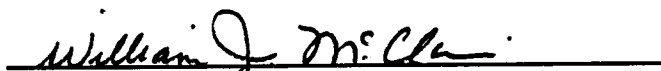


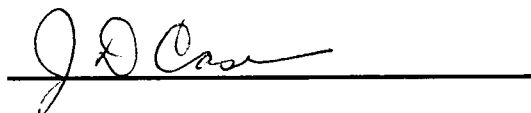
To the Graduate Council:

I am submitting herewith a thesis written by Matthew G. Hecht entitled "Exploring Security on an Extended Local Area Network." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science with a major in Computer Science.


William J. McClain, Major Professor

We have read this thesis
and recommend its acceptance:





Accepted for the Council:


Associate Vice Chancellor
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Matth Necht

Date December 1, 1992

**EXPLORING SECURITY ON AN
EXTENDED LOCAL AREA NETWORK**

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Matthew G. Hecht

December, 1992

DEDICATION

This thesis is dedicated to my parents, Dr. Harvey Hecht and Mrs. Carolyn Lea Hecht, for all their support through my twenty years of schooling and the two plus years it took to finish this thesis.

ACKNOWLEDGEMENTS

I thank Dr. McClain, my advisor, and Dr. Case and Dr. Dunigan for their ideas and assistance with this project.

I thank Jamey Maze, for his input and help with Kerberos during my summer internship at Martin Marietta and for proofing the Kerberos material in my thesis.

I thank my elementary, high school and college teachers for making learning a fun and interesting experience.

I thank my fellow graduate students for all the help and friendship they have shown me the past four years.

ABSTRACT

The purpose of this thesis was to report on the research of a study of three security products and how they improve security in Local Area Networks (LANs). The Kerberos Authentication System, Digital Equipment Corporation's Security Encryption System, and Xerox's Encryption Unit were the studied products. Kerberos Version 4 was installed and an automated teller machine (ATM) application was written and transformed into a Kerberos application. Other goals of this thesis were to show: (1) that using Kerberos does not seriously degrade performance, (2) that it is simple to install Kerberos and (3) that converting an application to use Kerberos can be done easily.

TABLE OF CONTENTS

CHAPTER	PAGE
1. Introduction	1
1.1 Overview	3
2. Kerberos	6
2.1 Goals	6
2.2 Kerberos Authentication	8
2.3 Kerberos Utilities	14
2.4 Kerberos Installation	17
2.5 Protocol Weaknesses	19
2.6 Version 5	22
3. DEC's SES	26
3.1 DESNC Controller	26
3.1.1 Operation of the DESNC Controller	27
3.1.2 Security and Network Management Functions	28
3.1.3 VAX Key Distribution Center System (VAX KDC System)	30
3.2 Setting Up the Security Encryption System	31
3.2.1 Network Configuration	32
3.2.2 Access Control	33
3.2.3 Initialization	34
3.2.4 Key Distribution and Management	34
3.2.5 Restrictions	37
3.3 Performance of the Security Encryption System	38

4.	Other Security Products	41
4.1	Xerox Encryption Unit	41
4.1.1	Functional Description	42
4.1.2	Features and Benefits	44
4.1.3	Customer Testing	45
4.2	Blacker	50
4.2.1	Blacker Front End	52
4.3	Other Products Considered for Study	57
5.	Automated Teller Machine	58
5.1	ATM Overview	59
5.2	ATM Client	61
5.3	ATM Server	62
5.4	X-Windows Interface	65
5.5	Kerberos Performance	66
5.5.1	SUN Platform Tests	68
5.5.2	DEC Platform Tests	68
5.5.3	TTCP Performance Tests	73
5.6	Traffic Analysis	79
6.	Conclusion	80
6.1	Xerox's XEU and DEC's SES	80
6.2	Kerberos	83
6.3	Speed Comparison	85
6.4	Availability	86
6.5	Future Research Ideas	87

LIST OF REFERENCES	88
APPENDIX	92
A. Kerberos Terms	93
VITA	96

LIST OF FIGURES

FIGURE	PAGE
1 Kerberos authentication scheme	9
2 Encryption and decryption process	29
3 An example of an Ethernet LAN using SES	30
4 Key Distribution Process	36
5 DESNC Performance	39
6 Typical Network Utilizing Xerox Encryption Units	46
7 Test scenario #1	49
8 Test scenario #2	51
9 Test scenario #3	51
10 Example of a network using a Blacker Front End	53
11 IP address	55
12 Client code overview	63
13 Server code overview	64
14 X-Windows Interface	67
15 System and user times for rcp tests performed between a SPARC- station 1 and a SPARCstation ELC	69
16 Actual times for all three rcp tests performed between a SPARC- station 1 and a SPARCstation ELC	70
17 System and user times for rcp tests performed between a DECsta- tion 2100 and a DECstation 5000/200	71
18 Actual times for all three rcp tests performed between a DECstation 2100 and a DECstation 5000/200	72

19	Transmission times for ttcp tests performed between a SPARCstation 1 and a SPARCstation ELC	74
20	Receiving times for ttcp tests performed between a SPARCstation 1 and a SPARCstation ELC	75
21	Transmitting CPU times for ttcp tests performed between a SPARCstation 1 and a SPARCstation ELC	76
22	Receiving CPU times for ttcp tests performed between a SPARCstation 1 and a SPARCstation ELC	77

CHAPTER 1

Introduction

Network security is a very important concern in the field of computers. Back in the 1960s and 1970s when computer professionals developed security measures for mainframes and other computers, computers were safe. Two developments have changed this situation in the last decade or so: (1) the heavy use of personal computers (PCs) in businesses, colleges, homes and other areas, and (2) the rapid expansion and use of computer networks (Local Area Networks or Wide Area Networks). In the former case, there are many unprotected computers and people using the computers who are not aware of the security problem. In the latter case, networks provide users the ability to access machines remotely. The use of networks multiplies the security threat by giving the intruder more opportunities and avenues to compromise the system. As examples, computer networks are used to send bank transactions, stock information, deposit workers' checks in their banks, military information, etc.

There are also three major security threats that a network can present:

1. Unauthorized persons on remote systems accessing unprotected PCs and workstations (WS). This could be done either by brute force or by masquerading as another user.
2. Unauthorized persons gathering unprotected information passing over a net-

work. The person could access this information on an intermediate node in a store-and-forward implementation or on a broadcast medium operating in promiscuous mode (e.g., a ring or bus topology).

3. Unauthorized persons collecting unprotected information from emanations from components of computer systems and networks.

To improve network security, four main methods can be used: (1) encryption, (2) access control mechanisms, (3) user and/or node authentication, and (4) different protocols. There are many security products in the field, but most of these products use only authentication and authorization, which provide some, but not enough security. These products can usually ensure that only authorized personnel are performing certain actions, but they do not provide data confidentiality. If an intruder desires to steal information, he/she needs only to read the message while it goes by on the wire. There is no need to alter the message (although this can be done on some systems). For companies transferring confidential stock prices, TOP SECRET military information, or bank transactions, most users or businesses select the option of using encryption, which provides data confidentiality.

Besides the problems of security on networks, the operating systems servicing most PCs and WSs acting as servers or clients on networks are not mature enough to supply the needed protection from unauthorized access. The network medium (especially bus and ring technology), typically, are not secure; these media provide inexpensive opportunities for nodes to be operated in promiscuous mode, which is both passive and undetectable by any network security or management device. The third problem mentioned above (emanations) has become less of a problem in recent years due to advances in technology, but it still must be considered to

assure a secure network.

These threats can be countered confidently by only one method: encryption. The encryption of messages between nodes provides a greater assurance that (1) the participants are authorized participants, because they have the same encryption key and (2) the information sent over the physical medium is protected since there is a high probability that an intruder could not decrypt the message even if it were captured promiscuously or by capturing and interpreting emanations.

When studying the use of encryption, there are four major problems:

1. How much of the message needs to be encrypted to provide security against the normal threats to the local area network (LAN)?
2. How does the encryption affect the throughput of the network?
3. How are the encryption keys administered (i.e., generated, distributed by a key server, or controlled) in the LAN or wide area network (WAN)? How can this be done efficiently?
4. Is the message encrypted end-to-end over both WANs and extended LANs via bridges and routers?

1.1 Overview

The purpose of this thesis is to report on the research of a study of three encryption-based security products and how they improve the security in LANs. The products chosen for the study were: (1) the Kerberos Authentication Service¹,

¹Kerberos was developed as part of the Athena project by the Massachusetts Institute of Technology

(2) DEC's Security Encryption System² and (3) Xerox's Encryption Unit³. The installation of Kerberos Version 4 is also discussed. This thesis also covers the implementation of an automated teller machine (client-server model) that uses Kerberos. The goal of this study was to provide a concise yet informative overview of these products for the interested party. Other goals were to show (1) that using Kerberos does not seriously degrade performance, (2) that it is simple to install Kerberos, and (3) that converting an application to use Kerberos can be done easily.

Chapter 2 presents M.I.T's Kerberos Authentication Service, including summaries of its authentication scheme, utilities and installation; the protocol weaknesses of Kerberos; and the differences between Kerberos Version 4 and Kerberos Version 5.

Chapter 3 presents Digital Equipment Corporation's Security Encryption System (SES). The two main hardware components of DEC's SES are the Digital Ethernet Secure Network Controller (DESNK) and the VAX Key Distribution Center (VAX KDC). The chapter describes the operation of these two components and the security and network management functions for the system. A description of the installation of the system and its performance are also provided.

Chapter 4 presents Xerox's Encryption Unit (XEU). A functional description of the XEU and the features and benefits of the system are provided. A detailed plan for testing the system is also included in the chapter. Blacker, a security product used by the Defense Communications Agency (DCA), is also described in

²The Security Encryption System is a marketed product of the Digital Equipment Corporation

³The Xerox Encryption Unit is a marketed product of the Xerox Corporation

Chapter 4.

Chapter 5 includes the main project for this thesis, a simulated automated teller machine (ATM), a client-server application that uses Kerberos for authentication and encryption. The chapter includes summaries of the ATM client, server, and the X-Windows⁴ interface for the users of the ATM. The results of performance tests on Berkeley's remote copy (*rcp*) utility using Kerberos are also included. The performance tests are run on (1) Normal *rcp*, (2) *rcp* running only with Kerberos authentication, and (3) *rcp* running with both Kerberos authentication and encryption. The results of performance tests on the *ttcp* utility are also included. Traffic analysis of Kerberos is also in Chapter 5.

Chapter 6 is the conclusion of the thesis. The pros and cons of the three main security products and their similarities and differences covered by this research are discussed. The products are evaluated on how they protect against network security threats and how the problems of encryption are addressed. Future research opportunities are also identified.

⁴Massachusetts Institute of Technology

CHAPTER 2

Kerberos

The first product to be examined in detail is Kerberos, an authentication system developed by S.P. Miller, B.C. Neuman, J.S. Schiller, and J.H. Salzer of M.I.T. and implemented by Miller and Neuman of M.I.T [1]. Since Kerberos was the only product that was available for testing, it was studied in more detail than were DEC's Security Encryption System, Xerox's Encryption Unit or Blacker. Some performance figures for Kerberos are included in Chapter 5. Unless stated otherwise, the discussion will be centered on Kerberos Version 4.

2.1 Goals

In any open network of workstations, such as M.I.T.'s Project Athena, most workstations are not under the control of any single organization. Users having a little knowledge about their systems can reboot their workstations and be granted **superuser** privileges on those machines. When describing Kerberos and the other security products, physical security will be ignored for the most part, because the necessity for physical security is understood; without it there is no security. When using Kerberos, users must prove their identity each time they want to use a service. It also would be beneficial for the server to prove its identity, which is called *mutual authentication*. Ideally, the user should rarely, if ever, notice a

performance degradation when the security is in place.

The requirements established for Kerberos included the following [1]:

- The system should be **secure** (i.e., hard to break). Intruders should not be able to impersonate other users unless they have managed to discover the encryption key.
- The system should be **reliable**. If the authentication service goes down, users should not be able to access any resources. The reliability should be as high as possible. Slave servers can be used to help improve the reliability.
- Use of the system should be **transparent** to the user. The user should not be aware of the security in the system unless the system goes down or authentication fails. The user should not be prompted repeatedly for passwords.

Kerberos does not provide authorization. It provides only authentication, but authorization can easily be added by using access control lists or similar mechanisms. Kerberos does not provide protection against viruses, worms, or trojan horses, except to the extent that encryption provides such protection.

There are three possible levels of security with Kerberos.

1. Authenticity at connection initiation only. All messages that follow from the same source are assumed to be valid. Athena's network file system uses this level.
2. KRB_SAFE messages, which are all authenticated but not encrypted. KRB_SAFE messages provide data integrity by using a cryptographic checksum; only

collision-proof (not easily compromised) checksums should be used with these types of messages in order to provide optimal protection.

3. KRB_PRIV messages, which are authenticated and encrypted. For example, these are used by the Kerberos server to send session keys over the network.

Kerberos uses the Data Encryption Standard (DES) [2] with cipher block chaining and cyclic redundancy 32 checksum (CRC-32). Other encryption methods and checksums are available in Version 5 of Kerberos.

2.2 Kerberos Authentication

A general depiction of the operation of the Kerberos authentication scheme is shown in Figure 1. Figure 1 is also an example of a Kerberos realm. A realm is usually the size of a local area network (LAN) or smaller. The following symbols will be used in the description of Figure 1:

- M = message
- K_s = TGS session key
- K_{serv} = key that the Kerberos-authenticated service and the TGS share.
- K_{sess} = session key that the Kerberos-authenticated service and the the user share.
- K_{tgs} = TGS key.
- K_u = the user's key.

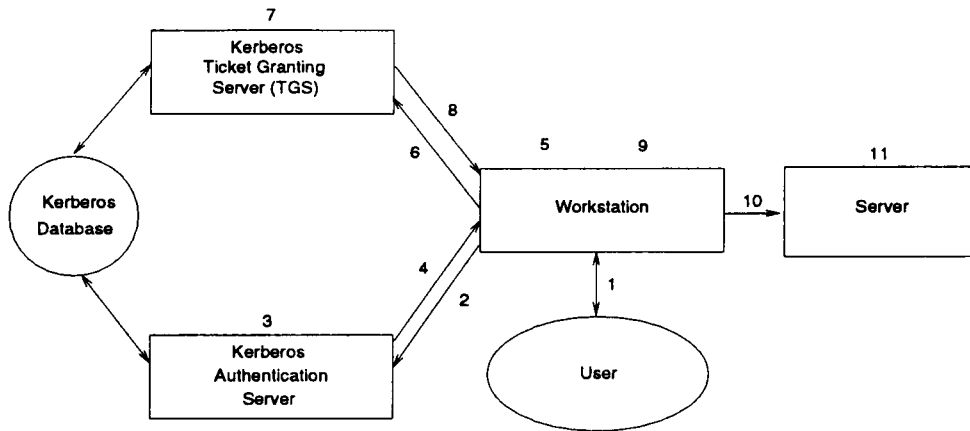


Figure 1: Kerberos authentication scheme

Source: W. Richard Stevens. *UNIX Network Programming*. Prentice-Hall, 1990.

The numbered steps, one to eleven in Figure 1, represent the following actions [3]:

1. The user begins a login session at a workstation by typing in his/her username in response to the **login:** prompt.
2. The name of the authentication server and the local Kerberos realm are retrieved from the Kerberos configuration file, *krb.conf*, which is in the */etc* directory on the workstation by default. The login program sends a message containing the username and the name of the Kerberos ticket-granting server (TGS) to the Kerberos Authentication server.

$$M_2 = \{ username, TGS-name \}$$

3. The authentication server uses the user's username and the TGS's name as an index into the Kerberos database and gets an encryption key for each.

These encryption keys are used to produce one-way encrypted passwords.

4. The authentication server generates a response to the login program on the originating workstation. This response contains a *ticket-granting ticket* (TGT), which gives the user access to the TGS. Tickets are never sent in cleartext. This ticket is encrypted using the encryption key of the TGS and usually looks like the following:

$$\text{ticket} = \{ \text{username}, \text{TGS-name}, \text{W-IP-address}, \text{TGS-session-key} \}$$

This message is then encrypted using the user's encryption key, which is derived by a special algorithm from his/her password into a one-way, non-invertible DES key. This key cannot be decrypted. It is used to encrypt and/or decrypt a message.

$$M_4 = \{ K_u[\text{TGS-session key}, K_{tgs}(\text{ticket})] \}$$

5. The login program receives this message and then prompts the user for his/her password. If the correct password is entered, it is turned into the proper decryption key; the message is then decrypted, leaving the workstation with a TGS session key and an unreadable ticket. The decryption key for the ticket should be known only by the TGS and the Kerberos Authentication Server. The workstation then saves a copy of the encrypted ticket and the session key for future use. The decryption key is needed only for this one time, so it is immediately erased from memory.

When users want to use a service such as printing a file or reading mail, they need a ticket for that particular service. This is done transparently, so the user should not notice anything out of the ordinary.

6. The workstation builds a message to send to the TGS.

$\text{authenticator} = \{ \text{username}, \text{W-IP-address}, \text{timestamp} \}$

The authenticator is encrypted using the TGS session key provided in step 5.

$M_6 = \{ K_{tgs}(\text{ticket}), K_s(\text{authenticator}), \text{destination server} \}$

7. The TGS receives the message and first decrypts the encrypted ticket using its TGS encryption key. The TGS session key is extracted from the ticket, and it is used to decrypt the authenticator. Comparison checks are made on the username, the Internet Protocol (IP) [4] address of the workstation, the server-name, and the timestamp. If these checks succeed, the TGS looks up the destination server (i.e., the service the user wants to use) in the database and gets the encryption key for the specified service. Otherwise, the message is deemed unauthentic and is discarded.

8. The TGS comes up with a new random session key and creates a new ticket that includes the new session key and the name of the service requested.

$\text{ticket} = \{ \text{username}, \text{server-name}, \text{W-IP-address}, \text{session-key} \}$

The ticket is encrypted using the key that the TGS and the server share. This key is distributed to the server at setup time and is stored in */etc/srvtab* by default.

$M_8 = \{ K_s(K_{sess}, K_{serv}(\text{ticket})) \}$

9. The workstation receives and decrypts the message using its TGS session key to get the session key required to communicate with the server and the ticket that it needs to send to the server.

10. The workstation then builds an authenticator to send to the server along with the encrypted ticket.

$$\text{authenticator} = \{ \text{username}, \text{W-IP-address}, \text{timestamp} \}$$

The authenticator is encrypted using the new session key it received in step 9. This is packaged with the encrypted ticket and the server-name and is sent to the server.

$$M_{10} = \{ K_{sess}(\text{authenticator}), K_{serv}(\text{ticket}), \text{destination server} \}$$

11. The server receives the message and decrypts the ticket with the key it shares with the TGS. From that it gets the session key it needs to decrypt the authenticator. The server then performs the same validation procedures that were performed in step 7.

Steps 1-5 are usually performed during the login process, and steps 6-11 are performed every time a user initially asks to use a kerberized service. Once the user has a ticket for a kerberized service, only steps 10-11 need to be done for KRB_SAFE and KRB_PRIV messages. When the ticket for the kerberized service has expired, the user must apply for another ticket. Tickets have a default lifetime of eight hours. On a normal day, users should only perform steps 1-5 once and steps 6-11 will usually be done once a day for every Kerberos-authenticated service that they use.

Timestamps represent an important idea in Kerberos. They are used in authenticators to prove the relative newness of a message and to prevent replay. The topology of the network using Kerberos is divided into realms, with one Kerberos server per realm. A realm is usually the size of a local area network or smaller, but it can be larger. There is only one main TGS server per realm, but there can

be secondary TGS servers (with read-only access) to provide some fault tolerance. When a user or service is checking a timestamp's value, it must be within the time skew for that realm. The time skew's value depends on the diameter of the network, the amount of communication delay, and clock accuracy; the default value is five (5) minutes. Network administrators should adjust this value according to what is needed on their network. So, if a message arrives with a timestamp that is more than *time-skew* minutes off from the system clock, the message is deemed unauthentic and is discarded. A time server is required for Kerberos. The idea of relatively synchronized clocks is vital to Kerberos. Without this, timestamps would not be of much use.

Session and private keys are never transmitted in cleartext; they are always encrypted before transmission. Because of this, only the Kerberos-authenticated service and the user should know the session key they share. The private-session key the user shares with the TGS should be known only by the TGS and the user. The private key shared between the TGS and the Kerberos-authenticated service should be known only by those two entities. Keys are usually stored on the workstation in a secure area where they can be accessed only by the **superuser**. The keys are stored in files with the permission bits set to read/write for the owner, and the owner of the file is **root**. Problems may arise because of the location of the key storage, especially on Kerberos client machines; these problems will be discussed in Section 2.5.

Tickets in Kerberos have lifetimes, which dictate how long they can be used. The default lifetime is eight hours. Making the lifetime larger or smaller has certain tradeoffs. If the lifetime is made larger, the transparency of Kerberos to the user increases, but the chance increases that the key for the ticket is stolen. If the

ticket lifetime is larger, it is more likely that an intruder will break the security of Kerberos. On the other hand, making the ticket lifetimes shorter increases the security of Kerberos and decreases the possibility of break-in. But this also affects the transparency. By making the ticket lifetimes shorter, the user is reminded of the security more often. For example, most users would not be interested in having to obtain new tickets every hour. Tickets for users are stored as files in the `/tmp` directory with the permission bits set to read/write for the owner and the user as the owner. This can be a problem on insecure machines or machines using the Network File System (NFS)¹ to mount the `/tmp` directory. In the latter case, the tickets would be going over the network, so anyone monitoring the network could steal other users' tickets for his/her own use until the tickets expire. In either case, if intruders can break the security on the local machine, they will have access to all tickets.

2.3 Kerberos Utilities

There are many utilities included with Kerberos. These include: *kinit*, *klist*, *kpasswd*, *kadmin*, *kdb_edit*, and Kerberos authenticated (kerberized) versions of Berkeley's *rcp*, *rlogin* and *rsh*.

The *kinit* utility enables the user to get a ticket-granting ticket (TGT). There is an `-i` option which allows the user to specify the username of the TGT; the `-r` option allows the user to specify the realm of the TGT; and the `-l` option specifies the lifetime for the TGT, which can range from 5 to 1275 minutes. After using the *kinit* command, the utility asks users for their password. If the correct password is

¹Sun Microsystems, Inc.

given, the user receives a new TGT. Because the *kinit* program should eventually be incorporated into the login program, only after the TGT expires should the user need be aware of its need to be able to use services. Once the TGT expires, the *kinit* program must be run again.

The *klist* utility allows users to get a list of the tickets they have at the moment. For example, this list includes the user's TGT and tickets for any Kerberos-authenticated (kerberized) services the user has requested, along with the expiration times for those tickets.

The *kpasswd* utility allows users to change their password. The options for *kpasswd* include: -h, for help; -n name, name of the user; -i instance, like **root** or **admin**; -r realm, kerberos realm; and -u username, where username is: name.instance@realm. After the *kpasswd* command is entered, users are asked for their old passwords; if it is correct, they are twice prompted for the new password they have chosen. If the new password is entered correctly two consecutive times, the user's password is changed in Kerbero's database. Since users are asked for their old password first, this should prevent another user from changing their passwords.

The *kadmin* utility is used to make changes to the Kerberos database. If users have the proper privileges and give the proper password, usually called an **admin** password, they can change any users' passwords, add new users, and get information on other users. This utility is meant to be used by system administrators, security officers, and other qualified personnel; it should be available only on the Kerberos server. One noticeable option missing from this utility is the *delete* option. The designers left out the *delete* option, because there are usually access control lists for services scattered all over the realm; if a user is deleted from

the Kerberos database, there would be the possibility that he/she would not be deleted from all access control lists. If another user was granted that same username, he/she would instantly gain access to all of the first user's privileges. To delete a user, the system administrator is required to edit the Kerberos database by using the *kdb_util* utility. In some newer implementations of Kerberos, a delete command is included.

The *kdb_edit* utility is used to enter new users or services into the database. The system administrator can establish the user's or service's password, ticket expiration date, ticket lifetime and attributes. The Kerberos master key must be entered before using this utility. This key is initialized when Kerberos is installed. This utility also should be available only on the Kerberos server. There are many other Kerberos utilities including: *kprop*, *kpropd*, *klogin*, *kstash*, and *kdb_init*.

The kerberized versions of the *rcp*, *rlogin* and *rsh* utilities are much like the usual versions of the utilities, but they have some extra options. The user can specify the realm for the command with the *-k* option; and the *rcp* and *rlogin* utilities have the *-x* option, which causes all communication to be encrypted using DES. One major drawback in using the *rlogin* command is that if the user does not appear in a *.klogin* file on the remote machine, the user's password is passed to the other machine in cleartext.

Another utility, *rkinit*, can be used to obtain a TGT on the remote machine before logging into that machine. It acts like the *kinit* command except that it gets a TGT for a remote machine. When the user already has a TGT on the remote machine, the *rlogin* command does not ask the user for his/her password if the user appears in the file, *.klogin*. The *.klogin* file is much like the *.rhosts* file, but a password must still be entered when using *rkinit* to get a TGT on the remote

machine.

2.4 Kerberos Installation

The Massachusetts Institute of Technology (M.I.T) provides installation and operation instructions for Kerberos [5, 6], so only a brief overview will be supplied. The network administrator must specify the local realm and the name of the TGS server in configuration files before Kerberos can be compiled. The file *krb.conf* contains the realm name and the name of the TGS server that Kerberos should use. This file should be placed in the */etc* directory. Another file, *krb.realms*, may also be needed. After this, the Kerberos source files are compiled. A DES test is performed so the system administrator can be sure that DES is working properly. If no errors occur, the command *make install* will install Kerberos in the proper system directories. Before Kerberos can start running, the *services* file in the */etc* directory must be changed to include entries for all of the Kerberos applications.

The operation instructions provide detailed instructions for the proper running of Kerberos. The system administrator should enter all users into the Kerberos database before starting Kerberos, but more users can be added later. Once this has been done, the Kerberos server is started. The system administrator should then test the Kerberos server by trying to use the *kinit* and *klist* commands. If these work properly, the Kerberos server should be working correctly, and the administration server should be started. The Kerberos and administration servers will usually run on the same machine. The *kadmin* command will not work unless the administration server is running. There are three access control lists that control use of the administration server. The files *admin_acl.mod*, *admin_acl.get*,

and *admin.acl.add* contain the names of users who can modify the database, get information from the database, and add information to the database, respectively. These files are stored on the Kerberos server in the same directory as is the Kerberos database, which is usually the */kerberos* directory. If the *kadmin*, *kpasswd* and *kinit* commands work properly, the administration server should be running correctly. Not every machine in the realm needs to have the Kerberos server installed. The Kerberos server should be installed only on the server machine. All of the other machines in the realm need the basic Kerberos utilities, the *krb.conf* file and the changes to the services file, but not the system administration utilities.

It is easy to add slave servers to the realm. These slave servers are read-only copies of the database and add some reliability to Kerberos. If the main server goes down, Kerberos will continue to run using the slave server, but no changes can be made to the Kerberos database. All slave servers should be listed in the *krb.conf* file on every machine using Kerberos. The Kerberos commands always try to use the Kerberos server; if that fails, the slave servers are tried. Slave servers are updated at fixed intervals, usually every four hours or so. In order for this propagation to occur, *kpropd*, a Kerberos utility program, should be running on the slave servers; the *kprop* command should be executed on the Kerberos server. The file, *krb_slaves*, contains the list of the slave servers, which allows *kprop* to send the database to all of the slave servers. The database can be sent either in cleartext, in cleartext but authenticated, or in encrypted form and authenticated by using command line options. There is one major drawback to the use of slave servers: the Kerberos master key must be installed on each machine that has a slave server.

It is very simple to add new client-server applications. First, add an entry

for the service in the Kerberos database by using *kdb_edit* or *kadmin*. Use the *ext_srvtab* utility to create a *srvtab* file for the server. There should be a *srvtab* file on each machine using Kerberos, and it should contain a list of all the kerberized services that are available on the machine. The new *services* and the *srvtab* files must be installed on all machines, and the client program must be installed on all of the machines that need to use the service. The *srvtab* file should be placed in the */etc* directory. A line for the new service is added, using any standard editor, to the *services* file on the server and on all client machines. On the server, a line must be added to the *inetd.conf* file, and *inetd*, the Internet services daemon, must be restarted on the server machine. The new service should be tested to make sure it works. As always, the user must have a TGT before using any Kerberos-authenticated services.

2.5 Protocol Weaknesses

There are many good things to be said about Kerberos, but there are also some weaknesses in Version 4 of the protocol. Some main weaknesses are listed below. For a more detailed summary, refer to [7, 8].

1. Use of authenticators to prevent replay attacks. The authenticator relies on timestamps to guard against reuse. Because of this, the server must keep a list of recent tickets to prevent replay attacks. There can be a storage problem on some machines.
2. Reliance on unsecure time servers. The system must rely on clocks that are relatively synchronized; therefore, occasionally, there must be communication with a time server. If the workstation can be fooled about what time it

really is, old authenticators could be reused. If timestamps are to be used in authenticators, Kerberos must use a secure time service (Ex: a secure version of the network time protocol (ntp)).

3. Password attacks. An intruder can record login attempts by users and try to guess passwords. If intruders get a copy of the encrypted message returned by Kerberos to the login program, they will know when they guess the correct password if part of the encrypted message should become readable. Users should have to prove their identity before the encrypted password is sent across the network.
4. Spoofing login. Kerberos has no protection against trojan horse login programs. If intruders start a trojan horse login program to record passwords, they could to become any of the password owners, because they would know users' passwords. Password security is still very important.
5. Ticket Forwarding: the *cascading trust* problem. For example, say realm A wants to use realm C, but it has to get a TGT from realm B before getting to realm C. Realm A trusts realm C, but doesn't trust realm B, and realm B is the only path to realm C. If the local realm does not trust one of the realms in the path to the desired realm or vice-versa, the user is out of luck.
6. Inter-realm authentication. Assume that the user wants to use a service in another realm. The request is sent to the originator's TGS, which then passes it along to another realm, and so on, until the proper realm is reached. That realm must decide whether to accept such a ticket. The biggest problem is deciding which realm should be its next hop. Each realm must have a private key for every other realm, too. This requires $O(n^2)$ keys to interconnect n

realms. This can get expensive very quickly.

7. Using Kerberos on multi-user hosts. There must be a way to save private and session keys. If workstations are single-user and remote login is disabled, this is not a problem; but it is otherwise. The keys should be stored on some external device and accessed or changed when needed.
8. Dependence on DES for encryption. DES cannot be exported from the U.S., so this severely limits the spread of Kerberos. Other types of encryption should be supported so the use of Kerberos can grow internationally.
9. Dependence on TCP/IP. All of the addresses in tickets and authenticators must be Internet addresses, and only one address can be specified. Problems arise when administrators try to use Kerberos on *multi-homed* hosts, which have more than one address. There should be some flexibility in the addressing scheme, and it should be possible to list more than one address for a host if it is multi-homed.
10. No proxy tickets. There is no way to forward a user's credentials from one host to another host and have the ticket be used by another user. What if an intermediate service needs to use another service to fulfill the user's command? The intermediate service would need the rights of the client to perform the other service. For example, the print service may need access to the file service to print a file for the user.

2.6 Version 5

A beta version of Kerberos Version 5 was released in late 1991 [9], so a brief discussion follows. Only changes or additions occurring in Version 5 will be provided.

1. Address flexibility. More than one address is allowed in tickets and authenticators, and these addresses do not have to be Internet addresses. Addresses are tagged with a type and length field to identify different types of addresses. This should allow the use of Kerberos in other environments. It also solves the multi-homed host problem.
2. Message ordering. There was a problem with message byte ordering in Version 4 because the *receiver makes right* philosophy was used. The sender transmitted the message in its own byte order, and it was up to the receiver to change the byte order if necessary. For example, if both the sender and receiver were *big-endian* or *little-endian* machines, there was not any problem. Version 5 uses *Abstract Syntax Notation One* (ASN.1) [10] to encode all messages. The use of ASN.1 is more prevalent in the OSI community, but is also used in the Internet community. ASN.1 provides a consistent ordering for messages over the wire, so messages should always be transmitted in the same format.
3. More efficient inter-realm authentication. Only $O(\log(n))$ keys are needed for n nodes because each realm does not have a key for every other realm in the network. There can be short-cut spanning links between nodes to cut down on the number of keys necessary for inter-realm authentication. The

short-cut spanning links are usually put in if there is heavy traffic between two realms. If the short-cut spanning links are not used, the *cascading trust* problem still exists. If the inter-realm authentication is approved, the ticket will contain an enumeration of all of the realms traveled to get to the desired realm.

4. Encryption flexibility. Other encryption algorithms besides DES may be used. For those people that use DES, it is recommended that a cryptographic checksum be used in addition to DES, which should make it harder to infiltrate. There are also more choices for the cryptographic checksum, including: CRC-32, DES's message authentication code (MAC), and MD4 [11]. CRC-32 is not keyed (perturbed by an encryption key) or collision proof, DES's MAC is keyed and collision proof, and MD4 is collision proof, but not keyed. Plaintext attacks are much harder to execute. Allowing other encryption algorithms permits Kerberos to be used in other countries.
5. Sequence numbers. Version 5 allows the option of using sequence numbers on messages instead of using timestamps. If the server chooses this option, it does not have to keep a list of recent tickets to prevent replay. All messages must come in order by sequence number. The server only needs to keep track of the sequence number for the last authentic message. If the sequence number for the next message is less than the last sequence number or is greater than the last sequence number plus one, the message can be thrown away.
6. More system independence. All system-dependent routines in Version 5 are kept in one library to keep their differences to a minimum. This allows other

environments to use Kerberos more easily. Sun, IBM, DEC, and Hewlett-Packard have versions of Kerberos. OSF/DCE, Transarc AFS and FTP software also have products that use Kerberos. There are *kerberized* versions of *ftp* and *telnet*, along with *lpr*, *POP* (post office protocol), *mh* (another mail service) and many others.

7. Renewable tickets. Tickets now have two times associated with them. The first is a renewable ticket time, and the second is the absolute last time for using this ticket. If the ticket is renewed before the first time passes, the user gets the ticket renewed for another increment of time. If the first time passes without renewing the ticket, the user must get a new ticket. The second time is the final deadline for this ticket. Once this time is reached, the user must get another ticket by using the *kinit* utility.
8. Session key negotiation. The service and user can negotiate for a session key instead of just using the one provided by the TGS. This should make Kerberos even safer.
9. Use of a challenge/response protocol. Kerberos has an option that requires the user to be authenticated before he/she enters the password. Because of this, Kerberos doesn't send the message encrypted by the password to the login program without authenticating the user first. This can be accomplished by using a hand-held authenticator to enter a code in response to the challenge. The key used to encrypt/decrypt the message could be altered using this code in addition to the password.
10. Postdated tickets. Postdated tickets are usually used for batch processing. The time associated with the ticket is set for some time in the future, pre-

sumably when the batch job should be processed, and the ticket is marked invalid. The ticket will not be valid until the specified time arrives and the ticket is presented to the TGS for a validated ticket.

11. Proxy tickets are allowed for service tickets, but not for TGTs.

CHAPTER 3

DEC's SES

This chapter provides a summary of DEC's Security Encryption System [12, 13, 14, 15, 16], which is composed of the Digital Ethernet Secure Network Controller (DESNK) and the VAX Key Distribution Center (VAX KDC).

This Security Encryption System allows businesses that buy the system to have a security-enhanced Ethernet local area network (LAN). Multiple nodes on the network are allowed to share a cable plant with the knowledge that only those nodes having the correct access codes can access secret or sensitive data. As stated earlier, this is the first DEC product that uses cryptography.

The Digital Ethernet Secure Network Controller and the VAX Key Distribution Center can give the customer better security and integrity on their LANs. The system is flexible enough to support either a single segment, a multiple segment or an extended Ethernet LAN as defined by the "DECconnect System Planning and Configuration Guide." [14]

3.1 DESNC Controller

The DESNC controller is a store-and-forward cryptographic controller that operates as a separate hardware unit on the network. It encrypts and decrypts

Ethernet frames placed on LANs.

There are three different types of communication allowed by the VAX KDC if the nodes have the same access class sets. Nodes connected to DESNC controllers are referred to as *client nodes*, while nodes connected only to the Ethernet are called *nodes*. Communication is possible between two client nodes, a client node and another node on the network, or two nodes not connected to DESNC controllers. Messages exchanged between two client nodes are encrypted because both of the nodes are connected to DESNC controllers. The other two possible types of communication are not encrypted because both nodes do not have DESNC controllers to encrypt or decrypt the messages.

The controller is installed on the transceiver interfaces between the client nodes (e.g., workstations, servers, and processors) and the LAN cable plant [14]. There are four ThinWire Ethernet ports on the DESNC controller to which the client nodes can connect. One to twenty client nodes can be distributed among the four ports of each DESNC controller. Multiple nodes using the same port are on "client-port subnetworks."

3.1.1 Operation of the DESNC Controller

The DESNC controller uses cryptography to protect messages if required; it also makes sure that the integrity of the messages is not violated.

The DESNC controller acts as a go-between for the client nodes and the Ethernet. It can receive requests for sending messages from the client nodes or receive messages from the Ethernet destined for one of the client nodes connected to it. Messages sent by client nodes are broken into Ethernet frames. If a client node

wants to send a message, the DESNC controller first notifies the VAX KDC about the request to make sure the client node is allowed to send a message to the destination it has requested. If the transmission of the message is cleared by the KDC, the DESNC controller accepts the message from the client node, encrypts it frame by frame, and sends it to the destination via the Ethernet. If the DESNC controller receives frames via the network, it decrypts the frames and sends the plaintext frames to the client node that was to receive the message.

The DESNC controller uses the Data Encryption Standard (DES) algorithm to encrypt the frames. The encryption keys used in the encryption and decryption processes are received from the VAX KDC or from a key supply that the business provides. The controller takes the plaintext message that is to be sent, along with the sixty-four bit long cryptographic key, as inputs, and then encrypts the message, producing the encrypted message, which is called ciphertext. When the DESNC controller performs the decryption process, the message is converted back to its plaintext form. A model of the encryption/decryption process is shown in Figure 2.

3.1.2 Security and Network Management Functions

Other security and network management functions that the DESNC controller provides are:

1. Authentication of client nodes before transmitting messages.
2. Communication with the VAX KDC management functions.
3. Making sure that the network access policy for nodes is always used.

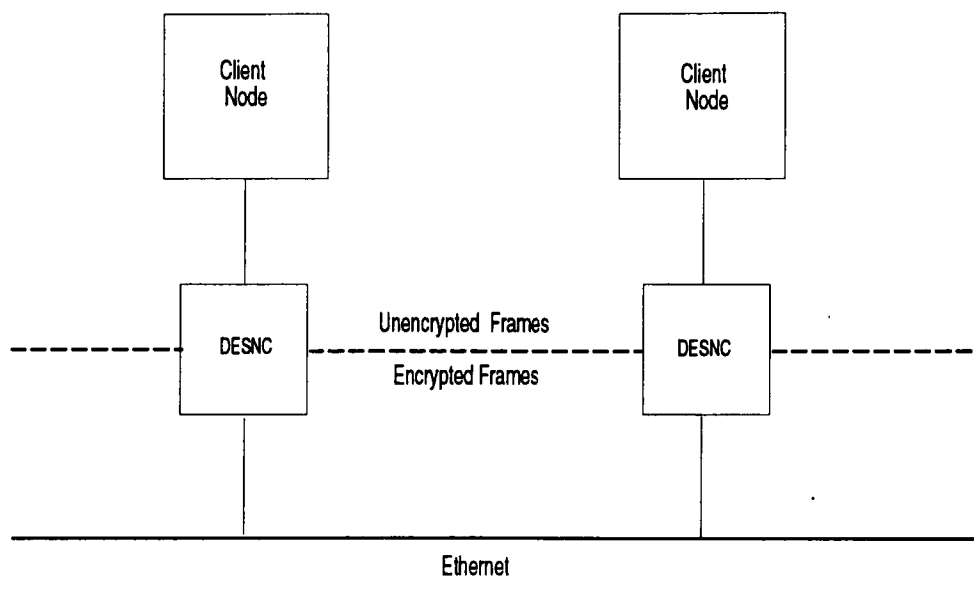


Figure 2: Encryption and decryption process

Source: Sales Update/Ethernet Enhanced Security System. Digital Equipment Corporation, 1987.

The access policy will be presented in more detail in Section 3.2. *Authentication* is the process of positively identifying who sent the message and assuring that the message has not been changed or modified. A model of a LAN using the Security Encryption System is shown in Figure 3.

Auditing and alarms can be used to monitor for possible security problems that could occur on the LAN. For more information, please refer to [16].

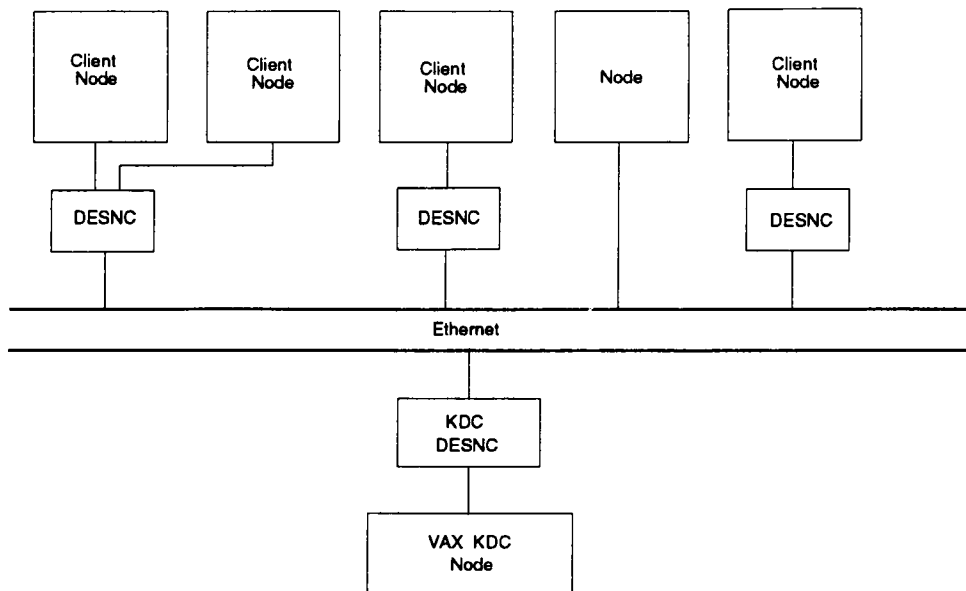


Figure 3: An example of an Ethernet LAN using SES

3.1.3 VAX Key Distribution Center System (VAX KDC System)

The VAX KDC, like the DESNC controller, is part of the LAN. The VAX KDC manages all the DESNC controllers that are part of the LAN. The system parcels out the cryptographic keys to the DESNC controllers and determines whether the two nodes are authorized to communicate. Communication is possible between two client nodes, between a client node and another node on the LAN, or between two nodes on the network. It is the VAX KDC system's job to decide whether these communications are allowed to occur.

The VAX KDC system has the following components [14]:

1. A VAX processor running the VMS Operating System (The VAX KDC

node).

2. The system's layered software running under VMS, which controls the DESNCs on the network.
3. DESNC controllers configured to use a VAX KDC node.

The system provides additional tools to help the LAN managers implement and control their LAN, such as to:

1. Determine the configuration of the LAN that will be best for security reasons.
2. Monitor the operational and security status of the LAN.
3. Control the DESNC controllers on the LAN.
4. Audit alarmed events that occur on the LAN.
5. Establish and enforce the network access control policy.

The security encryption system has many features that benefit the customer. Some of these will be discussed in Chapter 6. These features greatly increase the security of the network.

3.2 Setting Up the Security Encryption System

As expected, it is the purchaser's job to set up the system, so the network manager usually ends up developing the network configuration, establishing the access policy and initializing the DESNC controllers.

3.2.1 Network Configuration

Besides establishing the configuration of the LAN, it is the LAN manager's job to determine which nodes can talk to each other before the access control policy for the LAN can be established. There are also configuration rules that must be followed for the DESNC controllers, such as the following: [13]

1. There can be no more than two DESNC controllers between two communicating hosts.
2. There can be no more than two DESNC controllers and three bridges between the DECserver and its host.
3. Rules governing transceiver cable length must be followed.
4. No T-connectors are allowed on the DESNC client ports.
5. No repeaters are allowed on the DESNC ThinWire.

The main reason for rules one, two and three is the additional delays that are introduced by the using the DESNC controllers. The service would not be at an acceptable level if any more DESNC controllers or bridges were inserted between any two nodes or if the cable length between the nodes was too large.

When determining the configuration of the network, the LAN manager needs to think about which nodes on the network can talk to each other. It is usually a good idea to attach all nodes that work with sensitive information to a DESNC controller so that any sensitive information they send can be encrypted. Nodes that are not likely to transmit or receive sensitive information should not be attached to a DESNC controller.

3.2.2 Access Control

The main reasons for having an access control policy included in the security encryption system are to protect and restrict the communication between nodes; thus, only nodes that are designated to receive sensitive information actually receive it. Any message deemed sensitive enough to be encrypted can be sent only to nodes that are attached to a DESNC and that have the proper access-control classification.

Nodes on the LAN can be divided into classes. Only nodes with the same class can communicate. Dividing nodes into classes allow the LAN manager to define work-related groups of users whose data and information must be protected against other users' attempts to read or alter the information [15].

Each node on the network is given either a single access class (security level) or many access classes, called an *access class set*. Only nodes that have the same *access class set* can communicate with each other. If one node had an *access class set* of {7,11} it would not be able to communicate with another node that had an *access class set* of {7}, even though the two nodes had the access class, 7, in common. The reasoning behind this decision is that nodes that are trusted only with access class 7 must also have an access class of 11 if they are to be trusted with the information involved with access class 11 work. The nodes are not trusted with this information, so they only have an access class set of {7}.

3.2.3 Initialization

Once the configuration of the network and the access control policy have been worked out, the next step in setting up the system is that of initializing the network. This process must be done before any client nodes can communicate over the Ethernet. No encryption can take place until this process is completed. Initialization of the network involves the following steps [15]:

1. The LAN manager starts the initialization process at the VAX KDC node by asking for a printout of a list of the initialization key variables and for the address for each of the DESNC controllers.
2. Each VAX KDC node needs its own DESNC controller to generate the initialization key list.
3. The following steps must then be taken before actual initialization:
 - (a) The initialization key variable is entered into the controller using the DESNC's keypad.
 - (b) The physical key is inserted into the DESNC and turned on.
4. This process is repeated for every DESNC controller on the network. Should the DESNC controllers be turned off or have their power interrupted, they will retain their initialization information for at least seventy-two hours.

3.2.4 Key Distribution and Management

Key distribution is performed automatically by the VAX KDC node. Each node pair that is allowed to communicate (i.e., because they have the same access

class set) is given a unique association key by the VAX KDC to be used when communicating.

Key Distribution Process

There are seven steps involved in the key distribution process. These steps are performed each time a node requests communication with another node. Figure 4 illustrates the key distribution process, or what is commonly called *Establishing an Association*. (1) The client node sends a message to its DESNC controller requesting communication with another node. (2) The DESNC controller checks the addresses in the frame, authenticates the source address, and then sends a message to the VAX KDC asking for authorization for the connection with the node designated as the destination address. (3-5) If the nodes have the same access class set, the VAX KDC will send a unique association key in an encrypted packet to the DESNC controllers for the source and destination nodes. (6) If the communication is allowed, the message will be encrypted by the source DESNC controller, frame by frame, and forwarded onto the Ethernet. The association key is involved in the encryption process. (7) When the frames start arriving at the destination node's DESNC controller, the frames are decrypted using the association key and are then sent to the destination node.

These association keys are stored by the DESNC controllers for a period ranging from one hour to eight days. Cipher block chaining is used to encrypt the frames. Cipher block chaining combines the previous block of ciphertext with the current block of plaintext to provide, with the association key, the current block of ciphertext. Cipher block chaining provides additional protection against data manipulation.

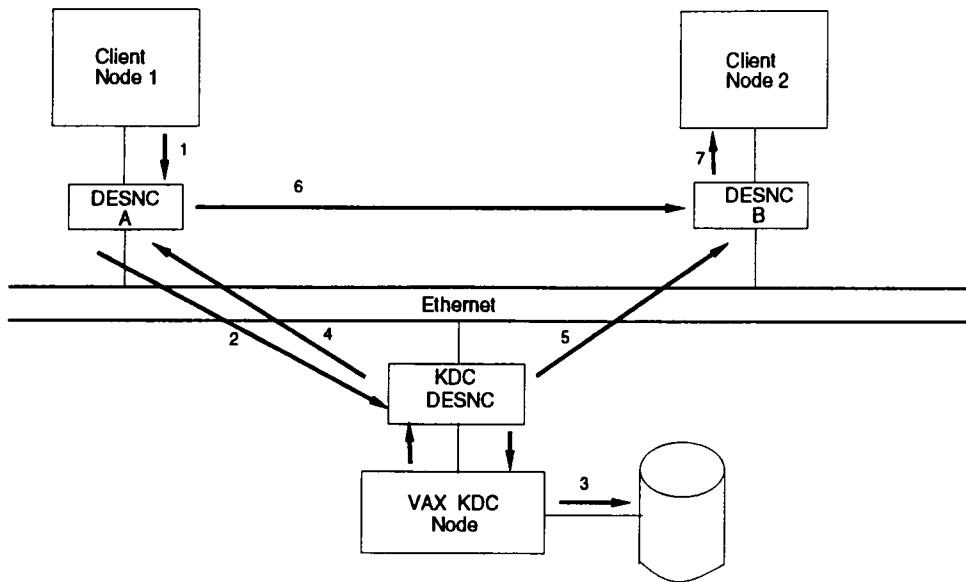


Figure 4: Key Distribution Process

Source: Ethernet Encryption: Digital Ethernet Secure Network Controller - (DESNC) and Key Distribution Center (KDC). Digital Equipment Corporation, 1987.

Encryption Keys Used by the VAX KDC and the DESNC Controllers

There are three different keys that are used to help encrypt frames sent over the network: (1) the master key, (2) the key generation key, and (3) the key generation seed. If any of these keys are lost, all DESNC controllers on the network must be reinitialized.

If the master key for the system is lost or thought to have been compromised, all encryption keys must be replaced. A new master key is then entered, and the initialization process is repeated. The key generation key works with the key

generation seed to generate the encryption keys used by the VAX KDC. Each VAX KDC on a network can use the same key generation key or totally different ones. If the key generation key is lost or thought to be stolen, the LAN manager must enter a new key generation key into each VAX KDC that used it, and the initialization process must be repeated. Each VAX KDC must have a unique key generation seed. If the seed is lost or thought to be stolen, only a new seed need be entered into the VAX KDC that used it. The initialization process is then repeated.

3.2.5 Restrictions

The LAN manager can apply additional restrictions besides the access control policy on both the nodes and the DESNC controller.

Restrictions on Nodes Attached to DESNC Controllers

There are essentially three other options that can be placed on the client nodes. The first option is whether the nodes can send frames to multiple destinations (i.e., send *broadcast messages*). Not allowing a node this option can affect the operation of the LAN, because the node could not use DECNET¹. The second option for a node is whether the node can communicate with nodes not attached to DESNC controllers. This is usually taken care of by the access control policy. The third option is whether a node can send or receive frames while its DESNC controller is in bypass mode. When the DESNC controller is in bypass mode, no encryption can be performed; the LAN manager would not want hosts to receive sensitive information during that time.

¹Digital Equipment Corporation

Modes on DESNC Controllers

There are three possible modes of operation for the DESNC controller; these modes are controlled by the VAX KDC. The modes are *regular operation* mode, *standby* mode and *bypass* mode. If a DESNC controller is in *standby* mode, it cannot process anything until the VAX KDC sends it a message approving the processing of messages. If all VAX KDC nodes on the network go down, operations on the network can continue, but only by sending cleartext messages. When this happens, the DESNC controller is in *bypass* mode; all DESNC controllers must be in this mode to restore normal communication.

When one VAX KDC comes back up, all of the DESNC controllers would return to their *regular operation* mode because the VAX KDC will start answering their queries. If a VAX KDC goes down on a network that has more than one VAX KDC functioning, another functioning VAX KDC node will answer the queries of the DESNCs assigned to the malfunctioning VAX KDC node until their VAX KDC comes up again.

3.3 Performance of the Security Encryption System

The performance of the Security Encryption System depends upon the performance of the DESNC controller and the VAX KDC. The performance of the DESNC controller, in terms of throughput (Mbps), increases linearly in proportion to the size of the packets. This is controlled by the access method (Ethernet). A graphic representation of the performance of the DESNC controllers is shown in Figure 5. The performance of the VAX KDC is heavily dependent on other factors such as CPU performance, the number of nodes attached to DESNC controllers,

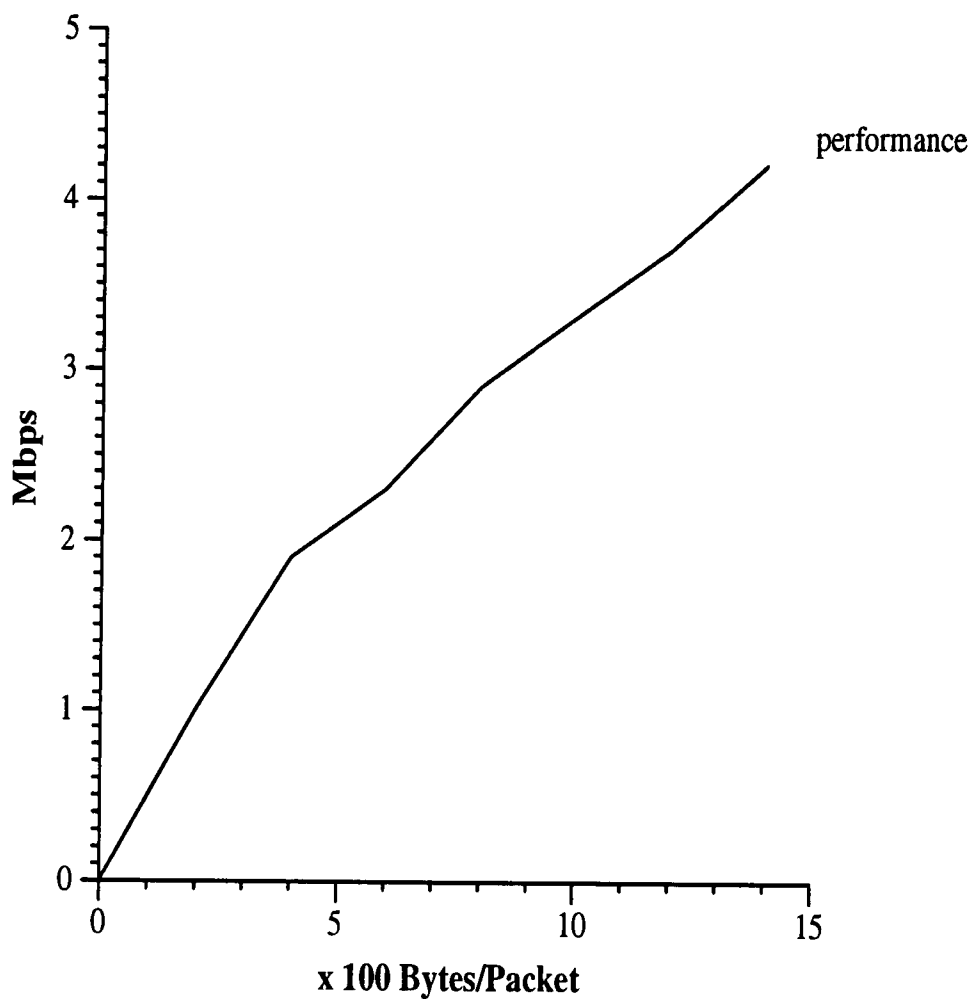


Figure 5: DESNC Performance

Source: Ethernet Encryption: Digital Ethernet Secure Network Controller: (DESN) and Key Distribution Center (KDC). Digital Equipment Corporation, 1987.

and the pattern of use on the network [15].

From the information available, the Security Encryption System provides increased security to networks that use it. The access control policy is a good way to control communications. Having three keys used in the encryption process, in addition to the DES algorithm for encryption, will make it difficult for intruders to read messages going across the network. The keys can always be changed if the system is thought to be compromised.

CHAPTER 4

Other Security Products

This chapter examines two other security products that are currently being used. The products that are examined in detail are Xerox's Encryption Unit (XEU) [17, 18, 19] and BLACKER [20]. This chapter provides a summary of how each of these products work.

4.1 Xerox Encryption Unit

The Xerox Encryption Unit (XEU) is an encryption unit that helps to protect classified information being transmitted over a network. The XEU unit resides between a processor and the network to which it is connected and is installed in the drop cable between the computer and the network. The XEUs help in securing communication channels between processors. As of 1989, the National Security Agency (NSA) was the only agency that could authorize the sale of XEUs [18]. Any company wanting to buy XEUs must have a NSA-assigned Communications Security (COMSEC) account. As with the Data Encryption Standard (DES), XEUs cannot be exported from the United States.

The XEU is mainly used to protect secret information for different companies and groups working for the U.S. government. The XEU can allow multiple levels of security. It can encrypt information up to the TOP SECRET level of classification,

and all levels (from unclassified to TOP SECRET) can be sent over the same transmission cable when encrypted.

The XEU can be used with networks and equipment that comply with the IEEE 802.3 standard. Once the XEU is initialized, the user should not notice the operation of the unit, because its operation is transparent to all application-layer protocols.

4.1.1 Functional Description

The XEU itself is a small device, measuring 3.25" x 9.5" x 11.5", and weighing only ten pounds [18]. The unit has two primary circuit boards, which hold the encryption chip, key storage, and LAN interface and control functions. The function of the encryption chip is to encrypt and decrypt the keys that are stored in the XEU key table.

The XEU is a special-purpose Media Access Control (MAC) level bridge that encrypts and decrypts the data portion of the IEEE 802.3 packet [18], operating in the datalink layer of the OSI seven layer model [21]. The data field in the packet may increase by fourteen to twenty-one bytes because of the encryption [18]. This increase suggests some extra headers in the data part of the packet. Manual switches on top of the unit permit selective control of its operation. The *Standby switch* makes the XEU stop processing packets; nothing will be encrypted, decrypted, sent or received after this switch has been turned on. The *Zeroize switch* will cause all configuration information to be erased; after this switch has been pressed, the XEU must be reinitialized to process packets again. The *Reset switch* will make the XEU start an internal diagnostics check.

Configuring the XEU is very simple. A physical object known as a *Xerox key-fill device* is used to load configuration information at initialization time. The key-fill device works much like a normal *door-locking* key in that it is inserted into a key-fill port in the front of the XEU. Each key-fill device contains a microprocessor, memory and interface circuitry [17]. After completion, each user will have a key-fill device that allows network communication. A Key Management Loader (KML) [18] loads information into the key-fill devices. The KML consists of menu-driven software that initializes the keys with their values. Each KML uses a PC-AT workstation with one floppy disk drive and one rigid disk drive [17]. One KML is sufficient for any number of local area networks (LANs).

There are many keys that can be used with a XEU. One set of these keys is composed of the *black keys*. These keys are used for the network side of the XEU to help with the configuration, encryption and decryption. Each XEU needs a *Crypto Ignition key* (CIK), because it enables the unit to encrypt and decrypt packets. Classified machines can communicate only if they have a XEU and a CIK. The *Initialization Key* configures the XEU at boot time. The *Black Key Fill Device* (BKFD) is also used at boot-up time to transfer network topology and black key information to the XEU. The XEU also may have a *Bypass key*, which allows packets to be sent or received by the XEU without any encryption or decryption being performed.

The NSA states that all companies wanting to use XEUs must order four red key-fill devices so that the XEU can support key management. The red keys are also used during initialization. After initialization, the information from the red keys is stored in encrypted form in the XEU key table in the XEU. The user needs a single key-fill device (the CIK) to activate the XEU, which enables the XEU to

encrypt and decrypt packets.

4.1.2 Features and Benefits

There are many advantages to the use of XEUs on a LAN. The throughput is 200 packets per second (600 bytes/packet) when the XEU is in send or receive mode [18]. The operation of the XEU is transparent to users and the processor to which it is connected, unless the unit crashes. Unclassified and classified users can share the same Ethernet cable, though they can't communicate with each other when the XEUs are in regular mode. The XEUs can accommodate up to one thousand protected network connections, which should be more than enough for most LANs [18]. It can also serve as a secure gateway between unclassified and classified subnetworks. A XEU can operate in *single host mode*, where it can be connected to one computer and control that computer's communication with that computer, or in *multiple host mode*, where it can be connected to more than one host and control their communication.

A big advantage of using XEUs is that only the data portion of the IEEE 802.3 packet is encrypted; the data-link and physical layer headers and checksums are left in plaintext form. This makes it much easier when transporting packets across bridges. A problem can arise if any intermediate device on the network (e.g., a gateway), wants to access the IP portion of the Ethernet packet. In this case, the device must have its own XEU to decrypt and reencrypt that part of the packet.

Should a XEU be damaged and require maintenance, it must be sent to a NSA-approved facility. The network manager should have extra XEUs for such emergencies, so the backup XEUs can be used while the damaged XEU is sent to

the approved facility to be repaired.

The configuration features need to be determined by the network manager before a XEU is ever used, because this information must be included in the Initialization key that is used when the XEU is turned on. If managers wish to have the XEU process packets from many network connections, they must so specify when ordering the XEU's. This mode of operation is the previously mentioned *Multiple Host Mode Option*. The *Diagnostic Loopback* feature allows the manager to test the operation of the XEU. Essentially, a packet is sent to the XEU, encrypted, sent out of the black output of the XEU, looped back into the black input end of the XEU, decrypted and sent back to the processor.

Another very interesting feature is called *Red and/or Black Interface jamming*. If this option is used, the XEU will cause collisions to occur when packets start coming in too fast, requiring retransmission of the packets. This is the XEU's form of flow control. This option allows XEU's to interact with faster, higher-performance equipment.

4.1.3 Customer Testing

Once the system manager has determined the network configuration for his/her LAN, he/she must perform a test to determine whether the XEU's can send and receive information correctly while operating in the cipher and bypass modes. Having repeaters and bridges in the local area network will not affect the operation of the XEU's. If the system manager would like to have a XEU operate in Multiple Host Mode (Community Mode), the test scenario can be changed accordingly.

An example of a network using XEU's is shown in Figure 6. The testing should

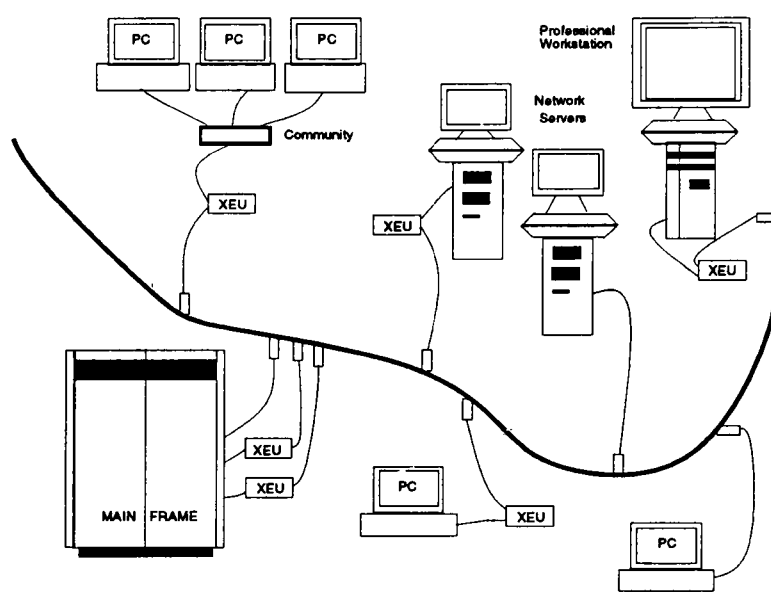


Figure 6: Typical Network Utilizing Xerox Encryption Units

Source: Xerox Corporation. Xerox Encryption Unit: Technical Product Description. Xerox Corporation, VA, 1989. Draft.

cover the following areas:

- Hosts with XEU's should be able to communicate with other devices that have XEU's.
- The XEU should compartmentalize traffic so that communication is allowed only among authorized stations.
- Network communication is unreadable by unauthorized hosts.
- Hosts without XEU's should still be able to communicate in plaintext.
- In event of an emergency, XEU's can be *zeroized*.

- The operation of the XEU is transparent to the user.

Initialization

When the system manager is designing the network, the following steps are required.

- Design a network layout that illustrates all of the communicating stations on the network.
- Make a list of the Ethernet addresses of all stations and determine whether they can communicate.
- Decide whether any stations need to send or receive multicast or broadcast messages.
- Fill out the keying matrix, which determines who can talk to whom.
- Initialize all keys before the test starts.

The network in Figure 6 has at least two virtual networks sharing a physical network. One virtual network's communication is protected by using XEUs, while the other virtual network's communication is not. Hosts using XEUs can send unencrypted information by having the XEU operate in bypass mode, but this mode should not be used by sensitive hosts.

Testing

The first test should be that of bringing up the XEU that is attached to the server. The XEU should be put in bypass mode. The server and hosts should still be able to communicate with each other. A graphic representation of this test is shown

in Figure 7. A network analyzer can be used to examine the traffic between the server and host.

The manager should then bring up a XEU in bypass mode for a host that was communicating with the server. The two should still be able to communicate. Once it has been determined that the two devices can communicate, the manager should switch both XEUs to cipher mode. If communication continues normally between the host and the server, this should show that the encryption and decryption done by the XEUs is working correctly. If the two stations are not able to communicate, the keys were initialized incorrectly.

A XEU will not allow any packets on the network that do not have an authorized address. If communication occurred normally, some performance tests can be done. Sending both small and large messages should be attempted. Small messages should not affect performance very much, but large messages may take more time because of the transmission delays incurred when the XEU must communicate with other network communication protocols [19].

To make sure that network traffic is being compartmentalized correctly, data should be sent from a host with a XEU to one that does not have one, and vice versa. Communication should not be allowed between the two hosts. The XEU will not send information to its host if the message did not arrive in encrypted form (or it was decrypted incorrectly) even though it may be authorized by the KML and the message had the correct address in the header. More hosts connected to XEUs should be brought up. Their XEUs should be placed in cipher mode, and an attempt should be made to send mail between hosts. This should prove that network routing between hosts and servers still functions correctly.

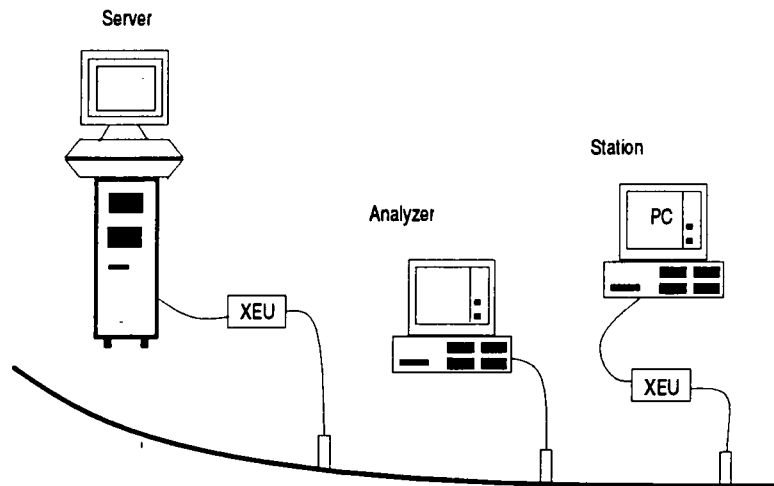


Figure 7: Test scenario #1

Source: Xerox Corporation. Xerox Encryption Unit: Customer Testing. Xerox Corporation, VA, 1989. Draft.

XEUs are supposed to be able to preserve keying information during short power outages and power spikes, because there is a back-up battery inside each unit. This capability should be tested to make sure it works correctly.

Bring up two hosts that are not connected to XEUs and try sending mail between the two hosts. Communication should still work normally. There is one last test that should be performed for the network with no bridges and repeaters. *Zeroize* one of the XEUs and have the host connected to that XEU try to communicate. This should prove that a host connected to a *zeroized* XEU cannot communicate with any host until the XEU has been reinitialized.

Once the elementary tests have been completed, the system manager should

try adding bridges or repeaters onto the network to show that packets can be transmitted through these devices without having to be decrypted (see Figure 8). This test should show that the Ethernet addresses of the packets are not encrypted and that bridges and repeaters will function as if the XEUs were not being used. If there are any gateways on the LAN, the gateways must have XEUs to decrypt the packets so that essential IP (routing) information (e.g., IP addresses) is available.

If the system manager has a *Community Mode* XEU to be used for a group of stations, another test should be conducted. An example network is shown in Figure 9. The test should show that all the stations in the community can communicate with each other and that a *Community Mode* XEU can handle packets from multiple hosts that have different Ethernet addresses.

4.2 Blacker

Blacker is another security product that has recently gone into production. It is a security system that allows the Defense Communications Agency (DCA) to interconnect three intelligence networks having different security classifications and to allow them to communicate as if they were on one virtual network [22]. The production of Blacker was started in 1982, but, according to Ronald Eliot, technical director of the Defense Intelligence Agency's Intelligence Communications Architecture Project [22], "it has taken until 1989 for Blacker to reach maturity." Unisys Corporation developed Blacker's software and hardware for the DCA [22].

Blacker provides end-to-end encryption between stations operating on a packet-switched network. As of late 1989, the National Computer Security Center was evaluating Blacker for an A1 security rating, which is the highest security rating

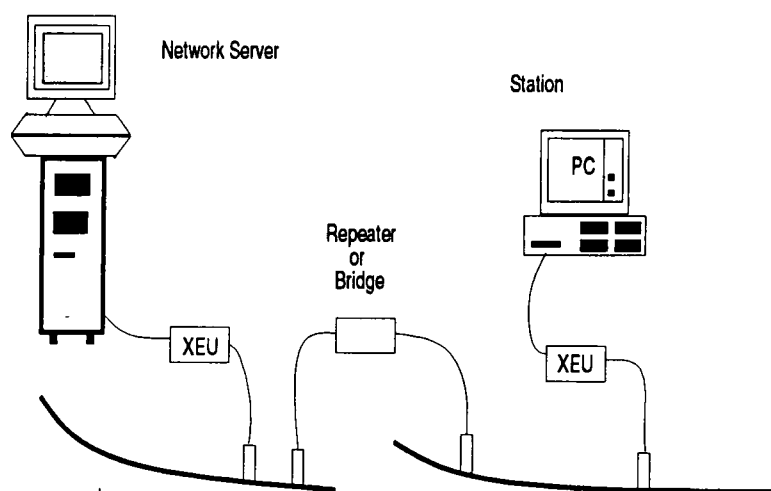


Figure 8: Test scenario #2

Source: Xerox Corporation. Xerox Encryption Unit: Customer Testing. Xerox Corporation, VA, 1989. Draft.

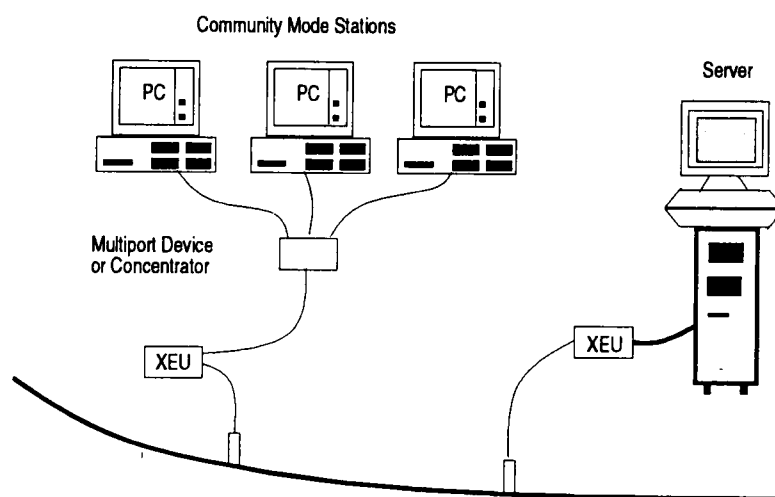


Figure 9: Test scenario #3

Source: Xerox Corporation. Xerox Encryption Unit: Customer Testing. Xerox Corporation, VA, 1989. Draft.

available, according to the Department of Defense's Trusted Computer Criteria [22]. Only the data portion of the message to be sent is encrypted, while the header, which contains the destination, is left unencrypted. Blacker was developed with the Defense Data Network (DDN) in mind, but it can be used in other government networks. Eliot said that Blacker could be added to the Department of State's network or to the Federal Telecommunications System 2000 [22].

4.2.1 Blacker Front End

A Blacker Front End (BFE) can be used as an interface between the DDN and an unclassified host or gateway. An example of this is shown in Figure 10. The BFE is used as the Data Communication Equipment (DCE) of a X.25 network to the host that is attached to it [20]. Because the BFE has security added to it, the host interface must have some requirements added to itself above the data link layer. So, much like the XEU, the BFE operates at the data link layer in the OSI seven layer model. The DDN interface on the BFE is used only as a local interface. Because of this, the standard end-to-end services of X.25 are not available. Parameters for the BFE are loaded using the BLACKER Initialization Carrier (BIC) [20]. Some of the parameters loaded are the Access Control Center (ACC) and Key Distribution Center (KDC) identification; security level; and site identification [20]. As with the XEU, the BFE is initialized when it is booted, and it is initialized again only when the BFE must be reset, zeroized, or has lost power.

Red-Side Host Interface

The Red-Side interface is modeled after the X.25 interface, which was developed in 1983 by the DDN [20]. The Internet Protocol must be used at the network layer.

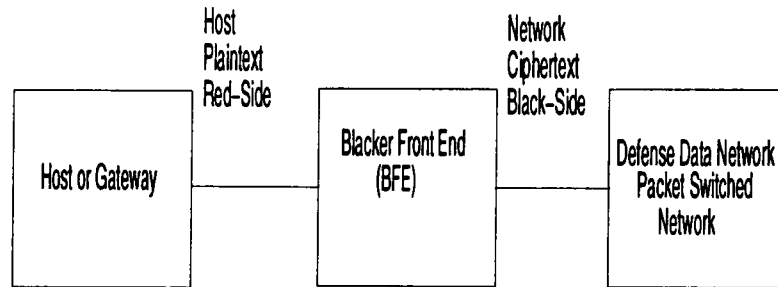


Figure 10: Example of a network using a Blacker Front End

Source: Blacker Interface Control Document. Defense Communications Agency, 1989.

For a more detailed summary of the BFE see [20].

Physical Level

The BFE will run at speeds of 1.2 to 64 kilobits per second, and will allow only full duplex synchronous operation [20].

Link Level

A Western Digital WD2512 Packet Network Interface Chip is used for the HDLC-LAPB interface in the BFE [20]. According to [20], "The chip controls bit-oriented, full duplex serial communications on its Level 1/Level 2 interface side."

Packet Level

On the host interface, only DDN's "Standard Service" X.25 is available. As far as addressing goes, only physical addressing is allowed. All BFE ports receive their

own physical address from the DCA. All addresses are twelve binary coded decimal (BCD) digits, and subnet addresses are not allowed. Since a host must use IP at the network level, a DTE must use the value defined for IP, 11001100 binary, as the only byte in the Call User Data Field of the CALL REQUEST packet [20]. Any deviation from this format will result in an error. A BFE can operate with the maximum packet size of either 128, 256, 512, or 1024 bytes (octets), but for BFE's operating at multiple security levels, 1024 octets must be used as the maximum packet size. The recommended maximum packet size to permit an IP datagram to fit into one packet is also 1024 octets [20]. In a complete packet sequence, the maximum number of data bits allowed is 896 octets [20]. Again, only physical addressing is supported. Logical addressing is not allowed.

Any device that is connected to the BFE will have a 32-bit IP address, which will contain two parts. The first part defines a network and the second is network specific. For the DDN Red Virtual Network (DDN-RVN), the first part is eight bits, which makes the network a class A network. The format of the IP address is shown in Figure 11. The port field is used to control routing for the BFE, and is three bits long. The domain number of the BFE and the BFE ID, which is its number within its domain, are ten bits each. To see the format of X.25 addresses for devices attached to the DDN-RVN, see [20].

The Blacker system is quite flexible, because it can operate even when the KDC and the ACCs are down. This mode of operation is called *Emergency mode* in which the BFEs can do one of three things: (1) refuse to communicate with other BFEs in Emergency mode, (2) make the transition to Emergency mode and let the host know about the transition, or (3) notify the host that the BFE could go into Emergency mode if the host wishes it. If the host's BFE is in Emergency

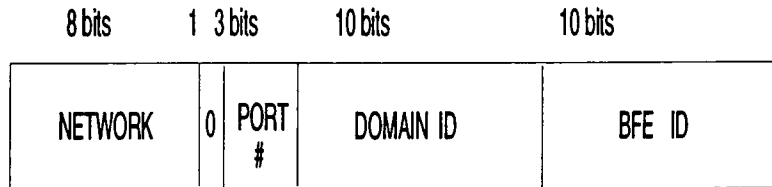


Figure 11: IP address

mode, the host will only be able to communicate with the hosts in its address translation tables, but only if all access controls succeed. A BFE can have up to 128 logical channels open simultaneously, where a logical channel is the intersection of a source and destination X.25 address and a X.25 precedence [20].

Internet Protocol Features

As stated earlier, a BFE must use the Internet protocol (IP) at the network level. For those people that are unfamiliar with IP, please look in [4] or [23]. There are some restrictions placed on IP for operation with BFEs, however. As always, the IP address is a thirty-two bit field composed of a network identifier and a network-specific host field. A DDN-RVN is a class A network, which has eight bits for the network identifier and twenty-four bits for the network specific host field. The DDN-RVN's network identifier is twenty-one. When a host that is connected to a BFE would like to communicate with another host, it must provide the Red IP address of the destination, along with the correct DDN-RVN X.25 address to the BFE. Usually the X.25 address supplied will correspond to the IP address given, but sometimes IP devices are not directly connected to a BFE; so the X.25 address will correspond to a Red gateway, which is the next stop on the way to

the destination [20].

A Blacker IP datagram can be up to 896 octets; however, datagrams of over 576 octets should be sent only if the host knows that the destination is ready to accept such a large datagram. If IP datagrams must be fragmented into multiple X.25 packets, no more than thirty-two incomplete datagrams can be sent at one time [20]. Hosts that can send datagrams at multiple security levels must send and receive datagrams that fit in one packet, but single-level hosts can send datagrams in packet sequences. All IP datagrams must have a sensitivity label; it is the first option on all datagrams.

One problem with a BFE is that it has limited buffer space. If a datagram already has authorizations, it is subject to normal flow control policy and adds no burden to the BFE's buffers. Outgoing datagrams that need new authorizations must wait in the buffer until the ACC sends back a permission or denial to send. Only five datagrams can be stored on a BFE. If any more outgoing datagrams need authorization, they are thrown away and the host must retransmit. To combat this problem, the system administrator should make sure that the keys for important and high-volume destinations are automatically set when the Blacker Initialization Carrier is ordered.

Blacker can handle multi-homed hosts (i.e., hosts with more than one network connection) by having a BFE for each interface that is connected to a network. Each BFE connected to such a host must have a different IP and network address, of course. If a host wants to communicate with a multi-homed host, the two hosts must decide which BFE and IP address will be used for communication. The BFE will also use the Internet Control Message Protocol (ICMP) [24] to send diagnostic information back to the host. By using ICMP, a BFE can tell the host what went

wrong. If a BFE receives an ICMP Echo Request message, it will reply with an ICMP Echo Reply message.

Black-Side Network Interface

The ciphertext side of the BFE is much like the cleartext side and needs to be reviewed only as background material or if a system administrator is trying to order the proper type of interface line from the DCA [20].

4.3 Other Products Considered for Study

There are many other security products being developed, and some of them are the following:

- An OSI security primer, which can be used as a guideline for system-wide security [25].
- The Intrusion Detection Expert System (IDES), which is being developed for the Navy by SRI [26]. In short, the IDES uses artificial intelligence (A.I.) techniques to compare the present activities of a user with profiles of previous activities. If there are any large discrepancies, appropriate action can be taken. Rockwell International Corporation and the Federal Bureau of Investigation (FBI) were hoping to use the IDES to complement their network security by this year [26].
- The Simple Network Management Protocol (SNMP), which is an important part of the network management structure for TCP/IP based internets [27]. A new version of SNMP with security extensions is being proposed in some new Internet Request For Comments (RFCs) [28, 29].

CHAPTER 5

Automated Teller Machine

When analyzing a product such as Kerberos, its impact on existing and new applications should be explored. Kerberos-authenticated (kerberized) versions of Berkeley's *rcp* (remote copy), *rsh* (remote shell) and *rlogin* (remote login) programs are provided with M.I.T.'s Kerberos Version 4 release, which should provide adequate sources for analyzing Kerberos effect on existing applications.

When designing a new application to use Kerberos, there are two main ideas considered. Obviously, the application should warrant the use of Kerberos. If the application has no real use for Kerberos, using it totally defeats the purpose of the project. Second, the application should be a network application, because Kerberos can be used to provide authentication and encryption (privacy) between a client (user) and a server. The server usually runs on a different machine than the client, which makes the program a network application, but occasionally the server and client are on the same machine. The main reasons for trying to use a new application with Kerberos are: (1) To see how well a new application interfaces with Kerberos. How easy is it to integrate authentication into the program and to use Kerberos to encrypt messages? (2) How does using Kerberos affect the application's performance?

After some deliberation, it was decided to simulate an automated teller machine

(ATM). An ATM seemed like a good choice because they are frequently used in the world and need some security. When banks are transferring data to other banks or between an ATM and a bank, they usually want the data to be protected.

5.1 ATM Overview

The model for the ATM was loosely based on real ATMs, so the expected functions are available. Users must first enter a name (instead of their bank card) and their personal identification number (PIN). The ATM gives the user three chances to enter the PIN correctly. Once the user is logged in, he/she is able to do as many transactions as he/she wishes. Some possible ATM transactions are: deposit, withdrawal, balance inquiry, and transfer. Users may perform any of these transactions using their checking or savings accounts, but only if they have those accounts. Users can also ask for help in using the ATM. Once users are finished using the ATM, they can exit the application.

The ATM client and server use Berkeley sockets to communicate. A socket is one of the two main application layer interfaces (APIs) used on UNIX¹ based machines to interface with communication protocols. The Transport Layer Interface (TLI) is the other API, but it was not available on the machines that were used for the testing, so only sockets were used for network communication.

When using sockets, information is usually sent using two main transport protocols: TCP [30] and UDP [31]. TCP, the *transmission control protocol*, is a connection-oriented protocol, which means that a connection is set up between the client and the server before any communication can occur. Using a connection-

¹UNIX is a registered trademark of AT&T

oriented transport protocol is like making a phone call. A connection must be established before any information can be sent. On the other hand, UDP, *the user datagram protocol*, is a connectionless protocol. No connection must be established before data transfer can begin.

TCP is more suitable for applications that will be sending large amounts of data. TCP is considered *reliable* because it takes care of duplicated, corrupted, out of order, dropped packets and packets sent to the wrong destination; whereas UDP does not take care of them unless it is enhanced. *Nonidempotent* applications, ones that need to have their actions done *at most once*, are better suited for TCP. TCP also provides flow control. Most of the cost in using TCP is expended in the connection establishment, so applications such as bulk transfer are more suited for TCP. It is difficult to open a TCP connection in a lossy network because TCP uses a three-way handshake during connection setup. TCP achieves *reliability through retransmission*, but this will not do any good if the network is very lossy or down.

UDP is better for short, low-traffic applications and when the network is very lossy. UDP applications should be *idempotent*. Reliability can be built into UDP by using sequence numbers on packets and timers to keep track of outstanding packets; but if reliability is needed and the network is stable, TCP is the best choice because of its built-in reliability.

At first glance, one might think that an ATM client-server application would be tailor-made for UDP. The ATM will be sending many small requests and getting many small replies in return. The problem is that the actions of the ATM must be nonidempotent. As an example, suppose a user wishes to withdraw two hundred dollars from his/her account. He/she would be quite upset if six hundred dollars were debited against the account (due to two transmission errors) while receiving

only two hundred dollars. The user would not mind having twelve hundred dollars deposited instead of the three hundred dollars actually deposited. However, there cannot be any duplicate requests, and the user does not want the requests to be lost. Imagine the user depositing twelve hundred dollars and then writing bad checks the next week because the deposit did not register in his/her account.

So, TCP was chosen as the transport protocol to be used with sockets. For a future project, it might be interesting to try UDP as the transport protocol to compare the performance of the two protocols.

5.2 ATM Client

The ATM client handles all transactions between the user and the server. There is a one-to-one mapping between the number of client processes and the number of users. In other words, there is one client process for each user connected to the ATM. After the client opens a TCP socket with the ATM server, Kerberos is used to authenticate the client to the server and vice-versa. The Kerberos functions, *krb_sendauth* and *krb_recvauth*, perform the authentication on the client and server, respectively. At this point, the client should be relatively sure that it is talking to the real ATM server and not an impostor. The client then asks the user to enter his/her name and PIN, which replaces inserting an ATM card and entering the PIN on a real ATM. Users are given three chances to enter their name and PIN before the connection with the client is severed, which simulates a real ATM holding a user's card after entering an invalid PIN.

Once the user has been authenticated by entering the correct name and PIN, a

X-Windows² client is brought up to provide a better interface for the user. After starting the X-Windows interface, the client must open a TCP socket between itself and the interface to receive information from the user and to be able to relay to the user information received from the server.

At this point, the client goes into a loop that runs until either the user quits or the socket communication fails. A *select* system call is used to listen on the server and user sockets for information. Whenever information comes in on one or both of the sockets, the *select* call returns and a *read* is performed on the first flagged socket. The ATM client acts much like a pass-through. If something comes in from the user, that message is sent to the server, and vice versa. The only message on which the client takes action is the quit message from the user. For every message sent by the user, there should be at most a single message returned by the server. For a visual overview of the client code, see Figure 12.

5.3 ATM Server

The ATM server is a *concurrent server*, which means that many clients can use the ATM simultaneously. The server could have been an *iterative server*, which would have allowed only one user at a time (much like a regular ATM), but the concurrent server was chosen for efficiency and for the users' benefit. It is much like having many ATM locations for a bank. When the server is started by an authorized user such as the **superuser**, the server's port and hostname are supplied on the command line to advise the program what port to use and what machine it is running on. If any other ATM servers are running, the current ATM

²Massachusetts Institute of Technology.

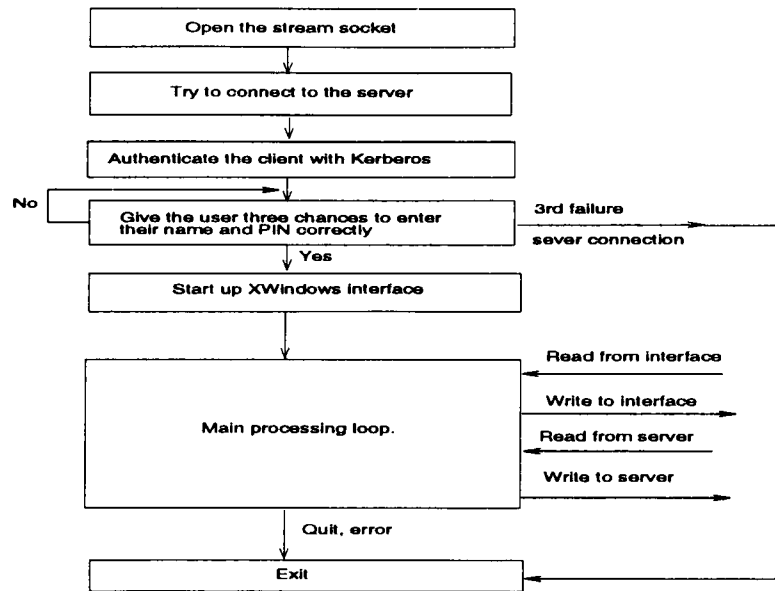


Figure 12: Client code overview

server prints an error message and exits.

An overview of the ATM server code is shown in Figure 13. Once the client has a connection with the server, the server uses Kerberos to authenticate the client. After performing the authentication, the server should be assured that the client is authentic. The server gives users three chances to authenticate themselves by entering their name and PIN. If the user fails the authentication test, the server severs the connection with the client and a note is made in the system log. All user authentication failures are recorded.

Once the user has been authenticated, the server enters its main processing loop. The server listens on the client's socket for information by using a *select* call. The *select* system call will return when the server receives a message from the

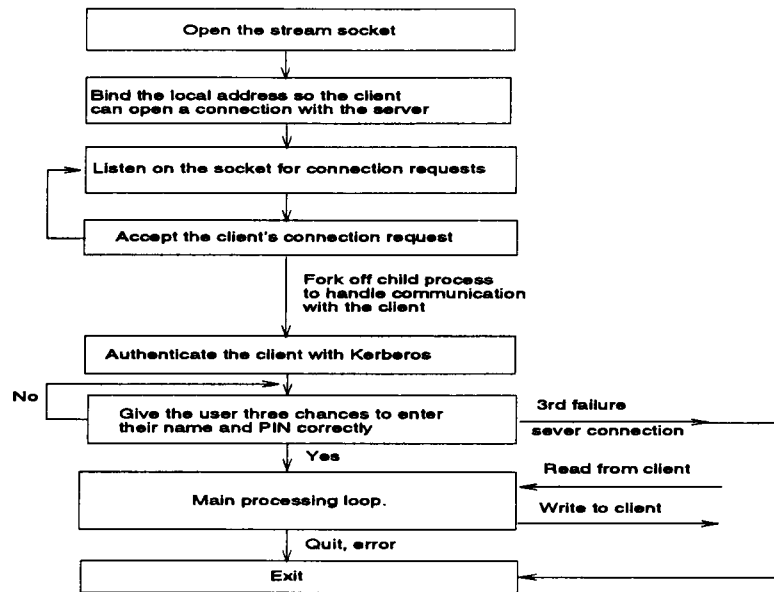


Figure 13: Server code overview

client or if an error occurs. The server tries to read the client's message, but if a read error or a normal disconnect occurs, the server exits. Otherwise, the message is correctly received and is parsed to determine the client's desired transaction. Once the message has been successfully parsed and executed, a message is sent to the client indicating success or failure.

The client can perform the following transactions:

1. Select the checking account.
2. Select the savings account.
3. Deposit money in the selected account.
4. Withdraw money from the selected account.

5. Transfer money from the selected account to the client's other account.
6. Get the balance for the selected account.
7. Fast Cash. (e.g., withdraw \$25 from checking)
8. Ask what account is selected?
9. Quit.
10. Any other transaction types are invalid.

Appropriate error checking is performed. For example, the server makes sure that users do not overdraw their account(s) or select an account that they do not have. Users' accounts are represented by files in the UNIX environment. File locking is used to ensure that only one process has access to a user's account at a time. This should prevent any *race conditions*, which could have resulted in incorrect account balances.

5.4 X-Windows Interface

The X-Windows interface was added for the user's benefit and to make the ATM client look more like a real ATM. The interface should make it much easier for users to access the ATM. The interface includes buttons for all of the possible ATM transactions, the numbers 0-9, the decimal point, and Enter, Clear, Quit and Help buttons. The user can also type in money amounts using the keyboard if the cursor is in the text window. The interface itself is a *vpane* widget, which is made up of a label, text and box widget. The box widget in turn is made up of many

command buttons. A representation of the X-Windows ATM interface is shown in Figure 14.

5.5 Kerberos Performance

To get an idea of the impact of Kerberos on existing applications, it was decided to analyze the performance of normal *rcp* (Kerberos Version 4 *rcp* without authentication and encryption), *rcp* using authentication, and *rcp* using authentication and encryption. Analyzing the performance of the ATM was considered for a short time but this idea was dismissed because the messages sent during ATM operation are so small. It was thought that the difference between using no security, Kerberos authentication, and Kerberos authentication and encryption would be negligible. The file sizes used for the *rcp* performance tests were one megabyte (MB), two MB, four MB and eight MB. Two sets of tests were performed: (1) between a Sun SPARCstation 1³ and a Sun SPARCstation ELC⁴ running SUNOS 4.1.1 and 4.1.2, respectively, and (2) between a DECstation 2100⁵ and a DECstation 5000/200⁶ running Ultrix 4.2. Running these tests on two different platforms should yield a good testbed of data.

³Sun Microsystems

⁴Sun Microsystems

⁵Digital Equipment Corporation

⁶Digital Equipment Corporation

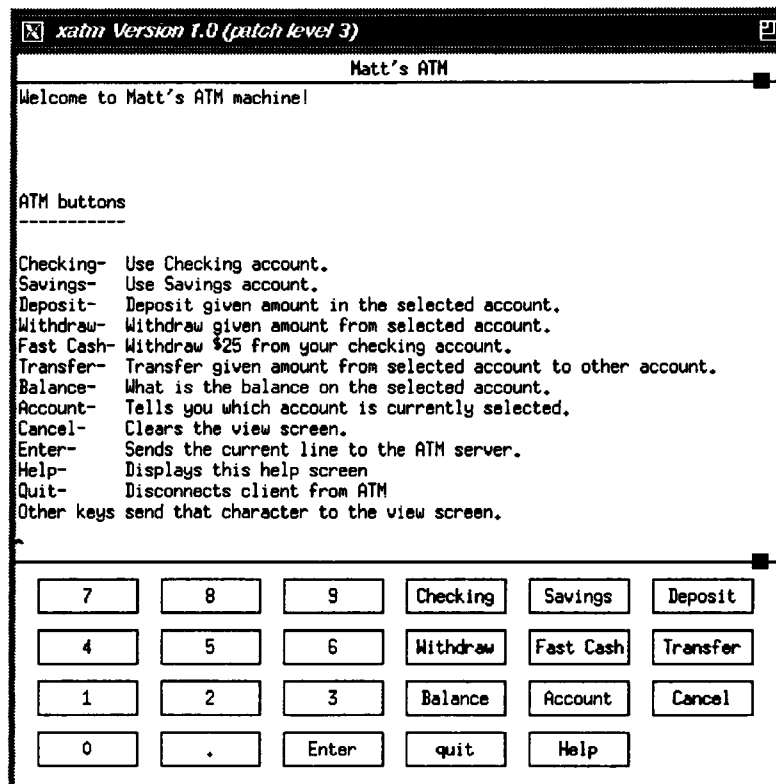


Figure 14: X-Windows Interface

5.5.1 SUN Platform Tests

The results are shown in Figures 15 and 16. Figure 15, representing the amount of system and user time used by each of the three sets of tests, produced similar results for each set of tests, but Figure 16, which represents the actual time it took for the tests to complete, yielded the most information. The granularity of the clock used for the system and user times shown in Figure 15 was not fine enough to get any meaningful results. Each of the results in Figure 16 increases approximately linearly as the file size increases. The *rcp* test using authentication and encryption took the most time to run by a large margin, so encryption should only be used when absolutely necessary. The results of the tests for regular *rcp* were a little faster than the results of the tests for *rcp* using Kerberos authentication. The two lines in Figure 16 roughly parallel each other. The main difference between the two tests is the Kerberos authentication used at the beginning of the program in the Kerberos authenticated *rcp* program. These results did vary slightly from test to test, so there was some experimental error in the results (especially in the regular and Kerberos-authenticated *rcp* results).

5.5.2 DEC Platform Tests

The results of the DEC platform tests were very similar to the SUN platform tests. The smaller granularity of the system clock led to the more precise results shown in the system and user time graph in Figure 17. The actual amount of time that elapsed during each set of tests is shown in Figure 18. This set of tests was quite similar to the ones run on the SUN platform, except that the DEC Ultrix *rcp* was used for the regular *rcp* tests instead of the one supplied with Kerberos

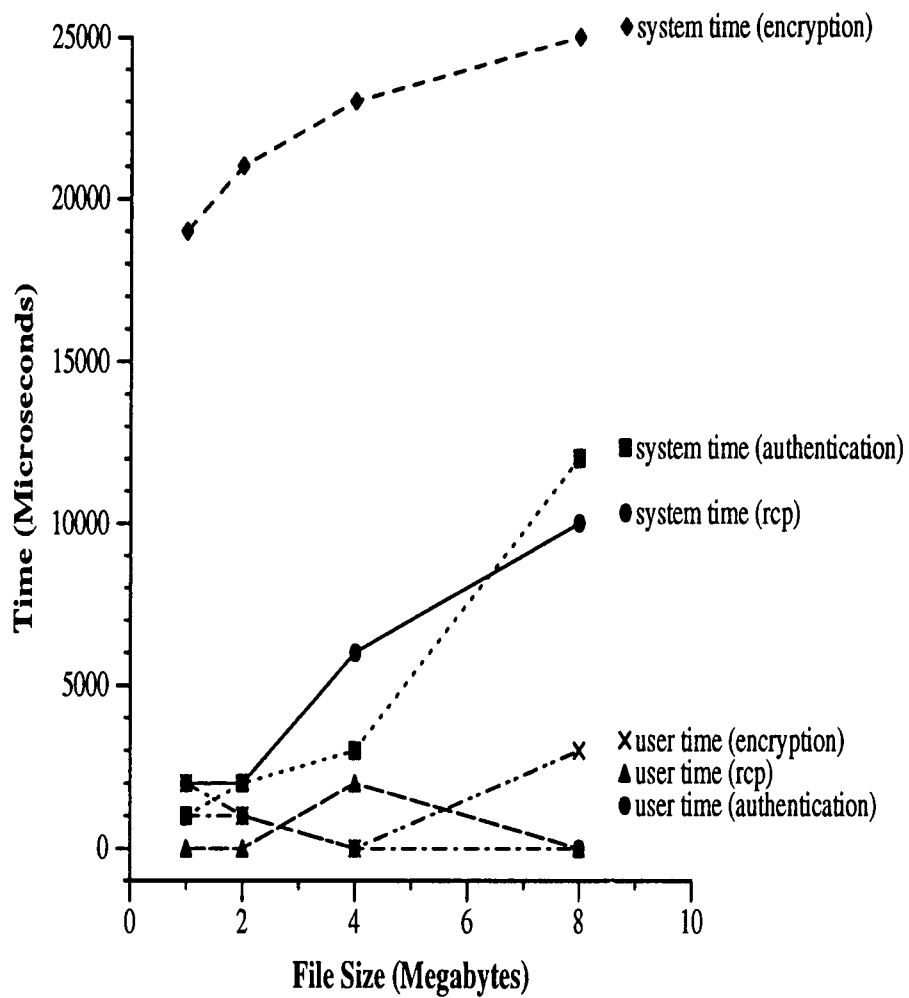


Figure 15: System and user times for rcp tests performed between a SPARCstation 1 and a SPARCstation ELC

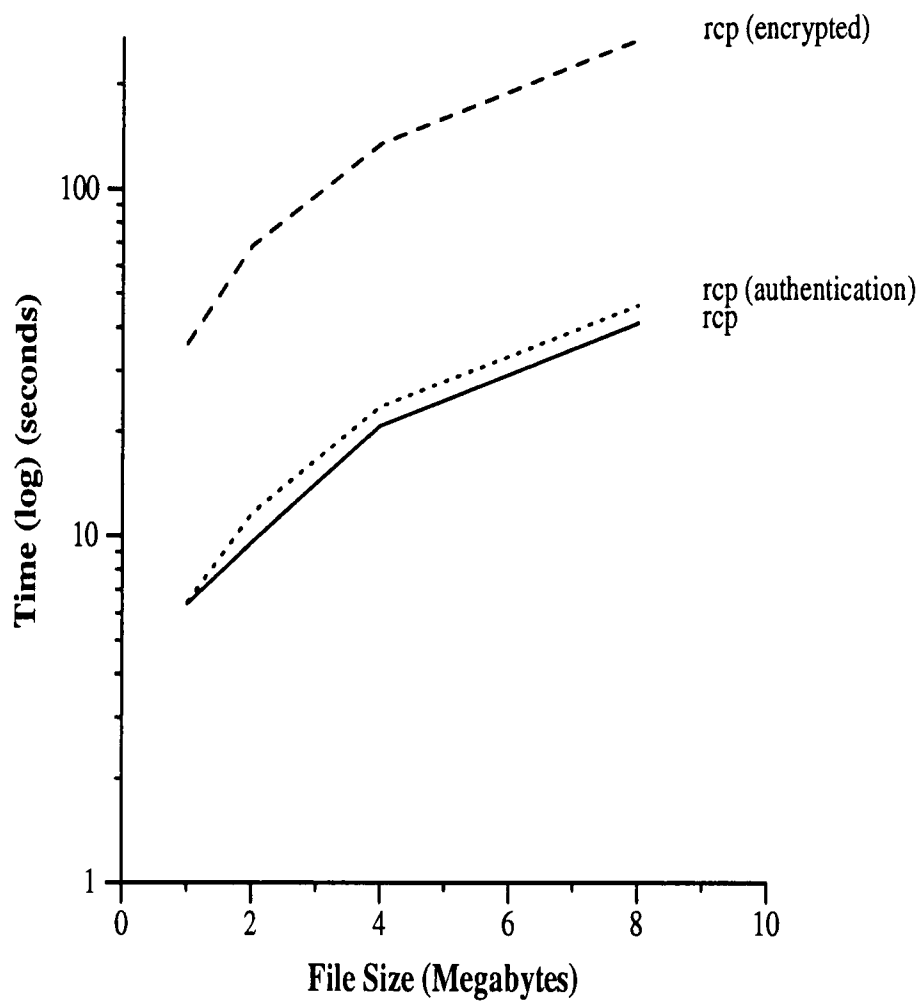


Figure 16: Actual times for all three rcp tests performed between a SPARCstation 1 and a SPARCstation ELC

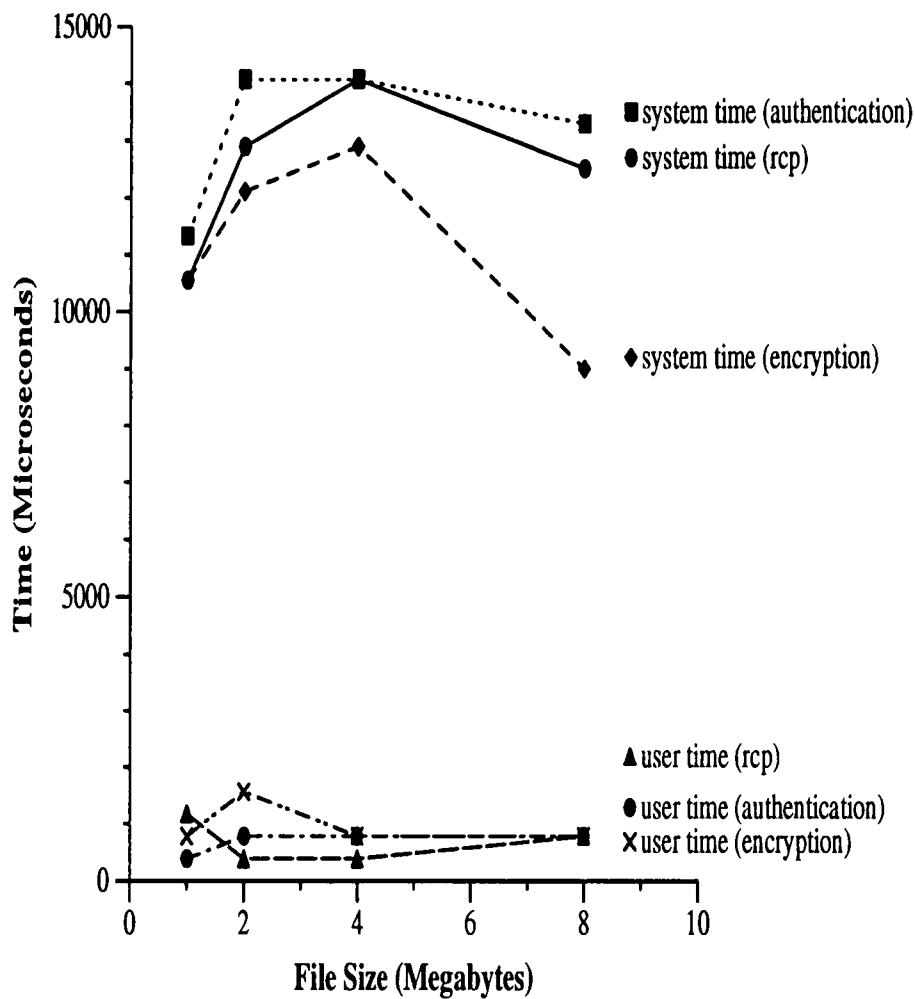


Figure 17: System and user times for rcp tests performed between a DECstation 2100 and a DECstation 5000/200

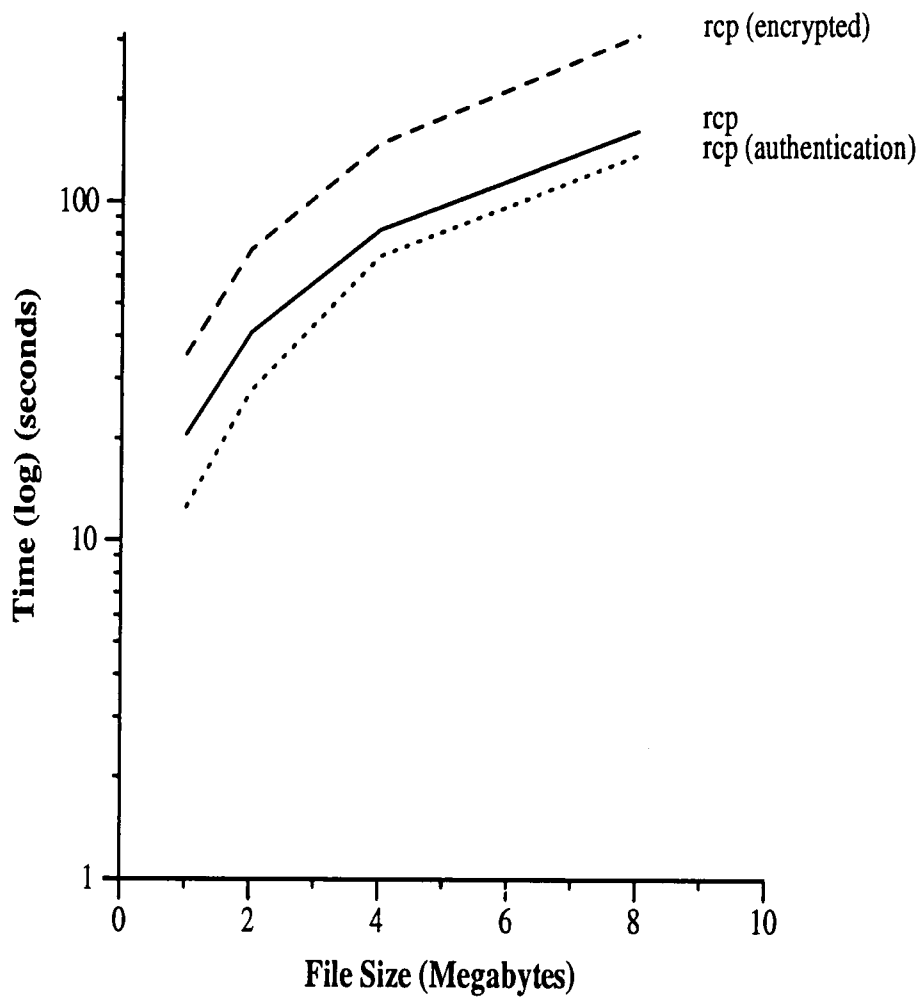


Figure 18: Actual times for all three rcp tests performed between a DECstation 2100 and a DECstation 5000/200

Version 4. All of the times were a little slower on the DEC platform.

5.5.3 TTCP Performance Tests

In order (1) to obtain timing requirements without file I/O interference and (2) to evaluate further the effect of Kerberos on message throughput, similar tests were run on the *ttcp* program. The *ttcp* program opens a TCP connection (or UDP if specified) to a *ttcp* server, transfers data, and closes the connection. It also computes actual and CPU times for the transmission on the receiving and transmitting end. Approximately three hours were required to add Kerberos authentication and encryption to the *ttcp* program. When using Kerberos, *ttcp* performs *mutual authentication* (i.e., the client authenticates itself to the server, and vice versa).

The tests were run using communications between a Sun SPARCstation 1 and a SPARCstation ELC with transfers of one MB, two MB, four MB and eight MB. The results of each test type varied slightly from test to test, and the best times for each type of test were used as the data points in the graph, so there was some experimental error involved in these tests. As expected, the *ttcp* tests with encryption took the most time (see Figures 19, 20, 21, and 22). The measured times for regular *ttcp* and *ttcp* with Kerberos authentication were comparable. The regular *ttcp* ran faster on the one MB and two MB transfers because the Kerberos authenticated *ttcp* performed mutual authentication (message to server, message to client) before the transfer, which is a dominant delay.

The Kerberos authenticated *ttcp* ran faster on the four MB and eight MB files for three reasons: (1) the time needed for mutual authentication became less significant compared to the total time, (2) because of a small timing difference

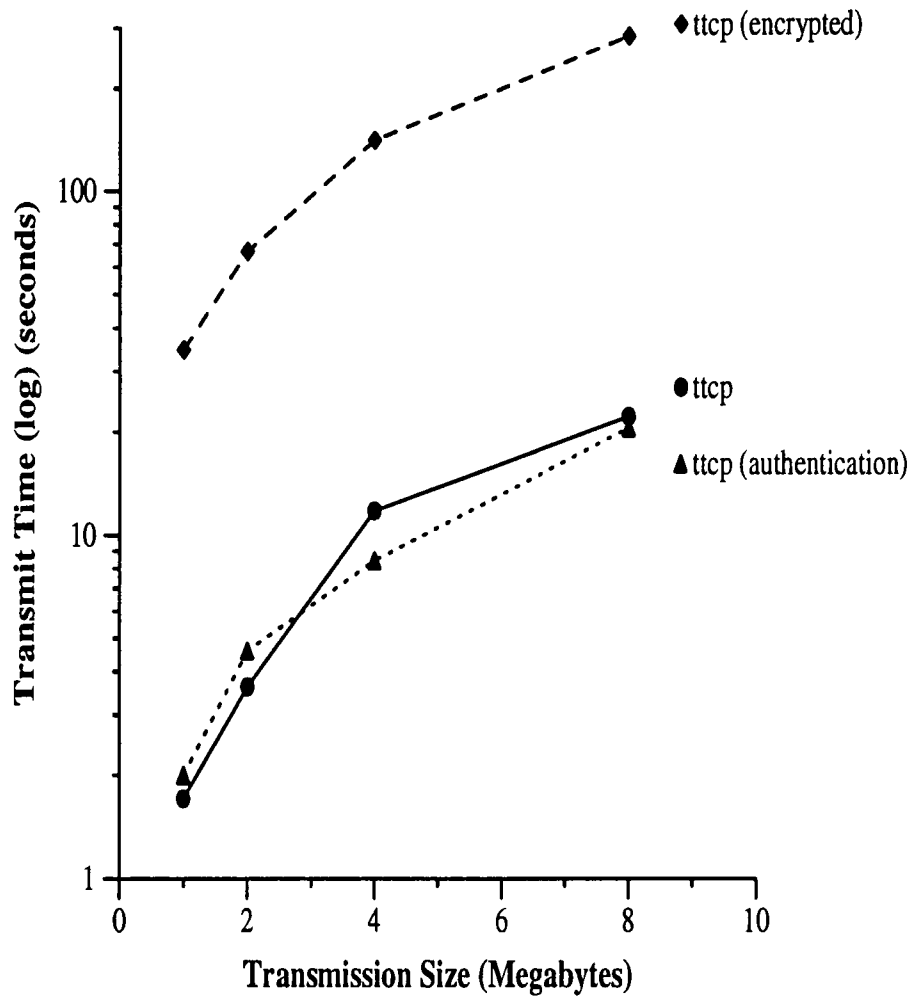


Figure 19: Transmission times for ttcp tests performed between a SPARCstation 1 and a SPARCstation ELC

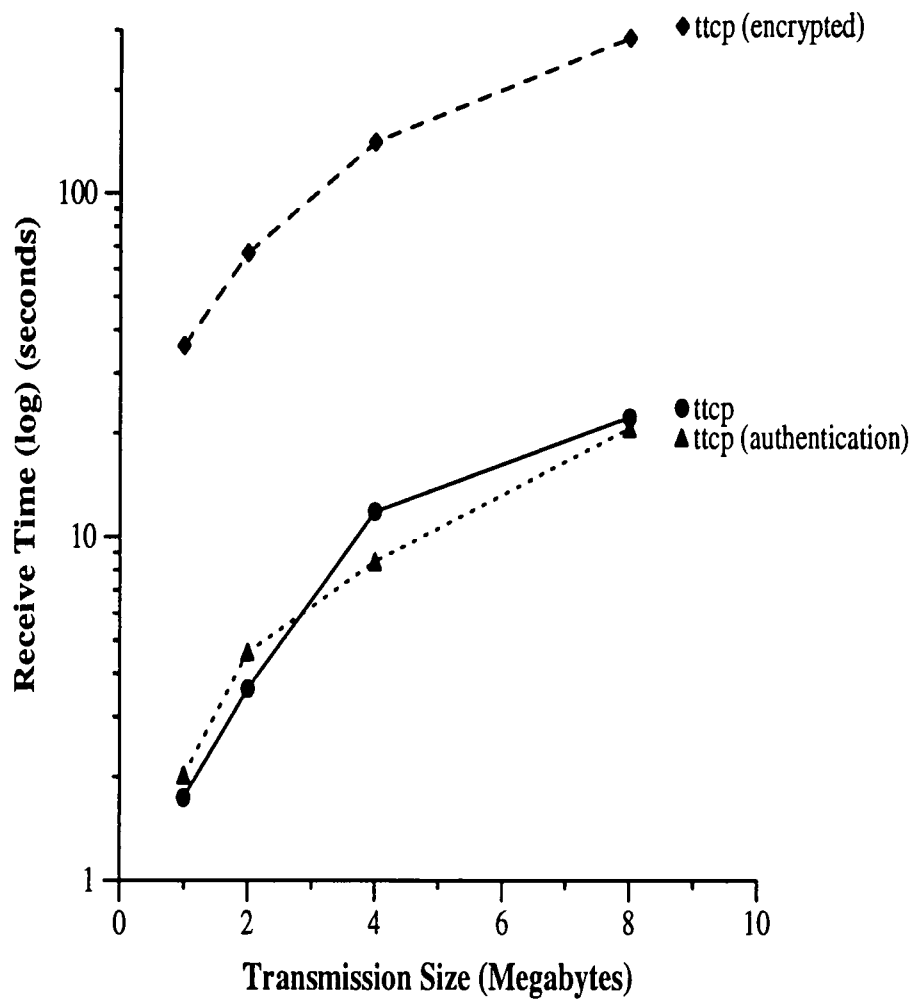


Figure 20: Receiving times for ttcp tests performed between a SPARCstation 1 and a SPARCstation ELC

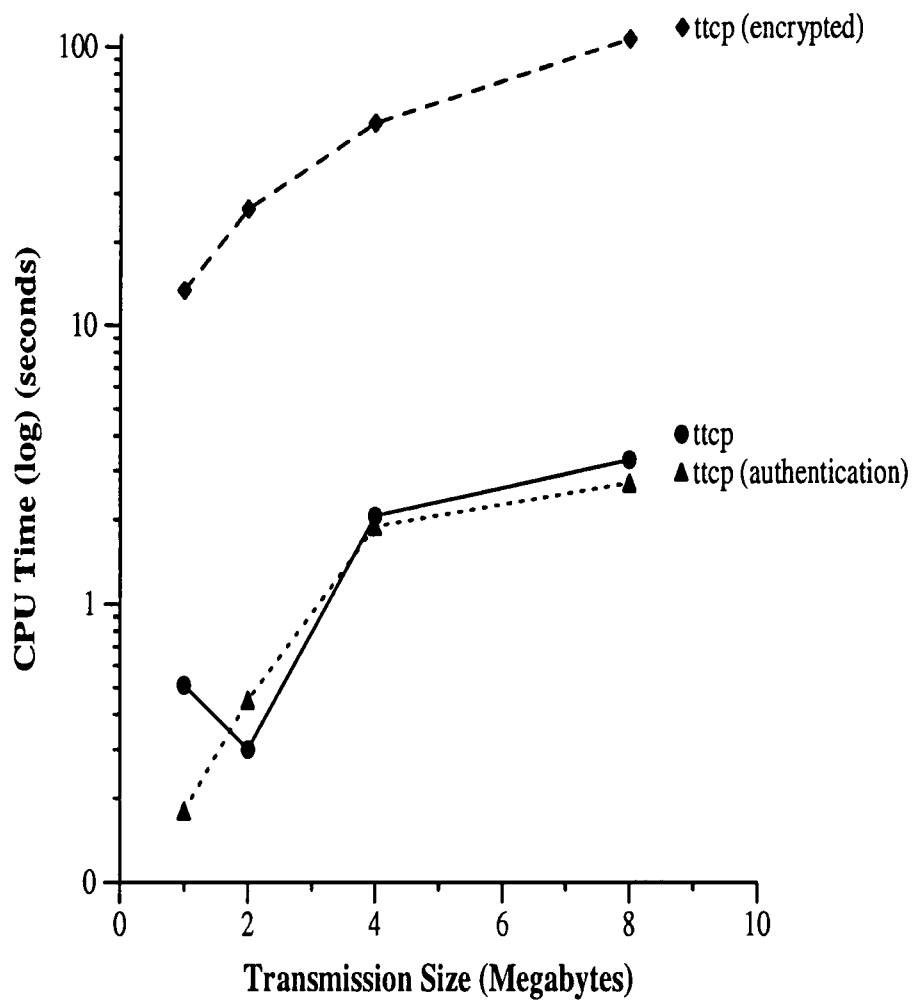


Figure 21: Transmitting CPU times for ttcp tests performed between a SPARCstation 1 and a SPARCstation ELC

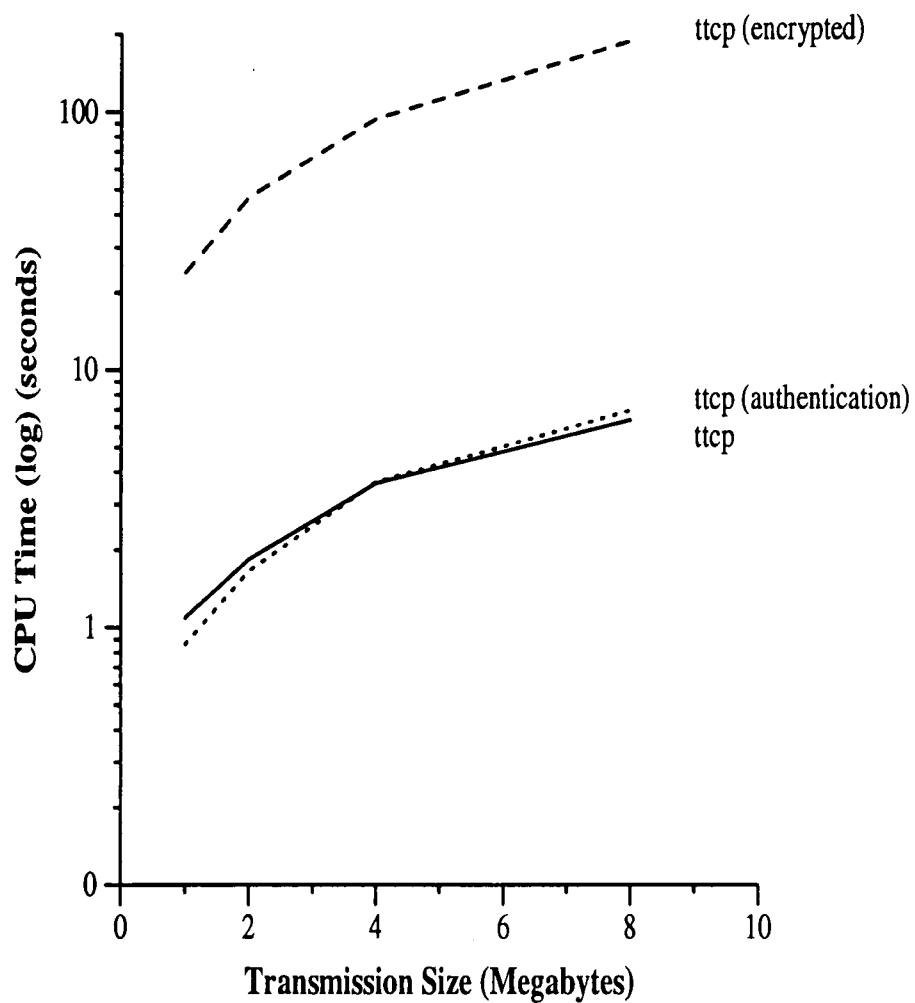


Figure 22: Receiving CPU times for ttcp tests performed between a SPARCstation 1 and a SPARCstation ELC

between the two programs, and possibly (3) experimental error. A comparison of the code showed that, in addition to the mutual authentication code, a few more lines of code were executed repetitively in standard *ttcp* than in the Kerberos authenticated *ttcp*. In the regular *ttcp*, two functions, *Nread* and *Nwrite*, were called to perform the transmission and receiving functions; in the Kerberos *ttcp*, *des_read* and *des_write* were called to perform the respective functions. The *Nread* and *Nwrite* functions do three comparisons and two comparisons, respectively, before calling *read* and *write*, respectively, while *des_read* and *des_write* do only one comparison before calling *read* and *write*. Each of these calls (*Nread*, *Nwrite*, *des_read*, *des_write*) is in a loop that runs for thousands of iterations; the extra time required for executing the *Nread* and *Nwrite* functions could add up to provide the very perceptible difference in execution time of the regular *ttcp* and Kerberos authenticated *ttcp* for four and eight MB transfers. The difference between the two tests could also be caused by experimental errors in the tests.

As in the *rcp* tests, the tests with encryption took the most time. These tests (i.e., using *ttcp*) presented a truer picture of the encrypt/decrypt times when using Kerberos, because there was no file I/O dependence as there was in the *rcp* tests. When running in encrypted mode, *ttcp* had a peak transmission rate of about 31 kilobytes (KB)/sec. The times for the regular and Kerberos authenticated *ttcp* ranged from four to eight percent as large as the encrypted results. Regular *ttcp* had a peak transmission rate of about 580 KB/sec. (ranging from 350 to 580 KB/sec.) and Kerberos authenticated *ttcp* had a peak transmission rate of 511 KB/sec. (ranging from 399 KB/sec. to 511 KB/sec.). Therefore, Kerberos encryption degraded the *ttcp* transfers by greater than an order of magnitude.

5.6 Traffic Analysis

The performance of Kerberos itself is not easily measurable because there were not a sufficiently large number of machines available for stress testing the Kerberos server. Hopefully more machines will be made available later if further research is performed.

Kerberos does not send out many extra messages, so the effective performance of the nodes on the network should not be affected very much unless there are many encrypted messages being sent. When the *kinit* command is used, one message is sent to the Kerberos server and one message is sent back. The *klist* command, which lists the Kerberos tickets the user has, does not send out any network traffic. When it is executed, it looks for the user's ticket file and prints it on the screen. When a user is using a Kerberos authenticated service like *rcp* for the first time (he/she does not have a ticket-granting ticket (TGT) for the service), a maximum of four messages is sent across the network: (1) from the user to the TGT server, (2) from the TGT server back to the client, (3) from the client to the service (*rcp* in this example), and (4) (optionally) a message from the service to the client if the client desires mutual authentication. Once the initial request of any service is performed, there should not be any extra traffic unless the user's ticket for the service expires. The messages are slightly longer when using Kerberos, but this extra overhead is negligible unless the messages are being encrypted.

CHAPTER 6

Conclusion

After analyzing three security products separately in the previous chapters, it is now time to compare the products and their effect on LAN security. Some ideas for future research are also discussed.

6.1 Xerox's XEU and DEC's SES

The security products from Xerox and DEC are quite similar. The Xerox Encryption Unit (XEU) and DEC's Security Encryption System (SES) both have store-and-forward cryptographic units operating transparently at the data-link layer between the host (e.g., server, workstation) and the network. The Digital Ethernet Secure Network Controller (DESNC) is the store-and-forward unit for the SES. All traffic traveling between the host and the cryptographic unit is in cleartext, while all traffic going from or coming into the unit is encrypted unless the unit is operating in *bypass* mode. Most hosts are not allowed to communicate in *bypass* mode because they have access to sensitive information, so the cleartext traffic generated by these units is usually quite low.

Both systems use encryption to protect the data, but the data-link layer headers are left in cleartext, which is an advantage in an extended network. If any devices (e.g., gateways) need to access the IP portion of the packet, they must have their

own XEUs to decrypt the message, get the information they need, and reencrypt the message. Unless the gateway is physically secure, this could be a weakness in the system. When the encryption is used, the messages sent will increase in size by fourteen to twenty-one bytes and the effective throughput (i.e., real data) of the system will deteriorate.

DEC's SES uses the Data Encryption Standard (DES) for encryption and Xerox's XEU uses an unspecified government algorithm. Initialization keys are entered into each unit to load configuration information at initialization time. A numeric key is entered on the keypad on the front of the DESNC along with a physical key, and a physical key is entered into the XEU. The keys for the XEU also load in network topology information. The XEU does not have a central unit for key distribution, authentication and authorization, so most of this information must be stored on one of the primary circuit boards in the XEU. DEC's SES also can monitor the network for alarms and audit the system. The XEU has been approved by the U.S National Security Agency (NSA) for government use.

The VAX Key Distribution Center (KDC) manages all of the DESNCs on the local area network (LAN). Besides distributing keys, it determines whether nodes may communicate and can monitor the operational and security status of the LAN. Each user connected to a DESNC has an *access class* or *access class set* that controls his/her communication.

When a host wants to communicate with another host using DEC's SES, the message is sent to the DESNC, which sends a message to the VAX KDC to see if the two hosts are allowed to communicate; if they are, an encryption key is sent to the sending and receiving DESNCs and the message is encrypted frame by frame and sent out on the network. DEC's SES and Xerox's XEU incorporate

both authentication and authorization in their systems. Encrypted traffic for TOP SECRET users and cleartext traffic for users not connected to a cryptographic unit can travel over the same broadcast medium (e.g., Ethernet).

In DEC's SES, the encryption keys used by the DESNC on each side of the communication are usually retrieved from the VAX KDC or from a key supply that the business provides. The XEUs, on the other hand, are the only units in its security system, so the keys are stored in each XEU. There are pluses and minuses to having one entity responsible for key distribution. One plus is that a DESNC does not have to store keys for every node on the network with which it communicates. The keys used for communication are remembered by the DESNC for up to eight days. It is also much easier to change or send new keys from one central location instead of physically having to visit every unit and enter a new key. If the master key for the VAX KDC is compromised, all encryption keys on the network must be replaced. A minus is the extra traffic generated by the key exchanges. There can be up to three messages over the network before communication between two hosts can commence for the first time: (1) the message to the VAX KDC from the sending DESNC asking whether this communication is allowed, (2) the VAX KDC sending the encryption key to the sending DESNC, and (3) the VAX KDC sending the encryption key to the receiving DESNC.

The throughput for the DESNC increases as the packet size increases but eventually it hits a limit (the frame size for the underlying broadcast medium). The performance of the VAX KDC depends on CPU utilization, network performance, and the number of nodes attached to DESNCs in the LAN. Unfortunately, no real definitive answer for the performance of DEC's SES was given, but the encryption of most messages and the extra messages sent between the DESNCs and the VAX

KDC must degrade the effective performance of the nodes on the network. There were no performance figures available for a LAN with and without the use of DEC's SES and Xerox's XEU. The only performance information given for the XEU is that it can process 200 packets per second (600 bytes per packet). Again, the encryption of most messages will degrade the effective performance of the nodes on the network. Xerox's information indicated that encryption would, on average, add about fourteen to twenty-one bytes per packet.

6.2 Kerberos

Unlike the other two products discussed, Kerberos is a *software-only* system that provides authentication for users and services. Authorization is not a part of Kerberos, but it can be added by using access-control lists (e.g., a daemon) or some similar concept. Kerberos has a little more flexibility when sending messages, because it has three levels of security: (1) authentication at connection time, (2) KRB_SAFE messages: all messages are authenticated, but none are encrypted, and (3) KRB_PRIV messages: all messages are encrypted and authenticated. As with DEC's SES, the DES is used for the encryption process in Kerberos. Cipher block chaining and the cyclic-redundancy checksum (CRC-32) are used along with DES. Other encryption algorithms are allowed in Version 5 of Kerberos, so the worldwide availability of Kerberos implementations should grow. KRB_PRIV messages sent using Kerberos will remain encrypted end-to-end over wide area networks and extended LANs using routers and bridges, because the encryption is performed at the application layer. All the headers will remain in cleartext and be accessible by any of the lower layers.

As with DEC's SES, Kerberos has centralized servers that distribute keys to the users on the network. The KDC sends ticket-granting tickets (TGTs), while the Ticket-granting server (TGS) distributes service tickets. These tickets contain encryption keys, which are never sent in cleartext because tickets are always sent in encrypted form. The extra traffic generated by Kerberos is kept to a minimum. Three network messages are needed to get a TGT, and three messages are sent when using a Kerberos service for the first time (or when a ticket expires). If the user desires mutual authentication, there are four messages sent over the network before regular communication can begin. The default lifetime for tickets is eight hours, which should keep the overhead low.

The problems with Kerberos were well documented in chapter 2, so they will not be discussed here. As with the other two products, Kerberos should be very transparent to the user. The users should not notice that Kerberos is being used unless the Kerberos server goes down or their TGT or service ticket expires. The utilities available for use with Kerberos are very good except for the notable exception of a delete operation for the Kerberos database. Adding new Kerberos-authenticated services is as easy as adding a line in the Kerberos server for every machine that will use the new service and adding an entry to the *srvtab* file for each machine.

The ATM project totalled approximately thirty-three hundred lines of code (including comments), with approximately eighteen hundred lines for the ATM server, six hundred lines for the ATM client, and nine hundred lines for the X-Windows interface. Of the eighteen hundred lines of code for the ATM server, only about forty lines of code were needed to perform the Kerberos authentication with the user. For the client, only about twenty-five lines of code are needed. To add the same functionality to other services, the programmer would essentially be able to

copy the code directly into the new service and use it. The only variables involved are: (1) whether mutual authentication is done, and (2) are all other messages after connection establishment sent in cleartext, authenticated, or authenticated and encrypted?

In the Kerberos-authenticated *rcp* code, there are approximately twelve hundred lines of code. It took approximately sixty lines of code to send authentication messages, thirty lines of code to receive authentication messages, forty lines of code to send encrypted messages, twenty-five lines of code to receive encrypted messages, and about another thirty lines of code spread throughout the program to total approximately fifteen percent of the code. The performance figures in Chapter 5 show that the overhead for Kerberos is quite low, especially when encryption is not being used. When compared to sending messages using encryption, sending messages in cleartext took twenty-five percent to thirty-seven percent as long. Because of these results, encryption should only be used when needed.

6.3 Speed Comparison

When running in encrypted mode, the Kerberos authenticated *ttcp* had a peak transmission rate of about 31 KB/sec., which is at least an order of magnitude less than the rates published for DEC's DESNC and Xerox's XEU. The DESNC's published peak transmission rate was 300 KB/sec. and Xerox advertised a peak transmission rate of 120 KB/sec. for its XEU. The principal reason for this degradation is simple: Kerberos is a software product; it must use the CPU on the workstation to perform its encryption and decryption. Both the DESNC and XEU have their encryption/decryption routines implemented in hardware, making

them much faster than those of Kerberos.

6.4 Availability

Availability of the three security systems varies widely. XEUs are sold only to National Security Agency (NSA) approved customers, so its availability is limited. Kerberos is by far the easiest of the three products to obtain. Anyone in the United States with Internet access can retrieve Kerberos Version 4 or 5 via anonymous ftp (over the Internet) for no cost, and international users can get a *skeleton* version of Kerberos (i.e., without DES) over the network. International users can then add encryption to Kerberos.

As of 1990, the price for a XEU was approximately \$5,000 and a Key Management Loader (KML) cost approximately \$12,000. As of September 1992, the price for a DESNC was approximately \$8,532 and the price for the VAX KDC was approximately \$10,900. The cost for the XEU and KML has probably increased.

Any one of the three products can be used to improve the security of a LAN. Both Kerberos and DEC's SES use private key encryption, which does not scale well when the network becomes large. Kerberos is slightly more flexible, because it allows other encryption algorithms to be used. Public key encryption is an alternative to private key encryption. Public key encryption uses two keys, a public and a private key. The public key can be known by anyone, so it can be distributed to name servers and the like. The private key only needs to be known by a few entities. With private key encryption, each communicating pair of nodes must share a key, which can lead to the aforementioned explosion of keys if the network gets large. As time progresses, more security products for LANs are being

introduced, so network security as a whole should be improving.

6.5 Future Research Ideas

While doing this thesis, other possible research topics were discovered. These topics include: (1) installing and testing Kerberos Version 5, (2) SPX, which is essentially a public-key version of Kerberos, and (3) secure SNMP, which allows authentication and authorization for SNMP based managers and agents.

LIST OF REFERENCES

LIST OF REFERENCES

- [1] Jennifer G. Steiner, Clifford Neuman, and Jeffrey I. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *Proceedings of the Winter 1988 USENIX Conference*, March 1988.
- [2] National Institute of Standards and Technology. Data Encryption Standard. Federal Information Processing Standard (FIPS) FIPS Publication 46-1, January 1988.
- [3] W. Richard Stevens. *UNIX Network Programming*. Prentice-Hall, 1990. ISBN 0-13-949-876-1.
- [4] J.B. Postel. Internet Protocol. Request for Comments 791, DDN Network Information Center, SRI International, September 1981.
- [5] Bill Bryant, Jennifer Steiner, and John T. Kohl. Kerberos Installation Notes. Draft, Project Athena, M.I.T., Cambridge, Massachusetts, January 1990.
- [6] Bill Bryant and John T. Kohl. Kerberos Operation Notes. Draft, Project Athena, M.I.T., Cambridge, Massachusetts, January 1990.
- [7] John T. Kohl. The Evolution of the Kerberos Authentication Service. In *Proceedings of the Spring 1991 EurOpen Conference*, May 1991.
- [8] Steven M. Bellovin and Michael Merritt. Limitations of the Kerberos Authentication System. In *Proceedings of the Winter 1991 USENIX Conference*, January 1991.
- [9] John T. Kohl and B. Clifford Neuman. The Kerberos Network Authentication Service. Request for Comments Revision 5, Project Athena, M.I.T., Cambridge, Massachusetts, April 1992.
- [10] International Organization for Standardization. Information processing systems - Open Systems Interconnection - Specification of Abstract Syntax Notation One (ASN.1). International Standard 8824, December 1987.
- [11] Ronald L. Rivest. The MD4 Message Digest Algorithm. Request for Comments 1186, DDN Network Information Center, SRI International, October 1990.
- [12] Digital Equipment Corporation. DESNC Preliminary Installation/User's Guide for Field Test. Digital Equipment Corporation, U.S.A., 1987.
- [13] Digital Equipment Corporation. Ethernet Encryption: Digital Ethernet Secure Network Controller: (DESN) and Key Distribution Center (KDC). Digital Equipment Corporation, U.S.A., 1987.

- [14] Digital Equipment Corporation. Sales Update/Ethernet Enhanced Security System. Digital Equipment Corporation, U.S.A., 1988.
- [15] Digital Equipment Corporation. Software Product Description: VAX Key Distribution Center, Version 1.0. Digital Equipment Corporation, U.S.A., 1987.
- [16] Digital Equipment Corporation. The VAX Key Distribution Center Installation Guide. Digital Equipment Corporation, U.S.A., 1987.
- [17] Xerox Corporation. The Xerox Encryption Unit. Xerox Corporation, VA, 1990.
- [18] Xerox Corporation. Xerox Encryption Unit: Technical Product Description. Xerox Corporation, VA, 1989. Draft.
- [19] Xerox Corporation. Xerox Encryption Unit: Customer Testing. Xerox Corporation, VA, 1989. Draft.
- [20] Blacker Interface Control Document, 1989.
- [21] Marshall T. Rose. *The Open Book: A Practical Perspective on Open Systems Interconnection*. Prentice-Hall, 1990. ISBN 0-13-643-016-3.
- [22] Mitch Betts. Guarding the Network Gates. (Network security system from Unisys Corp.). *Computerworld*, 24:57, 1989.
- [23] Douglas E. Comer. *Internetworking with TCP/IP: Principles, Protocols, and Architecture*. Prentice-Hall, 1988. ISBN 0-13-470-154-2.
- [24] J.B. Postel. Internet Control Message Protocol. Request for Comments 792, DDN Network Information Center, SRI International, September 1981.
- [25] William Stallings. A Network Security Primer. *Computerworld*, 24:63-66, 1990.
- [26] Susan Kerr. Using AI to Improve Security. *Datamation*, 36:57-59, 1990.
- [27] Jeffrey D. Case, Mark S. Fedor, Martin L. Schoffstall, and James R. Davin. A Simple Network Management Protocol. Request for Comments 1157, DDN Network Information Center, SRI International, May 1990.
- [28] James M. Galvin, Keith McCloghrie, and James R. Davin. SNMP Security Protocols. Request for Comments 1352, DDN Network Information Center, SRI International, July 1992.

- [29] James R. Davin, James M. Galvin, and Keith McCloghrie. SNMP Administrative Model. Request for Comments 1351, DDN Network Information Center, SRI International, July 1992.
- [30] J.B. Postel. Transmission Control Protocol. Request for Comments 793, DDN Network Information Center, SRI International, September 1981.
- [31] J.B. Postel. User Datagram Protocol. Request for Comments 768, DDN Network Information Center, SRI International, August 1980.

APPENDIX

APPENDIX A

Kerberos Terms

- **authenticator** contains information that can prove that it was recently generated and then encrypted by the session key, which only the client and service should know. The authenticator contains the client's principal identifier, the machine they are on, a checksum and a timestamp.
- **authorization** indicates the access rights of a particular authenticated client to use the requested services.
- **credentials** is a ticket plus a secret session key that is needed to use the ticket for an authentication exchange. After the service decrypts the ticket, it can use the enclosed session key to decrypt the authenticator that is sent by the client.
- **instance** could be the name of the machine the service is on or a special title like **admin** or **root**.
- **KDBM** is the Kerberos Database Manager, which provides a read-write interface for updates of the Kerberos master database.
- **Kerberos master database** is a single database that contains all principals and private keys for clients and services in the local realm. Slave machines may have read-only copies of this database.

- **KKDS** (Kerberos Key Distribution Service) is a network service that supplies Kerberos tickets and temporary session keys. It provides the initial ticket and ticket-granting ticket (TGT) that are needed when logging into a machine.
- **principal** is the Kerberos client, which can be either a user or a server. The principal must have a unique name, which usually consists of the username, instance and realm, with the form: `name.instance@realm`.
- **private (secret) key** is the key that is known only by Kerberos and the principal. This key has a longer lifetime than a session key.
- **realm** is Kerberos's local realm, a portion of a network. It is usually the size of a Local Area Network (LAN) or smaller. For most of the discussion, all communication will be in the local realm. There is one main Kerberos server per realm, with the possibility of having many secondary servers that have read-only access to the Kerberos database. The secondary servers are used to provide some amount of fault tolerance.
- **session key** is an encryption key that is used by the service and the client, but only for the lifetime of the service ticket. The key is then thrown away. The service and client can always negotiate for new session keys.
- **TGS** (Ticket granting service) is the service that gives out tickets for Kerberos-authenticated services such as `rlogin`, `rcp`, `rkinit` and `sample`.
- **ticket** is the mechanism that authenticates a client to a service. It contains a client's principal identifier, the name of the server, the IP address of the client and a session key, which is all encrypted using the service's private key.

- **timestamp** is the number of seconds since 00:00:00 GMT, January 1, 1970
(for Kerberos Version 4).

VITA

Matthew Gregory Hecht was born in Knoxville, Tennessee on May 29th, 1966. He moved to Cape Girardeau, MO in 1969. He graduated from Cape Central High School in June, 1984, and started school at Southeast Missouri State University in August, 1984. He graduated cum laude with a Bachelor of Science degree in Computer Science and Mathematics in May, 1988.

In August, 1988, he entered the graduate program in Computer Science at the University of Tennessee. From Spring 1989 to Fall 1991, he worked as a graduate teaching assistant in the Computer Science Department. He currently works for SNMP Research, Inc., in Knoxville, TN.