207

# PRIVACY AND SECURITY OF AN INTELLIGENT OFFICE FORM

by

KUM-YU ENID LEE

B.A., University of Texas at Austin, 1972

---

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

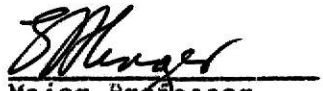
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

Approved by:



Major Professor

CONTENTS

1. Chapter 1: Introduction..... 2
1.1 Form-Based Office Automation Systems..... 3
1.2 Abstract Data Types..... 5
1.3 Form Definition/Manipulation Languages..... 6
1.4 Office Activities..... 7
1.5 Intelligent Data Object..... 8
1.6 Security and Privacy ..... 11
1.6.1 Security - Access Rights..... 13
1.6.2 Security - Abstract Data Type..... 14
1.6.3 Security - Authentication and Identification..... 15
1.6.4 Security - Authorization Rules..... 15
1.6.5 Security - Integrity Constraints..... 16
1.6.6 Security - Data Encryption..... 16
1.7 Statement of Problems..... 17
1.8 Overview..... 18

2. Chapter 2: Definition of an IOF..... 19

3. Chapter 3: Security System of an IOF - Requirements..... 25
3.1 Requirements - User Profile..... 25
3.2 Requirements - Time Stamping..... 26
3.3 Requirements - Data Encryption..... 29

4. Chapter 4: Security System of an IOF - Design..... 31
4.1 Design - User Profile..... 31
4.2 Design - Time Stamp..... 33
4.3 Design - Data Encryption..... 37

5. Chapter 5: Implementation of the Security System of an IOF..... 39
5.1 Introduction..... 39
5.2 Implementation of User Profile..... 41
5.2.1 Introduction..... 41
5.2.2 USER.parms..... 41
5.2.3 USER.disk..... 41
5.2.4 USER.inits..... 42
5.3 Implementation of Time Stamp..... 44
5.3.1 Introduction..... 44
5.3.2 Stamp.edit..... 44
5.3.3 Stamp.proc..... 46
5.4 Implementation of Data Encryption..... 48
5.4.1 Introduction..... 48
5.4.2 Encrypt..... 48
5.4.3 Decrypt..... 48

**THIS BOOK CONTAINS NUMEROUS PAGES WITH DIAGRAMS THAT ARE CROOKED COMPARED TO THE REST OF THE INFORMATION ON THE PAGE.**

**THIS IS AS RECEIVED FROM CUSTOMER.**

5.5 Summary..... 49

6. Chapter 6: Conclusion and Extensions..... 52
6.1 Conclusion..... 52
6.2 Extensions..... 54

7. Bibliography..... 56

8. Appendix 1 - Source Code Listings..... 65

9. Appendix 2 - Source Code Structure Chart..... 90

LIST OF FIGURES


Figure 1. The Employment Verification Procedure in IDO [mcb83]..... 22

Figure 2. The modified ICN for the Employment Verification Procedures in IOF..... 24

Figure 3. Files Executed by User Profile of a Node..... 32

Figure 4. Executing transactions with time stamps..... 35

Figure 5. Data Encryption..... 38

Figure 6. The Directory Layout of an IOF Security System..... 40

Figure 7. Files/Command Executed by User Profile of a Node..... 43

Figure 8. UNIX System Command/File Called by Stamp.edit/Stamp.proc..... 45

Figure 9. Structure Layout of a Stamp.ctl Record Used in Stamp.edit..... 46

Figure 10. Structure Layout of a Stamp.ctl Record Used by Stamp.proc..... 47

Figure 11. UNIX System Command Executed by Encrypt and Decrypt..... 49

MISSING PAGE

# 1. Chapter 1: Introduction

Currently, a great deal of interest is directed towards automating of the office environment [fikes81]. [Zis78] defines office automation as "... the application of computer technology, communication technology, system science and behavioral science to the vast majority of less structured office functions ... ". The functions of an office involve text editing, filing, organization, copying, transforming, analyzing, storing and transmission of text [ellis80, lad80]. These functions can be automated to reduce clerical work load and thereby reducing total expenditure.

Office automation leads to widespread use of electronic equipments such as word processing systems, communication equipments, data processing systems, etc. For the most part, these systems are not integrated and thereby leading to inefficiency. The challenge of an office information system is to integrate the above components in order to reduce the complexity of the users' interface to the system, control the flow of information and enchance the overall efficiency of the office [ellis80]. Electronic forms, to be discussed later, enable this kind of integration. As a result of integration, different groups

of users are accessing information from the same office information system. To protect the system against unauthorized alteration or destruction, a security system must be established.

The design of a central "security manager" to protect an IOF against unauthorized alteration or destruction is presented. Three different security measures are implemented. The first is a user profile to specify the data objects that the user is authorized to access and the operations the user is authorized to perform on those objects. The second is a time stamp ordering technique to specify when operations are allowed to be performed on the the data stored in IOF. Examples of allowable operations are read, write, delete and insert. The technique also sets the priority of IOF procedures. The third is an encryption procedure to transform IOF into non readable format. Thus, it helps to protect IOF during transmission in the communication network.

## 1.1 Form-Based Office Automation Systems

The traditional paper form is a printed document with spaces for insertion of required information. An electronic form is similar to a paper form. It is the

computer analog of a paper form. Forms have been used as the basis for the design of office automation systems ( e.g., OFFICETALK [ellis80], ODYSSEY [fikes81], OFS [tsic80]). Form-based office automation systems have the following advantages over other systems [geha82]:

(1) Forms permit logically related data to be treated as an entity.

(2) Electronic forms are similar to paper forms as they retain many properties of paper forms.

(3) Forms can be traced to determine the processing state in an office procedure.

(4) Access rights can be automatically enforced for a form.

(5) Automatically routing information can be designed to associate with forms.

(6) Forms allow easier transition from a manual office environment based on paper forms to the automated environment based on electronic forms.

(7) Paper copies of electronic forms can be generated automatically upon request.

With the introduction of electronic forms, an office automation system can be established whereby forms can be communicated via messages with minimal manual intervention.

## 1.2 Abstract Data Types

[Smith77a] and [smith77b] defined two kinds of data abstraction. The first is the aggregation in which a relationship among objects is regarded as a high-level object, with lower-level details suppressed. The second is a generalization in which a set of similar objects is regarded as a high-level object, with lower-level details suppressed.

The main goal of data abstraction is to allow users to focus on the relevant information and ignore the irrelevant details. Through the use of abstract data types, the user can achieve modularity, increased maintainability and higher program quality[lis74].

An abstract data type defines a set of abstract objects and specifies the operations available for use with those objects [lis74]. Knowledge of where the data are stored, which system environments are being used, where the security constraints are located, etc., should be hidden

from the user. Using this concept, the user can be more assured that security constraints will not be tampered with. Using the concept of an abstract data type, one can define office forms.

## 1.3 Form Definition/Manipulation Languages

Several form definition/manipulation languages exist in research and in practical applications. Some of them are:

(1) OFS (Office Form System) [tsic80]

(2) QBE/OBE (Query-By-Example/Office-Procedure By Example) [zlo81]

(3) ODIN (On Line Data Integrity System) [ferr82, dipi83]

[Tsic82, ferr82, dipi83] discussed the following form characteristics:

(1) fields - blank spaces found in forms for inserting requested information.

(2) operations - e.g., modify, delete, insert and route.

(3) abstractions - forms are used as a tool for data abstractions.

(4) access rights - control of the user's authority to insert, update, delete, create or add forms.

Among these characteristics, control of access rights and restriction of types of operations for specific forms are important security measures. Only personnel with the proper access rights should be able to perform certain operations on forms. Access rights must be determined based upon the needs of the office, the data being accessed and the operation being performed.

[Tsic82] suggests that workstations, not users, should control access rights. For example, an entry station can only fill out forms and a query station can only ask queries about forms. A query station can not be used as an entry station, thus providing security. On the contrary, [geha82] advocates that users, not workstations, should control access rights. Restriction of access rights by workstations seems to be somewhat inflexible.

## 1.4 Office Activities

[Tsic81] described three types of office activities in OFS (Office Form System):

(1) desk activity - for specifying actions which are triggered by specific conditions at the node;

(2) coordination activity - for providing form types and conditions to initiate specific procedures;

(3) mail activity - for routing the mail automatically.

Currently, electronic mail contains primarily texts or data structures. Items such as procedures and graphics may also be included. [Hogg84] defines an "intelligent message" which can be routed to multiple users and can send responses back to the original sender. It is viewed as an interactive program which permits a dialogue with the recipient.

## 1.5 Intelligent Data Object

McBride and Unger [mcb83] developed a model of office systems based on an intelligent form. This form is used to control the flow of information in a distributed processing environment.

Their paper [mcb83] on an "intelligent data object" (IDO) serves as an induction to this report.

The IDO may be defined as an elaboration of an abstract data type, which consists of "local data" (text), and procedures (routing commands, processing, error handling and history log). An IDO can be used to represent an electronic form.

The "local data" of an IDO corresponds to the fields in a form [mcb83]. Each field has a data type. For example, age is usually a field of the type integer and the last name is a field of the type character. One or more of the fields (key) in the "local data" must serve to uniquely identify each IDO instance among all IDO instances of the same type. Different IDO types may be differentiated according to their contents and operations, e.g., accounting, hospital admissions, college student report. Each IDO type is defined by its stored "local data". An IDO instance, which may be defined as an occurrence of the type, includes information specific to the form attributes. An IDO templates is the mapping of an IDO instance to a particular communication medium. Thus, an IDO may be defined as an elaboration of an abstract data type.

This object can be routed from node/terminal to node/terminal in a network. Each node has its own data base, its own procedures, its own sets of commands and its

own node manager [gantt85].

There are several procedures which provided for routing/navigation, processing commands, error handling and updating a history log. The routing procedures enable an intelligent data object (IDO) to navigate itself within the distributed data base [gantt85].

Using processing commands, IDO has the ability to perform computations and various functions based on those computation. Processing procedures in IDO also include those with the capability to do data base retrievals [gantt85].

An error handling procedure for an IDO may be a part of IDO itself or at the node or a combination of both. The node manager invokes the appropriate error handling procedure based upon the results of IDO processing operations [gantt85].

Each node manager maintains a history log to record the processing that an IDO instance has undergone and where it has been. This history log is used for tracing the activities of an IDO when needed. It is routed as part of the IDO from node to node [gantt85].

[Sewc85] proposed a form definition language (FDL) for use with the IDO to allow a user to specify data fields and required actions. The FDL also performs intra-form integrity checks.

To protect the IDO against unauthorized alteration, disclosure or destruction, a security system must be established. [Sewc85] designed the IDO FDL which specified access right/security constraints for the fields on the form. [Gantt85] designed a node manager of IDO's which " must execute the IDO's in a restricted environment to provide security".

## 1.6 Security and Privacy

Security of information in computer systems involves legal, administrative, physical and technological controls [camp73]. The followings are some examples [date84, ull82, park76, park79, park81, park83, camp73]:

(1) Legal, social and ethical aspects. For example, if a person requests information from a system, does he have the legal right to such information?

(2) Administrative controls. Administrative controls

determine what data gets collected and to whom it is disseminated.

(3) Physical control. For example, should the terminal/computer room be locked?

(4) Policy question. For example, how does a company determine the accessibility of the data?

(5) Operational problems. For example, how does the password information be protected?

(6) Hardware controls. For example, does any security feature exist in the hardware components?

(7) Operating system security. For example, does the underlying operating system provide any security measures concerning file contents?

(8) Threat monitoring. For example, does the system provide any "audit log" to monitor attempts to obtain sensitive information or attempts to write into protective files.

(9) Access rights. Access rights refer to the control of user's authority to insert, update, delete or create forms.

(10) Issues that are the specific concern of the data base system itself. These issues may include the authorization of users to access records in data base.

To deal with the last security problem in an IDO, several possible security issues such as access rights, abstract data type, authentication, authorization rules, integrity constraints (to protect IDO from users/programs at the node ) and data encryption are focused upon in this report.

### 1.6.1. Security - Access Rights

Access rights refer to the identification of who is authorized to insert, delete, update or add IDO types. Only personnel with the proper access rights should be able to perform certain operations on IDO types. Access rights must be determined based upon the needs of the office, the data being accessed and the operation being performed. Access rights in an IDO can be:

(1) Association of access rights to the fields on the form. The rights of a user to operate on the fields

may be restricted. For example, only accounting personnel may access fields reserved for accounting purposes.

(2) Association of access rights to IDO instances. The rights of a user to operate on IDO instances may be restricted. For example, only authorized users are allowed to delete an IDO instance.

[Sewc85] designed an IDO FDL which specified access rights for the fields on the form.

### 1.6.2 Security - Abstract Data Type

An abstract data type defines a class of abstract objects and specifies the operations available for use with those objects [lis74]. Using this concept, one can define office forms. Knowledge of where the security constraints are located, which system environments are being used, etc., are hidden from the user. Using this concept, the user can be more assured that security constraints will not be tampered with.

### 1.6.3 Security - Authentication and Identification

The process of authentication and identification involves setting up a password file for users and directing the identified users to the appropriate user profile. The user profile is a record which specifies the data objects that the users are authorized to access and the operations that the users are authorized to perform [date84, camp73, conway72].

### 1.6.4 Security - Authorization Rules

The authorization rules can be expressed in high level languages such as SEQUEL (Structured English Query Language) [Astr75, Astr75a, Cham74], SQL (Structure Query Language) [Denny77]. For example:

SQL GRANT statement -

```
GRANT SELECT on R to Tom
```

This SQL statement specifies that user Tom is authorized to perform SELECT operation R. Authorization rules are compiled and stored in the system. Some systems have authorization rules at the level of relation, others are at the level of individual fields [date84].

### 1.6.5 Security - Integrity Constraints

An integrity constraint is a security measure with the following important aspects:

(1) to prevent two or more users from changing the same data item concurrently;

(2) to specify which operations are allowed to be performed on a data item;

(3) to specify the order in which operations are to be performed.

The most popular ways of preserving the integrity of data items are locking and time stamping [date84, lynch83, ul182].

### 1.6.6 Security - Data Encryption

This security measure stores and transmits all data, messages, passwords, etc., in an encrypted form. This format which is unintelligible to any one not knowing the encryption key, is stored in the data base or transmitted from node/terminal to node/terminal [date84, feis73, gudes76, diff76, rivest78]. This method is useful for

protecting IDO during transmission in the communication network.

## 1.7 Statement of Problems

An IOF allows several different groups of users to process its local data concurrently. It also permits concurrency to exist among its internal procedures.

Of all the security problems stated in section 1.6, three security issues are implemented in this project:

(1) Authentication and identification problem. User profile is set up to identify users, to specify the data objects that the user is authorized to access and the operations the user is authorized to perform on those objects. Thus, user profile provides a way of executing the IOF's in a restricted environment.

(2) Integrity constraint problem. Time stamp ordering technique is used to specify when operations are allowed to be performed on the data stored in an IOF. This technique also sets the priority of IOF operations.

(3) Wiretapping problem. An encryption procedure is set up to transform IOF into non readable format. Thus,

it helps to protect IOF during transmission in the communication network.

## 1.8 Overview

The following chapters will give the definition of an IOF, describe the design and outline the implementation of three IOF security measures as stated in section 1.7. Chapter 2 will give the definition of an IOF. Chapter 3 will describe the overall requirements of three IOF security measures. Chapter 4 will discuss the design. Chapter 5 will discuss the implementation. Finally, chapter 6 will provide a summary and extensions of security system.

# 2. Chapter 2: Definition of an IOF

As a result of integration in office automation, different groups of users are accessing information from the same office information system. The IOF allows different groups of users to access its local data and it permits concurrent internal procedures.

A Petri Net is viewed as a directed graph consisted of two node types. A node type drawn as a circle is called a place and a node type drawn as a bar is a transition. Places with edges directed into transitions are input places and those with edges directed out of transitions are output places. The state of a Petri Net is determined by the presence or absence of markers called tokens over its places. The transition is said to be active if all input places for that transition have tokens [pete77, mcb83]. Petri Net is a good modeling tool for office procedures as it has the ability to show decisions and concurrency.

Zisman [zis77] described an Augmented Petri Net (APN) to combine process and knowledge representations into one model. In this model, transition firing is not instantaneous. It depends on the presence of tokens at

input places and the production rules associated with the transition. It is used for modeling offices as asynchronous concurrent processes.

McBride and Unger[mcb83] proposed the use of a control Petri Net for modeling an office environment. The token, in this model is viewed as an intelligent form. In order for an intelligent form to be efficiently processed in an office, McBride and Unger in [mcb83] proposed to use the FORK transition to partition the form into subforms and the JOIN transition to merge the completed subforms back into a completed copy of the original form.

The McBride and Unger [mcb83] model is the basis of the IDO concept as described in section 1.5. The IOF has the following properties:

(1) An IOF allows different groups of users to process its local data concurrently.

(2) An IOF allows concurrency to exist among its internal operations.

An expanded version of a control graph based on a Petri net was developed by Cook [cook80], called an ICN

(Information Control Net). Figure 1 illustrates how a modified version of the ICN can be used to describe the processing of a job for the employment verification procedure in an IDO.

ICN uses squares to denote input and output files [el180]. In figure 1, the personnel file is the input to the transition bar. It is included in the diagram to show the dependence of transitions on the availability of data in the personnel file. T1 is the path between the FORK and the JOIN transitions

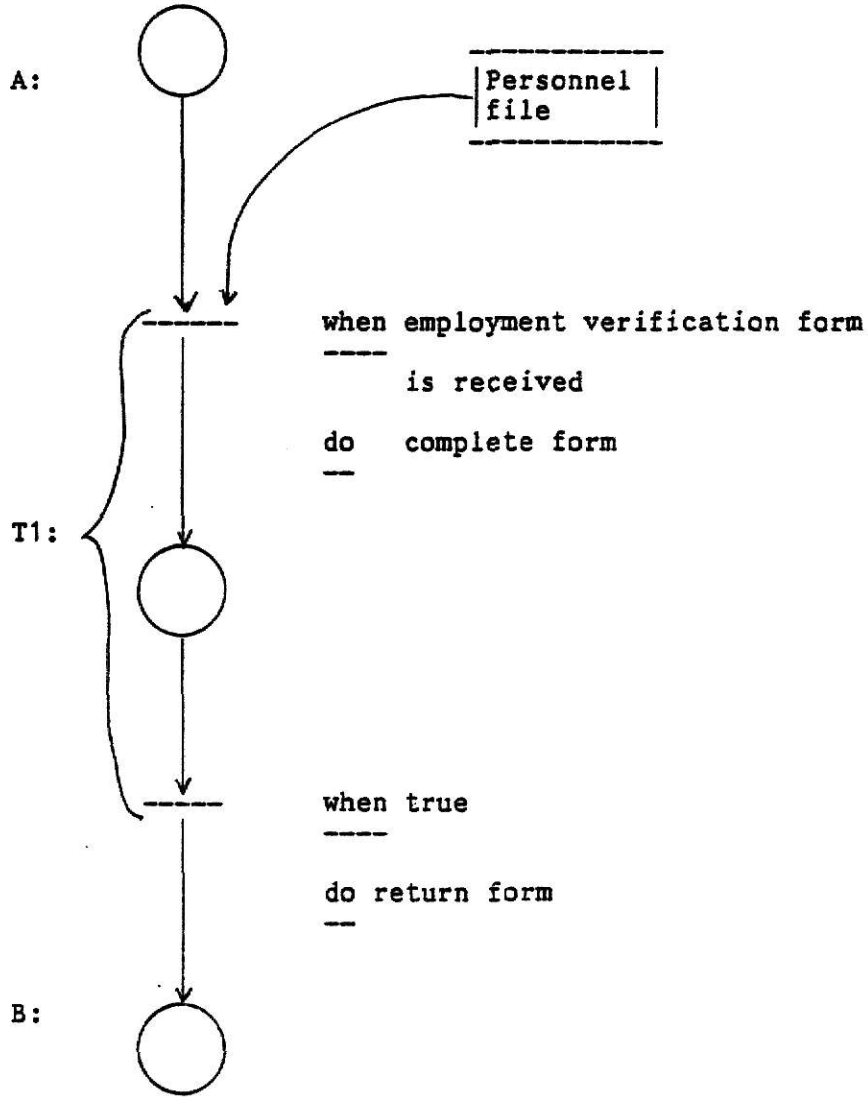



Figure 1. The Employment Verification Procedure in IDO [mcb83]

Figure 2 shows how a modified ICN directs a job for the employment verification procedures in an IOF. The personnel file is the input to the transition bar. It is used to show the dependence of transitions on the availability of data in the personnel file. T1 is the replicated paths between the FORK and the JOIN transitions. It illustrates several verification procedures of the same type to proceed simultaneously on the same token or copies of the same token.

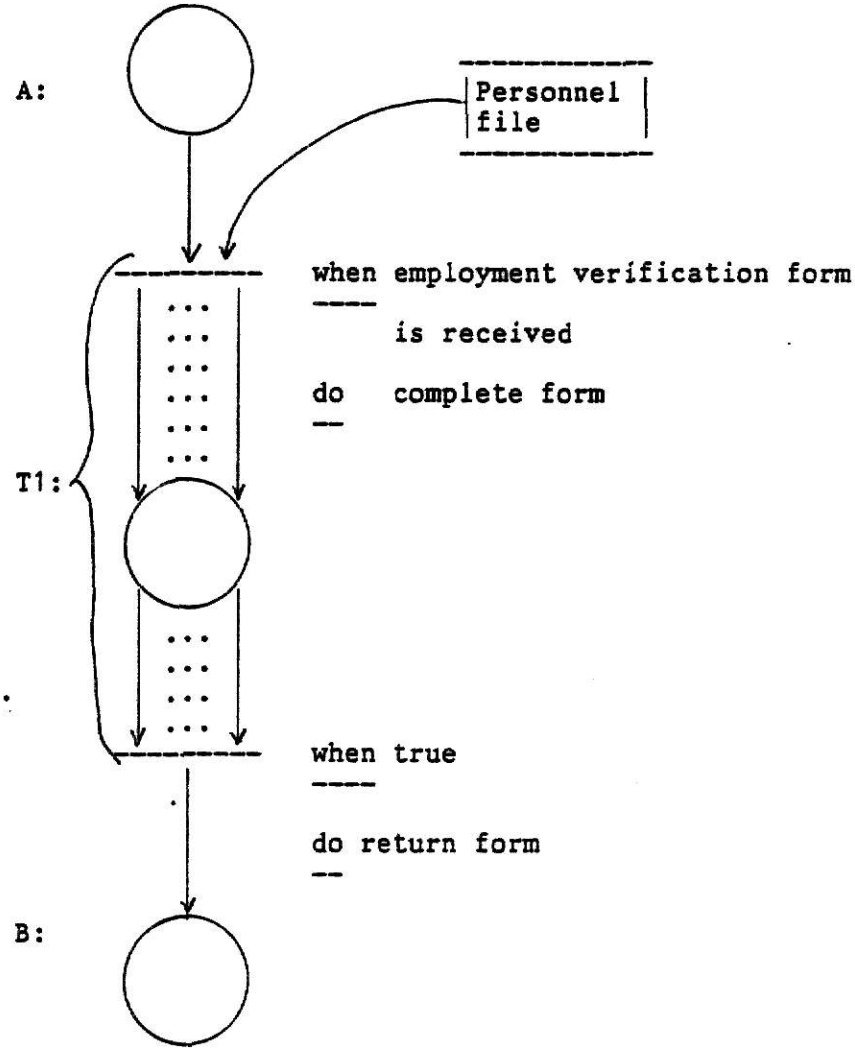



Figure 2. The modified ICN for the Employment Verification Procedures in IOF

In order to protect an IOF against unauthorized alteration, a security system must be established. In the next chapter, several security issues related to the IOF will be analyzed and the overall requirements will be defined.

# 3. Chapter 3: Security System of an IOF - Requirements

## 3.1 Requirements - User Profile

The user profile is a record which specifies the data objects that the users are authorized to access and the operations that the users are authorized to perform [date84, camp73, conway72]. The process of authentication and identification involves setting up a password file for users and directing the identified users to the appropriate user profile.

[Daley65] viewed the data base as a file hierarchy with data files as terminal nodes. Access control to these data files is handled by a control file which specifies the permitted users of each data file and each user's permitted "modes" (such as write or read only).

Before accessing the data base, the users have to identify themselves. The process of identification may involve the use of specific login information such as the name of the system or an operation number. Voice prints or finger prints have been used as an identification method in some specific terminals [date84].

The process of authentication may involve the use of

passwords or answering specific questions from the system [date84, hoff69]. Such questions may ask for personal information unknown to anybody but the authorized user. Alternatively the user may be asked to supply a secret function to a pseudo-random number generated by the system. The dial-up and call-back method can be used in authentication. In this method, the operator calls the user back to make sure the latter is authorized [peter67, hoff69].

Therefore, the requirements of user profile are:

(1) To identify users;

(2) To direct the identified users to the appropriate system environment.

## 3.2 Requirements - Time Stamping

Time stamping is an integrity constraint technique using a unique identifier equivalent to the start-time of a data item. This technique must specify which operations (e.g., read, write, insert) are allowed to be performed. It must also specify the priority of operations. Time stamping has the following aspects [date84]:

(1) every transaction is assigned a globally unique time stamp.

(2) updates are not physically applied to the data base until (successful) end-of-transaction.

(3) every object in the data base carries the time stamp of transaction that last read it and the transaction that last updated it (at least conceptually, although several optimizations are possible).

(4) if a transaction T1 requests a data base operation that conflicts with some other data base operation already executed on behalf of a younger transaction T2, that transaction T1 is restarted. An operation from T1 is in conflict with T2 if and only if

   (a) it is a read, and the object in question has already been updated by T2 or

   (b) it is updated and the object in question has already been read or updated by T2. This case can occur only during T1's commit processing, by virtue of (2) above. By "object" here we mean a physically stored object (that is , one of possibly many replicas of a logical object).

(5) If a transaction is restarted it is assigned a new time stamp."

A transaction in this report is referred as an operation or a sequence of operations performed on data objects. The above aspects of time stamping can be implemented in an IOF.

[Bern80] specified the following approaches to optimize the synchronization in the time stamp technique:

Let tw = write time on record x

tr = read time on record x

(1) Thomas Write Rule [bern80]: If write(x) has smaller time stamp than tw, the Thomas Write Rule suggests that write(x) be ignored.

(2) Multi-version time stamp ordering [bern80]:

(a) for a data item x, a set of tr which records the time stamps of all read operations on x is maintained.

(b) a set of (tw, value) pairs called versions which record time stamps of all write operations on x.

If an operation reads data item x with a time stamp ts, then this operation can be processed by reading the version(x) with the largest time stamp less than ts. This technique can achieve synchronization without ever rejecting any read operation on x.

(3) Conservative time stamp ordering [bern80]: This technique eliminates restarts by using the FIFO (first-in-first-out) queuing technique. When a scheduler receives an operation (OP) that might cause a future restart, it delays the OP until it is certain that it will not cause any restarts.

Therefore the requirements of time stamping are:

(1) to prevent two or more users from changing the same data item concurrently;

(2) to specify which operations are allowed to be performed on a data item;

(3) to specify which operations are authorized to be performed first.

## 3.3 Requirements - Data Encryption

Data encryption stores and transmits all data, messages, passwords, etc., in an encrypted form. The plaintext which is the original data is encrypted by subjecting it to an encryption algorithm whose input are the plaintext and an encryption key. The output from this encryption algorithm, the encrypted form of the plaintext, is called ciphertext. The ciphertext which is unintelligible to any one not knowing the encryption key, is stored in the data base or transmitted from node/terminal to node/terminal [date84, feis73, gudes76, diff76, rivest78]. Diffie [diff76] designs the ideas of public-key encryption. Specific schemes have been proposed by Rivest, Shamir and Adleman [rivest78] to design public-key encryption and decryption.

Therefore, the requirements of data encryption are:

(1) To transform readable IOF into a non readable format before transmission;

(2) To transform non readable IOF into a readable format.

The requirements defined in this chapter will ensure the protection of an IOF against unauthorized alteration. The following chapter will outline the design of the security system based on the above requirements.

# 4. Chapter 4: Security System of an IOF - Design

## 4.1 Design - User Profile

A user profile is a record which provides a restricted environment whereupon the user can access and perform operations on data objects. Figure 3 shows the files executed by the user profile of a node. The user profile consists of:

(1) unique identifier for each node;

(2) location of each node;

(3) a set of commands that the user can use;

(4) the restricted environment for users.

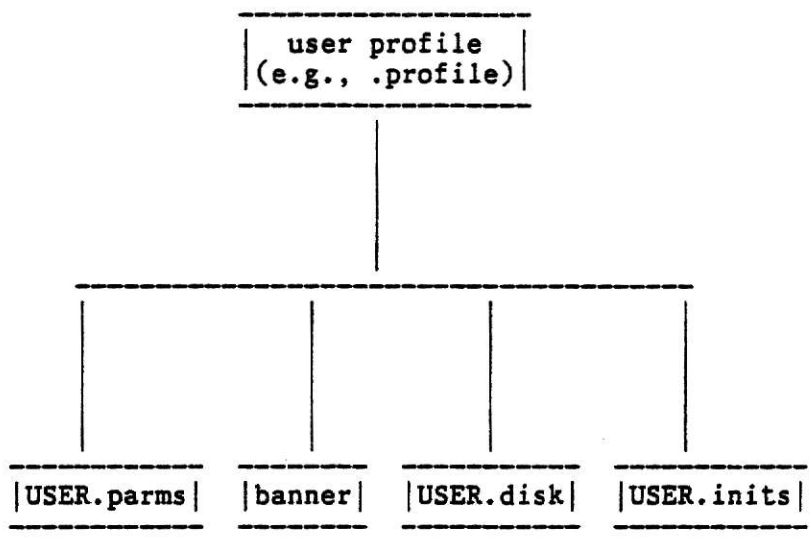



Figure 3. Files Executed by User Profile of a Node

When a user logs into a node, the node's user profile is executed. The "user profile" finds the node's dependent variables in the file USER.parms which exists in the node's login directory. Next, the user profile executes the system file 'USER.disk'. This file sets the variable P to the device that the system resides on (e.g., p=/a3). Then, the user profile executes USER.inits to set up the restricted environment for the user.

A process of identification is necessary for a user to log

into an appropriate environment. This involves supplying information known only to the authorized user. This is done by answering questions from the UNIX shell scripts. The answers to the questions are verified against the information in the user profile.

## 4.2 Design - Time Stamp

The user has the ability to destroy or modify an IOF. The IOF is loaded into the local node either from the result of having been transmitted to that node or from the result of being created/modified by the user at that node [gantt85]. A time stamp ordering technique is used to prevent unauthorized alteration of an IOF by the user at that node. The technique also sets the priority of IOF procedures. Thus, the purpose of time stamp is to prevent unauthorized alteration of an IOF at an inappropriate time. It is chosen over the locking mechanism because it has no locks, therefore deadlock is impossible and there is no overheads as required for locking and deadlock detections [date84].

The time stamp must be:

(1) globally unique;

(2) consists of site ID, IOF (form) name and clock value.

Conservative time stamps can eliminate the possibility of conflicting operations, and hence eliminate the need to restart transactions. This can lead to less work wasted and less communication overhead [date84]. However, it can lead to less concurrency. Therefore, the following Ullman's algorithm [ull82] is implemented in this project:

> "Suppose we have a transaction with time stamp t that attempts to perform an operation X on an item with read time tr and write time tw.
>
> (a) Perform the operation if X=READ and t >= tw or if X=WRITE, t >= tr, and t >= tw. In the former case, set the read time to t if t > tr, and in the latter case, set the write time to t if t > tw.
>
> (b) Do nothing if X=WRITE and tr <= t <= tw.
>
> (c) Abort the transaction if X=READ and T< tw or X=WRITE and t < tr."

| T1 | data item D | result |
|---|---|---|
| 200 | tr = 0<br>tw = 0 | |
| write D | tr = 0<br>tw = 200 | D will be allowed to be updated |

(A)

| T2 | data item D | result |
|---|---|---|
| 200 | tr = 220 | |
| write D | | abort |

(B)

| T3 | data item D | result |
|---|---|---|
| 200 | tr = 170<br>tw = 250 | |
| write D | | T3 can not update D |

(C)

Figure 4. Executing transactions with time stamps

Figure 4 illustrates three transactions (T1, T2, T3) operating on a data item D under different and unrelated situations. However, all three transactions start at the exact same time. That is, they have an identical time stamp (t) equals 200. In (A), the write operation is allowed since t >= tw and t >= tr. In (B), when T2 is updating a

copy of D, another user has already read the original copy of D and changed its time stamp (tr) to 220. Thus, T2 can not put its updated version of D back to the storage area. In (C), when T3 is changing a copy of D, the original copy of D has been updated by another user. Thus, T3 will not be allowed to copy its updated version back to the storage area.

This approach is quite efficient when it is used in the following situations [ull82]:

(1) In an environment where the probability of two transactions conflicting is small.

(2) When each user will access only a tiny fraction of a large data base.

The above algorithm is chosen because it allows more concurrency than the conservative time stamp method. The latter will not perform a data base operation until it can be guaranteed that it cannot possibly cause a future conflict. Thus, Ullman's algorithm [ull82] is implemented together with his approach for restarting an aborted transaction. He suggested that a random amount of time that an aborted transaction must wait before restarting. His explanation is that " few transactions conflict makes the

probability of having to restart a given transaction k times shrink like $c^k$, where c is some constant much less than one" [ul182].

UNIX shells are used to implement the time stamp procedure. Any user who wants to perform any operation on any data object has to call this time stamp procedure first. The procedure checks the access time, modification time or completion time of the requested file against its corresponding time in the central time stamp table. The user is then notified whether he/she is allowed to process that particular file.

## 4.3 Design - Data Encryption

The purpose of data encryption is to prevent unauthorized tapping into the IOF system. The UNIX command "crypt" is used in the data encryption procedure. UNIX shell scripts are set up to use "crypt" along with an encryption key. A shell script is used to transform an IOF into a ciphertext for storage and transmission. Another shell script using the same UNIX command and the identical encryption key are used to convert the ciphertext back to plaintext. Figure 5 shows the input and output of data encryption.


Figure 5. Data Encryption

The design presented in this chapter provides the foundation for the security system of an IOF. In the next chapter, the implementation of this IOF security system will be described.

# 5. Chapter 5: Implementation of the Security System of an IOF

## 5.1 Introduction

Using the requirements presented in chapter 3 and designs presented in chapter 4, a security system of an IOF is implemented. A structured, top down design is used. The entire implementation is written in UNIX shell scripts, developed on a VAX 11/780 machine, written for operating systems of Berkley UNIX 4.2 and AT&T UNIX 5.2.

To demonstrate the implementation of this project, different directories residing in a common machine are used. Figure 6 shows the directory layout of an IOF security system. The commands/files which pertain to user profile are prefixed with "USER". All commands/file which pertain to time stamp ordering technique are prefixed with "stamp". The commands which pertain to data encryption are suffixed with "crypt" (reference appendix 2).

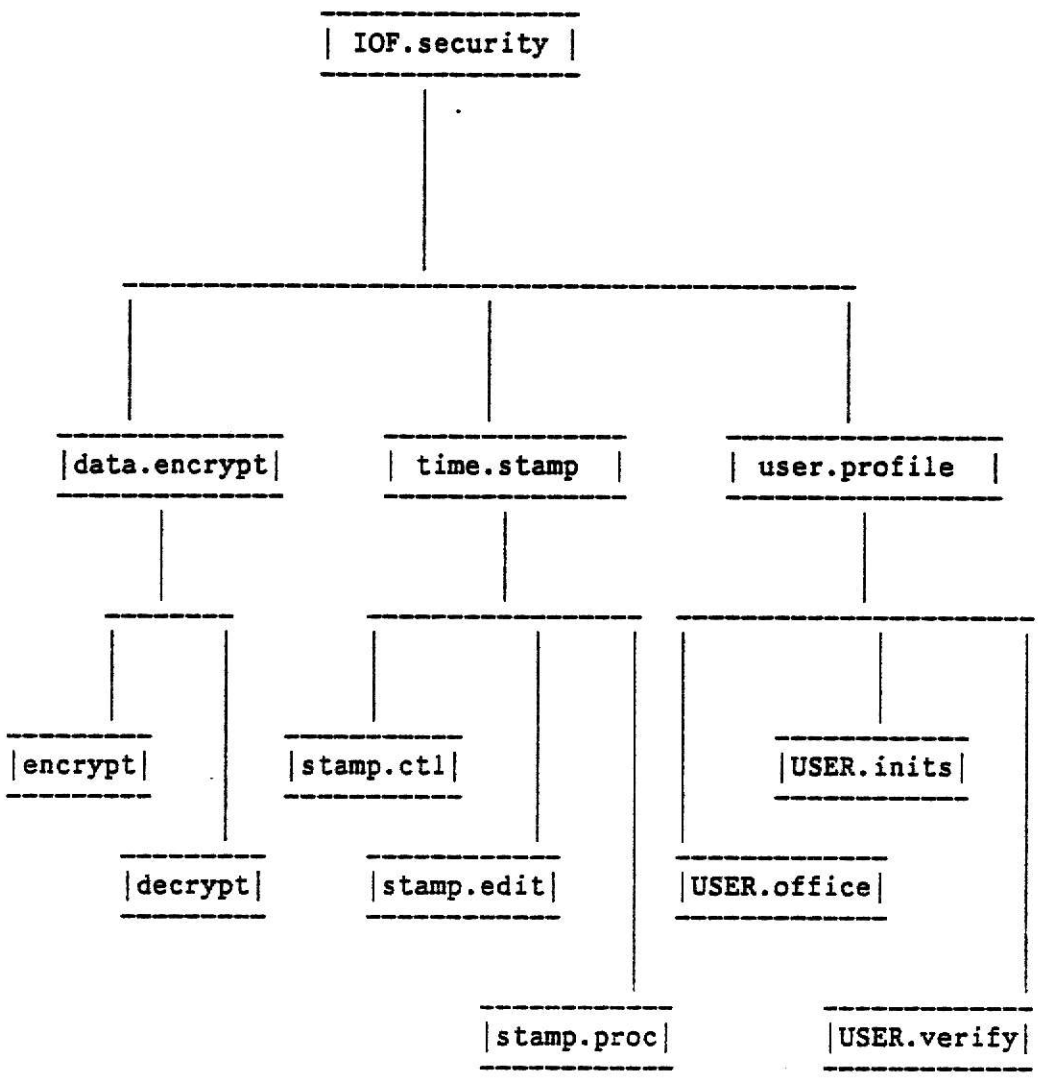



Figure 6. The Directory Layout of an IOF Security System

## 5.2 Implementation of User Profile

### 5.2.1 Introduction

When a user logs into a node, the node's user profile (e.g., .profile) is executed. Figure 7 shows the files and banner command executed by a user profile. The user profile of a node executes three different files: USER.parms, USER.disk and USER.inits.

### 5.2.2 USER.parms

The nodes' dependent variables are in the USER.parms file which resides in the node's login directory. The dependent variables can be node's name, node's location and node's account number (reference appendix 1.1a and 1.1b). User profile then executes the UNIX system command "banner" to display the name of the node onto the screen of the terminal.

### 5.2.3 USER.disk

The file USER.disk sets the variable P to the device in which the system resides (e.g., P=/a1/). Thus, the user profile can find the system codes and commands (reference appendix 1.2).

### 5.2.4 USER.inits

The amount of codes kept in a node's login directory should be kept to a minimum. Any code that can be put into a common routine should be kept in the directory of the node manager, so that multiple copies of codes are not maintained. USER.inits is that part of common codes. USER.inits is a node's initialization procedure called by the node's user profile. It sets the search path for the user. It also provides a process of identification for a user to log into an appropriate environment by calling USER.verify. The questions coded in USER.verify are used to verify the user against the information coded in the user profile. Then, USER.inits will call USER.office which will display the user's office information onto the screen of the terminal. (reference appendix 1.3, 1.31, 1.32).

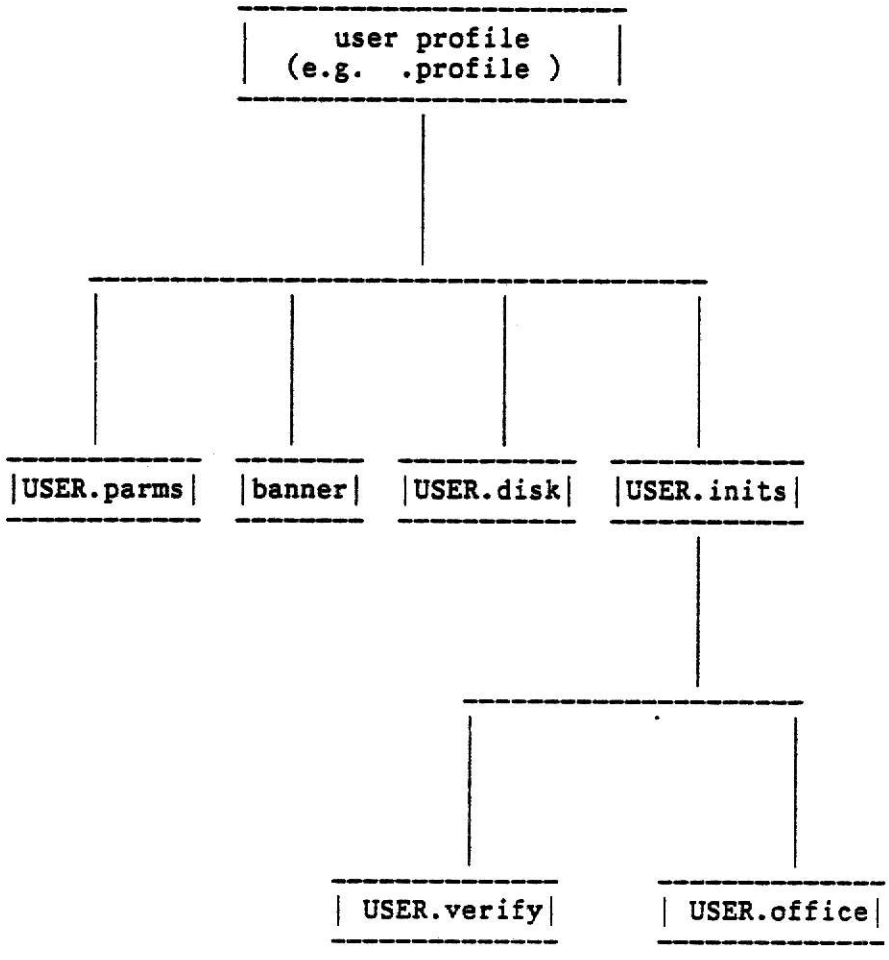



Figure 7. Files/Command Executed by User Profile of a Node

## 5.3 Implementation of Time Stamp

### 5.3.1 Introduction

Two different UNIX shell scripts were written to protect the integrity of an IOF so as to prevent its unlawful alteration. Both shell scripts utilize the unique features of UNIX files.

### 5.3.2 Stamp.edit

A UNIX shell script "stamp.edit" is written to use the UNIX system command "date" to extract the current time (T) of an operation X on a data item. "Stamp.edit" calls stamp.ctl which is a control file with site ID for each IOF, read time and write time in year, month, day, hour, minute and second. Ullman's algorithm [ull82] is then used to decide whether an operation X should be performed or not. Therefore, "stamp.edit" is used to synchronize read or write operations on an IOF (reference appendix 1.4, 1.4a, 1.4b). In doing so, "stamp.edit" helps to prevent alteration of an IOF in an inappropriate time. Figure 8 shows the UNIX system command and the stamp.ctl file executed by stamp.edit. Figure 9 shows the structure layout of a stamp.ctl record used in stamp.edit.

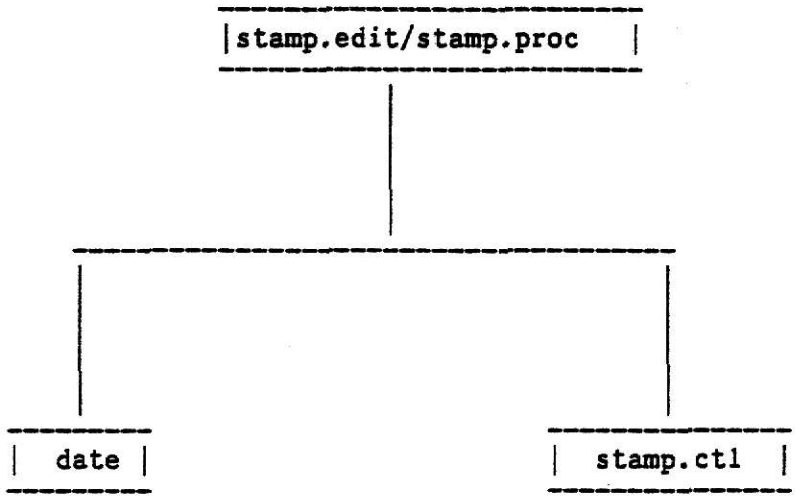



Figure 8. UNIX System Command/File Called by Stamp.edit/Stamp.proc

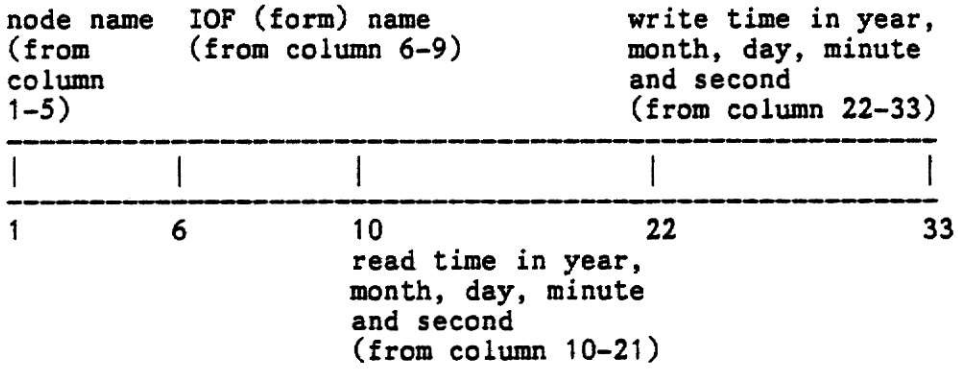



Figure 9. Structure Layout of a Stamp.ctl Record Used in Stamp.edit

### 5.3.3 Stamp.proc

Another UNIX shell script "stamp.proc" is written to set the priority of IOF operations other than read and write. It uses the UNIX system command "date" to extract the current time of an operation in terms of year, month, day and minute. In this procedure, the stamp.ctl file is updated for each operation at completion time. Assume that operation Z must precede operation Y. Operation Y is allowed to perform

if and only if the previous operation Y completion time (y-time) in the stamp.ctl file is less than the operation Z completion time (z-time). When the operation Y is completed, y-time in the stamp.ctl file is updated with the current time at completion. Thus, this y-time in the stamp.ctl can be used to turn on and off the other operations that follow operation Y (reference appendix 1.5, 1.6a, 1.6b). Figure 8 shows the UNIX system command and file executed by stamp.proc. Figure 10 shows the structure layout of stamp.ctl record used by stamp.proc.

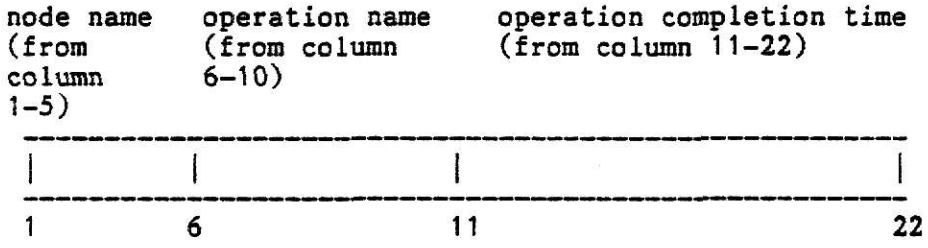



Figure 10. Structure Layout of a Stamp.ctl Record Used by Stamp.proc

## 5.4 Implementation of Data Encryption

### 5.4.1 Introduction

Two UNIX shell scripts "encrypt" and "decrypt" are written using the UNIX commands "crypt" (figure 11).

### 5.4.2 Encrypt

"Encrypt" (reference appendix 1.7, 1.7a) reads the standard readable input (plaintext) and transforms it into a non readable output (ciphertext) using the key provided by the user.

### 5.4.3 Decrypt

"Decrypt" (reference appendix 1.8, 1.8a) reads the encrypted non readable input (ciphertext) and transforms it back into a readable format (plaintext). The key used in "decrypt" must be the same key used in "encrypt".

Security of the IOF is enhanced by storing and transmitting of IOF in ciphertext format. Figure 11 shows the UNIX command executed by encrypt and decrypt.

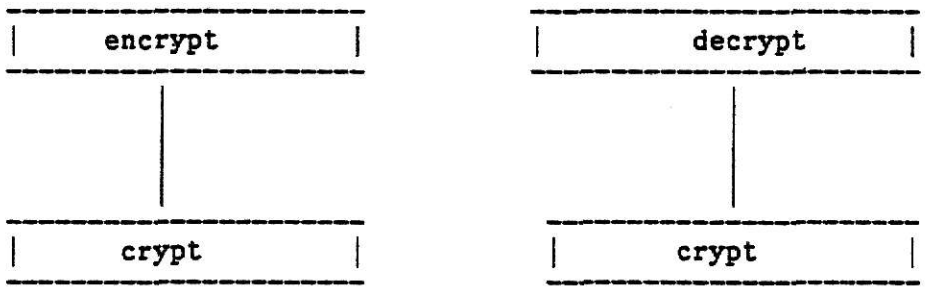



Figure 11. UNIX System Command Executed by Encrypt and Decrypt

## 5.5 Summary

The implementation of an IOF security system was done in one machine using different directories to represent different nodes. This implementation did show the performance and functionality of user profile, time stamp and data encryption.

The following UNIX scripts/files were created to demonstrate the functionality of this implementation project:

(1) .profile
(2) USER.parms
(3) USER.disk
(4) USER.inits

(5) USER.office
(6) USER.verify
(7) stamp.ctl
(8) stamp.edit
(9) iof.editor
(10) stamp.proc
(11) iof.lprint
(12) iof.print
(13) encrypt
(14) iof.encrypt
(15) decrypt
(16) iof.decrypt

During the implementation of the user profile, the node did establish the different environments for different passwords and thereby restricting a user to a particular environment with its authorized commands and data objects.

Time stamp, using the shell script stamp.edit, did establish the integrity constraint which controlled the read and write operations on an IOF, thereby preserving its integrity. The shell script, stamp.proc, did set the priority of operations other than read and write. Therefore, these two UNIX shell scripts contributed in the prevention of unlawful alteration of an IOF.

Finally, implementation of data encryption did establish the transformation of file into a non-readable format and back to a readable format. It provided users a method of storing and transmitting data safely.

The following chapter will include the conclusion of this report. It will also outline suggestions to enhance this implemented security system.

# 6. Chapter 6: Conclusion and Extensions

## 6.1 Conclusion

The following represents a summary of this project:

(A) Changed an "intelligent data object" (IDO) to an "intelligent office form" (IOF). An IOF permits access by multiple users to its local data and it allows concurrency to exist among its internal procedures.

(B) Analyzed the following security issues related to the IOF:

(1) Authentication and identification. The process of authentication and identification involves setting up a password file for users and directing the identified users to the appropriate environment.

(2) Integrity constraint. An integrity constraint is a security measure for preventing two or more users to change the same data item concurrently. It also specifies the order in which operations are to be performed.

(3) Data encryption. This security measure stores and transmits all data in an encrypted form.

(C) Designed a security system for the IOF involving the following:

(1) User profile: was designed to identify users, to specify the data objects that the user was authorized to access and the operations the user was authorized to perform on those objects.

(2) Time stamping technique: was employed to specify when operations were allowed to be performed on the data stored in an IOF. This technique was also designed to set the priority of IOF operations.

(3) Encryption procedure: was utilized to transform an IOF into a non readable format to protect IOF during transmission in the communication network.

(D) A security system for the IOF as described in (C) was implemented.

## 6.2 Extensions

There are several possible extensions to this project to improve the system.

In data encryption, algorithms other than that used in UNIX system command "crypt" may be implemented.

Performance has not been addressed at all in this implementation project. The performance of user profile, data encryption and time stamp should be considered. The effect of the length of keys used in data encryption on the speed of transformation may be explored. Another area for consideration is the optimal frequency of password replacement for maximal security.

In the user profile, the password has been used to restrict users to a particular environment. This password may be kept in encrypted format and a UNIX shell script may be set up to update that password periodically.

The security system of this report is directed mainly towards prevention. Application programs may also be set up to detect whether the current security system is sufficient or not. Recovery and correction procedures may also be set

up as part of the security system of an IOF.

## 7. Bibliography

[astr75]

Astraham, M. M. and Chamberlin, D. D., "Implications of a Structured English Query Language", Communications of the ACM, Vol. 18, No. 10, Oct. 1975.

[astr75a]

Astraham, M. M. and Lorie, R. A., "SEQUEL - XRM: a Relational System", Proc. ACM Pacific Conference, San Francis, April 17-18, 1975.

[bern80]

Bernstein, P. A. and Goodman N., "Timestamp-Based Algorithms for Concurrency Control in Distributed Database Systems", IEEE 1980, pp. 285-300.

[bern81]

Bernstein, P. A. and Goodman, N., "Concurrency Control in Distributed Database Systems", Computing Surveys, Vol. 13, No. 2 (June 1981) pp. 185-221.

[bern83]

Bernstein, P. A. and Goodman, N., "Multiversion Concurrency Control Theory and Algorithm", ACM Transactions on Database Systems, Vol. 8, No. 4 (Dec. 1983) pp. 465-483.

[camp73]

Campaigne, R. W. and Hoffman, L. J., "Computer Privacy and Security", Computers and Automation, July 1973, pp. 12-17.

[cham74]

Chamberlin, D. D. and Boyce, R. F., "SEQUEL: A Structured English Query Language", Proceedings ACM SIGMOD, pp. 249-264.

[conway72]

Conway, R. W., Maxwell, W. L. and Morgan, H. L., "On the Implementation of Security Measures in Information Systems", Communications of the ACM, Vol. 15, No. 4 (April 1972), pp. 211-220.

[cook80]

Cook, C. L., "Streamlining Office Procedures - an Analysis Using the Information Control Net Model", AFIPS NCC 1980, pp. 555-565.

[daley65]

Daley, R. C. and Neumann, P. G., "A General Purpose File System for Secondary Storage", Proc. FJCC 27 (1965).

[date84]

Date, C. J., "An Introduction to Database Systems", Vol. 2, The System Programming Series, 1984.

[denny77]

Denny, G. H., "An Introduction to SQL, a Structured Query Language", Tech. Rep. RA93(28099), IBM Res. Lab., San Jose, CA.

[diff76]

Diffie, W. and Hellman, M. E., "Multiuser and Cryptographic Techniques", National Computer Conference, 1976, pp. 109-112.

[dipi83]

Dipirro, J. E., Ferrans, J. E. and Juszczak, C. "A Form Management System for Switching Database Administration", Proceedings IEEE International Conference on Communications, Boston, MA., Vol. 1 (June 19-22, 1983), pp. 125-130.

[ell80]

Ellis, C. A. and Nutt, G. J., "Office Information Systems and Computer Science", Computing Surveys, Vol. 12, No. 1 (March 1980).

[feis73]

Feistel, H., "Cryptography and Computer Privacy", Scientific American, Vol. 228, No. 5 (May 1973).

[ferr82]

Ferrans, J. C., "SEDL - A Language for Specifying Integrity Constraints on Office Forms", Proceedings ACM - SIGOA Conference on Office Information Systems, Philadelphia, PA., June 21-23, 1982, pp. 123-130.

[fikes81]

Fikes, R. E., "ODYSSEY: A Knowledge-Based Assistant", Artificial Intelligence 16 (1981), pp. 331-361.

[gantt85]

Gantt, D. M., "Management of an Intelligent Data Object", Masters Report, Kansas State University, Manhattan, Kansas 1985.

[geha82]

Gehani, Narain H., "The Potential of Forms in Office Automation", IEEE Transactions on Communications, Vol. COM-30, No. 1 ( Jan. 1982), pp. 120-125.

[gray75]

Gray, J. N., Lorie, R. A. and Putzolu, G. R., "Granularity of Locks in a Large Shared Data Base", Proc. International Conference on Very Large Data Bases, 1975, pp. 428-451.

[griff76]

Griffiths, P. P. and Wade, B. W., "An Authorization Mechanism for a Relational Data Base System", ACM Transactions on Database Systems, Vol. 1, No. 3 (Sept. 1976), pp. 242-255.

[gudes76]

Gudes, E., Koch, H. S. and Stahl, F. A., "The Application of Cryptography for Data Base Security", National Computer Conference, 1976, pp. 97-107.

[hoff69]

Hoffman, L. J., "Computers and Privacy: A Survey", Computer Surveys, Vol. 1, No. 2 (June 1969), pp. 85-103.

[hogg84]

Hogg. J. and Gamvroulas, S., "An Active Mail System", SIGMOD record, Vol. 14, No. 2, 1984.

[lad80]

Ladd, I. and Tsichritzis, D. C., "An Office Form Flow Model", Proceedings AFIPS Office Automation Conference, National Comp. Conf., Mar. 1980, University of Toronto, Ontario Canada.

[lamp78]

Lamport, L., "Time, Clocks and the Ordering of Events in a Distributed System", CACM21, No. 7, July 1978.

[lis74]

Liskov, B. and Zilles, S., "Programming with Abstract Data Types", SIGPLAN, April 1974.

[lynch83]

Lynch, N. A. "Multilevel Atomicity - a New Correctness Criterior for Database Concurrency Control", ACM Transactions on Database Systems, Vol. 8, No. 4 (Dec. 1983), pp. 484-502.

[mcb83]

McBride, R. A. and Unger, E. A.,

"Modeling Jobs in a Distributed System", ACM SIGSMALL Conference Proceedings, Dec. 1983, pp. 32-41.

[mena80]

Menasce, D. A., Popek, G. J. and Muntz, R. R., "A Locking Protocol for Resource Coordination in Distributed Database", ACM Transactions on Database Systems, Vol. 5, No. 2 (June 1980), pp. 103-138.

[park76]

Parker, D. B., "Crime by Computer", Scribner, 1976.

[park79]

Parker, D. B., "Ethical Conflicts in Computer Science and Technology", AFIPS, 1979.

[park81]

Parker, D. B., "Computer Security Management", Reston Publishing Company, Inc., 1981.

[park83]

Parker, D. B., "Fighting Computer Crime", Scribner, 1983.

[pete77]

Peterson, J. L., "Petri Nets", ACM Computing Surveys, Vol. 9, No. 3 (Sept. 1977), pp. 223-252.

[peter67]

Petersen, H. E. and Turn, R., "System Implications of Information Privacy", Proc. SJCC 30 1967, pp. 291-300.

[reis77]

Reis, D. R. and Stonebraker, M. R., "Effects of Locking Granularity in a Data Base Management System", ACM Transactions on Database Systems Vol. 2, No. 3 (1977), pp. 233-246.

[rivest78]

Rivest, R. L., Shamir, A. and Adleman, L., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM, Vol. 21, No. 2 (Feb. 1978), pp. 120-126.

[sewc85]

Sewczwicz, R. P., "Form Definition Language for Intelligent Data Object", Masters Report, Kansas State University, Manhattan, Kansas 1985.

[shave80]

Shave, M. J. R., "Problems of Integrity and Distributed Database", Software-Practice and Experience 1980, Vol. 10, pp. 135-147.

[smith77a]

Smith, J. M. and Smith, D. C. P., "Database Abstractions: Aggregation", CACM 20, No. 6, June 1977.

[smith77b]

Smith, J. M. and Smith, D. C. P., "Database Abstractions: Aggregation and Generalization", ACM TODS 2, No. 2, June 1977.

[stone74]

Stonebraker, M. R. and Wong, E., "Access Control in a Relational Data Base Management System by Query Modification", Proc. ACM National Conference, 1974.

[tsic80]

Tsichritzis, D., "OFS: An Integrated Form Management System", Proceedings of the ACM International Conference on Very Large Data Bases, 1980.

[tsic81]

Tsichritzis, D. C., "Integrating Database and Message Systems", Proceeding 7th International Conference on Very Large Data Bases, 1981, pp. 356-362.

[tsic82]

Tsichritzis, D., "Form Management", Communications of the ACM, Vol. 25, No. 7 (July 1982), pp. 453-478.

[ull82]

Ullman, J. D., "Principles of Database Systems", Computer Science Press, 1982.

[zis77]

Zisman, M. D., "Representation, Specification, and Automation of Office Procedures", Ph.D. Dissertation, Wharton School, Univ. Pennsylvania, Philadelphia, Pa., 1977.

[zis78]

Zisman, M. D., "Office Automation: Revolution or Evolution", Sloan Management Review, spring 1978, pp. 1-16.

[zlo81]

Zloof, M. M., "QBE/OBE: a Language for Office and Business Automation", IEEE Computer (May 1981), pp. 13-22.

## 8. Appendix 1 - Source Code Listings

Appendix 1.1a

```
#.profile - a user profile
#
#Introduction:
#
#     When a user logs into a node, the node's .profile
#     is executed. '.profile' finds the node's dependent
#     variables in the file called 'USER.parms' which exists
#     in the node's login directory. Next, the '.profile'
#     executes UNIX system command 'banner' to display
#     a node name onto the terminal. Then, the '.profile'
#     executes 'USER.disk' which sets the variable P to the
#     device that a node resides on. Lastly, '.profile'
#     executes 'USER.inits' to initialize the common codes
#     used by a node.
#
#
  . USER.parms

if test -z "$BATCH"
then banner $OFFICE
fi

if test -z "$ENV"
then ENV=dev
fi

   . USER.disk

IOFDIR=$P/iof$ENV
ADMIN=$P/iofadm

if test ! -d $IOFDIR
then echo "base node directory '$IOFDIR' does not exist"
       exit 1
fi

  . $ADMIN/IOF.security/user.profile/USER.inits
```

Appendix 1.1b

```
#USER.parms - A node dependent variables
#
#
#Description:
#
#ACCT - account number of a node
#NAME - name of the developer
#ENV  - environment to which a user can be log into
#
#
#
ACCT=nwtgs4
NAME=lee
OFFICE_ORDER_NO=12345678
OFFICE=node1
OFFICE_STATE=I1
OFFICE_DESCRIPTION='This is a development office'
ENV=dev
```

```
Appendix 1.2
#USER.disk - sets the variable P to the device
#            in which a node resides
#
#
P=/usrb/att/enid/master
export P
```

```
Appendix 1.3
# USER.inits -  A user's initialization procedure
#               called by an office's .profile.
#
# Introduction:
#
#       When a user logs into an office,
#       the office's .profile is executed.
#       '.profile' finds the office dependent
#       variables in the file 'USER.parms'
#       which exists in the the office's login
#       directory.
#       Next, the .profile executes the system
#       file 'USER.disk". This program sets
#       the variable P to the device that
#       the system resides on (e.g. P=/a1). In this
#       way the .profile can find the system code.
#
# Description:
#       The amount of code kept in an office's
#       login directory should be kept
#       to a minimum.  Any code that can
#       be put into a common routine should
#       be kept in the directory of the node
#       manager, so that multiple copies
#       of the code are not maintained.
#       "USER.inits" and "USER.verify"
#       are examples of those common
#       routines. The users with special
#       login IDs (e.g. en) are
#       privilege users while others are not.
#       A privileged user gets a different
#       SPATH.
#
# Parms:
#       none
# Globals:
#
#   set in .profile via USER.parms
#       OFFICE_STATUS:
#       OFFICE:
#       OFFICE_STATE:
#       OFFICE_DESCRIPTION:
#       ACCT:
#
#  set in .profile directly
#       ENV:    set to 'tst' unless USER.parms
#               already set it.
#       P:      set to disk via system program USER.disk
```

```
#       IOFDIR: full path to ioftst or iofdev
#       ADMIN:  full path to iofadm
#
#  set in USER.inits
#       USER_ID:  first 2 characters of user's LOGNAME
#       UNIQUEID: used in the creation of temporary files
#       MAIL:     user's mail file
#       SPATH:    the search path for commands,
#                 restricted to only user commands,
#                 unless user id is privileged in which case
#                 he may access the restricted commands also.
#       Innerpath:path called by user's commands to access
#                 /bin, etc.
#       PS1:      prompt is set ot 'Enter command:'
#
# Input:
#       interactive responses from the user
#
# Output:
#       log on messages on the user's terminal
#
# Calls:
#       messages - to display the system messages
#
# Returns:
#       none
#
#
#
# all variables set in .profile and USER.parms
# are exported from here.
readonly OFFICE_STATUS OFFICE OFFICE_STATE ACCT
export OFFICE_STATUS OFFICE OFFICE_STATE ACCT
readonly ENV P IOFDIR ADMIN
export   ENV P IOFDIR ADMIN


if test -z "$BATCH" # are we in interactive mode?
then
        stty erase "


        PS1='
Enter command: '
        echo "Welcome to the IOF system  "
        echo " "

# verify the environment that the user is going
# to use
```

```
  . $ADMIN/IOF.security/user.profile/USER.verify
if test "$VERIFY" = "$ENV"
then echo " "
     echo "You are now in $ENV environment"
     echo " "
     echo " "
else
 echo "The Environment in your USER.parms is not the same"
 echo "as the environment that you  want to get into"
 exit 1
fi
fi

USER_ID=`expr $USER : ')'`

if test -z "$BATCH"
then

 # see if there is a message of today and if so print it
        if test -s $ADMIN/admmsgs/msgof2day
        then    cat $ADMIN/admmsgs/msgof2day
        fi
        # Concatenate and display system messages, if any
        if test -s $ADMIN/admmsgs/messages
        then    cat $ADMIN/admmsgs/messages
        fi

fi




# Set up the path to the user's shell
# commands IOFDIR/cmds/cmds
# unless he is a privileged user, in which
# case he gets the privilege commands
#  as well IOFDIR/cmds/privcmds.
cmds=$IOFDIR/cmds/cmds
pcmds=$IOFDIR/cmds/privcmds
if test "$USER_ID" = en
then  echo "You are a privileged user. Your id is $USER"

        SPATH=$pcmds
else    SPATH=$cmds
fi
export SPATH
```

```
# Set inner path which will be used inside user's commands
# innerpath must be in terms of the original path in order
# to incorporate whether the user is privileged
Innerpath=:/bin:/usr/local:/usr/bin:/local/bin:$SPATH
export Innerpath


# Display office information
# by calling the 'USER.office' command
if test -z "$BATCH"
then    . $ADMIN/IOF.security/user.profile/USER.office
fi
```

```
Appendix 1.31
#USER.verify - used to verify the environment to which
#               the user can be logged into
#
#
#ask for identification, and verify that it is
#a valid environment.
#


while
 echo " "
 echo 'Enter the password for the right environment:'
      stty -echo
      read VERIFY
do
      case $VERIFY in
           finished)
               echo " "
               echo ' A testing environment has been set '
               VERIFY=tst
               break
           ;;
           updated)
               echo " "
               echo ' A developing environment has been set'
               VERIFY=dev
               break
           ;;
           '')
              echo " "
              echo 'You do not have the right password to the'
              echo ' appropriate environment'
              exit 1
           ;;

          esac
     done
stty echo
export VERIFY
```

Appendix 1.32

```
#USER.office - office identification
#
#Description:
#
#
#    This file is used to tell the user what office he is
#    logged into, who he is and his account number
#
#Parms:
#       none
#
#Globals:
#        IOFDIR -
#        OFFICE -
#        OFFICE_DESCRIPTION -
#        OFFICE_STATE -
#        USER_ID -
#
#Input:
#       none
#
#Output:
#        none
#
#Calls:
#        none
#
#Return:
#        none

#set the path
PATH=$Innerpath
export PATH

#display the user's office information on
#the user's terminal
echo "The office you have logged into is:
Office Abbreviation:        $OFFICE
Office number:              $OFFICE_ORDER_NO
Account number:             $ACCT"
```

Appendix 1.4

```
 #stamp.edit - to synchronize read and write operation
 #
 #description:
 #
 # (1) use UNIX system command 'date' to extract the current
 #       time
 #
 # (4) use stamp.ctl to store the site ID together with
 #       IOF name, read time and write time.
 #
 # (5) use Ullman's algorithm [ull82] to decide whether an
 #       operation should be allowed to perform or not
 #
 #
 #
 #input: prompt on the terminal (insert, change, delete or read)
 #
 #
 #output: none
 #
 #parms:       none
 #
 #

ID=`who am i | cut -c18-22`

echo " "
echo "Enter form name to be changed/read:"
echo " "
read FILE


 #To extract the current time

Date=`date`
Cyr=`echo $Date | cut -c26-27`
Cmon=`echo $Date | cut -c5-7`
Cday1=`echo $Date | cut -c8-8`
if test "$Cday1" = " "
   then  Cday1=0
fi
Cday2=`echo $Date | cut -c9-9`
Cday=$Cday1$Cday2
Chr=`echo $Date | cut -c11-12`
Cmin=`echo $Date | cut -c14-15`
Csec=`echo $Date | cut -c17-18`

 #to translate the current month into numeric value
```

```
case $Cmon in
     [jJ]an*) Cmon=01;;
     [fF]eb*) Cmon=02;;
     [mM]ar*) Cmon=03;;
     [aA]pr*) Cmon=04;;
     [mM]ay)  Cmon=05;;
     [jJ]un*) Cmon=06;;
     [jJ]ul*) Cmon=07;;
     [aA]ug*) Cmon=08;;
     [sS]ep*) Cmon=09;;
     [oO]ct*) Cmon=10;;
     [nN]ov*) Cmon=11;;
     [dD]ec*) Cmon=12;;
     *) ;;
esac

echo " "
echo 'Operations are: Insert=I, Change=C or Read=R '
echo ' '
echo 'Enter operation: '
read ANS
case $ANS in

C | c |  I | i)


if test -f $HOME/text/$FILE
then


cp $HOME/text/$FILE $HOME/temp/$ID$FILE.tmp
ed $HOME/temp/$ID$FILE.tmp

if (test -f $HOME/temp/*.ctl) ||
   (test -f $HOME/temp/*.rd)
then
  echo 'Timestamp is in conflict!!'
  echo "$FILE has already been updated/read by other users."
  echo 'Update transaction has been aborted!!'
  rm $HOME/temp/$ID$FILE.tmp
  sleep 1
else
cat $ADMIN/IOF.security/time.stamp/stamp.ctl |
while
read Ctl
do
sid=`echo $Ctl | cut -c6-9`
if test "$sid" = "$FILE"
```

```
then

upd_rt=`echo $Ctl | cut -c10-21`
upd_wt=`echo $Ctl | cut -c22-33`


if test $Cyr$Cmon$Cday$Chr$Cmin$Csec -ge $upd_rt &&
   test $Cyr$Cmon$Cday$Chr$Cmin$Csec -ge $upd_wt
then
 Date=`date`
 Chr=`echo $Date | cut -c11-12`
 Cmin=`echo $Date | cut -c14-15`
 Csec=`echo $Date | cut -c17-18`
 Ctl=$OFFICE$FILE$upd_rt$Cyr$Cmon$Cday$Chr$Cmin$Csec
 echo $Ctl >> $HOME/temp/$ID$FILE.ctl
 mv $HOME/temp/$ID$FILE.tmp $HOME/text/$FILE
 echo 'updated operation has been completed'
else
  echo 'Timestamp is in conflict!!'
  echo "$FILE has already been updated/read by other users."
  echo 'Update transaction has been aborted!!'
  Ctl=$OFFICE$FILE$upd_rt$upd_wt
  rm $HOME/temp/$ID$FILE.tmp
  echo $Ctl >> $HOME/temp/$ID$FILE.ctl
  sleep 1
 fi
else
  echo $Ctl >> $HOME/temp/$ID$FILE.ctl
fi
done
mv $HOME/temp/$ID$FILE.ctl $ADMIN/IOF.security/time.stamp/stamp.ctl

fi
else
    echo `Invalid input form`
    echo `Read operation has been aborted`
fi
;;

R | r)
if test -f $HOME/text/$FILE
then
cp $HOME/text/$FILE $HOME/temp/$ID$FILE.read
chmod 444 $HOME/temp/$ID$FILE.read
ed $HOME/temp/$ID$FILE.read

if test -f $HOME/temp/*.ctl
then
  echo 'Timestamp is in conflict!!'
```

```
  echo "$FILE has already been updated by other users."
  rm -f $HOME/temp/$ID$FILE.read
else
cat $ADMIN/IOF.security/time.stamp/stamp.ctl |
while
read Ctl
do
 sid=`echo $Ctl | cut -c6-9`
 if test "$sid" = "$FILE"
 then
 ctl_rt=`echo $Ctl | cut -c10-21`
 ctl_wt=`echo $Ctl | cut -c22-33`


 if test $Cyr$Cmon$Cday$Chr$Cmin$Csec -ge $ctl_wt
 then
   mv $HOME/temp/$ID$FILE.read $HOME/text/$FILE
   chmod 644 $HOME/text/$FILE
   Ctl=$OFFICE$FILE$Cyr$Cmon$Cday$Chr$Cmin$Csec$ctl_wt
   echo $Ctl >> $HOME/temp/$ID$FILE.rd
   echo "Read operation has been completed"
  else
     echo $Ctl >> $HOME/temp/$ID$FILE.rd
     echo 'time stamp is in conflict!!'
     echo 'The file that you have just read is'
     echo 'differed from the one in the current file.'
     echo 'Read transaction has been aborted!!'
     rm -f $HOME/temp/$ID$FILE.read
     sleep 1
  fi
else
   echo $Ctl >> $HOME/temp/$ID$FILE.rd
fi
done

mv $HOME/temp/$ID$FILE.rd $ADMIN/IOF.security/time.stamp/stamp.ctl


fi
else
    echo 'Invalid form'
    echo 'Read operation has been aborted'
fi
;;
*);;
esac
```

Appendix 1.4a

```
#stamp.ctl - A time stamp control file
#
#
#description:
#
#     This control file consists of two type of records:
#
#     (1) used for stamp.edit with site ID together with
#         IOF name, read time and write time in year,
#         month, day, hour, minute and second.
#
#     (2) used for stamp.proc with node name, operation
#         switch name and operation completion time.
#
#
#
node1edint860619184245
node1iof2860603122040860603132345
node1edout860619184245
node1iof1860619190924860619190904
```

Appendix 1.4b

```
# iof.editor - IOF command used to demonstrate the
#              functionalities of time.edit
#
#
  . $ADMIN/IOF.security/time.stamp/stamp.edit
```

```
Appendix 1.5
#stamp.proc - to synchronize operations other than
#             read or write
#
#
#description:
#
#
#     It extracts the time of an operation Z (indicated by
#     "edout" in stamp.ctl) by which the current operation
#     Y must follow. If the time that associated with
#     "edout" (in this report) is great than the time
#     that associated with "edint", then Y will be
#     allowed to execute.
#
#input:
#
#       switch of the process such as 'edout'
#
#output:
#
#  A switch with the value of "y' to indicate whether the
#  next operation will be performed or not.
#
#
#
#
PATH=$Innerpath
export PATH

switch=`cat $HOME/temp/proc.sw`

cat $ADMIN/IOF.security/time.stamp/stamp.ctl |
   while
   read Ctl
   do
   sid=`echo $Ctl | cut -c6-10`
   ctl_time=`echo $Ctl | cut -c11-22`
   ctl_rtime=`echo $Ctl | cut -c23-33`
   if test "$sid" = "$switch"
   then
         echo "$ctl_time" >> $HOME/temp/edout_time
   fi
   if test "$sid" = edint
   then
         echo "$ctl_time" >> $HOME/temp/edint_time
    fi
  done
```

```
edint_t=`cat $HOME/temp/edint_time`
edout_t=`cat $HOME/temp/edout_time`
if test $edout_t -gt $edint_t
then
   echo "y" >> $HOME/temp/$switch.ctl
else
   echo " "
   echo " "
   echo "Time stamp is in conflict!!"
   echo " "
   echo "iof.print has to be performed after iof.lprint!!"
fi


        rm $HOME/temp/edint_time
        rm $HOME/temp/proc.sw
        rm $HOME/temp/edout_time
```

Appendix 1.6a

```
#iof.lprint -
#  UNIX shell script used to demonstrate the usage of
#  stamp.proc. It is used together with another UNIX
#  shell script "print". It should be used before
#  "print". If the  order is reversed, an error message
#  of "Time stamp is in conflict!" will be printed.
#
#
#description:
#
#  (1) it lists a file and executes "date" command
#      to extract the current time which is then
#      concatenated with "edout" in stamp.ctl (an IOF
#      control file).
#
#   (2) "iof.print" UNIX shell will execute stamp.proc
#       to find out whether the time associated with
#       "edout" is greater than the time associated
#       with "edint". If it is, then, "iof.print"
#       will be allowed to execute. Using stamp.proc,
#       "iof.print" command has to be executed after
#       "iof.lprint" command and not before.
#
#
PATH=$Innerpath
export PATH


 #to extract the current time
Cyr=`date | cut -c27-28`
Cmon=`date | cut -c5-7`
Cday1=`date | cut -c9-9`
if test "$Cday1" = " "
   then  Cday1=0
fi
Cday2=`date | cut -c10-10`
Cday=$Cday1$Cday2
Chr=`date | cut -c12-13`
Cmin=`date | cut -c15-16`
Csec=`date | cut -c18-19`


 #to translate the current month into numeric value
case $Cmon in
     [jJ]an*) Cmon=01;;
     [fF]eb*) Cmon=02;;
     [mM]ar*) Cmon=03;;
     [aA]pr*) Cmon=04;;
```

```
      [mM]ay)  Cmon=05;;
      [jJ]un*) Cmon=06;;
      [jJ]ul*) Cmon=07;;
      [aA]ug*) Cmon=08;;
      [sS]ep*) Cmon=09;;
      [oO]ct*) Cmon=10;;
      [nN]ov*) Cmon=11;;
      [dD]ec*) Cmon=12;;
      *) ;;
esac

cur_time=$Cyr$Cmon$Cday$Chr$Cmin$Csec

echo " "
echo 'Enter the name of file to be listed'
echo " "
read Ans

 # place an IOF command that has to be performed first here

ls -l  $Ans


cat $ADMIN/IOF.security/time.stamp/stamp.ctl |
while
read CTL
do
    Proc=`echo $CTL | cut -c6-10`
    ID=`echo $CTL | cut -c1-5`
    Proc_time=`echo $CTL | cut -c11-22`
    Re_time=`echo $CTL | cut -c23-33`

    if  test "$Proc" = "edout"
    then
    Proc_time=$cur_time
    fi
    CTL=$ID$Proc$Proc_time$Re_time
    echo $CTL >> $HOME/temp/$OFFICE.ctl
    done
mv $HOME/temp/$OFFICE.ctl $ADMIN/IOF.security/time.stamp/stamp.ctl
```

Appendix 1.6b

```
#iof.print -
#  UNIX shell used to demonstrate the usage of stamp.proc.
#  It should be executed after another UNIX shell script
#  "iof.lprint". If the order is not in sequence, then an
#  error message of "Time stamp is in conflict!" will
#  be printed.
#
#description:
#   (1) it executes stamp.proc to find out whether the
#       the time associated with "edout" is greater than
#       that associated with "edint". If it does, then
#       a switch such as 'edout.ctl' will be set up
#       in the directory
#
#    (2) if 'edout.ctl' exists, then 'iof.print' will be
#        allowed to perform and its execution time
#        will be concatenated with "edint" and "edout"
#        in the stamp.ctl file
#
#
#
#
PATH=$Innerpath
export PATH

 switch="edout"
 echo $switch >> $HOME/temp/proc.sw
if test -s $HOME/temp/edout.ctl
then
echo " "
echo 'Enter the name of file to be printed'
echo " "


 # Place an IOF command that has to be performed next here

read Ans
pr $Ans

Cyr=`date | cut -c27-28`
Cmon=`date | cut -c5-7`
Cday1=`date | cut -c9-9`
if test "$Cday1" = " "
   then  Cday1=0
fi
Cday2=`date | cut -c10-10`
Cday=$Cday1$Cday2
```

```
Chr=`date | cut -c12-13`
Cmin=`date | cut -c15-16`
Csec=`date | cut -c18-19`

 #to translate the current month into numeric value
case $Cmon in
     [jJ]an*) Cmon=01;;
     [fF]eb*) Cmon=02;;
     [mM]ar*) Cmon=03;;
     [aA]pr*) Cmon=04;;
     [mM]ay)  Cmon=05;;
     [jJ]un*) Cmon=06;;
     [jJ]ul*) Cmon=07;;
     [aA]ug*) Cmon=08;;
     [sS]ep*) Cmon=09;;
     [oO]ct*) Cmon=10;;
     [nN]ov*) Cmon=11;;
     [dD]ec*) Cmon=12;;
     *) ;;
esac

cur_time=$Cyr$Cmon$Cday$Chr$Cmin$Csec


cat $ADMIN/IOF.security/time.stamp/stamp.ctl |
while
read CTL
do
    Proc=`echo $CTL | cut -c6-10`
    ID=`echo $CTL | cut -c1-5`
    Proc_time=`echo $CTL | cut -c11-22`
    Re_time=`echo $CTL | cut -c23-33`
    if  (test  $Proc = edout) ||
        (test $Proc = edint)
    then
    Proc_time=$cur_time
    fi
    CTL=$ID$Proc$Proc_time$Re_time
    echo $CTL >> $HOME/temp/$OFFICE.ctl
  done

 mv $HOME/temp/$OFFICE.ctl $ADMIN/IOF.security/time.stamp/stamp.ctl
 rm $HOME/temp/edout.ctl
fi
```

```
Appendix 1.7
#encrypt - used in data encryption procedure
#
#Description -
#
#     This module reads form the readable standard input and
#     transforms into a non readable format using the key
#     provided as a means of transformation.
#
#
#Input -
#
#       standard readable format
#
#
#Output -
#
#        Non readable format
#
#
PATH=$Innerpath
export PATH
echo 'Enter key for encryption: '
     read KEY
echo 'Enter name of file to be encrypted: '
     read FILE
echo 'Enter name of output file of encryption: '
     read OUTPUT
crypt $KEY < $FILE > $OUTPUT
echo 'Encryption is done'
```

Appendix 1.7a

```
# iof.encrypt - IOF command used to demonstrate the
#               functionalities of data encryption
#
#

  . $ADMIN/IOF.security/data.encrypt/encrypt
```

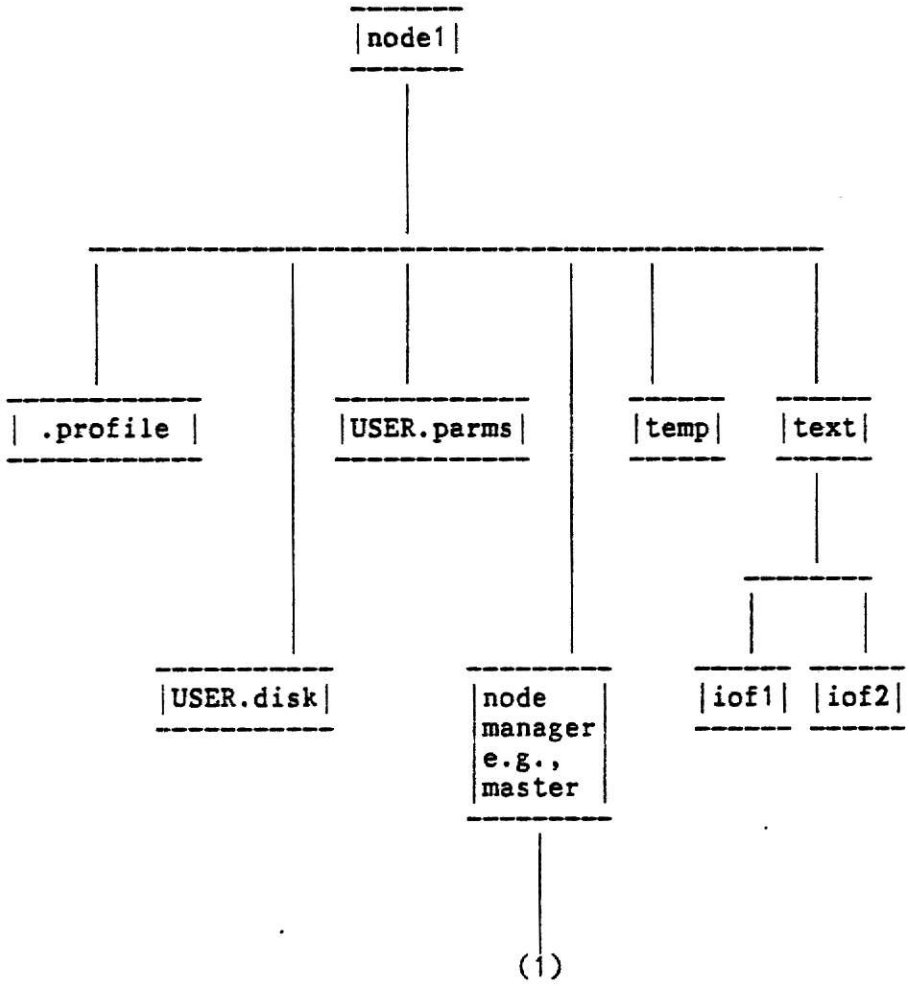
```
Appendix 1.8
#decrypt - used in data encryption procedure
#
#Description  -
#
#      This module reads in the non readable input and
#      transforms it back to a readable format using
#      the key provided.
#
#
#Input -
#
#       Non readable format
#
#
#Output -
#
#       Readable format
#
#
#
PATH=$Innerpath
export PATH
echo 'Enter key for decryption: '
     read KEY
echo 'Enter name of file to be decrypted: '
     read FILE
echo 'Enter name of output file of decryption: '
     read OUTPUT
crypt $KEY < $FILE > $OUTPUT
echo 'Decryption is done'
```

Appendix 1.8a

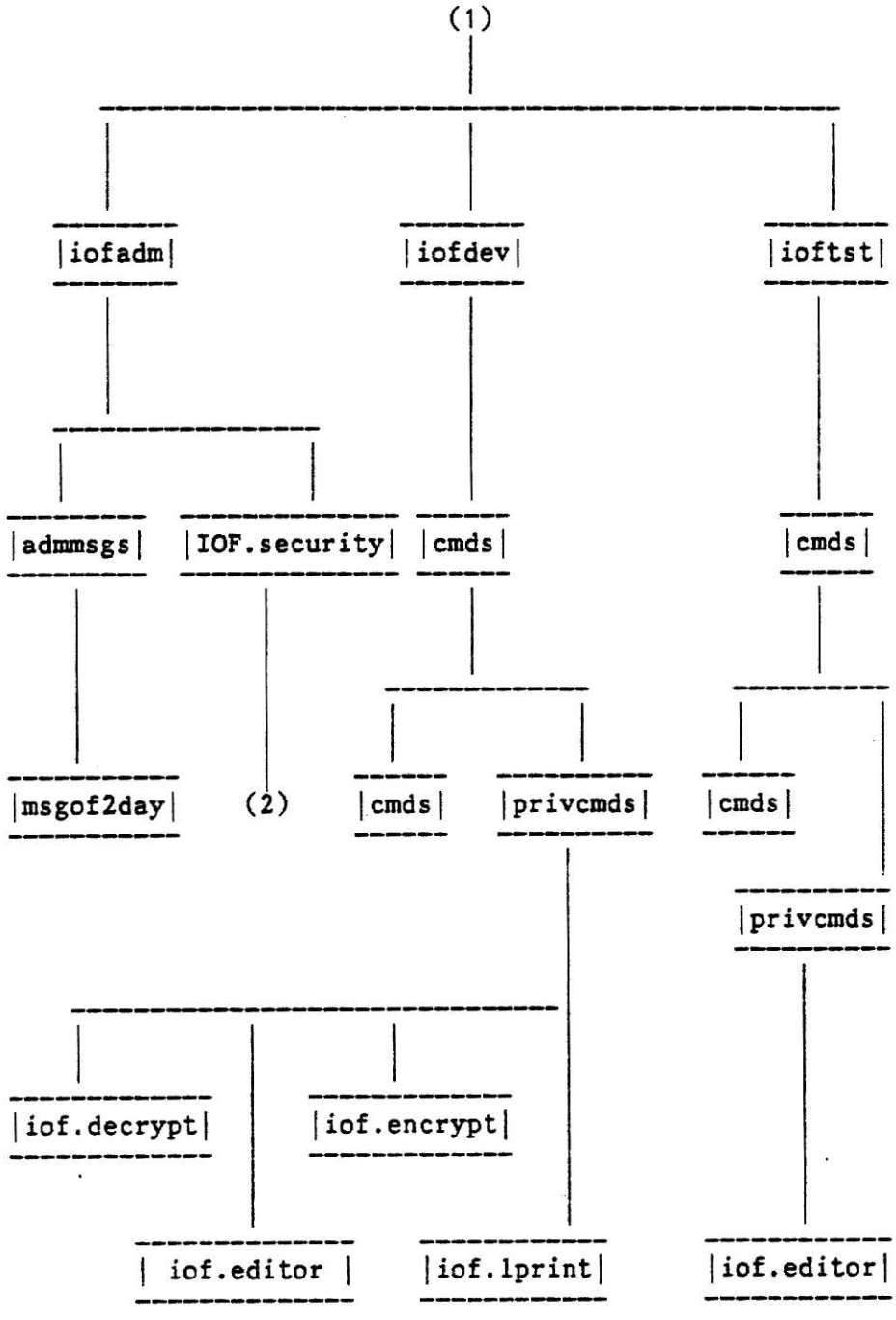
```
# iof.decrypt - IOF command used to demonstrate the
#               functionalities of data decryption
#
#

  . $ADMIN/IOF.security/data.encrypt/decrypt
```

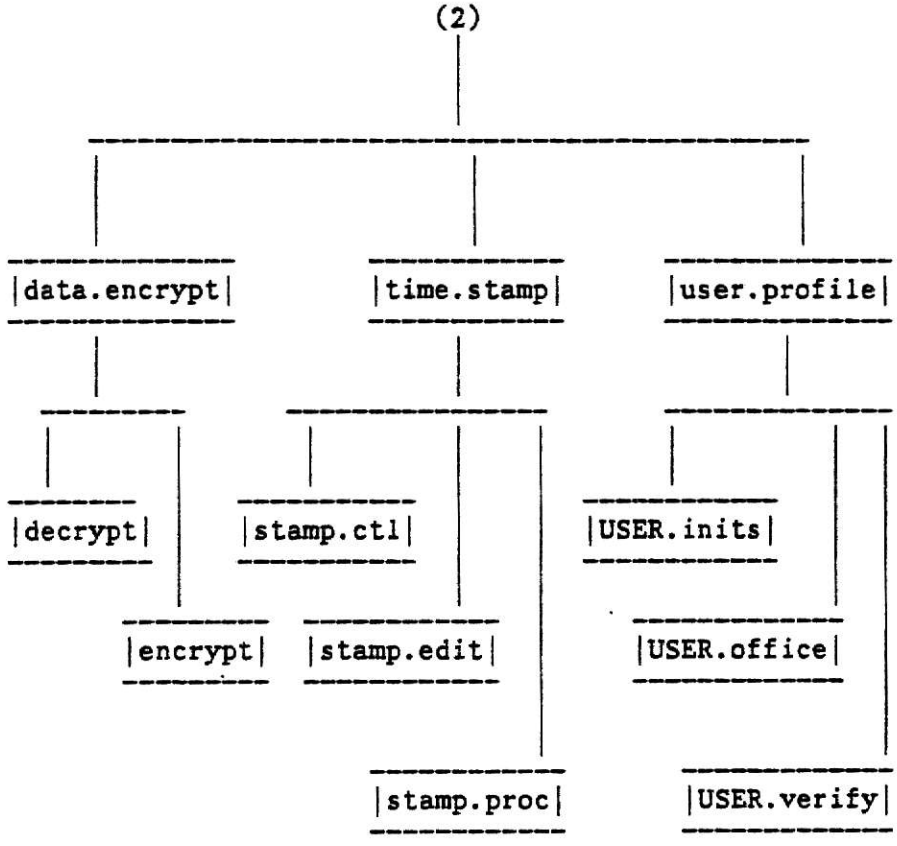
## 9. Appendix 2 - Source Code Structure Chart



Appendix 2 - Source Code Structure Chart



Appendix 2 - Source Code Structure Chart



# PRIVACY AND SECURITY OF AN INTELLIGENT OFFICE FORM

by

KUM-YU ENID LEE

B.A., University of Texas at Austin, 1972

---

AN ABSTRACT OF A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1986

## ABSTRACT

PRIVACY AND SECURITY OF AN INTELLIGENT OFFICE FORM

An Intelligent Data Object (IDO) is defined as an elaboration of an abstract data type. It consists of routing information, specific processing information, a history log, text ( data structures) and a set of operations defined on the data object.

In this report, a version of an IDO, termed an "intelligent office form" (IOF), is discussed. An IOF permits access by multiple users to its local data and it allows concurrency to exist among its internal procedures.

The design of a central "security manager" to protect an IOF against unauthorized alteration or destruction is presented. Three different security measures are implemented. The first is a user profile to specify the data objects that the user is authorized to access and the operations the user is authorized to perform on those objects. The second is a time stamp ordering technique to specify when operations are allowed to be performed on the data stored in IOF. Examples of allowable operations are read, write, delete and insert. The technique also sets the priority of IOF procedures. The third is an encryption procedure to protect the data during transmission in the communication network.

The proposed security measures are programmed in UNIX shells.

1430-89
CD-68