on the type of cipher used, since in the most general case $n=1$, i.e. no "partition" is done.

UE statistic ciphers. As examples, a monoalphabetic substitution makes the single character statistic cipher both UE and UD. A character homophonic cipher makes the single character statistic cipher UD but not UE.

One partition of the file is particularly important and it is also discussed in Chapter 8. This is the partition of the file into fields. (We assume only one field is enciphered in each record.) The corresponding statistic is the field statistic. The following definitions will be used in Chapter 8.

Definition 6 A file cipher is almost retrievable if its corresponding field statistic cipher is UD.

Definition 7 A file cipher is retrievable if its corresponding field statistic cipher is both UE and UD.

As examples, a VERNAM cipher with key length equal to the field length is retrievable. The homophonic field cipher in which homophones are applied to fields instead of to characters (see details in Chapter 8) is almost retrievable.

We now introduce the concept of distribution. Given the clear file M and a partition

$$M = (m_1, m_2, \dots, m_n)$$

some of the elements in the partition may be equal. That is, it may be that $\exists i, j$ s.t. $m_i = m_j$ for $i \neq j$. We can divide M into equivalence classes $M_1, M_2, \dots, M_k$ where if $m_i \in M_p$ and $m_j \in M_p$ $i = j$ then $m_i = m_j$. The number of elements in an equivalence class is called the class frequency. We define the distribution corresponding to partition M as the set of equivalence classes

$$M_1, M_2, \dots, M_k$$

and their corresponding frequencies

$$f_1, f_2, \dots, f_k$$

denoting it as $D(M)$.

Similarly, we define $D(E)$.

We define a statistic S as a pair of distributions $D(M)$ and $D(E)$. This is shown in Figure 29.

A Statistic S

<u>Distribution D(M)</u>		<u>Distribution D(E)</u>	
	<u>frequency</u>		<u>frequency</u>
M_1	f_1	E_1	g_1
M_2	f_2	E_2	g_2
$\vdots$	$\vdots$	$\vdots$	$\vdots$
M_s	f_s	E_t	g_t

Figure 29 - A Statistic S

Clearly, in Figure 29 $n = \sum_{i=1}^s f_i = \sum_{j=1}^t g_j$.

The importance of the definition of "statistic" and "statistic cipher" should be clear. A cryptanalyst will partition the ciphered file and measure its statistical properties. He will then compare them to his knowledge of the clear file statistical properties. Thus, a cryptanalyst will have several statistics $S_1, S_2, \dots, S_k$ for analysis. Some of the corresponding statistic ciphers will be UD, some will be UE, and some will be neither UE or UD.

Before we can use the definitions above in developing our measure we need to state our basic assumptions.

7.3.3 The Assumptions

An important variable on which the work factor depends is the assumed knowledge of the enemy about the clear file. If the enemy knows all the distributions of the clear file then he knows the clear file. (Actually, one distribution for which length (m_i) = length (file) is enough.) We therefore do not assume that the enemy knows all the statistical distributions of the clear file.

We limit ourselves to two different assumptions:

- 1) The strong assumption. Assume the enemy knows some (but not all) of the statistical distributions of the clear file $D_1(M), D_2(M), \dots, D_k(M)$. For example, he might know single character or digram frequency or field frequency. Since the enemy knows the whole ciphered file, this means that the enemy knows the statistics $S_1, S_2, \dots, S_k$. The assumption is that this knowledge is enough to get a unique solution by enumeration.

- 2) The weak assumption. The enemy knows only part of some distributions of the clear file. For example, he might know what is the most frequent field in the clear file, but no more than that. The enemy therefore knows only parts of some statistics $S_1, S_2, \dots, S_k$. The enemy's knowledge may or may not be enough for obtaining a unique solution.

In reality, the weak assumption is more often applicable than the strong assumption, but it makes the analysis much harder. In this chapter we will use the strong assumption only. This can be justified by the observation that the strong assumption is analogous to a "worst case" analysis. If we find a high work factor based on the strong assumption then we are "pretty safe".

Two other assumptions which are used throughout this thesis are that the enemy has access to all the ciphered data and that he knows the type of the cipher used.

7.4 Combinatorial Approach to the Work Factor Measure

7.4.1 The Cryptanalyst's Task

In order to define the work factor we need to look on the way a cryptanalyst works. A cryptanalyst may use the method of enumerating all possible keys in the key space. This will be done as follows:

- 1) Choose next key and decipher the file.
- 2) Does resultant clear file match the known distributions $D_1(M), D_2(M), \dots, D_k(M)$? (These distributions are assumed known by the strong assumption.)
- 3) If yes - STOP (we assume a unique solution)
If no - go to 1.

This process is of course, in all but trivial cases, too expensive. Usually, the cryptanalyst uses his knowledge of the ciphered file and the statistics $S_1, S_2, \dots, S_k$ to reduce the number of possible keys. This can be demonstrated by the following example. Assume a monoalphabetic file cipher. Assume that the most frequent character in the clear distribution is E, while the most frequent character in the ciphered distribution is X. Since the statistic cipher C_c is both UD and UE we are assured that X is indeed the substitution for E. By this "match" we have reduced the size of the key space from $26!$ to $25!$. The process above can be repeated for each character and statistic until only one key remains possible. (The case of monoalphabetic substitution is interesting because the corresponding statistic cipher is both UD and UE and therefore the clear and ciphered distributions are identical. Even then there may be more than one possible key. This occurs if

several clear (ciphered) characters have the same frequency, since in that case we don't exactly know which ciphered character corresponds to which clear character.)

Another example is shown below.

Character homophonic cipher

message: EABE

cryptogram: UVWX

Single character distributions D(M) and D(E)

D(M)		D(E)	
E	2	U	1
A	1	V	1
B	1	W	1
		X	1

Without using any of the distributions above, the number of possible keys is $3^{**}4 = 81$. With the use of these distributions and the assumed known fact that the cipher is character homophonic, which is "uniquely decipherable", we have

Possible keys (mappings)

E → U, V	E → U, V	E → U, W	E → U, X
A → W	A → X	A → V	A → X
B → X	B → W	B → X	B → V
			...

All together, there are 12 possible keys.

So we have reduced the number of points in the key space from 81 to 12. For further reduction we need to use another statistic (e.g. the diagram statistic). Clearly this reduction would not have been possible without the cipher being UD.

In general, the amount of work the cryptanalyst has to do depends on the character of the various known statistics. More precisely, it depends on the number of possible mappings between the clear distribution and the ciphered distribution. We call this number of possible mappings NP. NP depends on the particular statistic used and we sometimes denote it as NP_i for NP of statistic S_i .

In the examples above NP was equal to the number of keys in the reduced key space which remained to be checked by the cryptanalyst. This is the reason why NP is a natural measure for the amount of work the cryptanalyst has to do. Even if,

for some statistics, NP is not equal to the number of possible keys (as, for example, in the case of the "header" statistic to be described in Chapter 8), it is still a good indication for the amount of work the cryptanalyst has to perform when using that statistic. So our measure of a work factor will be some function of this number NP.

One comment is appropriate here. NP_i measures only the effect of a particular statistic S_i . Since a cryptanalyst uses all known statistics, the real work factor is a function of all $NP_1, NP_2, \dots, NP_k$. In some instances the work factor will depend only on the minimum of $NP_1, NP_2, \dots, NP_k$, but in general it will be a very complex function of all NP_i . We will not try to obtain this function here. Some suggestions will be discussed in Section 7.5. It is clear, however, that a designer would ideally like to have high values of NP_i for all assumed known statistics. In this section we try to estimate the amount of work associated with using a particular statistic. We call it the work factor of a statistic. It measures the amount of work (resources, time, money, etc.) the cryptanalyst has to invest in using a particular statistic for "breaking" the cipher. Examples which relate the work factor of a statistic to the total work factor will be given in Chapter 8.

The work factor of a statistic will depend mainly on the number NP. Our major goal is then to compute this number for each statistic. Most of this section will be devoted to this goal.

The problem of computing NP depends on the type of statistic cipher we have. We can distinguish between two main cases:

- 1) The statistic cipher is uniquely decipherable (UD). In this case a ciphered "character" comes from only one clear character, even though one clear character may result in several ciphered characters. (*)
- 2) The statistic cipher is not uniquely decipherable (NUD). In this case no restrictions are made on the mappings between ciphered characters and clear characters.

Corresponding to these two cases, the problem of computing NP can be transformed into two combinatoric problems.

(*) A "character" here stands for an element of the partition m_i or e_i .

7.4.2 The Combinatoric Problems

The two combinatoric problems are based on Figure 30. using the following analogy.

<u>Balls</u>		<u>Boxes</u>	
<u>colors</u>	<u>frequency</u>	<u>box</u>	<u>occupancy</u>
x_1	f_1	y_1	g_1
x_2	f_2	y_2	g_2
.	.	.	.
.	.	.	.
x_m	f_m	y_K	g_K

Figure 30 - The Combinatoric Problems

The colors $x_1, x_2, \dots, x_m$ and their frequencies correspond to the clear characters whose frequencies are given by the distribution $D(M)$. The boxes $y_1, y_2, \dots, y_K$ and their frequencies correspond to the ciphered characters whose frequencies are given by the distribution $D(E)$. The number of mappings between the two distributions, NP , is equal to the number of ways we can arrange the balls in the boxes.

This kind of terminology is traditional in combinatorics, and we will occasionally use this language in talking about the original problem. We are therefore faced with the following two problems:

Problem A (NUD case)

Given N balls of m different colors $x_1, x_2, \dots, x_m$. There are f_i balls of color x_i , i.e.

$$\sum_{i=1}^m f_i = N.$$

Balls of the same color are indistinguishable. We are also given K distinguishable boxes $y_1, y_2, \dots, y_K$. How many ways are there to arrange the N balls in the K boxes, such that there are exactly g_j balls in box y_j i.e.

$$\sum_{j=1}^K g_j = N.$$

Problem B (UD case)

The same problem as problem A but with the restriction that in each box all balls are of one color.

It should be clear that solving these problems will in effect compute NP, the number of possible mappings for a particular statistic in the two cases (UD and NUD). Furthermore, a computation of NP for problem B will always yield a value less than NP for problem A.

The combinatoric problems stated above are very hard to solve. We have consulted several references in trying to find a solution (Niven [NIV65]), Berman and Frayer [BER72b] p. 216, Riordan [RIO58] pp. 99-100, Wilson [WIL75].) None of these references provide a closed formula or a reasonable algorithm for the solution to either of these problems.

The problem is a counting problem which can be solved in general by the use of generating functions.

However, using generating functions will, as will be shown later, require an algorithm with essentially exponential time requirements in N - the total number of balls. Since N is usually quite large (it represents the number of "characters" in the file) the use of generating functions is generally impractical. It is instructive, however, to see several examples of solving some simple cases using generating functions.

7.4.2.1 Generating Functions

The function

$$f(t) = a_0 + a_1 t + a_2 t^2 + \dots + a_n t^n + \dots$$

is a generating function of the sequence

$$(a_0, a_1, a_2, \dots, a_n, \dots)$$

The function

$$H(X, Y, Z \dots) = \sum_{i_1, i_2, i_3 \dots} a_{i_1, i_2, i_3 \dots} X^{i_1} Y^{i_2} Z^{i_3} \dots$$

is a generating function of the coefficients

$$a_{0,0,0,\dots}, \dots, a_{i_1, i_2, i_3, \dots}, \dots$$

The function $H_1(X, Y, Z \dots)$

which generates the coefficients 1, 1, 1 ... is frequently used in combinatorics. For example in

$$H_1(X, Y, Z) = 1 + X + Y + Z + X^2 + Y^2 + Z^2 + XY + XZ + YZ + \dots$$

The terms of degree j correspond to the ways j indistinguishable

balls can be distributed in 3 distinguishable boxes X, Y and Z. For example, the term XY means one ball in box X and one ball in box Y. The number of such terms can be obtained by substituting $X = Y = Z = 1$ in all terms of degree j and summing them. Suppose now that we have j indistinguishable balls of one color, and k balls of another color. The number of ways the $j+k$ balls can be distributed in the three boxes is equal to the number of terms of degree $(j+k)$ in the product

$$H_1(X, Y, Z) \cdot H'_1(X, Y, Z) \quad (H'_1 \text{ stands for the second color})$$

If we restrict the problem to i_x balls in box X, i_y balls in box Y and i_z balls in box Z, where

$$i_x + i_y + i_z = j + k$$

then the number of ways to distribute the $(j + k)$ balls with these constraints is equal to the coefficient of $X^{i_x} Y^{i_y} Z^{i_z}$ in the product $H_1(X, Y, Z) \cdot H'_1(X, Y, Z)$

The relevant part of this generating function is a polynomial of degree $(j+k)$, so we need consider no terms of H_1 or H'_1 of higher degree than j or respectively, k i.e., the relevant parts of H_1 or H'_1 are also polynomials.

In the problem of interest we now have m polynomials corresponding to the m colors. Each polynomial involves K variables corresponding to the K boxes X, Y, Z ... The degree of polynomial P_i is f_i (frequency of color i). Each term in P_i corresponds to one distribution of f_i balls of color i in the different boxes. For example, the term X in P_i means one ball of color i in box X. The term XYZ means 3 balls of color i , one in box X, one in box Y, and one in box Z. We have to include all and only terms with degree = f_i which satisfy the constraints put on the boxes (e.g. x^3 in P_i is "illegal" if the number of balls in box X is only 2).

The reason that we can omit terms of degree less than f_i in polynomial P_i is that they can not contribute to the term of degree N in the product, whose coefficient we are looking for.

In the product $P_1 P_2 \dots P_m$ we seek the coefficient of the term of degree N

$$x^{g_1} y^{g_2} z^{g_3} \dots$$

Each contribution to this coefficient corresponds to one possible configuration in our problem. This coefficient is thus equal to NP, the number of mappings we want to compute.

Let us give some examples. In the following examples we will say generating function when we mean the relevant polynomial part of it.

A) Non-uniquely decipherable (problem A)

Example 1

<u>colors</u>		<u>boxes</u>		<u>mappings</u>	
a	2	x	2	a	a
b	1	y	1	a	b
				b	a

The (relevant part of the) corresponding generating function is:

$$(x^2 + xy)(x + y)$$

where the first factor is for color-a and the second for color-b. We are looking for the coefficient of x^2y (2 balls in box x, one in box y).

The result:

$$x^3 + \underline{x^2y} + \underline{x^2y} + xy^2$$

As there are two terms of x^2y , NP is 2.

Example 2

<u>colors</u>		<u>boxes</u>	
a	2	x	2
b	2	y	2
c	1	z	1

There are 11 mappings in this case, namely

	1	2	3	4	5	6	7	8	9	10	11
X	a	a	a	a	a	a	a	b	b	b	b
	a	a	b	b	b	c	c	b	b	c	c
Y	b	b	a	a	b	b	b	a	a	a	a
	b	c	b	c	c	b	a	a	c	a	b
Z	c	b	c	b	a	a	b	c	a	b	a

The generating function, where first factor-color a, second factor-color b (same as for a), third-c, is

$$(x^2 + XY + Y^2 + XZ + YZ)^2 (X + Y + Z)$$

We are looking for the coefficient of:

$$x^2 y^2 z$$

which after multiplying out and collecting terms is 11.

B) Uniquely decipherable (Problem B)

Here we have to set some constraints on the polynomials, e.g. if box x contains exactly 2 balls, the term X is "illegal" since it means only one ball of color i in box x.

Example 3

a	2	x	2
b	1	y	1
c	1	z	1

There are two possible mappings:

a		a
a	or	a
b		c
c		b

Generating function:

$$(x^2 + YZ) (Y + Z) (Y + Z)$$

The coefficient of x^2yz is 2.

Example 4

a	2	x	1
b	1	y	1
c	1	z	1
		u	1

There are 12 possible mappings

$$\frac{4!}{2! \cdot 1! \cdot 1!} = \frac{24}{2} = 12$$

Generating function:

$$(\underline{XY} + \underline{XZ} + \underline{XU} + \underline{YZ} + \underline{YU} + \underline{ZU}) (X + Y + Z + U)^2$$

The coefficient of $XYZU$ is 12 (each of the 6 underlined terms contributes 2).

The last examples illustrate the fact that the number of terms which have to be examined in order to obtain the desired coefficient grows rapidly with N , the number of balls (the number of characters in the file). In general, the number of terms in one polynomial P_i in K variables of degree f_i is at most equal to the number of non-negative integral solutions of the equation:

$$X_1 + X_2 + \dots + X_K = f_i \quad X_j \geq 0$$

which is known to be

$$\binom{f_i + K - 1}{f_i}$$

Therefore the number of terms we have to check is at most NT , where

$$NT = \prod_{i=1}^m \binom{f_i + K - 1}{f_i}$$

Actually, the number of terms we have to check is less than NT because of the constraints put on the boxes (e.g. the solution $X_K = f_i$ is "illegal" if $f_i > g_K$).

To show, however, "exponential behavior" it is enough to show one "bad" case. Assume that we have N balls of N colors, i.e. $m=N$ and $f_i=1$. Then the number of terms in each

polynomial is exactly $\binom{K}{1} = K$

and the number of terms we have to check is exactly NT where

$$NT = \sum_{i=1}^N \binom{K}{1} = K^N$$

Therefore the number of terms we have to check is exponential in N.

It is thus reasonable to assume that in many cases (especially those where K and f_i are "far" apart) the time required by the generating function method is exponential in N and so generally impractical. We therefore take a different approach to the above problems.

We abandon the search for an exact solution and seek an approximate solution. Since our original problem was to get high values for the work factor, which means high values for NP, then high lower bounds for the solutions of the combinatorial problems will be satisfactory. In that case we are sure that the exact solution is at least that high, and therefore the corresponding work factor is high, and the designer can rely safely on that cipher relative to that statistic. We will not be able to get much information from low values of the bounds, since low values for the bound may occur while the real NP is quite high. (See further discussion of this point in the next section.) We are looking therefore for "high lower bounds".

7.4.3 Bounds

7.4.3.1 Bound for the NUD Case.

Suppose we have N balls of m colors: $f_1, f_2, \dots, f_m$, and N distinguishable boxes, each containing one and only one ball. The number of possible configurations, denoted by N_C , is known to be

$$N_C = \frac{N!}{f_1! \cdot f_2! \cdot \dots \cdot f_m!}$$

Suppose now that we have g_1 indistinguishable boxes of type 1, g_2 indistinguishable boxes of type 2, ... g_K indistinguishable boxes of type K. The number of possible configurations, NP, is now smaller than N_C found above, because permutations of indistinguishable boxes which were counted in computing N_C should not be counted now (N_C can thus be used as an upper bound). Let us divide N_C by the number of possible permutations of indistinguishable boxes. The result is denoted by N_{CB} .

$$N_{CB} = \frac{N_C}{g_1! g_2! \dots g_K!} = \frac{N!}{f_1! \cdot f_2! \cdot \dots \cdot f_m! \cdot g_1! g_2! \dots g_K!}$$

But now N_{CB} is smaller than the real NP. Dividing by g_i reduces N_C excessively. For example, the permutation of boxes i and k where the two boxes contain balls of the same color j was already considered in dividing by $f_j!$ We conclude that

$$N_{CB} \leq NP$$

and N_{CB} can serve as a lower bound.

We apply this to examples 1 and 2 above.

<u>Example 1</u>	<u>colors</u>	<u>boxes</u>
	a	x
	a	x
	b	y

$$N_{CB} = \frac{3!}{2! \cdot 1! \cdot 2! \cdot 1!} = \frac{6}{4} = 1.5 < 2 \quad (\text{exact NP})$$

<u>Example 2</u>	<u>colors</u>	<u>boxes</u>
	a	x
	a	x
	b	y
	b	y
	c	z

$$N_{CB} = \frac{5!}{2! \cdot 2! \cdot 1! \cdot 2! \cdot 2! \cdot 1!} = 7.5 < 11 \quad (\text{exact NP})$$

From the examples above N_{CB} seems a reasonable lower bound. However, for some distributions N_{CB} is very low (less than 1) while the real NP is much higher. An example of such a distribution is shown in Chapter 8. N_{CB} is frequently a very conservative lower bound. A large value for N_{CB} guarantees that NP is large. If N_{CB} is low, NP might still be large and we might try to compute a better lower bound for that specific case. We made some attempts to get a general better lower bound than N_{CB} , but we did not succeed. More research here is desirable. It is also desirable to characterize cases for which N_{CB} will not give excessively low values, since it could then serve as a guideline for the designer. It can be shown, using Stirling approximation, that if $f \cdot g \leq N$ where f is the maximum of f_i and g is the maximum of g_j , that N_{CB} will be very high. This is also a "conservative" characterization, and N_{CB} will be high for many more cases.

In the rest of this thesis we use N_{CB} as the lower bound for NP, keeping its shortcoming in mind.

To summarize, in the NUD case we set

$$NP = N_{CB} = \frac{N!}{f_1! f_2! \dots f_m! g_1! g_2! \dots g_k!}$$

7.4.3.2 Bound For the UD Case

Here we do not give a formula but show an algorithm to compute the lower bound. We call it "algorithm AUD".

Algorithm to compute the lower bound for problem B (UD case)

The main idea here is that after assigning balls of some color to a box, we remove that box.

Algorithm AUD

1. Set $LB = 1$. Order colors and boxes in decreasing order of their respective frequencies.

$$f_1 \geq f_2 \geq \dots \geq f_m$$

$$g_1 \geq g_2 \geq \dots \geq g_K$$

2. If $g_1 = 1$ go to 5.

3. If $f_1 = f_2$ find largest J such that $f_1 = f_J$.
/* This is called a "tie" between $f_1, f_2, \dots, f_J$ */
Set $LB = LB * J$

4. Remove g_1 i.e. set $g_{i-1} = g_i$ $i = 2 \dots K$
and set $K = K-1$
Set $f_1 = f_1 - g_1$
If $f_1 = 0$ then remove f_1 (i.e. set $f_{i-1} = f_i$ & $m = m-1$)
Else sort $f_1, f_2, \dots, f_m$
go to 2.

5. Set $LB = LB * \frac{K!}{f_1! f_2! \dots f_m!}$
Halt.

The resultant LB is the lower bound for which we are looking.

Proof of the algorithm

In each step of the algorithm, (except 3), we count a smaller number of possibilities than the actual number because we assign only one color to each box. The only question is: can we count any possibility twice because of step 3?

This is impossible, since in each execution of step 3, the case of a "tie", we have J colors to choose from for inserting in box g_1 . After that we remove a unique box g_1 and we never come to it again. QED

ExamplesExample 3 (above)

<u>colors</u>	<u>boxes</u>	<u>NP</u>	<u>LB</u>
2	2		
1	1	2	2
1	1		

Example 4 (above)

<u>colors</u>	<u>boxes</u>	<u>NP</u>	<u>LB</u>
2	1		
1	1		
1	1	$\frac{4!}{2! \cdot 1! \cdot 1!} = 12$	12
	1		

Example 5

<u>colors</u>	<u>boxes</u>	<u>NP</u>	<u>LB</u>
4	2		
2	2	36	24
1	1		
1	1		
	1		
	1		

From algorithm AUD:

$$\begin{array}{rcl}
 \text{LB:} & \begin{array}{ccccccc} 4 & 2 & 2 & 2 & & & \\ 2 & 2 & 2 & 1 & 2 & 1 & \\ 1 & 1 & \rightarrow 1 & 1 & \rightarrow 2 & 1 & \\ 1 & 1 & 1 & 1 & 1 & 1 & \\ & 1 & & 1 & & & \\ & 1 & & & & & \end{array} & = 2 \cdot \frac{4!}{2 \cdot 1 \cdot 1} = 24
 \end{array}$$

Example 6

<u>colors</u>	<u>boxes</u>	<u>NP</u>	<u>LB</u>
5	3	1640	360
3	2		
2	2		
2	2		
1	1		
1	1		
	1		
	1		
	1		

From algorithm AUD:

$$\begin{array}{ccccccc}
 & 3 & 2 & 2 & 2 & 2 & 2 & 1 \\
 & 2 & 2 & 2 & 2 & 2 & 1 & 1 \\
 & 2 & 2 & 2 & 2 & 1 & 1 & 1 \\
 \rightarrow & 2 & 1 & \rightarrow & 1 & 1 & 3 \cdot & 1 & 1 & 6 \cdot & 1 & 1 \\
 & 1 & 1 & & 1 & 1 & & 1 & 1 & & 1 & 1 \\
 & 1 & 1 & & 1 & 1 & & 1 & 1 & & & 1 \\
 & & 1 & & 1 & & & & & & & \\
 & & 1 & & 1 & & & & & & & \\
 & & & & 1 & & & & & & &
 \end{array}$$

$$6 \cdot \frac{5!}{2!} = 6 \cdot 60 = 360$$

We see from the last examples that LB grows very fast, especially if we have a long tail of 1's in the boxes column. We will then take LB as the lower bound for NP.

To summarize, in the UD case, we set

$$NP = LB$$

LB is computed by algorithm AUD.

Complexity of Algorithm AUD

Step 1 is executed only once. The sort takes $K \log_2 K + m \log_2 m$ comparisons (see Knuth [KNU75] pp. 181 ff). Steps 2, 3, 4, are executed at most as many times as there are boxes, i.e. at most K times. Step 3 which looks for a "tie" takes at most m comparisons. Step 4 has to sort only one element. So it takes at most $\log_2 m$ comparisons. Maximum complexity (no. of comparisons) is then: $K \times (m + \log_2 m + \log_2 K) + m \log_2 m$. This is modest in comparison to the complexity of using generating functions or actual counting which is essentially exponential or factorial. Note that the average complexity is reduced because of the shortcut in step 5 for the long "tail" of 1's.

7.4.4 Cipher Dependency

In computing NP we did not use all of our (enemy) knowledge of the type of the cipher used. In the NUD case, it is very hard to use the knowledge of the general cipher for reducing NP. For example, suppose we have a digram substitution cipher (each pair of characters is replaced by a different pair), and suppose we are interested in the single character statistic. This statistic is clearly non-uniquely decipherable, but it is very hard to use our knowledge about the type of cipher to reduce the number of possible keys for this statistic.

In the UD case, the statistic cipher is equivalent to a general homophonic cipher. Here our knowledge about the cipher, for example, the number of homophones each character has, or the way in which these homophones are generated, can be used to reduce the number of possible keys.

The most general case is when the enemy does not know anything about the homophonic cipher (beside the fact that it is homophonic - i.e. UD). In this case our bound LB is applicable. In case the enemy does have some knowledge about the way homophones are generated, the number of possible keys might be much less than the general bound LB. It is clear, however, that we cannot analyze all different types of "knowledge".

One important cipher, which we also use in our experiment is the "retrievable" cipher, or in general, a UD & UE statistic cipher. In this case we assume that the enemy knows that the cipher is "retrievable" and therefore a different bound on the number of possible keys is needed.

A "retrievable" cipher, by definition, will cause the two distributions $D(M)$ and $D(E)$ to be equal. An example of a "retrievable" cipher is shown in Figure 31. The most common way to construct a "retrievable" cipher is to have one key used to encipher each field (message), and in the rest of this chapter we assume that the "retrievable" cipher is constructed in this way.

<u>D(M)</u>	<u>D(E)</u>
5	5
4	4
4	4
2	2
2	2
1	1
1	1

Figure 31 - A "Retrievable" Cipher

For example, in Figure 31 only one key is used for each field and by deriving the key for one field we can decipher any other field. An obvious example for such cipher is the CAESAR cipher. Another example is the VERNAM cipher with the same key for each field. In the case of "retrievable" cipher it is enough to check the section with the narrowest "tie". A "tie" of width J is a set $(f_i, f_{i+1}, \dots, f_{i+J}, g_i, \dots, g_{i+J})$ such that $f_i = f_{i+1} = \dots = f_{i+J} = g_i = \dots = g_{i+J}$. In the

example above the narrowest "tie" is the 5 - 5 section. (width 1); the 4 - 4 "tie" is of width 2. The narrowest "tie" is the most vulnerable, and once the key for it has been found, the cipher is broken.

If we denote NR as the number of possible keys for the "retrievable" cipher, then NR can be computed using the following algorithm - Algorithm AR.

Algorithm AR

1. Find the narrowest "tie". Suppose its width is J.
2. If $J = 1$ then $NR = 1$.
(That is, there is only one possible mapping and the key can be derived immediately.)
3. If $J > 1$ then $NR = \binom{J}{2} \cdot 2$ i.e. $NR = J \cdot (J-1)$
(It is enough to check only two fields from column D(M). Each of them is checked against all fields in the "tie" section of D(E). Once we get the same key for the two fields then this is the right key. (*))

Example

<u>D(M)</u>	<u>D(E)</u>
3	3
3	3
3	3
2	2
2	2
1	1
1	1
1	1

The width of the narrowest "tie" in this example is 2, therefore $NR = 2 \cdot 1 = 2$.

In Chapter 8 we use a "field homophonic" cipher which can be divided into two parts:

- H - the homophonic part. We assume that the enemy does not know how the homophones are generated.

(*) This is true only if the cipher is uniquely decipherable relative to keys, i.e. $(M, K_1, E) \in C$ and $(M, K_2, E) \in C \Rightarrow K_1 = K_2$. Most "retrievable" ciphers, e.g. VERNAM, are of this type.

R - the "retrievable" part. All fields are enciphered with the same key.

We assume that the enemy knows exactly which fields from D(M) belong to part H, and which belong to part R. The problem is that for some fields in D(E), it is not known to the enemy to which part they belong. The reason for this is that the enemy does not know how many homophones were used in part H for each field. If the frequency of Y_j (a ciphered field) is smaller than the maximum frequency in part R of D(M), then Y_j may have come either from part H or from part R.

If we denote the number of keys for part R as NR, and the number of keys for part H as NH, then $NP = NR + NH$. This is because the two parts are completely independent and can be analyzed one after the other. We call this "two part" cipher - partially retrievable. An example is shown in Figure 32.

Figure 32 - A "Partially Retrievable" Cipher

In the right half of Figure 32, part R of column D(E) is larger than part R of column D(M), because for some fields the enemy does not know from which part they come.

The following algorithm, Algorithm AH, will compute NP for the "partially retrievable" cipher.

Algorithm AH

1. Using the partition of column D(M) into R and H partition column D(E) into H and R. (If the frequency of ciphered field Y_j is larger than any other frequency of field X_i of part R of D(M), then Y_j belongs to part H of column D(E).)

2. Compute NR using Algorithm AR.
3. Remove part R from both columns. (We now know exactly which ciphered fields belong to part R in D(E), since we have found the right key for part R.)
4. Apply algorithm AUD to compute LB. Set $NH = LB$.
5. Set $NP = NH + NR$.

Example

Using Figure 32, from step 1 and 2 we get $NR = 5 \cdot 4 = 20$. After removing part R in step 3 we get

$$\begin{array}{ccc}
 5 & 2 & \\
 3 & 2 & \longrightarrow LB = 4 \\
 & 1 & \text{using algorithm} \\
 & 1 & \text{AUD}
 \end{array}$$

This will give $NH = 4$ and $NP = 20 + 4 = 24$. Another example of a "partially retrievable" cipher called the field homophonic cipher will be shown in Chapter 8.

It is important to note that if only partial deciphering is required, for example only the retrievable part, then it is enough to check only NR possible keys.

7.4.5 Summary

The goal in the last section was to compute NP - the number of possible mappings (keys) between the clear and ciphered distributions of a statistic cipher, assuming the enemy knows the statistic S and the type of cipher used. We were not able to compute NP exactly, but did find some lower bounds.

For the NUD case our bound is:

$$NP = \frac{N!}{f_1! \cdot f_2! \cdot \dots \cdot f_m! \cdot g_1! \cdot g_2! \cdot \dots \cdot g_k!}$$

and it is independent of the type of cipher used. For the UD case we gave an algorithm to compute a lower bound, called Algorithm AUD. If the enemy has no knowledge of the way homophones are generated, this bound is computed by algorithm AUD. In the case the enemy does have this knowledge, we computed the bound for "retrievable" and "partially retrievable" ciphers. For a "retrievable" cipher $NP = NR$, where NR is computed by algorithm AR. For the "partially retrievable" cipher $NP = NR + NH$, where NH is

computed by algorithm AH. NR is the exact number for the "retrievable" part, while NH is a lower bound for the homophonic part.

In the next section we show how NP is used to define a security measure called "the work factor of a statistic".

7.5 Definition of the Work Factor

7.5.1 The Work Factor of a Statistic

In Section 7.4.1 we showed that the work factor of a statistic depends mainly on the number NP. We, however, are interested in estimating the cost for the enemy of using a particular statistic for "breaking" the cipher. This cost will depend not only on NP but also on the type of cipher used, since each of the possible NP keys has to be further checked by deciphering the ciphered file using this key. The work factor of a statistic will be equal to this cost.

This cost is composed of several factors:

- 1) The cost of deciphering for each possible key. For each possible key, the clear file has to be generated by the transformation

$$M = C^{-1}(E, K)$$
 so that we can check this file further with the use of other statistics. This cost of deciphering is, of course, cipher dependent and file dependent. Assume that for a single key this cost is C_k . Then the total cost for all possible keys is:

$$CK = C_k * NP$$
- 2) The cost to obtain distribution $D(E)$. As is shown in the experiment in Chapter 8, this cost grows rapidly with the growth of the number of symbols in the ciphered alphabet. Since we have to search and sort, we approximate this cost as:

$$CD = C_d * K \log_2 K$$
 where K is the number of symbols in alphabet $D(E)$, and C_d is a constant.
- 3) A special cost for the "retrievable" and "partially retrievable" ciphers. We saw that in this case NP was reduced significantly, however we must pay for this reduction. This is because when we compute NR we must check for two fields if they are enciphered by the same key. We need to find this key from the assumed clear and ciphered fields. That is, we have to perform the transformation

$$K = C^*(M, E)$$

The transformation C^* may be very difficult to perform for some ciphers, specially for "key non-invertible" ciphers described in Chapter 6. For example, in the case of the NBS Block cipher [FED75] we have to solve 64 simultaneous non-linear equations with 64 unknowns, or alternatively search sequentially through the whole key space. We call the cost involved with the "retrievable" cipher, the "key deciphering" cost. This cost will be

$$CR = C_r * NR$$

where C_r is the cost to find one key, and NR is as defined above. In some cases, the cost C_r is so large that even though $NR \ll NP$, this reduction might not help us in practice. (*)

We can now define the total cost associated with the use of a particular statistic. This cost is called:

WFS - the work factor of a statistic

WFS is computed as follows:

$$WFS = CD + CK + CR$$

where CR is zero for the non "retrievable" cases. For most statistics only the second term is significant. For some "retrievable" ciphers, with very complex transformations on the keys, the last term will be the most significant. And for some statistics, especially if the ciphered alphabet $D(E)$ is very large, even the first term is significant. (**) It is clear then that WFS depends both on the statistic and on the type of cipher. In our experiment, in Chapter 8, we get values for K , NR , and NP . We also get some "rough" values for C_d , C_k , and C_r , and therefore get a "feeling" for WFS.

7.5.2 Discussion

WFS - the work factor of a statistic is not a complete security measure. It is a good measure for evaluating the enemy's work involved in utilizing a particular statistic. However, if we make the strong assumption then more than one statistic is known. In that case a composite measure which takes into account all known statistics is needed. The desired measure is a complex function of all the various NP 's. This can be demonstrated by the following example.

(*) Hellman & Diffie [DIF76] estimate the cost C_r for the NBS block cipher as \$10,000 per day.

(**) This may eliminate the possibility of using some statistics. For example, checking "word" statistic may involve checking the minimum of, 2^{32} and the number of different words in the file, which might be very large.

Assume that we have a monoalphabetic substitution cipher. Assume also that NP (single character statistics) is 100, and NP (digram statistics) is 1000. From this we might conclude that we have to check just 100 possible keys corresponding to NP of single character statistics. In reality, however, the digram statistics can be used to reduce the 100 even further since some of the 100 keys will create digram frequencies which contradict the given digram statistic and therefore they can be discarded. So the real NP is less than 100. In general this reduction of NP by combining several statistics is a very complex process. This causes the composite work factor to be a complex function of all the known statistics. We will not achieve here a measure for a composite work factor. This can be a subject of future research.

Even though the work factor of a statistic is not a complete measure, it can be very useful in the following cases:

- 1) It measures the strength of a cipher relative to a statistic. Therefore, if a designer finds that for some statistics the cipher is "weak", then he now knows where it should be improved.
- 2) In reality we will have the more common case of the weak assumption. With the weak assumption, only few (one, two) statistics and maybe only parts of them are known to the enemy. In that case, combining the "knowledge" from two "partial" statistics (which is the enemy's task) could be very difficult. The work factor of these statistics will then be a realistic measure for the security provided by the file cipher.

To summarize, the work factor of a statistic, even though it is not a complete security measure, is very useful in many realistic cases. Examples of computations of this measure will be given in Chapter 8.

7.6 Other Factors

The work factor of the file cipher is not the only factor which can affect the security of file enciphering. We can classify all factors which affect the security of a cryptography based file system into four broad classes:

- a) Factors which relate to the content of the file. As is apparent from the discussion above, the security provided by a cipher depends on the statistical properties of the source file. If one would "flatten"

these statistics before enciphering, then the work factor of any cipher applied to this file will increase. "Flattening" these statistics can be achieved by several methods. The first one is the method of data compression. As an example, compressing the "leading" or "trailing" blanks will remove a big "peak" from several statistics (see examples in Chapter 8). In general, the method of data compression is used to decrease the redundancy of the data in the file. It also has the advantage of saving secondary storage. A method of using data compression before enciphering was described by Stahl [STA74]. The second method is to add "spurious" information to the file in order to flatten its statistics. This may be used for a direct organization file where we are supposed to have many "empty" records. We can use these "empty" records (the enemy does not know that they are empty...) to add "garbage" which will flatten the desired statistics. The third method is to use a "combination" cipher, such as the transposition/substitution cipher described in Chapter 2, since such ciphers "scramble" the order of fields in the file, making the task of getting D(E) difficult. The fourth method is to "cut" a large file into several small files, and encipher each of them with a different key. This will reduce the amount of ciphered and clear data from which meaningful statistics can be derived. However, this method might cause some problem with controlling the different parts of the file and their corresponding keys.

To summarize, some preprocessing which is done on the file, might increase the work factor of the cipher applied to this file, significantly.

- b) Factors which relate to the cipher type. Here the consideration should be a cipher with a high work factor. Our measure, the work factor of a statistic can be very useful for this purpose. Sometimes, however, we have special requirements from a cipher. We might want a cipher with some "processing" capabilities, for example a retrievable cipher. Again, the work factor of a statistic (especially, the field statistic) can be used for choosing a retrievable cipher which provides reasonable security. Another requirement can be a cipher with "difficult to perform" C* transformation. Such requirement is very important if we use the key record scheme (Section 6.2).

To summarize, the cipher type and the requirements put on the cipher are very important factors which affect the security of file enciphering.

- c) Factors which relate to the keys. The manner in which keys are chosen will affect the security of file enciphering. For achieving high work factor keys should be chosen at random from a very large key space. "Obvious" keys such as the "user's name" should not be chosen. Also, as Shannon pointed out, keys should be used wisely to increase the work factor (by the methods of diffusion and confusion, see Section 7.2). Since the keys are the most important item in cryptography based systems, they should be guarded and controlled carefully. Some of the problems involved in controlling these keys, such as the "human engineering problem" were described in Chapter 6. A very effective method of increasing security is to change the cryptographic keys occasionally. Clearly, this cannot be done too frequently because of the need to reprocess the whole file. One possibility is to change these keys every time reorganization of the file is done.

To summarize, the ways in which the cryptographic keys are produced, distributed, controlled and changed are major factors which affect the security of file enciphering.

- d) Factors which relate to access control. As was pointed out in Chapter 6, any cryptography based system must be supplemented by some access control mechanism. If clear keys or data are compromised during file processing then the security of the cryptographic system is compromised. Therefore, the correctness and reliability of the access control mechanism is an important factor which affects the security of file enciphering.

The factors mentioned above are some of the major factors which affect the security of a cryptography based system. One would like to include all these factors in a complete security measure. For this, each of the above factors must be further analyzed and made quantitative. This can be a subject for future research.

7.7 Summary

In this chapter we discussed the problem of evaluating the security of file enciphering. This problem is important in cryptography based systems, since the environment is not ignored and methods of proving correctness are not relevant. We discussed Shannon's work and we showed the difficulties in applying it to file enciphering. We then developed a measure for the work factor of a file cipher called: the work factor of a statistic. This is not a complete security

measure but it is a useful measure in many real life cases. Computing the work factor of a statistic is not a trivial problem and we have presented several algorithms for its computation. Finally, we discussed several additional factors such as the frequency of changing keys, which affect the security of file enciphering and which should be included in the development of a complete security measure of cryptography based systems.

CHAPTER EIGHT

Experiments with File Enciphering

8.1 Introduction

In this chapter we describe an experiment for implementing cryptographic transformations in a small scale file system. Several types of ciphers mentioned in previous chapters are implemented. Other concepts such as "processability of ciphers" or "security provided by a file cipher" which were mentioned previously are discussed again in light of the experimental results.

Of the two goals of this experiment, the first is the evaluation of the cost-effectiveness of file enciphering. This evaluation is done by measuring two quantities:

- 1) The effectiveness of a cipher is measured mainly by the security it provides. For measuring this security we use WFS - the work factor of a statistic, which was defined in Chapter 7. WFS is measured (or approximated) for several types of ciphers. Thus, the (minimal) security provided by different file ciphers can be estimated.
- 2) The cost of enciphering depends mainly on the CPU time associated with the enciphering/deciphering process. This CPU time overhead is measured for all ciphers used. Thus, the cost of using different file ciphers can be computed.

Since the experiment is conducted on one specific file, it cannot be considered a "complete" evaluation of file enciphering. It serves mainly to show what can be done, or the methodology to use, in order to do such an evaluation. However, since our choice of a file can be considered as a "random choice", and since no special effort was made to program one cipher more efficiently than the others (except for the NBS block cipher), the experiment's results are expected to be representative. A comparison is also made between our results, in relation to cost, and Friedman and Hoffman's results [FRI74] as a further check on our measurements.

The second goal of this experiment is to gain insight into implementation problems of file enciphering. Such insight is important in systems oriented research such as ours. Also, not much data on experience with file enciphering has been published (publicly). Many practical considerations associated with the implementation are not apparent until some implementation is tried. Examples include considerations related to the generation of keys, the use of pseudorandom number generators, the length of fields vs. length of keys, etc. As is shown below, such considerations are very important since they affect both the security and the cost of the various ciphers. The above implementation considerations have been discovered as a result of the experiment and the experience it provided. Getting "one's hands dirty" has proven useful in our case.

Due to lack of time and resources (computer funds) not all planned experiments were actually conducted. However, enough experiments were performed and enough data was gathered to fulfill major parts of the two goals stated above, namely evaluating cost-effectiveness of file enciphering and getting insight into implementation problems of file enciphering.

8.2 General Description of the Experiment

8.2.1 The File

The main input to this experiment is a portion of an accounting file which was used by the Instruction and Research Computer Center at The Ohio State University. The file contains 1000 records, each 80 bytes long. The layout of the file is shown in Figure 33. A record is composed of fixed length fields. Sample fields are "account", "user name", "charge quantity". This is a typical file which might require protection using cryptographic transformations. It has the advantage of having fields with different lengths, so that questions relating to field length can be investigated. It also has the advantage that different fields have completely different statistical properties, therefore ciphers which are applied to different fields might have different work factors. As an example of what these statistical properties might be, let us look at part of the field statistic for the "user name" field shown in Figure 34. (Some of the "user names" are not shown for privacy reasons.) Two facts are apparent. First, the "user name" field does not contain only names. Actually, it contains anything which has appeared in the "id field" of the "job card". Secondly, several values appear many times. For example, the value "TSO" appears 113 times. This repetition of field values is very common in files and it is

[illegible]

Figure 33 - The Layout of the File

*** SINGLE FIELD STATISTICS ***

FIELD=	TSD	FREQUENCY=	113
FIELD=		FREQUENCY=	82
FIELD=	HRRC BYS NEWOMP	FREQUENCY=	53
FIELD=		FREQUENCY=	47
FIELD=	HRRC WMN REG	FREQUENCY=	40
FIELD=	HRRC WMN NEWOMP	FREQUENCY=	35
FIELD=	BYS REG	FREQUENCY=	30
FIELD=	HRRC BYS TRANS	FREQUENCY=	24
FIELD=	HRRC WMN 126K	FREQUENCY=	24
FIELD=		FREQUENCY=	24
FIELD=	HRRC WOM TRANS	FREQUENCY=	21
FIELD=	LOGSHEFT	FREQUENCY=	20
FIELD=	CARD/DISK	FREQUENCY=	20
FIELD=		FREQUENCY=	19
FIELD=		FREQUENCY=	15
FIELD=	DRUGS	FREQUENCY=	15
FIELD=	EDIT	FREQUENCY=	14
FIELD=	HRRC BYS 126K	FREQUENCY=	11
FIELD=	TRP GIRLS O&I	FREQUENCY=	11
FIELD=		FREQUENCY=	11
FIELD=	DS CARD REG	FREQUENCY=	11
FIELD=	<<<<DUMP>>>>	FREQUENCY=	10
FIELD=	BYS6R2	FREQUENCY=	10
FIELD=	WOMEN UPDATE	FREQUENCY=	10
FIELD=	PRETAB / TRANSLATE	FREQUENCY=	9
FIELD=		FREQUENCY=	9
FIELD=	ANALYZE	FREQUENCY=	9
FIELD=		FREQUENCY=	8
FIELD=		FREQUENCY=	7
FIELD=	UPDATE XREFTABLE	FREQUENCY=	7
FIELD=	HRRC WMN SPSS	FREQUENCY=	7
FIELD=	SUMMARY	FREQUENCY=	7
FIELD=		FREQUENCY=	6
FIELD=	250	FREQUENCY=	6
FIELD=	FIX	FREQUENCY=	6
FIELD=	GRLS TRP O&I	FREQUENCY=	6
FIELD=	TUTORING	FREQUENCY=	6
FIELD=	PLI TEST	FREQUENCY=	6
FIELD=	BOYS 5 O&I	FREQUENCY=	6
FIELD=		FREQUENCY=	5
FIELD=	TEST FUNK	FREQUENCY=	5
FIELD=	UPDATE MONITOR	FREQUENCY=	5
FIELD=	TAPEMAP	FREQUENCY=	5
FIELD=	HRRC GRLS REG	FREQUENCY=	5
FIELD=	DSCARD REG	FREQUENCY=	5

Figure 34 - Part of the Field-Statistic

used in retrieval applications (e.g. one can ask to retrieve all records for which "user name" = 'TSO'). The "user name" field has then "desired" statistical properties which enable us to check concepts such as "retrievable ciphers" and to measure retrieval efficiency.

The file described above is the only one used in our experiment.

8.2.2 Methodology

In Section 8.1 we pointed out that we want to measure two main quantities. The first is the security provided by different file ciphers, measured by WFS, the work factor of a statistic. WFS depends on the type of cipher used and on the statistic chosen. The second is the cost which is determined by the CPU time overhead of the enciphering process (for all the ciphers used in our experiment, the deciphering time was equal to the enciphering time). Clearly, many parameters which can be changed in the experiment can affect the above two quantities. The main parameters are the following:

- 1) We can encipher the file on the field level (i.e. a ciphered field i is the enciphered form of only the clear field i), on the record level (i.e. ciphered record j is the enciphered form of clear record j only), or on the file level.
- 2) We can use the same key for different fields (records) or different keys for different fields (records).
- 3) We can use different types of ciphers. Examples are monoalphabetic substitution, VERNAM, NBS block cipher.
- 4) We can use different file organizations. The simplest type of organization is the sequential file. However, in order to investigate "access oriented" transformations (see Chapter 5) we need other file organizations such as direct or inverted file organizations.
- 5) We can compute WFS for many different statistics. Examples are single character frequency, digram frequency, header frequency or field frequency.
- 6) We can measure the effect of enciphering on different file operations. For example COPY, RETRIEVE or SORT operations.

Because of the cost involved we cannot perform an experiment which exhausts all the possible combinations of the above parameters. We therefore limited ourselves to only . about 50 important combinations. These are described in the next section. We call one unique combination of the above parameters an experiment. We therefore have several different experiments. One experiment may be repeated several times. In that case we talk about the instances of an experiment.

In each experiment we perform the following measurements:

- 1) We measure the CPU time required to encipher the whole file. Using the CPU time we compute the Enciphering Time Coefficient (ETC - defined by Friedman and Hoffman in [FRI74]) which measures the relative cost of the different ciphers. ETC is defined as the time needed to encipher and copy a segment of data divided by the time to just copy this data.
- 2) We measure the CPU time required to answer a query. With this time we can compare the retrieval overhead of different ciphers.
- 3) We compute NP for several different statistics. From NP we compute or approximate WFS for these statistics and cipher.

Because of the multiprogramming environment in which the experiments were run, both measurements 1 and 2 give a distribution in their results. We assume that the CPU times come from a Normal distribution (the same assumption is made by Friedman and Hoffman [FRI74]). We have performed several instances of each experiment, so that we can estimate the mean and the 95% confidence interval for CPU times. This estimation is done by the use of the student-t distribution (see Lordahl [LOR67]). The number of instances which we have performed (which is equal to 1+ the number of degrees of freedom for the t distribution) was determined by the desire not to have more than 10% relative error, although in most cases the relative error was much smaller.

The third quantity which we compute, NP, is also not computed exactly as was explained in Chapter 7. Since our computed NP is a lower bound for the real NP, our computed WFS will be a lower bound to the real WFS. Furthermore, for computing NP we use the Stirling approximation (see Knuth [KNU73] p. 46) which in most cases will give lower than real values for NP. Finally, we also consider lower bounds for the cost constants Ck, Cr, and Cd (see

Section 7.5.1). We therefore get very "conservative" values for WFS.

In the next section we give a detailed description of our experiments.

8.3 Detailed Description of the Experiment

8.3.1 The Parameters

As pointed out in the last section, we limit ourselves to a few combinations of parameters affecting the experiments. Our specific parameters are as follows.

- 1) The enciphering is done on the field level only. Only one field in each record is enciphered in each experiment, although different fields are used in different experiments. All of the statistical properties measured before and after enciphering relate to this field only.
- 2) Two cases of "key use" are tried. In the first case the same key is used in each record. This causes the resultant cipher to be retrievable. In the second case we use different keys for different records. This causes the resultant cipher to be non-retrievable in general, but in our specific experiment it is almost retrievable (see explanation below). Both retrievable and almost retrievable ciphers are analyzed in this experiment.
- 3) Only sequential file organization is investigated.
- 4) Two file operations are tried: COPY and RETRIEVE.
- 5) Several types of ciphers are used. They are described in the next section.

8.3.2 The Ciphers

We now describe the different types of ciphers used in the experiment. The reader is referred to Chapter 2 for details and examples.

1. CAESAR Cipher

This is the simplest form of the monoalphabetic ciphers.

Each clear character is shifted with a constant shift which is the key for this cipher. This is clearly a retrievable cipher.

2. VERNAM Cipher

Here we use a key whose length is equal to the field length, and the same key is used in each record. This makes the cipher retrievable. The key is either read as input or is generated using another "seed" for a pseudo-random number generator.

3. NBS Block Cipher

This is the block cipher suggested by NBS [FED75] (see also Feistel [FEI73] for a similar cipher.) It is a very complicated block cipher composed of a sequence of permutations and nonlinear substitutions operating on 8 bytes (64 bits) of clear data and using 8 bytes of key. The enciphering process is composed of two parts: a) From the 8 byte key, 16 different key schedules are generated by a complex algorithm. b) Using the 16 key schedules, 16 iterations of the enciphering operations are performed. A simplified diagram of part (b) is shown in Figure 35. Part (a) has to be done only when a new key is used. The overhead associated with this cipher varies significantly depending on whether part (a) is done or not.

In this experiment the NBS block cipher is used on fields of different lengths, not necessarily multiples of 8. "Blanks" are used for completing the field length to a multiple of 8 (characters other than "blanks" can be used), and the same key is used for each record. Thus, the resultant cipher is retrievable, and part (a), the key schedules generation, is done only once.

Figure 35 - Part (b) of the NBS Cipher

4. PSEUDO-RANDOM VERNAM Cipher

This is a VERNAM cipher with different keys for different records. The keys are generated from a short "seed" using a pseudo-random bit generator. A bit generator was used instead of a random number generator mainly because of the different length fields which we need to encipher (a pseudo-random number generator always produces fixed length random numbers). Our generator is based on a shift register with linear feedback described by Krishnaiyer and Donovan [KRI73]. As is shown later, it gives very good results in terms of the "randomness" of the generated keys.

In general the PRVERNAM (pseudo-random VERNAM) cipher is not uniquely decipherable and therefore not almost retrievable. However, in our experiment this cipher is practically uniquely decipherable and therefore almost retrievable.

To show this note the following. For our 1000 record file there are at most 1000 different values for a particular field (actually there are less). What is the probability that the ciphered forms of two different values will be equal? (this will make the cipher NUD). The answer depends on the field length. The shortest field we use in our experiment is 9 bytes (72 bits) long. We therefore can have 2^{72} different keys applied to this field. However, there are $2^{71} \cdot (2^{72} - 1)$ different pairs of keys, of which only 2^{72} pairs will give the right result (for every key chosen to encipher value A there is only one key which gives the same ciphered form if used with value B). The number of ways to choose pair of values from the 1000 values is $500 \cdot 999 \sim 2^{19}$. If the keys chosen are random, then the probability of getting an NUD cipher is

$2^{19} \cdot 2^{72} / (2^{71} \cdot (2^{72} - 1)) \sim 2^{-52}$ which is very small.

Even for our pseudo-random bit generator the results are very good and in all the relevant experiments the PRVERNAM cipher was UD, i.e. almost retrievable (that is, it never happened that two different values had the same ciphered form).

The result above is important since it is used in the retrieval process of ciphers which are based on the pseudo-random bit generator.

5. FIELD HOMOPHONIC Cipher

This cipher is based on the concepts of classical homophonic ciphers (see Section 2.1 and Stahl [STA74]). However, in this case the homophones are generated for complete fields instead of for single characters. Each clear field is enciphered as a complete unit and is substituted by one or more forms of the ciphered field, called field homophones. The generation of homophones is based on the clear field distribution. Frequent fields require more homophones than non-frequent fields. The main goal is to "flatten" the "field statistic" curve, although other statistics may also be "flattened". (*)

There are many ways to generate the homophones. We chose a method based on the pseudo-random bit generator. That is, the homophones are generated rather than read as input. The number of fields with homophones, and the number of homophones each field has, affect both the security provided by the cipher and the overhead associated with it. We use two types of field homophonic ciphers in the experiment:

PHOMOPHON 1 - only 11 fields have homophones. Maximum number of homophones for one field is 5.

PHOMOPHON 2 - 17 fields have homophones. Maximum number of homophones for one field is 10.

All the fields which do not have homophones are enciphered with the same key. Therefore, each of the two fields homophonic ciphers described above has a large retrievable part. For the same reasons as for the PRVERNAM cipher the nonretrievable parts are practically UD or almost retrievable. Our field homophonic ciphers are then composed of two parts: retrievable and UD. They are therefore partially retrievable ciphers (defined in Section 7.4.4).

6. PSEUDO-RANDOM NBS Block Cipher

Here we use different keys for different records. The keys are generated by the pseudo-random bit generator. For the PRNBS cipher (pseudo-random NBS) part (a), the key schedules generation has to be done separately for each record since a "new" key is used for each record. This, as we shall

(*) As far as we know this cipher has not been published before.

see below, causes the cipher to be very expensive. Again, from the same reasoning as for the PRVERNAM cipher, the PRNBS cipher is practically almost retrievable.

To summarize, we have three retrievable ciphers (CAESAR, VERNAM, NBS) and four nonretrievable ciphers (PRVERNAM, FHOMOPHON 1, FHOMOPHON 2, PRNBS), of which two are partially retrievable (FHOMOPHON 1 and 2) and the other two are practically almost retrievable.

8.3.3 The Measurements

For the retrievable ciphers four measurements are made (for 1 and 2 refer to Figure 33):

- 1) Time to encipher the "user name" field (20 bytes) for the whole file.
- 2) Time to encipher the "charge quantity" field (9 bytes) for the whole file.

For measurements 1 and 2 the enciphering time coefficient is computed as the time to encipher the whole file divided by the time to copy the whole file without enciphering (the NULL cipher).

- 3) Time to retrieve one query: VALUE = 'TSO'; Since the file is sequential a sequential search is used. The retrieval overhead (RO) is computed as the time to retrieve from the ciphered file divided by the time to retrieve from the clear file.
- 4) NP is measured for six different statistics: single characters, digrams, trigrams, headers (of three characters), trailers (of three characters), and fields. Using NP, WFS is computed for each statistic and cipher.

Since we deal with "rough" approximations for NP and for the costs C_k , C_d , C_r , we decided, instead of giving exact values to WFS, to classify it in three broad categories: SMALL, LARGE, and VERY LARGE. SMALL means that not many resources (e.g. CPU time) are needed to utilize this particular statistic in order to "break" the cipher. LARGE means that a large amount of resources is needed to utilize this statistic. VERY LARGE means that utilizing this statistic by the enemy is impractical. If $WFS > 10^{**7}$ seconds we call it VERY LARGE, and if $WFS < 10^{**3}$ seconds we call it SMALL. Now, if we assume that the smallest cost (of C_k , C_d , C_r) is 10^{**6} seconds then an $NP > 10^{**13}$ will result with WFS VERY LARGE. We therefore

call an NP which is larger than $lg^{**}13$ also VERY LARGE. We note again, for future reference, that NP is a lower bound which is exact for the retrievable case.

For the non-retrievable ciphers measurements 1, 3 and 4 were done.

8.3.4 The Programs

The experiment involved a fair amount of programming. Most of the programs, including all but one cipher procedure are written in PL/I. The two programs which are written in ASSEMBLER are:

NBSCPR - The program to implement part (b) of the NBS block cipher.

VRAND - the program to implement the pseudo-random bit generator which is used by many ciphers.

The other major programs are:

FILEMNG - the program which processes the file, enciphers it, and executes the retrieval queries. It also measures the CPU time overhead.

STAT - the program which computes the statistical properties of the clear and ciphered files and prints the histograms. It uses a SORT routine to sort the various frequencies.

WFS - the program which computes NP for the various statistics and ciphers.

COMP - the program which computes means, 95% confidence intervals, and relative errors for all the measurements.

8.4 The Results

The results of all experiments are reported in Tables 1 through 7. The following abbreviations are used:

ETC - enciphering time coefficient
 RO - retrieval overhead
 UD - uniquely decipherable
 NUD - nonuniquely decipherable
 R - retrievable
 PR - partially retrievable
 VL, VLARGE - very large

TABLE 1 - CLEAR

	Cost Measurement - msec						Time for whole file-msec	Time per byte-msec	ETC
20 byte field	83.2	83.2	93.2	89.8	96.5	89.8	89.3 ±5.3	0.0045 ±0.0003	1
9 byte field	83.2	83.2	93.2	89.8	96.5	89.8	89.3 ±5.3	-	1
									EO
Time for retrieval	46.6	53.2	53.2	43.7	49.9	43.7	48.3 ±4.8	-	1

TABLE 2 - CAESAR

	A. Cost						Time for whole file-msec	Time per byte-msec	ETC
	Measurements - msec								
20 byte field	1234.7	-1234.7	-1234.7	1238.0	1317.9	1311.2	1262.1 +42.1	0.063 ±0.002	14.1 ±0.9
9 byte field	675.6	685.6	678.9	652.3	658.9	665.6	667.5 +13.3	0.074 ±0.001	7.5 ±0.5
Time for retrieval	46.6	46.6	56.6	46.6	49.9	56.6	50.4 ±5.1	-	1.04 ±0.14

B. Security

	Cipher type	NR	NH	NP	Dominant cost-sec	WFS
Single charac- ters	R	1	-	1	$C_k \sim 1$	SMALL
Digrams	R	1	-	1	$C_k \sim 1$	SMALL
Trigrams	R	1	-	1	$C_k \sim 1$	SMALL
Headers	R	1	-	1	$C_k \sim 1$	SMALL
Trailers	R	1	-	1	$C_k \sim 1$	SMALL
Field	R	1	-	1	$C_k \sim 1$	SMALL

TABLE 3 - VERNAM

167

	A. Cost						Time for whole	Time per	ETC
	Measurements - msec						file-msec	byte-msec	
20 byte field	359.4	362.8	362.8	369.4	382.7	369.4	367.8 ±8.7	0.018 ±0.001	4.1 ±0.3
9 byte field	329.5	326.1	352.8	349.4	329.5	342.8	338.3 ±12.0	0.037 ±0.001	3.8 ±0.3
Time for retrieval	39.9	53.2	53.2	53.2	56.6	43.3	49.9 ±7.0	-	RO 1.03 ±0.18

B. Security

	Cipher type	NR	NH	NP	Dominant cost-sec	WFS
Single characters	NUD	-	-	VL	$C_k \sim 0.5$	VLARGE
Digrams	NUD	-	-	VL	C_k	VLARGE
Trigrams	NUD	-	-	VL	C_k	VLARGE
Headers	R	1	-	1	C_k	SMALL
Trailers	R	1	-	1	C_k	SMALL
Field	R	1	-	1	C_k	SMALL

TABLE 4 - NBS

A. Cost

	Measurements - msec						Time for whole file-msec	Time per byte-msec	ETC
20 byte field	20141	19765	19731	19761	20347	20167	19985 ±278	0.999 ±0.014	223.8 ±13.6
9 byte field	13175	13229	14380	14470	14037	14297	13935 ±613	1.548 ±0.068	156.0 ±11.5
Time for retrieval	46.6	49.9	56.6	46.6	49.7	53.2	50.5 ±4.0	-	1.04 ±0.13

B. Security

	Cipher type	NR	NH	NP	Dominant cost-sec	WFS
Single characters	NUD	-	-	VL	$C_k \sim 20$	VLARGE
Digrams	NUD	-	-	VL	C_k	VLARGE
Trigrams	NUD	-	-	VL	C_k	VLARGE
Headers	NUD	-	-	VL	C_k	VLARGE
Trailers	NUD	-	-	846	C_k	SMALL
Field	R	1	-	1	C_R ~LARGE	LARGE

TABLE 5 - PRVERNAM

169

	A. Cost						Time for whole	Time per	ETC
	Measurements - msec						file-msec	byte-msec	
20 byte field	1228.0	1234.7	1264.6	1164.8	1171.5	1181.4	1207.5 ±42.6	0.060 ±0.002	13.5 ±0.9
9 byte field	-	-	-	-	-	-	-	-	-
Time for retrieval	1098.2	1128.2	1154.8	1124.8	1144.8	1128.2	1129.8 ±20.4	-	23.4 ±2.4

B. Security

	Cipher type	NR	NH	NP	Dominant cost-sec	WFS
Single characters	NUD	-	-	1.53×10^{10}	$C_k \sim 1$	LARGE
Digrams	NUD	-	-	VL	C_d or C_k	VLARGE
Trigrams	NUD	-	-	VL	C_d or C_k	VLARGE
Headers	NUD	-	-	VL	C_k	VLARGE
Trailers	NUD	-	-	VL	C_k	VLARGE
Field	UD	-	-	VL	C_k	VLARGE

TABLE 6 - FHOMOPHON 1

170

A. Cost							Time for whole file-msec	Time per byte-sec	ETC
	Measurements - msec								
20 byte field	529.1	485.9	545.8	542.5	525.8	502.5	521.9 ± 24.5	0.026 ± 0.001	5.8 ± 0.4
9 byte field	-	-	-	-	-	-	-	-	-
Time for retrieval	93.2	89.9	93.2	103.2	83.2	73.2	89.3 ± 10.7	-	RO 1.85 ± 0.29

B. Security

	Cipher type	NR	NH	NP	Dominant cost-sec	WFS
Single characters	NUD	-	-	0.01?	$C_k \sim 0.5$	SMALL
Digrams	NUD	-	-	VL	C_k	VLARGE
Trigrams	NUD	-	-	VL	C_k	VLARGE
Headers	UD	0	VL	VL	C_k	VLARGE
Trailers	UD	0	0.08	0.08?	C_k	SMALL
Field	PR	1	576	577	C_k	SMALL

TABLE 7 - PHOMOPHON 2

171

	A. Cost						Time for whole file-msec	Time per byte-msec	ETC
	Measurements - msec								
20 byte field	632.3	595.7	599.0	559.1	582.4	552.4	586.8 ± 30.6	0.029 ± 0.002	6.6 ± 0.5
9 byte field	-	-	-	-	-	-	-	-	-
Time for retrieval	106.5	123.1	113.2	119.8	116.5	113.2		115.4 ± 6.1	2.37 ± 0.27

B. Security

	Cipher type	NR	NH	NP	Dominant cost-sec	WFS
Single characters	NUD	-	-	VL	$C_k \sim 0.6$	VLARGE
Digrams	NUD	-	-	VL	C_k	VLARGE
Trigrams	NUD	-	-	VL	C_k	VLARGE
Headers	UD	0	VL	VL	C_k	VLARGE
Trailers	UD	0	0.35	0.35?	C_k	SMALL
Field	PR	6	696708348	696708354	C_k	LARGE

Recall also from Chapter 7 that for UD ciphers

$$NP = NR + NH,$$

and that,

$$WFS = CK + CD + CR = C_K * NP + C_d * K * \log_2 K + C_r * NR.$$

CLEAR

The results for the clear file are shown in Table 1. There is no difference between the time to process a 20 byte field and the time to process a 9 byte field, since in both cases a full record is copied from one file to another. The differences between these fields appear only when we try to encipher one of them.

The statistical properties of the clear file are shown in Figures A.1 - A.6 of Appendix A.

CAESAR

The results for the CAESAR cipher are shown in Table 2. Our implementation of the CAESAR cipher is not very efficient, so its cost is higher than expected. It is clearly a retrievable cipher, therefore its $RO \sim 1$.

The file after enciphering is shown in Figure A.7 (only the "user name" field is enciphered). Seemingly, we have a "good" cipher, however, from Table 2b it is evident that the security provided by this cipher is very low. It might be surprising that we get so many "blanks" as part of the ciphered field. The reason is that these are not really "blanks" but they are actually unprintable characters. These unprintable characters can be distinguished by the statistical analysis routines. Therefore the security provided by the unprintable characters is illusory.

As can be expected, NP for all statistics is 1 and therefore WFS for all statistics is SMALL. Figures A.8 and A.9 show two statistics and their corresponding NP for the CAESAR cipher, the character statistic and the field statistic.

VERNAM

The results for the VERNAM cipher are shown in Table 3. As is seen from the table this cipher is very efficient. It is also a retrievable cipher and therefore $RO \sim 1$.

The file after enciphering is shown in Figure A.10. Three statistics for this cipher are shown in Figures A.11, A.12, A.13 and A.14. The statistic ciphers for Header, Trailer and Field statistics are retrievable, therefore their WFS is SMALL. One surprising result is seen in Figure A.11. The single character statistic for the VERNAM cipher is NUD. Using the bound for the NUD case we get a very low value, 0.01 (actually the real result is even smaller, since NP is rounded up to the second decimal point). As was pointed out in Chapter 7, our bound for NP in the NUD case is very conservative for some distributions, and this is such a case. As an alternative approach we tried to compute the bound using algorithm AUD for the UD case. This is justified because UD is a special case of NUD, and therefore a bound for UD is also a bound for NUD. The result, shown in Figure A.12, is that NP is actually VERY LARGE, and therefore the corresponding WFS is VERY LARGE. This phenomenon, where we get a very low value with the bound for NUD, but a better bound with algorithm AUD occurs repeatedly. We therefore use algorithm AUD whenever we get a low bound for NUD.

During this experiment we discovered an interesting phenomenon for the VERNAM cipher. In one instance we tried to use an alphabetic key as a VERNAM key. The results are shown in Figure A.23. Surprisingly the key appears as part of the file! The reason is the large number of "blanks" (H'40') which exist in the file and if are EXORED with a capital letter give a small letter as a result (e.g. 'A' + H'40' = 'a'). We therefore recommend avoiding alphabetic keys for the VERNAM cipher, generating them instead from another "seed" using a pseudo-random number (bit) generator.

NBS Block

The results for the NBS block cipher are shown in Table 4. Clearly this cipher is much more expensive than the last two ciphers. It is also a retrievable cipher and therefore RO ~ 1.

The file after enciphering is shown in Figure A.15. Two statistics are shown in Figures A.16 and A.17. Notice that WFS for the header statistic which was SMALL for the VERNAM cipher is now LARGE. This is an advantage of the NBS block cipher. NP for the trailer statistic remains small because of the many cases in which we have an 8 byte "blanks" trailer. NP for the field statistic is 1 because this cipher is retrievable. However, the cost of getting

the key from the pair (cipher, clear), i.e. C_r , is very high (since, transformation C^* for the NBS block cipher is difficult to perform (*)). This means that the cost C_r is large and therefore we estimate WFS for the field statistic as LARGE.

PRVERNAM

The results for the PRVERNAM cipher are shown in Table 5. Since this cipher is not retrievable its RO is much larger than 1.

The file after enciphering is shown in Figure A.18. Two of its statistics, the character statistic and the field statistic, are shown in Figure A.19 and A.20. Again, for the character statistic we get an NP which is too low, but using algorithm AUD we get the value 1528756224 which is LARGE. (**)

As seen in Table 5, all other WFS's are VLARGE. Here, we encountered a new problem. We were unable initially to generate the distributions of the ciphered digrams and trigrams because of insufficient core storage (in our programs all tables are kept in core). We had to use up to 630k bytes in order to get these distributions. This is a typical case where the cost to get a particular statistic, CD, is high. This cost can therefore be significant in the computation of the work factor, defined in Chapter 7.

Referring again to Table 5 we see that the PRVERNAM cipher is very secure since five of the WFS values are VERY LARGE and one is LARGE.

PHOMOPHON 1 and 2

The results for the two field homophonic ciphers are shown in Tables 6 and 7. It is evident that even though these ciphers are not retrievable, their retrieval efficiency

(*) Diffie and Hellman estimate this cost as \$10,000 per day [DIF76].

(**) It is also surprising that we get here, for the PVERNAM cipher, a smaller NP than for the simple VERNAM cipher. The explanation is that algorithm AUD, too, behaves "strangely" for some distributions and produces for them very "conservative" values. The longer tail of 1's in the VERNAM cipher distribution causes AUD to give a larger NP.

is quite good.

The field statistics for the two ciphers are shown in Figures A.21 and A.22. Notice that the values for NP in case of the field statistic were obtained by the use of algorithm AH (Section 7.4.4), since the ciphers are partially retrievable. Here, too, we have the problem that for both ciphers, for two statistics (character, trailer) our bounds were too low. Using algorithm AUD helped only in case of single character statistic for the FHOMOPHON 2 cipher, but not in the other three cases. One important observation is that by a slight increase of the number of homophones, WFS for the field statistic increased from SMALL for FHOMOPHON 1 to LARGE for FHOMOPHON 2. We see then the great importance of a sufficiently large number of homophones.

PRNBS

No tables are included. The main reason is that enciphering of the file with this cipher takes more than 5 minutes of CPU time (more than \$45 per run). The reason for this large CPU time overhead is the need to generate new key schedules for every record. This cipher is thus impractical, although we expect its security properties are at least as good as those of the PRVERNAM cipher.

The last result is important. It means that the NBS block cipher with a different key for different records is impractical in a software implementation. In this case a high performance hardware implementation is essential.

8.5 Analysis of the Results

8.5.1 Comparison With Published Results

In order to check our results we compare them with some published results. We could not find any published results on the security of file ciphers. Therefore we concentrate on their cost. Friedman and Hoffman [FRI74] measured the overhead associated with some ciphers and defined enciphering time coefficient (the measurements of interest are resulted from their FORTRAN routines; our routines were coded in PL/I which is "comparable"). Our experiment is slightly different since our "null transformation" includes all CPU processing needed to copy a file, while their "null transformation" is just "moving" from one location in memory to another. We consider our experiment more realistic in the case of files. Even with this difference our results are quite similar to Friedman and Hoffman's results.

For the VERNAM cipher, their result for ETC is between 2.68 ("one word" key) to 4.03 ("long" key). In our case, ETC is between 3.8 for a 9 byte key to 4.1 for a 20 byte key.

For the pseudo-random VERNAM cipher, their ETC is 9.96 while we got the value 13.5. The difference can be attributed to the different pseudo-random number generator used, and to the way the keys are generated.

For checking the results for the NBS block cipher we compare them to the results of Bright and Enison [BRI76]. They only tested an 8 byte key, and their result, 10 msec for an 8 byte key on a "comparable" computer, is very close to ours (8 msec for 8 bytes - Table 5).

The above comparisons are a good indication that our results for the cost of various file ciphers are reasonable.

8.5.2 Comparison Between Ciphers

We now compare the results for the different ciphers from several viewpoints.

Figure 36 shows the relative cost of the different ciphers on a logarithmic scale (for the 20 byte field). The order of these ciphers in increasing cost is: NULL, VERNAM, PHOMOPHON 1, PHOMOPHON 2, PRVERNAM, CAESAR, NBS. The last cipher is at least 20 times more expensive than the first five.

The relatively small cost of the first five ciphers (at most 1 second per 1000 records file) disproves the common objection to cryptography, i.e. that using cryptography is too expensive. As can be seen from our experiment the overhead associated with file enciphering is not excessive. We therefore believe that our hope that cryptographic transformations is a feasible protection mechanism has been fulfilled.

It is interesting to note that for different ciphers the increase in cost when going from a 9 byte field to a 20 byte field is different. The % increase in time to encipher is shown in Figure 37. The smallest increase occurs for the VERNAM cipher, the largest for the CAESAR cipher. This could be expected since any character oriented cipher such as the CAESAR depends very much on field length. As for the NBS block cipher, the best results are when the field length is a multiple of 8. It seems that for "long" fields the VERNAM cipher is the most efficient.

Figure 38 shows the retrieval overhead of the different ciphers. We see that the PRVERNAM cipher is far less efficient in retrieval than all the other ciphers. The field homophonic ciphers are quite efficient in retrieval even though they are not retrievable. Figure 38 can be used to justify the use of retrievable ciphers. They do not increase the retrieval time over the time to retrieve from a clear file, and some of them, e.g. the NBS block cipher provide good security even with the constraint of "retrievability". Clearly, if the constraint of "retrievability" is removed, as is the case with the PRVERNAM cipher, then we can get very secure ciphers.

It is quite difficult to compare the ciphers from the security point of view. The reason is that we don't know which statistics are more important (or are known to the enemy). Let us take an arbitrary scale where all statistics have the same weight. Let us also assign the following weights: SMALL - 0, LARGE - 1, VLARGE - 2. Let us define the "security" provided by a cipher as the sum of the weights of its WFS's. Figure 39 shows the security provided by the different ciphers.

One should not take Figure 39 too literally because of the special scale that we chose to express "security". However, some trends can be seen. For example, it appears that the CAESAR cipher provides very little security, the NBS block cipher and the field homophonic type 2 cipher provide quite good security, and the pseudo-random VERNAM cipher provides very good security.

Figure 36 - Relative Cost of Ciphers

Figure 37 - Comparison Between the Two Fields

Figure 38 - Retrieval Overhead

Figure 39 - The Security of File Ciphers

Using Figures 36-39, the designer can now choose a cipher according to his own tradeoffs and criteria. If we had to recommend a cipher based on our results, we would choose the field homophonic 2 cipher. It provides good security, and it is quite efficient both in the cost of enciphering and in the cost of retrieval.

8.6 Summary

In this chapter we described an experiment measuring the cost and security of different ciphers applied to a typical file. Our security measure, WFS, enabled us to compare the security of different ciphers. The experiment, however, shows that the bounds for NP, developed in Chapter 7, are sometimes too "conservative". Developing improved bounds is a subject for future research. The cost of enciphering and the cost of retrieval were measured and shown to be reasonable in most cases. We also found it feasible to use "retrievable" ciphers and still provide a reasonable degree of security. We therefore achieved our goal of evaluating the cost-effectiveness of several file ciphers. The experience we got from this experiment is valuable, and we hope that this report of it will help others to implement file enciphering in the future.

CHAPTER NINE

Summary and Suggestions for Future Research

"All's well that ends well..."

9.1 Summary and Main Contributions

The research reported in this dissertation has identified and addressed the problems of applying cryptography to data base security. It covers three important areas: data security, data base systems and cryptography and makes contributions in each of these areas.

In the introduction we reviewed the subjects of cryptography and data base security. We showed that although cryptography was recognized in the past as an effective protection tool, its use for protection of data in computer systems, except for protecting messages in communication lines, has been limited. This gave us the main motivation for looking into the problems of using cryptographic transformations as a protection mechanism in data base systems. After reviewing the subject of data base security and several of the known data base security models, we concluded that a new model was needed to permit convenient discussion and analysis of problems related to the application of cryptography to data bases. This motivated the development of our multi-level model of a data base in Chapters 4 and 5.

We view the multi-level model of a data base as one of the two main contributions of this dissertation. This is believed to be the first model which shows the relationship between the data base structure and the cryptographic transformations applied to the data base. The identification of cryptographic transformations as transformations between the physical levels of a data base is essential for the understanding of this relationship. We also introduced the classification of cryptographic transformations into two classes: user controlled transformations and system controlled transformations, and we discussed the advantages and disadvantages of each. The analysis of user controlled transformations led to the conclusion that access control mechanisms complement the mechanism of cryptographic transformations and

vice versa. That is, the cooperation of both access control and cryptographic transformations is needed to assure the security of a data base system.

The multi-level model of a data base is important from two other points of view. From the data security point of view, it shows explicitly the ways to achieve decentralization of protection in data base systems by allowing the "spreading" of protection specifications and mechanisms through all levels of the data base. From the point of view of data base systems architecture, it is believed to be the first model to recognize and utilize the existence of several physical levels of a data base. As discussed later, both of these areas can be studied further using this model.

After presenting the conceptual/structural model of a data base in Chapters 4 and 5, we directed our attention to the question of system design. In Chapter 6 we described the design of a secure file system based on user controlled cryptographic transformations. We showed several cryptographic schemes to implement different protection policies. We presented the designer with several schemes and with criteria to choose between them. We believe that these schemes are a practical contribution in the field of designing secure systems using cryptographic protection.

Evaluating the security provided by a system is a complex problem. In Chapter 7 we developed a measure for evaluating the security provided by file enciphering. We believe this is the first attempt to estimate the work factor associated with different file ciphers, and it differs from the classical, information theory oriented approach of Shannon. The development of the measure involves solving difficult combinatoric problems, and though our "solutions" to these problems are not "perfect," we were able to use them for (approximately) evaluating the security of several file ciphers in Chapter 8. We view our measure, the work factor of a statistic, as a practically useful measure and as a step toward developing a more complete and accurate security measure. We view this measure as the second main contribution of this dissertation.

The cost-effectiveness of cryptographic transformations has been the subject of debate for some time. In Chapter 8, by conducting a small scale experiment in file enciphering, we showed the economic feasibility of using cryptographic transformations. Using our security measure and by measuring the CPU time overhead, we compared both the security and cost of various ciphers. Several contributions are made by this experiment. First we presented a methodology to evaluate cost-effectiveness of file enciphering. Second, we showed the feasibility of using an important class of

ciphers, the "retrievable" ciphers. Third, we showed the advantages of our own suggested cipher, the field homophonic cipher. Fourth, we reported some of the practical considerations involved in file enciphering, which may be helpful to others who use cryptography in the future.

As the research reported in this dissertation is in a new and developing area, it is natural that it opens many questions for future research. Some are discussed in the next section.

9.2 Points for Future Research

Our research combines several areas, namely: data security, data base systems and cryptography, and raises questions in all of these areas. In the following paragraphs we enumerate several questions needing further research, in the order of their importance in our view (others may use a different ordering).

The first and most urgent problem for future research is the problem of developing more complete security measures for computer systems. We attacked part of the problem by developing a measure for the security of file enciphering. It needs further development and refinement in several aspects. Our "bounds" for the combinatoric problems need improvement. A work factor which includes multiple statistics needs to be developed. Lastly, methods need to be discovered to deal with the dependence of the work factor both on the interactions between different statistics and on the "knowledge" of the enemy.

One should not stop with just a measure for the security provided by cryptography. As was pointed out in Chapter 1, cryptographic transformations are just one of the possible protection mechanisms. At this stage, much research has been done on other protection mechanisms such as access control. Measures for the security provided by different access control mechanisms should be developed. The current approach of "proving" the correctness of an access control mechanism by ignoring the external environment does not seem sufficient. Finally a composite measure which takes into account the security provided by different interacting protection mechanisms should be developed.

Another area for future research is the investigation of ways to combine access control mechanisms with cryptographic transformations. Some suggestions were made in Section 5.3.3, and we also mentioned the work by Muffic [MUF76] in this area. The crucial area in which cryptography needs the help of access control is in the operating system. The interaction between the operating system and the data base system, and the interaction between the pro-

tection mechanisms provided by both, is a key issue in investigating this problem of cooperation between access control and cryptography. Future research in this area should center around the operating system rather than the data base system as stressed in this dissertation

The multi-level model of a data base can be used as a research tool in two areas. First, there is the area of security engineering mentioned in Chapter 4. For example, what protection specifications should be used in each level? What protection mechanisms are suited to each level? Both cost and security measures are needed in order to answer such questions. These questions are related to problems of performance evaluation in data base systems (most protection mechanisms cause some performance degradation). Tools used in the area of performance evaluation can be helpful here.

The second area of research which stems from the multi-level model is the area of data base systems architecture. Multiple physical levels, which are central to our model, affect issues such as: multi-level relational data bases, distributed data bases in computer networks, data translation and data migration. Since these topics deal with data residing on several physical media, they can benefit from the multiple physical levels concept, and can be modelled using extensions of our model.

The above are just a few of the questions opened by this research. We hope they will stimulate others to do research in this interesting and important area.

BIBLIOGRAPHY

- [AST72] Astrahan, M.M., Altman, E.B., Fehder, P.L., Senko, M.E., "Concepts of a Data Independent Accessing Model", SIGFIDET Workshop, 1972.
- [BAR64] Baran, P., "On Distributed Communications - Security, Secrecy and Tamper-free Considerations", Memorandum, Rand Corp., RM-3765-PR, 1964.
- [BAR74] Bartek, D.J., "Encryption for Data Security", Honeywell Computer Journal, 1974.
- [BAU75] Baum, R.I., "The Architectual Design of a Secure Data Base Management System", Ph.D Dissertation, The Ohio State University, 1975.
- [BAY75] Bayer, B., Metzger, J.K., "On the Encipherment of Search Trees and Random Access Files", ACM Transactions on Data Base Systems, Vol. 1, No. 1, March, 1976.
- [BER72] Bergart, J.G., Denicoff, M., Hsiao, D.K., "An Annotated and Cross-Referenced Bibliography on Computer Security and Access Control in Computer Systems", Technical Report, The Ohio State University, OSU-CISRC-TR72-12, 1972.
- [BER72b] Berman, G., Freyer, K.D., Introduction to Combinatorics, Academic Press, 1972.
- [BRI76] Bright, B.S., Enison, R.L., "Cryptography Using Modular Software Elements", NCC Proceedings, 1976.
- [CAR70] Carrol, J.M., Mclelland, P.M., "Fast 'Infinite Key' Privacy Transformations for Resource Sharing Systems", AFIPS Conference Proceedings, FJCC, 1970.
- [COD70] Codd, E.F., "A Relational Model of Data for Large Shared Data Banks", CACM, Vol. 13, No. 6, June, 1970.
- [COD71] CODASYL Data Base Task Group Report, April, 1971.

- [CON72] Conway, R. W., Maxwell, N. L., Morgan, H. L., "On the Implementation of Security Measures in Information Systems", CACM, Vol. 15, No. 4, April, 1972.
- [COS72] Cosserat, D. C., "A Capability Oriented Multi-Processor System for Real Time Applications", ICCC, 1972.
- [DIF76] Diffie, W., Hellman, M. E., "A Critique of the Proposed Data Encryption Standard", CACM, March, 1976.
- [DIJ68] Dijkstra, E. W., "The Structure of THE Multiprogramming System", CACM, Vol. 11, No. 5, May, 1968.
- [ELS73] Elson, M., Concepts of Programming Languages. Science Research Associates, 1973.
- [EVA74] Evans, A., Kantrowitz, W., Weiss, E., "A User Authentication Scheme Not Requiring Secrecy in the Computer", CACM, Vol. 17, No. 8, August, 1974.
- [FAB74] Fabry, R. S., "Capability Based Addressing", CACM, Vol. 17, No. 7, July, 1974.
- [FED75] Federal Register, March 17, 1975.
- [FEI73] Feistel, H., "Cryptography and Computer Security", Scientific American, Vol. 228, No. 5, May 1973.
- [FEI75] Feistel, H., Notz, W. A., Smith, L. J., "Some Cryptographic Techniques for Machine to Machine Data Communications", Proceedings of the IEEE, November, 1975.
- [FER75] Fernandez, E. B., Summers, R. C., Coleman, C. D., "An Authorization Model for Shared Data Bases", ACM SIGMOD Proceedings, 1975.
- [FRE73] Freeman, P., Software Systems Principles, Science Research Associates, 1973.
- [FRI70] Friedman, T. D., "The Authorization Problem in Shared Files", IBM System Journal, Vol. 7, No. 4, 1970.
- [FRI74] Friedman, T. D., Hoffman, L. J., "Execution Time Requirements for Encipherment Programs", CACM, Vol. 17, No. 8, August, 1974.
- [GAI56] Gaines, H. F., Cryptanalysis, Dover, 1956.
- [GRA68] Graham, R. M., "Protection in an Information Processing Utility", CACM, Vol. 15, No. 5, May, 1968.

- [GRA72] Graham, S. G., Denning, P. J., "Protection Principles and Practice", AFIPS Conference Proceedings, SJCC, 1972.
- [GUD75] Gudes, E., Stahl, F. A., "An Annotated List of Patents Dealing with Cryptography", Unpublished Notes, 1975.
- [GUD76] Gudes, E., Stahl, F. A., Koch, H. S., "Applications of Cryptographic Transformations to Data Base Security", NCC Proceedings, 1976.
- [HAN74] Hanlen, J., "UK White Paper Due this Month, May Disappoint Privacy Backers", Computer World, December 11, 1974.
- [HAR75] Hartson, H. R., "Languages for Specifying Protection Requirements in Data Base Systems - A Semantic Model", Ph.D Dissertation, The Ohio State University, 1975.
- [HOF69] Hoffman, L. J., "Computers and Privacy: A Survey", Computing Surveys, Volume 1, No. 2, 1969.
- [HOF71] Hoffman, L. J., "The Formulary Model for Flexible Privacy and Access Controls", AFIPS Conference Proceedings, FJCC, 1971.
- [HOF73] Hoffman, L. J., Security and Privacy in Computer Systems, Melville Publishing Co., 1973.
- [HOF75] Hoffman, L. J., Private Communication, 1975.
- [HSI70] Hsiao, D. K., Harary, F. "A Formal System for Information Retrieval from Files", CACM, Vol. 13, No. 2, February, 1970.
- [HSI73] Hsiao, D. K., "Logical Access Control Mechanisms in Computer Systems", OSU-Report, No. 4, 1973.
- [IBM72] IBM Systems Reference Library, OS Data Management Services Guide, Order No. GC26-3746, 1972.
- [JON73] Jones, A. K., "Protection in Programmed Systems", Ph.D Dissertation, Carnegie -Mellon University, 1973.
- [KAH67] Kahn, D., The Codebreakers, Macmillan, 1967.
- [KNU73] Knuth, D. E., The Art of Computer Programming, Volume 1, Addison-Wesley, 1973.
- [KNU75] Knuth, D. E., The Art of Computer Programming, Volume 3, Addison-Wesley, 1975.

- [KRI73] Krishnaiyer, R., Donovan, J. C., "Shift Register Generation of Pseudorandom Binary Sequences", Computer Design, April, 1973.
- [LAM69] Lampson, B. W., "Dynamic Protection Structures", AFIPS Conference Proceedings, FJCC, 1969.
- [LOR67] Lordahl, D. S., Modern Statistics for Behavioral Sciences, The Ronald Press Co., 1967.
- [MAR73] Martin, J., Security, Accuracy and Privacy in Computer Systems, Prentice-Hall, 1973.
- [MAT74] Matyas, S. M., "A Computer Oriented Cryptanalytic Solution for Multiple Substitution Enciphering Systems", Ph.D Dissertation, University of Iowa, 1974.
- [McC75] McCauley, E. J., "A Model for Data Secure Systems", Ph.D Dissertation, The Ohio State University, 1975.
- [McC75b] McCauley, E. J., "File Partitioning and Record Placement in Attribute Based File Organizations", Proceedings of the third USA-JAPAN Computer Conference, 1975.
- [McL73] McLaughlin, R. A., "Equity Funding: Everyone is Pointing at the Computer", Datamation, June, 1973.
- [MIN74] Minsky, N., "On Interaction with Data Bases", ACM-SIGFIDET Workshop on Data Description, Access and Control, May, 1974.
- [MUF76] Muftic, S., "The Design of a Secure Computer System", Ph.D Dissertation, The Ohio State University, 1976.
- [NBS75] National Bureau of Standards, "Computer Security Guidelines for Implementing the Privacy Act of 1974", FIPS PUB 41, May, 1975.
- [NEE73] Nee, C. J., Hsiao, D. K., Kerr, D. S., "Context Protection and Consistent Control in Data Base Systems", OSU-Report, No. 9, 1973.
- [NIV65] Niven, I., Mathematics of Choice, New Mathematics Library, 1965.
- [ORG72] Organic, E. I., The MULTICS System, The MIT Press, 1972.
- [PAR73] Parker, D. B., Nycum, S., Oura, S. S., "Computer Abuse", Stanford Research Institute, 1973.

- [PET67] Petersen, H.E., Turn, R., "Systems Implications of Information Privacy", AFIPS Conference Proceedings, SJCC, 1967.
- [POP74a] Popek, G.J., Kline, C.S., "Verifiable Secure Operating System Software", NCC Proceedings, 1974.
- [POP74b] Popek, G.J., "Protection Structures", Computer, June, 1974.
- [PUR74] Purdy, G.B., "A High Security Log-In Procedure", CACM, Vol. 17, No. 8, August, 1974.
- [REI71] Reiter, A., Clute, A., Tenenbaum, J., "Representation and Execution of Searches over Large Tree Structured Data Bases", IFIPS Congress Proceedings, 1971.
- [REI72] Reiter, A., "The HODS Data Storage Management System", Technion Technical Report, No. 2, 1972.
- [RIO58] Riordan, J., An Introduction to Combinatorial Analysis, John Wiley, 1958.
- [SAL74] Saltzer, J., "Protection and the Control of Information in MULTICS", CACM, Vol. 17, No. 7, July, 1974.
- [SCH72] Schroder, M.D., Saltzer, J.H., "A Hardware Architecture for Implementing Protection Rings", CACM, Vol. 15, No. 3, March, 1972.
- [SCH74] Scheuermann, P., Heller, J., "A View of Logical Data Organization and its Mapping to Physical Storage", Third Texas Conference on Computing Systems, November, 1974.
- [SCH75] Schmid, H.A., Bernstein, P.A., "A Multi-Level Architecture for Relational Data Base Systems", Proceedings of the International Conference on Very Large Data Bases, 1975.
- [SHA49] Shannon, C.E., "Communication Theory of Secrecy Systems", The BELL System Technical Journal, Vol. 28, No. 4, 1949.
- [SHA70] SHARE-GUIDE, Data Base Management System Requirements, November, 1970.
- [SIB73] Sibley, E. H., Taylor, R. W., "A Data Definition and Mapping Language", CACM, Vol. 16, No. 12, December, 1973.

- [SIB74] Sibley, E. H., "On the Equivalence of Data Base Systems", SIGFIDET Workshop on Data Description, Access and Control, May, 1974.
- [SKA69] Skatrud, R. D., "A Consideration of the Application of Cryptographic Techniques to Data Processing", AFIPS Conference Proceedings, FJCC, 1969.
- [SMI72] Smith, J. L., Notz, W. A., Osseck, P. R., "An Experimental Application of Cryptography to a Remotely Accessed Data System", IBM Report, 1972.
- [STA73] Stahl, F. A., "A Homophonic Cipher for Computational Cryptography", NCC Proceedings, 1973.
- [STA74] Stahl, F. A., "On Computational Security", Ph.D Dissertation, The University of Illinois, 1974.
- [STO74] Stonebraker, M., Wong, E., "Access Control in a Relational Data Base Management System by Query Modification", BERKELEY ERL Report, No. M438, May, 1974.
- [SWE73] "Sweden's Data Act", Computer Decisions, November, 1973.
- [TSI74] Tsichritzis, D., "A Note on Protection in Data Base Systems", IRIA International Workshop on Protection in Operating Systems, August, 1974.
- [TUR73] Turn, R., "Privacy Transformations for Databank Systems", NCC Proceedings, 1973.
- [TUK70] Tuckerman, B., "A Study of the Vignere-Vernam Single and Multiple Loop Enciphering Systems", Report RC - 2879, IBM Research Laboratory, 1970.
- [UNI73] UNIVAC 1100 Series, Data Management System Schema Definition, (EP-7907), 1972.
- [US74] U.S. Code, Title 5, Section 552a (Privacy Act of 1974).
- [VAN69] Van Tassel, D. L., "Cryptographic Techniques for Computers", AFIPS Conference Proceedings, SJCC, 1969.
- [VAN72] Van Tassel, D. L., Computer Security Management, Prentice-Hall, 1972.

- [WAL73] Walter, K. G., Orden, W. F., Rounds, W. C., Bradshaw, F. T., Ames, S. R., Shumway, D. G., "Primitive Models for Computer Security", Case Western Reserve University, NTIS AD-778 467, 1973.
- [WEI74] Weiss, H., "Computer Security, An Overview", Data-mation, January, 1974.
- [WES72] Westin, A. F., Baker, M. A., Databanks in Free Society, Quadrangle Books, New York, 1972.
- [WIL75] Wilson, R., Private communication, Mathematics Department, The Ohio State University, 1975.
- [WON71] Wong, E., Chiang, T. C., "Canonical Structures in Attribute Based File Organization", CACM, Vol. 14, No. 9, September, 1971.
- [WUL74] Wulf, W., Cohen, E., Corwin, W., Jones, A., Levin, R., Pierson, C., Pollack, F., "HYDRA, The Kernel of a Multiprocessor Operating System", CACM, Vol. 17, No. 6, June, 1974.
- [YAM74] Yamaguchi, K., Merten, A. G., "Methodology for Transferring Programs and Data", SIGFIDET Workshop on Data Description Access and Control, May, 1974.

Appendix A

Sample Results of the Experiment

Figure A.1 -CLEAR-Character Statistic

Figure A.2 - CLEAR - Digram Statistic

Figure A.3 - CLEAR-Trigram Statistic.

Figure A.4 - CLEAR-Header Statistic.

Figure A.6 - CLEAR-Field Statistic

Figure A.8 - CAESAR-Character Statistic.

Figure A.9 - CAESAR-Field Statistic

HISTOGRAM OF THE 70 MOST FREQUENT CHARACTERS

** = 16,838 OCCURRENCES

Figure A.11 - VERNAM-Character Statistic-NUD

HISTOGRAM OF THE 70 MOST FREQUENT CHARACTERS

*** = 16,838 OCCURENCES

Figure A.12 - VERNAM-Character Statistic-UD

Figure A.13 - VERNAM-Header Statistic

Figure A.14 - VERNAM-Field Statistic

HISTOGRAM OF THE 70 MOST FREQUENT CHARACTERS

** = 1.42% OCCURRENCES

Figure A.19 - PVERNAM-Character Statistic

Figure A.20 - PRVERNAM-Field Statistic

Figure A.21 - PHONOPHON-1-Field Statistic