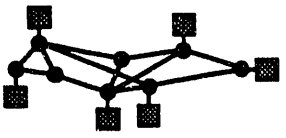
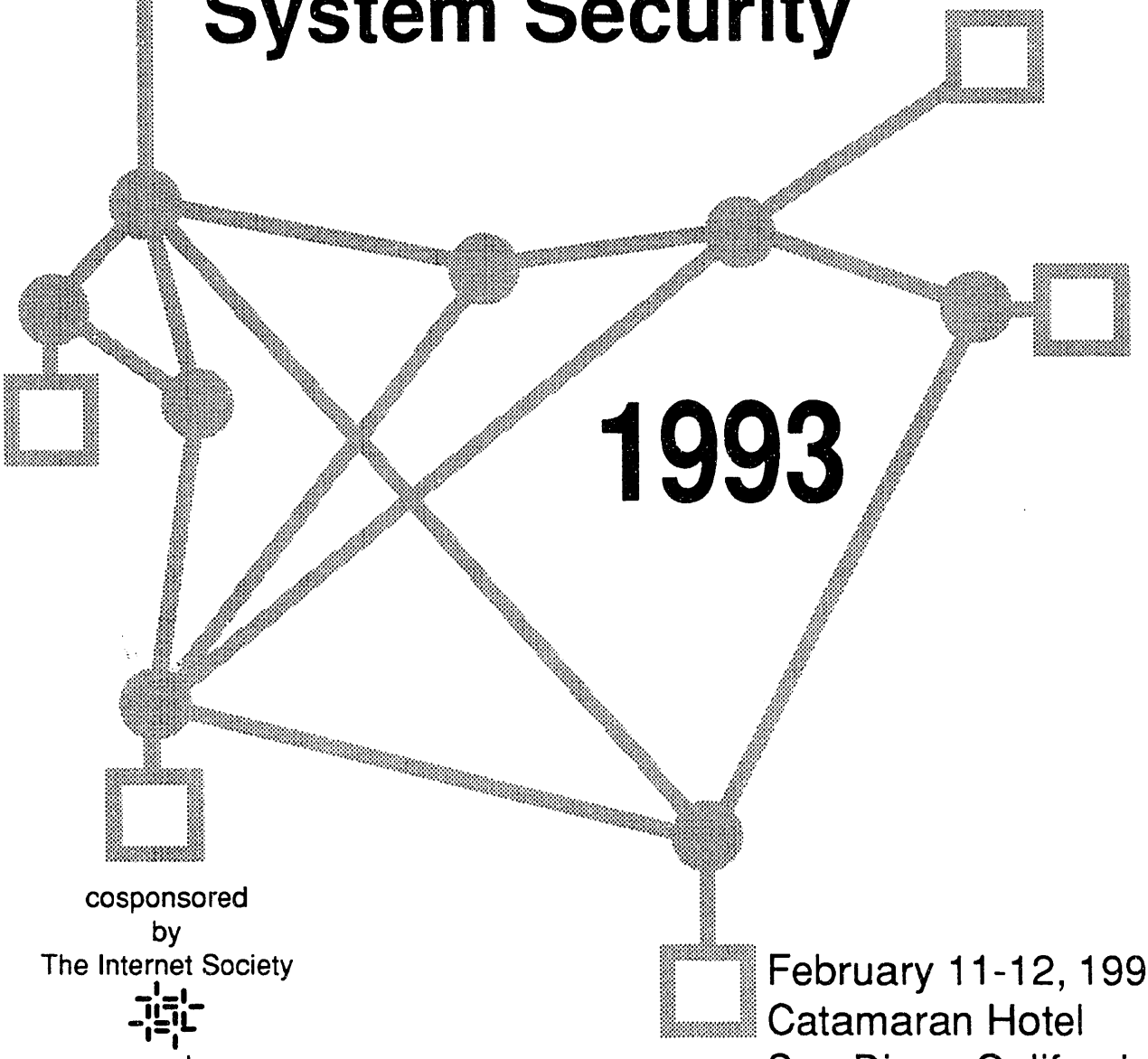
PROCEEDINGS

Privacy and Security Research Group



Workshop on Network and Distributed System Security



1993

cosponsored
by
The Internet Society

and
Lawrence Livermore
National Laboratory

February 11-12, 1993
Catamaran Hotel
San Diego,California

Contents

Privacy for Large Networks

Panel Session - Layer Wars: Options for placement of security in the OSI Reference Model (Position Paper)

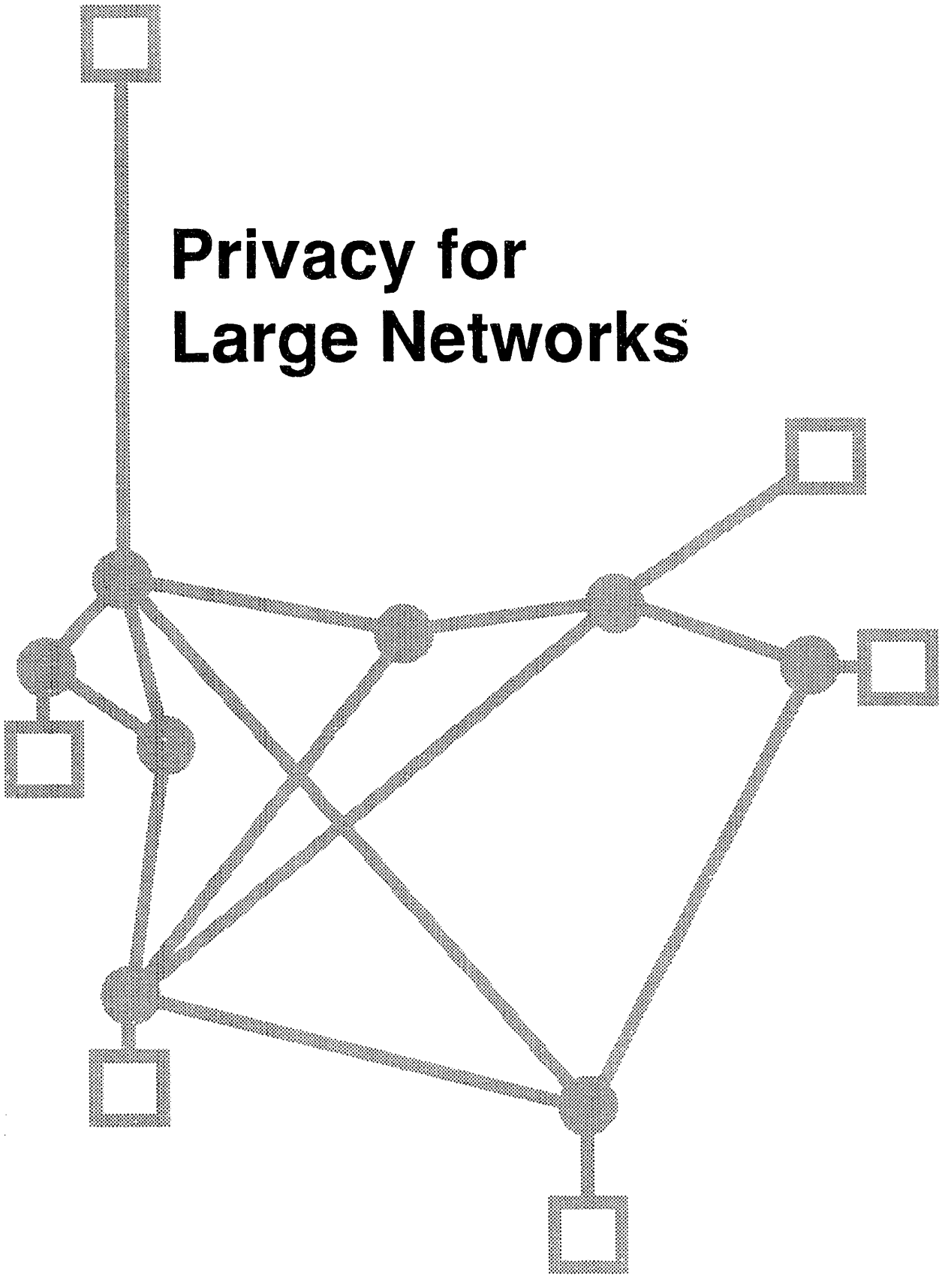
Electronic Documents

Privacy Enhanced Mail

Distributed Systems

Panel Session - Network Security Using Smart Cards (Position Papers)

Privacy for Large Networks



NREN Security Issues : Policies and Technologies
Invited Paper

Dennis K. Branstad
Computer Systems Laboratory
National Institute for Standards and Technology
Gaithersburg, Maryland

Robert J. Aiken
Lawrence Livermore National Laboratory and the
Office of Scientific Computing, U.S. Department of Energy
Germantown, Maryland

ABSTRACT

The National Research and Education Network (NREN) is a part of the federal initiative to significantly improve the availability and capability of the nations information technology. Both the administrative and congressional arms of the government have identified this as a high priority program. Security of the NREN is one goal of the program. This paper discusses several of the security issues that must be addressed when developing and fielding the physical part of the NREN.

HISTORY OF THE NREN

The National Research and Education Network (NREN) component of the High Performance Computing and Communications (HPCC) Program is composed of two interrelated and complementary subcomponents: the Interagency Interim NREN (IINREN) and the Gigabit Research and Development subcomponent. The IINREN, which is based primarily on DARPA's internet technology, builds on the National Science Foundation's NSFNET, DOE's Energy Sciences network (ESnet), NASA's Science Internet (NSI), and other networks supporting research and education. These network backbones interconnect and peer with the autonomous regional networks (e.g. BARRNET, NEARNET, CERFNET, etc.) which generally provide the "last mile" of connectivity to the research and education community.

The Internet comprises these networks in addition to various international R&E networks and domestic commercial service providers. Both the IINREN and the Internet are "networks of networks". The IINREN backbones currently peer and interconnect at two Federal Internet eXchanges (FIXes), one on the east coast at College Park in Maryland and the other on the west coast at NASA AMES in California. The Commercial Internet eXchange (CIX) is the peering point for U.S. commercial service providers such as Sprintlink, ANS, CERFnet, ALternet, PSI and others. In addition, CIX members interconnect with federal networks at various places for the purpose of exchanging R&E traffic. The IINREN and the regional also connect with other international R&E networks for the purpose of scientific and educational collaborations.

The evolution of the current IINREN, with its peering and interconnection technologies, into the future gigabit NREN will be accomplished with the introduction of pre-competitive advanced network services. The gigabit research subcomponent is responsible for working closely with industry in identifying and performing the research required to sustain the advances necessary to achieve widely deployed gigabit networks and applications by 1997. As these technologies and services become available for "beta testing," they will be integrated into the IINREN.

Addressing the broad and diverse requirements of the NREN constituency requires imaginative technological and administrative solutions. Many of the HPCC agencies directly support their own specific computation and research centers. They also support many Principle Investigators (PI's), students and staff throughout academia who access these resources in addition to other non-federally funded resources. The success of the HPCC will rest on the ability of these administratively and operationally distinct entities to collaborate.

The IINREN and the NREN are a truly distributed network and do not have a centralized management structure. Each HPCC agency,academic institution, library, network service provider, local government and state government has its own set of rules and regulations for negotiating contracts, determining budgets and providing operational support to its staff. In addition, both federal and civilian users of the NREN may include physicists, chemists, biologists, computational scientists, economists, mathematicians, computer scientists, university administrators, librarians, graduate assistants, K-12 teachers, medical professionals, and even politicians; virtually anyone involved with research and education within the United States. Thus any management program, including a security program, is necessary distributed and will rely on voluntary compliance on the part of the participants.

Coordination of the HPCC and NREN activities is accomplished via various mechanisms. Each agency submits its own HPCC budget to the Office of Management and Budget

(OMB) in line with the coordinated activities agreed to by the agencies under the guidance of the Office of Science and Technology Policy (OSTP). Further HPCC coordination is accomplished through the newly formed HPCC Office of Coordination, which has evolved from and oversees the High Performance Computing and Communications and Information Technology (HPCCIT) task force. NREN specific coordination is currently performed in the Federal Networking Council (FNC) and its working groups, an evolution of the Federal Research Internet Coordinating Committee (FRICC), who coordinate the agencies' activities on issues such as policy, security, education, engineering and operation of the IINREN. The FNC advisory committee (FNCAC) includes distinguished members from industry, academia, and other groups interested in the NREN. They meet with the FNC at least twice a year to discuss NREN topics. In addition, the participating NREN federal agencies sponsor and attend many workshops and public meetings on issues relevant to the NREN and its evolution.

The NREN charter is to support unclassified research and education activities. As such, it is not intended to carry any sensitive or classified traffic. However, Public Law 102-194 and the "Grand Challenges 1993: High Performance Computing and Communications" both state the need to define, develop and deploy NREN security policy and mechanisms whenever possible. Keeping in mind that many of the NREN users will also be working in other physically and administratively distinct non-NREN domains, any NREN specific policy and implementation (e.g. security) must be developed so that all participants can and will both implement and adhere to it.

NREN SECURITY POLICY

In February, 1992, the National Institute of Standards and Technology (NIST) published NISTIR 4734 entitled "Foundations of a Security Policy for Use of the National Research and Education Network." This document was the result of a research activity by Dr. Arthur Oldehoeft, Chairman of the Computer Science Department at Iowa State University. He was granted a sabbatical leave at NIST and worked under the direction of Dr. Dennis Branstad, NIST Fellow. The objective was to establish the foundations of a security policy that could be used in developing an operational policy for use of the NREN.

Dr. Oldehoeft presented a background on the evolution of the NREN, the foundations for an NREN security policy and a proposed security policy for use of the NREN. Dr. Branstad then refined the security policy, coordinated it with security experts who had developed an INTERNET security policy and submitted the result to the Federal Networking Council (FNC). The FNC executive committee and the entire FNC approved the policy with minor changes. Implementation procedures for the policy are now being developed by the FNC security working group and are expected within the next few months.

The objectives of the security policy include:

1. establishing responsibility for NREN security to the users, managers, administrators, overseers, developers, vendors and service providers;

2. encouraging responsible security practices by all NREN organizations and network participants;

3. establishing guidelines for protecting information, computer systems and telecommunication systems that will:

   a. safeguard the rights of individuals with regard to personal privacy and protect intellectual and industrial property rights; and

   b. be effective and acceptable to both the public and private sectors.

Copies of the policy are available from the Federal Networking Council secretariat and the National Institute of Standards and Technology.

Organizational policy(ies)

From the general Security Policy for Use of the NREN, it is expected that organizations will establish an organizational policy for achieving the provisions of the general policy. Such policies should include assignments of responsibilities to organizational elements by job title and/or by individual names. In additional to a policy, each organization should also establish a security program which includes a security implementation plan (goals, actions, milestones, time table) and a security review program to assess if the plan is being implemented adequately.

Multi-policies

The issue of multiple security policies is raised whenever a secure distributed network is discussed. The purpose of a network is to facilitate communications among multiple entities (organizations, computers, people) whenever desired. The purpose of security is to facilitate the desired communication whenever the communication is authorized and then to protect the communication as desired. This is difficult, but doable, when both entities follow the same security policy (e.g., Bell-LaPadula policy, IBM Corporate policy, NIST computer security policy). However, when the two communicating entities have different policies, an unsolved problem is encountered.

Several papers have been written on varying approaches to solving the problem. The European Computer Manufacturers Association (ECMA) sponsored a security group that pub-

lished two technical reports on distributed system security. The first, entitled "Security in Open Systems: A Security Framework" presented a top down view of security and addressed security policies, domains, models, services, facilities and mechanisms. The second, entitled "Security in Open Systems: Data Elements and Service Definitions" amplified and elaborated on each of these topics. The topic of multi-policies in a distributed open system was treated fairly extensively but primarily from the definition viewpoint. Several requirements for an acceptable solution were stipulated but no solution was offered.

The approaches to circumventing this problem can be summarized as avoidance or acceptance. Networks which process classified information cannot be connected to any other network and hence the problem is avoided. Closed systems, rather than open systems, have been used to avoid the problem in some environments. Acceptance comes in several flavors: communicate and accept the risk; accept the fact that you cannot communicate with the organization; or request a level of protection and accept the statement that it will be provided. The latter is sometimes loosely called a discretionary security policy.

An outline of a good solution exists. If a security policy can be stated in a formal manner and if a secure computer system has a way of accepting the formal security policy and automatically building a trusted system that enforces that policy for a stated process or processing period, then security policies can be prepended to data and data processing requests and transferred within a distributed system for remote processing. The corollary of transmitting a compiler needed to translate a program along with the program for remote compilation and execution was defined nearly 30 years ago but never achieved acceptance in either theory or use. A more acceptable corollary is including data descriptions and definitions along with the data for remote processing.

IMPLEMENTATION PLANS

NREN Security Workshop

The National Science Foundation networking group requested that NIST sponsor a workshop to address the security of the NSFNET supercomputer centers and the NREN. This workshop was held at NIST on July 6-7, 1992. A report on this workshop has been published by, and is available from, NIST.

Near term solutions

The goal of the workshop was to develop a set of recommendations for near term solutions to security problems in the NSF network. The focus was on technologies that can be implemented immediately with specific attention given to enhancing the security of the NSF supercomputer centers - recognized as valuable resources on the NSFNET. The workshop concentrated on four topics: user authentication, access control, application security and security management, all in a distributed system environment. The following summarizes the recommendations that were established in these four areas.

In formulating recommendations, the participants considered a number of factors: currently available off-the-shelf technologies (hardware and software); interoperability; ease of use; cost; acceptability to the user; and, secondarily, exportability. The recommendations are intended to complement each other and should be selected across the areas so that the effective level of security increases as increased levels of security mechanisms are implemented.

User Authentication (Priority A)

- Avoid use of static (i.e., reusable) passwords
- Use a challenge/response system until Kerberos is available
- Use a public key based authentication system
- Use a publicly available authentication algorithm
- Use an exportable authentication algorithm
- Allow for user access from multiple sites, including from across international boundaries
- Allow implementation in software, smart tokens, or special hardware

Access Control (Priority B)

- Use Kerberos (Version 4 now with transition to Version 5)
- Review use of DASS as its development proceeds
- Assure conformity with application access control (e.g., rlogin, tenet, ftp)
- Support alternative ways of generating/providing Kerberos keys (e.g., user passwords/passphrases, software, EEPROM, smart cards)

Application Security (Priority B)

- Implement and use Privacy Enhanced Mail (PEM)
  - PEM should be available for users to protect mail (integrity, confidentiality, signature)
  - Initially use existing suite of cryptographic algorithms (DES/RSA/MD2/MD5)
  - Explore later use of proposed NIST suite of cryptographic algorithm standards (DES/DSS/SHS)
  - Support and use the certificate registration authority infrastructure
  - Initial applications of PEM should include security administration/management

Security Management (Priority B)

- Establish security responsibility (e.g. security officers)
  - Develop and maintain site security policy and procedures, including a policy for handling security incidents

- Support comprehensive security education programs for users
- Establish Forum of Incident Response and Security Team (FIRST) points of contact

• Establish configuration control for security purposes (priority A)
- Distribute/support use of automated security management tools (e.g. COPS)

• Establish a security perimeter protection capability (Priority B)
• Establish security audit information collection and review capability (Priority B)

Follow-on Activities

• Maintain cognizance of new security technology
• Establish security enhancement specification / implementation teams
• Establish follow-on activities such as additional workshops in specific security areas (e.g. signature certificate registration authority infrastructure)

Proofs of Concepts

Each of these recommendations can be implemented today to a certain degree. For example, a security management plan can be established for meeting the basic provisions of the NREN security policy within a short period of time. Kerberos is available from several sources and supported on many different networks. It provides a turnkey authentication and access control capability for a network but requires people to manage it. PEM is implemented in prototype systems but has not been fielded as a supported product.

More effort is needed in demonstrated proofs of concepts in distributed system security. A true distributed system is one in which a task cannot be completed at one physical location but requires resources at more than one location. A secure distributed system thus requires cooperating security mechanisms in more than one location. A task can be initiated at various places in the distributed system with the necessary resources made available when authorized. This requires several types and levels of standards. Authentication and access control standards are types and each can provide various levels of protection. A level appropriate for the application needs to be selected by the initiating user or process.

Kerberos is a distributed security system which utilizes cryptographic keys generated in one facility for transmission to other facilities in order to open logical security locks. The access control procedures built into Kerberos provide authentication service but not authorization service. Authorization is a function of each host. Access control is based on user/process identification information to determine host access privileges. Local access control must be provided to protect the local processing and data resources.

Impediments To Adoption

Export control(s)

The issue of export controls on security devices, especially those based on cryptography, arose several times during the workshop. Kerberos uses DES for authentication and integrity assurance but does not use DES for data encryption. One reason is export control. Security products that implement DES for data encryption cannot be exported from the United States except for limited applications (e.g., banking) in limited geographical areas. For international interoperability, especially for remote access by a U. S. user travelling overseas, it is highly desirable to use exportable algorithms and security devices. Thus export controls impede the implementation and use of the DES in security systems.

Conflicting security policies

The problem of multiple security policies was discussed previously. This problem assumed that the policies were simply different rather than conflicting. If the policies conflict (i.e., are either totally or partially mutually exclusive), the security mechanisms probably cannot be modified to accept and enforce a conflicting policy. Hierarchical policies may be able to be enforced given that the nomenclature and specification languages for the policies are unambiguous and universal. Disjoint policies that neither overlap nor conflict need to be investigated for implementation and enforcement in a distributed system.

Distributed management

The NREN is a distributed system with distributed management in the extreme. There is no hierarchical management structure nor any central coordination point (except for the Federal Networking Council for the federal portion of the NREN). The Internet has worked because it is in the best interests of all participants to make it work. In the area of security, managers and users must decide what protection is reasonable for their resources (data and processing capability) and assure that the protection is provided. Some security services can be provided on a selected basis but other services (e.g., assurance of network communication capability - denial of service prevention) may have to be pervasive in the network in order to be effective. A flat distributed management structure may be an impediment to providing this type of protection since the cost cannot be allocated easily.

Lack of priorities/resources

The NREN will probably be subject to a shrinking federal allocation of funds to support long term research. While the recently elected vice president was a sponsor of the bill which established the HPCC (and NREN) program in the Congress, emphasis of funds allocation will probably be on those areas which demonstrate immediate improvement in social and economic well being, especially if new job oppor-

tunities can be demonstrated. To date, development projects in security standards for computer networks have not been well funded. Resources must be allocated according to some set of priorities to support the needed efforts. Lack of these priorities and resources will prevent the development of needed standards. Lack of these standards will be an impediment to full utilization of the NREN in the highly competitive society anticipated for the future.

SECURITY TECHNOLOGIES

Available Technologies

The workshop dwelt on available technology in several areas. Privacy Enhanced Mail (PEM) and Kerberos were two of the areas. Steve Kent, BBN, has been chairman of the Internet group developing security RFC's for several years and gave an overview of PEM. Jeff Schiller, MIT, has been involved with the development of Kerberos for several years and discussed its present status.

PEM is defined by a set of four documents in draft form but intended to be published as Request for Comments (RFC's). PEM is a mail system consisting of several components. An editor is used to compose a message, the message is processed by the PEM program to provide the desired security, and the resulting privacy enhanced message is transmitted by the normal mail system. The receiving mail system notifies the addressee of available mail. The person then requests access to the message and requests the PEM component to open the secure envelope. The receiver then can read and process the now unprotected received message.

The security features include: data origin authentication, connectionless integrity and optional confidentiality protection. A "basis" for nonrepudiation of the sender is provided but supporting management services are required to provide the full nonrepudiation service.

PEM is expected to be implemented either as a stand-alone component that is activated between the operations of composing a message and sending the message or an integrated component in a mail system that includes a user agent and a transfer agent. The operations are equivalent in both. Interoperability among all implementations is a goal of the program.

Kerberos was developed at MIT as a part of the Athena computing environment. It also consists of several components implemented in different facilities (protected physically and logically at different levels). A Key Distribution Center (KDC) requires the highest protection while client workstations require less protection. The DES algorithm is implemented in software in all components and used as the basis for protecting keys and "tickets". The goals addressed by Kerberos are detection of spurious association initiation (authenticity), detection of message stream modification (data integrity) and prevention of disclosure of message contents (confidentiality). Traffic analysis and denial of service are not addressed in the Kerberos system.

The design criteria for Kerberos include no cleartext passwords transmitted over the network, no cleartext passwords in client servers and minimal exposure of cryptographic keys (through encryption and minimized key lifetime).

Kerberos is deployed and supported technology. Version 4 of Kerberos is presently distributed (from MIT free of charge) and Version 5 is in Beta test. DEC supports Kerberos in its ULTRIX system but has disabled any DES encryption capability for export control purposes. Kerberos is reported to operate reliably and to be user friendly. The ambiguity of naming conventions (actually multiplicity of conventions) has caused some delay in expanding Kerberos to new open systems environments.

Future Technologies

NIST technical staff members gave several presentations on research topics that may result in technologies useful for enhancing security in the future. Jim Dray described the Advanced Smart Card Access Control System, a cryptographic based authentication system implemented in smart cards and host computers. Each host supports an interactive login and each workstation has a smartcard reader/writer (R/W). The firmware in the smart card implements a set of commands which initiate and respond to interactions with the host computer. The smartcard presently is based on DES but public key cryptography is being added. A challenge/response protocol is implemented for bidirectional authentication. One card can grant access to up to 100 remote hosts if the user is so authorized. Smart cards range in cost from $10 to $100, depending on capability and quantity purchased, while the R/Ws range from $275 to $500.

While the goal of the workshop was available technology and smartcards did not satisfy that goal, the participants concluded that this technology should be developed and supported as future technology. The widely distributed NREN will require something that is very flexible, very inexpensive and very user friendly in order to be accepted. Smart cards have the necessary potential but costs will continue to be an impediment.

NIST staff also gave an update on the Digital Signature Algorithm and its status as a Federal Information Processing Standard. The DSA is a useful security tool for authentication and data integrity assurance purposes. A Secure Hashing Algorithm is also proposed as a FIPS by NIST for use with the DSA. A digital message, program, data file or

image can be input to the DSA which computes a 160-bit hash value that depends on the entire input (both contents and position). The 160-bit hash value is input to the DSA which computes two 160-bit values (r,s) which constitute the signature. The signature is stored or transmitted with the data. When the data are to be verified, they are input to the DSA along with the (r,s) values. The DSA then determines if the data or the (r,s) values have been modified. A private signature generation key is used to compute the signature and the matching public signature verification key is used to verify the signature and the data. The workshop participants believed that while the algorithms were currently specified, implementations of the algorithms were not commercially available and hence were not recommended for immediate deployment.

Trusted systems were discussed a little. While important, it was felt that their widespread use in multi-security policy domains was still in the future.

FUTURE GOALS

Designed/Integrated Security

For many years security experts have recommended that security services, mechanisms and facilities be designed into a data processing system early and integrated into the basic structure of the system. This recommendation still holds. Open distributed systems should have security as a fundamental design criteria and users must demand security in the systems before procurements are completed. This goal will then be satisfied to a greater degree than is presently happening.

High speed security mechanisms

The HPCC program, within which the NREN is being developed, is intended to field very high performance processing and communication capability. Security mechanisms always incur some overhead in computing and communications. In some cases, the overhead is either so small it is not discernable or can be performed in an off line mode with very little on-line delay. However, there is a need to develop high speed security mechanisms (e.g., data encryption, access authorization, security policy routing algorithms) that will not incur unacceptable overhead when the mechanisms must be installed in-line. Cryptographic devices operating around 100 megabits per second are being designed but will not operate at the gigabits per second range. Cryptographic algorithms generally require significant data manipulation or arithmetic computation in order to achieve high work factors against specific security threats. More research and development is needed in this area.

Evolution/transition plans

The NREN program anticipates transitions from the existing networks to the high performance networks desired. The NREN will evolve from present capability with bits and pieces upgraded as funds and equipment allow. The security program for NREN must similarly evolve, going from basically management based security procedures to technology based security mechanisms. Existing equipment and operating procedures will not be abandoned because of security concerns or desires. Even if the existing network breaks (or is penetrated much more often than currently exists), it will not be fixed by a complete replacement of existing resources. Plans for the inevitable transition must be made now.

Educational plans

The first provision of the security plan is to educate the users, managers, administrators, vendors and service providers of the network in the area of security. Assignment of responsibility, and notification of the assignment, are first on the list. Tutorials on existing security technology that can and should be used will be next. Most people are aware of potential problems (viruses, integrity losses, unauthorized disclosures, theft of computation capacity) but few incidents occur with respect to the potential. Therefore, people accept the residual risk at a high level because acceptance is easier than action. However, this should be an informed decision based on good facts and good education of the network participants.

Priorities/resources

Improvement of security in the evolving NREN will primarily be a matter of priorities and resources. The federal government will continue to sponsor research in high speed computing and communications with emphasis on research and education but resources will be allocated to priority programs with the highest positive payback. Security is sometimes thought of as preventing negative feedback but rarely thought of as having positive payback. However, if information technology is not utilized on a broader and grander scale, the lack of security provisions will prevent acceptance in many applications where great risks are encountered and this high ticket item will not develop to its full potential. Security should be considered in each of the exercises in which priorities are set and resource allocated.

ACKNOWLEGEMENTS

The authors wish to thank all the participants of the Workshop on NSFNET/NREN Security sponsored by the NSF and hosted by NIST. Specifically, we would like to thank Dr. Steve Wolff for sponsoring the workshop and Drs. Vint Cerf, Steve Crocker and Steve Kent for their leadership roles in the workshop. We would also like to thank Dr. Arthur Oldehoeft for his work on the NREN security policy and for his report of the security workshop.

REFERENCES

Grand Challenges 1993: High Performance Computing and Communications; A Report by the Committee on Physical, Mathematical and Engineering Sciences; Federal Coordinating Council for Science, Engineering and Technology.

High Performance Computing and Communication Act of 1991 (Gore); Public Law 102-194.

Interagency Interim National Research and Education Network Implementation Plan; Aiken, Braun and Ford; National Science Foundation.

NISTIR 4734, Foundations of a Security Policy for Use of the National Research and Education Network; Arthur E. Oldehoeft, National Institute of Standards and Technology.

Report of the NSF/NIST Workshop on NSFNET/NREN Security, Arthur E. Oldehoeft, National Institute of Standards and Technology.

Security in Open Systems: A Security Framework, European Computer Manufacturers Association, 1988.

Security in Open Systems: Data Elements and Service Definitions, European Computer Manufacturers Association, 1989. Security Policy for Use of the National Research and Education Network; Federal Networking Council.

Security Policy for Use of the National Research and Education Network : Federal Networking Council.

SECURITY AND MANAGEMENT IN IT2000

Seow-Hiong Goh
Yeow Meng Chee
Michael Yap

Planning and Infrastructure Department
National Computer Board
Singapore

ABSTRACT

The IT2000 project aims to use Information Technology in making Singapore an Intelligent Island by the year 2000. This paper discusses some issues on Security, Privacy and Management in the context of an on-going design and implementation of a National Information Infrastructure (NII) the vehicle for realising the Intelligent Island vision. The topics focused in the paper include the problems faced in the protection, security and privacy of data, the provision and accessing of services, and the management of the principals.

INTRODUCTION

Singapore has embarked on a National IT2000 plan, which aims to set up the country as an Intelligent Island. It heralds a brave new world in which Information Technology will play a pervasive role in the lives of the people. This paper will discuss some technical issues and experiences concerning security, privacy and management encountered in our on-going design and prototyping efforts of the National Information Infrastructure (NII).

BACKGROUND

The National Computer Board (NCB) initiated the IT2000 Study in January 1991 to examine how IT can be effectively exploited to create national competitive advantages and to enhance the quality of life. The goal is to transform Singapore into the Intelligent Island where an advanced national information infrastructure interconnects every home, office, school and factory. In our vision, the computer will evolve into an information appliance, combining the functions of the telephone, computer and television, to provide a rich array of communication modes and access to services. Text, sound, pictures, video, documents, designs and many other forms of media will be transferred and shared through the infrastructure which will be fibre optics based reaching all homes and offices, and with a pervasive wireless network working in tandem.

The approach taken at the onset of the IT2000 has been very much demand driven. The first stage of the effort calls for an extensive study of the needs and strategic application of IT in various sectors of Singapore over the next decade. 11 sectoral study groups, comprising some 200 knowledgeable and influential senior executives and academics from both the private and public sectors, were formed. After months of intensive study, the groups have surfaced more than 60 major projects that will help bring the nation to the next phase of its development, and whose implementation will be facilitated by the availability of the national information infrastructure. Studying the proposals of the all the sectoral groups, we recognised a set of common themes that collectively describe the vision of the Intelligent Island. The 5 themes include [1]:

- *Developing a Global Hub.* The infrastructure will help turn Singapore into a highly efficient switching centre for goods, services, capital, information and people, enhancing its role as the global business, services and transportation hub. Potential applications that would support this theme include remote collaborative work, remote delivery of expertise, integrated transportation networks and electronic marketplace for information services.

- *Boosting the Economic Engine.* The potential benefits of the infrastructure to the economy are immense, generating greater productivity and new business opportunities. It will help to strengthen inter-firm coordination, both within and across national borders, and enable flexible and just-in-time business operations. Applications suggested include integrated networks to support the exchange of documents, payments and drawings, and a nation-wide multimedia leisure

information and reservation system.

- *Enhancing the Potential of Individuals.* By providing more opportunities and bringing education to the convenience of the individuals, we hope to facilitate interactive and self-paced learning. Focus will be to greatly improve access to cultural digital repositories in Singapore and around the world, and to provide extra help for the disabled. Applications suggested include multimedia training, distance learning and adaptive computer technology.

- *Linking Community Locally and Globally.* The infrastructure can facilitate the creations of various community-based networks linking people with common causes or interests. It will help to support civic and cultural networking, and with overseas extension, helps to promote international goodwill.

- *Improving the Quality of Life.* The emphasis here is to enrich the lives of the people by increasing discretionary time and generating more opportunities and choices in the kinship, social, work and civic spheres. It will help facilitate living in an increasingly more congested, vibrant and cosmopolitan Singapore by shielding some of the complexity. Applications suggested include one-stop, non-stop government and business services, cashless transactions, dynamic transport information, round-the-clock health care information and telecommuting.

The National Information Infrastructure (NII)

In conceiving the development of the infrastructure, it might be prudent not to fully entrust the invisible hand to guide an otherwise uncoordinated evolution. The information infrastructure is so all-pervasive, and its needs for communication and computing standards so critical, that it might not arise spontaneously and optimally. The high financial investment and the possibility of even higher costs needed to rectify mistakes lend support to the need for a disciplined approach in planning and developing the national information infrastructure, particularly, for a resource scarce nation-state like Singapore.

For the competitiveness of the nation, the challenge is therefore to approach the issue in a holistic manner, by providing a nation-wide information network. It is an infrastructure consisting of efficient transport mechanism, information processing and service facilities that combine both computer and communication technologies. From it anyone, anywhere, at anytime could easily, quickly, and inexpensively get and share information services that they want. The needs of business and the people drive the definition of the infrastructure. Its goal is to increase the well-being of the people as a whole.

The government would be required to play a leading role in facilitating the realisation of the Intelligent Island. However, it must be emphasised that it is not the intention to centralise and monopolise all information services. The building of the national information infrastructure is to promote greater innovative use of information. It is a concept established to make it easier for individuals and enterprises to provide information services and exchange of information, bringing Singapore closer to the enjoyment of the full benefits of the information age.

What then is the NII? Below (Figure 1) is a reference model of the NII. It shows the major components of the NII. Applications are specific end-users systems which are constructed using common *Application Models* that are built over facilities provided by the *Common Services*. The layer known as the *Distributed Computing Architecture* abstract developers from the complexity of having to handle heterogeneous systems. Refer to [2] for a more detailed description of each component.

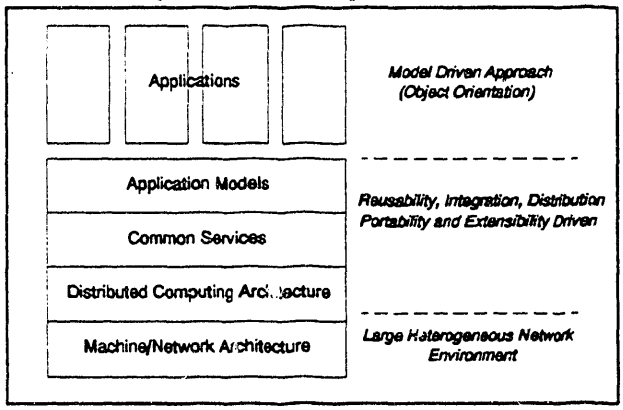


Figure 1. NII Reference Model

The NII is a distributed environment supporting a heterogeneous network of servers. The computing servers providing various services will be interconnected via a high speed network. Users access to the system will be through the public network system. Wireless connections at a lower bandwidth will be supported particularly for access by end-users. Support for user access will include devices with no processing capability (apart from supporting display). In the later subsection on *Types of Access Devices*, we discuss in greater details the type of access devices that we would be supporting.

Project Status

Apart from establishing the various organisational mechanisms to attain strong support and collaboration, work to examine the technical issues in developing the infrastructure has begun.

Various groups has now been formed to carry out research and develop prototypes of the network infrastructure and the software architecture of the NII. Efforts are now underway to define the technical reference models and functional specifications of the NII. To gain further understanding of the technology requirements and to assess the implications of application demands, prototypes are being constructed. Together, the reference models and prototypes will serve as a common definition and understanding of the infrastructure.

In evolving the physical infrastructure, an iterative approach is being used. It is recognised that the envisioned national information infrastructure is untested and should not be built using the "big-bang" approach, instead it needs to be prototyped and evolved. It is also recognised that Singapore is behind the technology leaders in various enabling components of the infrastructure. An importance thrust of our effort will therefore be to tap into the local R&D effort and to foster strategic alliances with various international technology sources.

The technology acquisition effort will focus on developing a national test-bed for experimenting with infrastructure feasibility and to demonstrate application viability. The current plan is to develop a fibre-based network as a backbone for experimenting with the infrastructure. There is also plan to build prototype showcase applications on the backbone to study their demands on the infrastructure and on how they may be integrated.

Scope of Paper

While there are many issues regarding how distributed computation is achieved in the NII, this paper will focus mainly on the Security, Privacy and Management issues and the mechanisms chosen for the implementation of the NII. The topics addressed include:

- Basic components for Security and Privacy
- Providing Services
- Accessing of Services
- Authentication and Authorization Mechanisms

DEFINITIONS

The following definition of Security, Privacy and Confidentiality is adapted from the definitions of the National Bureau of Standards and the Association for Computing Machinery [15]. These definitions will be used as a basis for our discussion on how we aim to achieve each of them in the NII.

Security is defined as follows:

> *Security is the protection of data against accidental or intentional destruction, disclosure, or modification. It refers to the technological safeguards and managerial procedures that can be applied to computer hardware, programs and data to assure that organizational assets and individual privacy are protected.*

Privacy is defined as follows:

> *Privacy is a concept which applies to an individual. It is the right of an individual to decide what information he wishes to share with others and what information he is willing to accept from others.*

There may not be a direct relation between Security and Privacy. A secure system does not necessarily ensure the privacy of its users. Information sharing may result in a conflict with the concept of the privacy of users. We must therefore achieve a system that provides a balance between data sharing and privacy.

Confidentiality is defined as follows:

> *Confidentiality is a concept which applies to data. It is the status accorded to data that has been agreed upon between the person or organization furnishing the data and the organization receiving it and which describes the degree of protection to be provided.*

Principal, Group and Role

A *Principal* is defined to be a person, a group, a machine or a service. It is an entity that is unique in the NII that holds certain attributes and resources.

A *Group* is a collection of Principals with some common characteristics. A group in itself is also a Principal.

A *Role* is a given set of authority in which a Principal acts in. An authority is defined by a set of role certificates. A role that requires additional attributes and resources is considered to be a separate Principal. A Principal that has

multiple roles will have to select the role in which he wishes to act before accessing any services.

Authentication and Authorization

Authentication is the process of verifying the identity of a principal. *Authorization* is the determination of whether a principal is entitled to use a protected resource [8].

Principal Roles

A given principal may have a number of different roles. For example, he may be a private citizen and a student, or he may be an employee of an organization and also the system administrator of the computers in the organization. A role allows a principal to do or not do something. It is thus represented in terms of a set of role certificates that the principal has.

Principals are associated with particular roles by obtaining the relevant role certificates from the Authorization Grantor. When requests for services are made by the principal, these certificates are used by the Grantor during the generation of authorization tickets to determine if a principal is entitled to use a service.

A person may own a large number of certificates and tickets due to the many roles that he assumes and the services he accesses. The principal will have to select the role in which he wishes to act before he accesses any services. When a Principal assumes a role, he will have access only to the certificates and tickets that are valid for the given role. Therefore, it is not the entire collection of authorizations that a principal has, but rather the set of authorizations he has *for a specific role* that he plays that determine what services he can access.

Each principal has two levels of "roles" - the private citizen level role and a specialized level role. An authorization to access a service that is available to the principal in general is assigned at the private citizen level. A role specific authorization is assigned at the specialized level. When a principal assumes a specialized role, he can use the authorization available to the role as well as the authorizations available at his private citizen level. The division into two levels of roles (each level supporting a number of different roles) relieves the principal from the need of obtaining separate authorization for some basic operation for every role that he may assume.

For example, if a person is categorized as under-18, he may be given an authorization ticket for viewing movies with this restriction. This ticket is placed at the private citizen level. If he also has the additional specialized role of a student, he may be given the authority to use a library. The library authorization ticket is placed at the specialized level. When this principal assumes the role of a student, he may use the library, as well as watch movies in the under-18 category. When he assumes the role of a generic private citizen, then he may only watch under-18 movies, but not make use of the library.

Directory Service

Our Directory Service is based on X.500. The Directory Service in the NII serves several important purposes:

- *Principal Directory*

  The primary purpose of the Directory service is to allow one principal to locate another principal based on some other known attributes regarding the principal. It also allows a principal to locate a Service or Role.

- *Principal Database*

  The collection of the principals' frequently used information is stored with the database that the Directory Service uses. The principals' name-value attribute pairs and certificates are stored at this database, though the value of the attribute may not necessarily be stored at the database for security reasons. A pointer to the storage location may be used instead.

- *Public Key Directory*

  The Directory Service in the NII doubles as the Key Directory in the NII. The Public Key of a principal is stored as an attribute of the principal.

Directory Service and Roles

As users may prefer associations to roles eg. "librarian" rather than to the actual name of the person who is the librarian, the Directory Service must allow for such a mapping between Roles and principals.

Within the Directory Service, we have a list of Principals, each with his own mailbox for receiving mail. These include Groups that contain a list of principals that perform a role collectively. In the Directory Service, we also have a list of Role names which users can use as a search key. A role entry is an alias to a Principal (including groups).

When a principal wishes to contact a particular role

through the Directory Service, there are two possibilities. The role may refer to a particular person (eg. the General Manager), or it may refer to a group of people (eg. the Librarians). In the case where it refers to a particular principal, the role entry is an alias to the Principal, and any mail directed to the role is automatically routed to the Principal's mailbox.

In the latter case where a role refers to a group of people, this may have two further possibilities. A message to the role may be meant for everyone in the group or just any *one* in the group. In the former case where the message is meant for everyone in the group, the role entry is an alias to all the member principals. If it is meant for anyone in the group, then the role entry points to a Group Principal that has its own mailbox. Members of the group can then log themselves on as the Group Principal to access the mailbox.

BASIC SECURITY COMPONENTS

The data available from various organizations that participate in the NII will require different levels of security protection. The nature of the transactions requested by the principal will also need to be secured. At the onset, the NII is designed to include a basic communication security mechanism so that all communication between principals on the network are protected against various forms of attack. We have a system based on both Public-Private Key and Single-Key technologies for providing secure communication.

In the subsections that follow, we will describe how we protect our data through the use of encryption. The authentication procedure is described later in the section on *Authentication*.

There are two basic premises that we base our work on:

- A Service Provider will not trust a client, unless the client has been authenticated.
- A user generally will not trust a machine. Such a trust is available normally only to machines personally owned by the user. The exception is through the use of Smart Cards. This will be described in greater detail in the section on *Authentication*.

Public Key Encryption

It is assumed that the reader has some working knowledge of the Public and Private Key Encryption techniques. This paper will not describe in great detail the Public Key mechanism or the relative merits of the system over a Single Key method. Details regarding Public Key systems can be found in [14,16].

In the NII, each principal is issued with a pair of Public and Private Keys. The Public Key of a principal is stored with the Directory Service, while the Private Key is kept with the principal. There is a central agency that is responsible for generating new keys and issuing the keys to principals so that two principals do not end up having the same set of keys.

Hybrid Encryption System

The main disadvantage of Public Key systems is the slow speed of encryption. The mechanism chosen for the NII is thus a hybrid Public-Private Key system and single-key Secret Key system.

For stream data, the Public Key system is used at start up time to establish an initial secure channel between the communicating parties. Once the channel is established, the communicating parties will agree on a random number to be used as the session key. Subsequent messages in the session will be encrypted with faster Single Key systems.

For block data, the sender decides on a random number to use as the session key to encrypt the bulk of the data with (using Single Key systems). This key is then encrypted with the receiver's public key and the encrypted key is sent to the receiver with the encrypted body.

Encryption Algorithm

For the security system, we are experimenting with various schemes, including the RSA algorithm [3] (with a 200-digit key) for encryption with the Public Key System and DES [4] (with a 56-bit key) for encryption with the Single Key system.

For Digital Signatures, we are examining MD5 [5] as the hashing function to generate a 128-bit message digest. This digest is in turn encrypted with the RSA Cryptosystem [3]. Digital Certificates are also generated with RSA.

We have tried to conform to the Public-Key Cryptography Standard (PKCS) [6] wherever relevant.

Key Storage

The preferred storage means for the principal's Private Key is a smart card. Among other things (see following subsection), one advantage is that the smart card with its

embedded processing capability can perform encryption using the Private Key. In this way, the principal can be certain about the secrecy of his Private Key, since it never have to leave the card.

In the absence of a smart card, the alternative is to store the Private Key in an encrypted form on a magnetic card, or even on a magnetic diskette. The encrypted Private Key may also be stored on the principal's personal machine which he usually uses for access to his sensitive data. The Private Key is encrypted with a Single Key Encryption system, protected by a password known only to the principal. Without a smart card, encryption of data with the principal's Private Key will therefore be performed by the local machine which the principal is logged on at.

Trust of Machine

It is important to note that without the use of a smart card, the principal must *fully* trust the machine that he is using. To prevent any unauthorized principal from easily retrieving the principal's Private Key from his magnetic card, diskette or personal machine, the Private Key is encrypted and can be obtained only with a password which the principal must enter during authentication. However, the Private Key will still be made known to the machine the principal is using (since the machine has to perform the encryption on his behalf). To entrust his Private Key to the machine, the principal must trust the machine sufficiently.

Unforgeable Smart Cards

Creating an unforgeable identification card is important for many applications in IT2000. We want to make it impossible for two adversaries to misrepresent themselves to each other with an unforgeable card. In addition, such cards are protected against loss and theft.

A smart card has the capacity to process information internally and store all transactions in non-erasable memory. Thus the smart card has a high degree of security and extended performance capabilities. We can use a protocol devised by Fiat and Shamir [7]. The scheme uses a trusted agency which issues the smart cards to principals after properly checking their physical identify. No further interaction with the agency is required either to generate or to verify proofs of identity. Interaction with the smart cards will not enable verifiers to reproduce them. Even complete knowledge of the secret contents of all the cards issued by the agency will not enable adversaries to create new identities or to modify existing identities. Since no information whatsoever is leaked during the interaction, the cards can last a lifetime regardless of how they are used.

To protect the card from theft, we can use a biometric system with which the cardholder's digitized fingerprint is stored in the smart card at the time of issue by the trusted agency. When the card is used for identification, the card presenter's fingerprint image is automatically compared with that stored on the card. If the images match, then the verification protocol by Fiat and Shamir is carried out. Using this method, even if a card is stolen, an adversary will not be able to use the card because his fingerprint image is different from that stored on the card.

However, using a biometric system is not always viable because of costs. A more practical alternative is to assign each card owner with a personal password. This password can be presented to the smart card in the place of a fingerprint for verification. In our later discussion on the use of smart cards for authentication, we will assume the use of a password system.

Types of Access Devices

With the proliferation of end-user computing and the increasing "intelligence" of various "household" appliances (eg TV and HDTV), the range of access machines to NII is potentially wide. To protect existing investment and to lower the cost of entry to users, the goal of NII is to support as many types of access devices as possible and to provide the connection in software.

Since there are wide differences in computing capability among the potential access devices, the functionality accorded to them will necessarily be different. The challenge is maximise the inherent power of the access device in a manner that is transparent to both the end users and developers. The users should not have to worry about the how the machines are being optimised. And the architecture should shield developers from having to wrestle with the differences among the devices.

To help us manage the wide spectrum of access devices, we have divided the interoperability capability that an access device have into various distinctive classes. However, for the purpose of this paper, we will only classified them into two classes:

- Display and User Interface only hosts.
- Hosts with additional local computational power.

This distinction between the availability of computational power at the local site is important as the NII security model requires encryption of data to be done at the local machine. At no time is the Private Key or password of the principal ever transmitted over any communication link. The principal must trust his local platform to encrypt his

data.

As a result of the above criteria, the Display and User Interface only hosts (eg. a public-access information terminal) cannot allow the principal to access any sensitive information, since the terminal has no capability of authenticating the principal. The only exception to this is if the public terminal is equipped with a Smart Card reader. In this case, the card may perform the required encryption.

The actual authentication procedure is described in greater detail in a later section on *Authentication*.

BASIC PRIVACY COMPONENTS

A closely related topic to the security of data is the Privacy of data. With the NII, information will be readily available to the users. There is therefore a need for the control of access of a user's private and personal information. Authorization procedures must prevent an unauthorized access or change to private information. In addition, apart from ensuring that the principals have the proper authority to access certain information, the principals themselves may not wish to receive certain services or information. The privacy mechanism must be able to provide both types of protection.

Following from the definition of Privacy described earlier, there are two things we need to protect:

- what information is shared.
- what information is received.

Protection of Information Shared

The topic on the protection of information sharing involves the following areas:

- *Proper Access Control.*

  We aim to partly achieve privacy of shared information through the enforcement of access control. Proper authorization must be obtained for the access of any information. The subsequent sections on *Providing Services*, *Accessing Services* and *Authorization* will go into greater detail on how the authorization procedure is used to help provide Privacy in the NII.

- *Protection of Information Abuse.*

  Authorization and access control alone do not provide the principal with full privacy of information shared. A legitimate user of information may still intrude on the privacy of another principal if the information provided to him is not used in the manner it was meant for. At the present time, the most effective way of preventing such an abuse of information provided to principals is through legal means.

- *Limiting Exposure of Information.*

  The aim here is to find ways of answering a legitimate principal's queries without actually revealing to the principal too much information. With common access control, an authorized principal in accessing a service may gain exposure to the structure of information. He is then able to manipulate the information, providing the possibility of abuse. The challenge is to find ways of providing information in a manner that even if a principal has the authority to access a service, he can only get information that he is specifically looking for and not beyond that. Service Providers will have to be selective in the types of information they provide. Zero-knowledge [13] queries is one way of allowing information sharing without actually revealing any information.

Protection of Information Received

The protection of information received is to prevent the principal from receiving undesired services. The NII solution to this problem is closely tied with the process of authorization. The mechanism will therefore be discussed after the discussion on the NII Authorization Mechanism, in the section of *Refusal of Services*.

Authorization Components

The main defence against privacy invasion is through the proper implementation of an authorization mechanism. Our system involves the use of role certificates and authorization tickets.

A *Role Certificate* is a signed string that proves that a principal is entitled to the rights associated with the given role. An *Authorization Ticket* is a signed string that allows a principal to access a service. Figure 2 illustrates the relationship between certificates and tickets. A principal makes a request to the Directory Service. The Directory Service presents the principal's role certificate to the Authorization Grantor. If authorization is approved, the Grantor will give the principal an Authorization Ticket. The principal may then present the ticket to the service for

accessing its services. The subsequent sections on *Providing Services* and *Accessing Services* will explain the mechanism in greater detail.

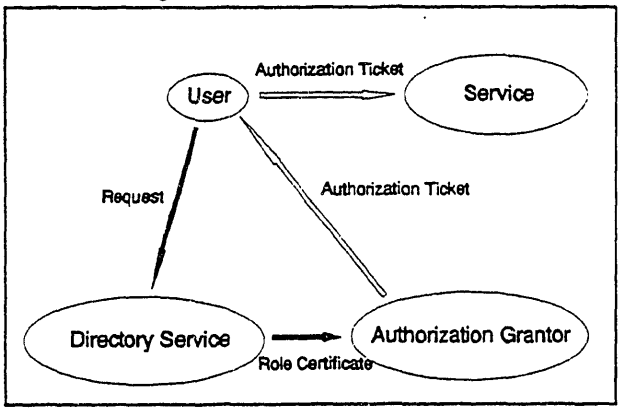


Figure 2. Relationship between role certificates and authorization tickets.

PROVIDING SERVICES

Different organizations may participate in the NII with their own systems for providing services. These systems do not necessarily integrate seamlessly into the NII in areas such as providing authentication and authorization of principals. Since these systems are autonomously managed, the task of the NII will be to provide a bridging mechanism that provides Authentication and Authorization of principals and allow these organizations to offer their services to the general public user. The mechanism that we have adopted is based on a method that is similar to Kerberos [9].

In addition, there is also a need for a mechanism for keeping track of the resources consumed by each principal so that accounting information may be collected and the principal billed for the usage of the services. We make use of a system of agents to assist in accounting.

Agents

Since it is not feasible for every host and service provider in the NII to have a separate account for each individual principal in the NII (due to the large number of users), we have a system of agents that will provide the mechanism for the proxy invocation of services. There are 3 main categories of agents at the system level:

- Proxy Agent (PA)
- Principal Agent (PrA)
- Public Agent (PuA)

The Proxy Agent is used by the service providers to allow NII principals to use their services. The Principal and the Public Agents are used by principals when accessing services in the NII. These Agents are provided by the NII. Figure 3 illustrates the relationship between the agents and the corresponding platforms that they run on.

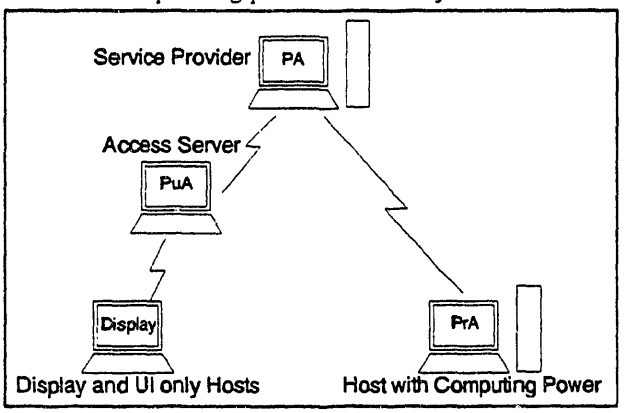


Figure 3. Types of agents

Providing Continuity

Among other things, the NII Agents aim to separate the technological aspects of the NII from the scheme by which services are accessed and provided. The Agent has a standard generic interface that allows the details about implementation to be hidden from the principal. This allows components to be replaced and upgraded as innovations in technology become available, while at the same time, the general paradigm by which the users and service providers make use of the NII is not affected.

In addition, authentication and authorization procedure is largely handled by the Principal Agent. The actual human users or end-principals are isolated from the security procedure details.

Proxy Agents (PA)

Each service provider has a Proxy Agent that coordinates the operations for the host. This agent performs the following:

- Receives requests to be performed on the local machine
- Performs authentication and authorization required by the service where appropriate
- Performs the necessary logging and auditing functions
- Invokes the requested task
- Returns results, if required
- Reports to the Billing and Accounting Agency regarding the usage of principals.

A request that is submitted to a Proxy Agent contains the name of the actual requester and the originating requester. The need for the distinguishing of the actual and originating requester arises from the possibility of one service invoking another to perform its assigned task. The charging for the request should be billed to the request originator and not to the intermediate services.

In situations where an intermediate service who needs to invoke a third service and wishes to bill original principal itself, it can invoke the third service with itself as the request originator. In this way, the third service will bill the intermediate service instead of the principal who made the request.

The service provider can set up the necessary interfaces between the services and the Proxy Agent. The agent will perform all the necessary administrative work such as authentication and logging on behalf of the service. This way, the service can concentrate on providing the service and not worry about checking the principal. The service provider may still perform its own authentication and authorization checking on top of what the Proxy Agent provides if it wishes to. Accounting is done through the Proxy Agent. This allows the NII to validate and audit the accounts of the service providers.

Security Issues of Proxy Agents

The use of Proxy Agents raises some additional security issues. Since the request originator's name is provided by the requester, the service provider must be able to verify the authenticity of the originator's identity. The request originator must authorize the intermediate agents when they perform tasks on behalf of the originator. A ticket issuing scheme is used for the authorization of Proxy Agents.

The originator generates an authorization ticket by encoding the originator's name and the Proxy Agent's name with the originator's Private Key. The ticket is given to the Service when the request is made by the originator. The Service will verify the certificate by decrypting the certificate with the originator's Public Key and checking the contents with the name of the originator and the requester. For nested invocation of services, the next authorization ticket is generated by using the previous ticket as the name of the originator.

Registration of Services

Each service in the NII must register itself with the Authorization Grantor. At the time of registration, access restrictions to the service is made known to the Grantor. For example, a service providing the records of a person's financial background is not available to the general public. Only a specially authorized group of principals (eg. banks and credit agencies) may access the records. Another example is the provision of materials which may be restricted by Age. A person under the age of 18 may not be allowed to access movies rated Restricted. Other information that needs to be made known to the Grantor include the credit limit to give to principals for the particular service. Changes to the access restrictions and other authorization information may be made to the Grantor as and when the need arises. Figure 4 illustrates the Registration of Services.

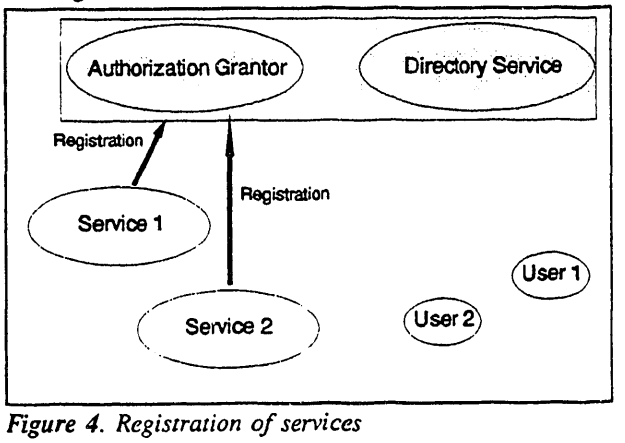


Figure 4. Registration of services

Authorization Granting is decoupled from the Service so that some form of policing on the type of services offered can be done. This requirement of registering services with the Grantor and the ability of the Grantor to negotiate the type of service provided will help prevent illegal services from being made easily available to other principals on the NII. The implementation policies regarding the provision of services can be enforced at the Authorization Grantor. However, this scheme of Service Registration and Authorization Granting does not prevent the provision of services through direct mail mechanisms.

Service Acceptance

After the negotiations between the service and the Authorization Grantor have come to a successful resolution and agreement, the Grantor will issue a set of Public and Private Keys to the service principal.

The Authorization Grantor will subsequently update the Directory Service to include the new service. The Directory Service will also be informed by the Grantor whether authorization is required for access to this service. If the service is available to the general public, the

Directory Service may send out the handles to the service without any reservations. The Authorization Grantor is involved only when a separate authorization of the principal is required. Figure 5 illustrates the Acceptance of a Service.

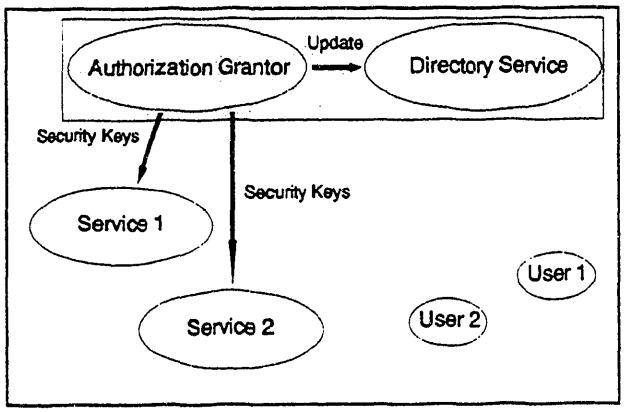


Figure 5. Acceptance of service

The Directory Service also offers an additional service of informing the principals about new services available. Such information is compiled in the form of condensed reports. A principal may request that the Directory Service send information about new services to him at regular intervals. Alternatively, if the principal prefers not to be flooded with such information, the principal may indicate that he will request for the information only when he desires to do so.

Local Autonomy

There is some concern that the requirement of registration of services may affect the ability of organizations to maintain their own autonomy. The NII allows the services to build any additional authorization and accounting mechanisms to enforce their own autonomy on top of the NII's mechanism. The NII does not impose any more restrictions as compared to what is required for setting up of a business in the real world today.

There needs to be a compromise between freedom and privacy. The registration mechanism helps to curtail the provision of illegal services through the NII.

The NII Authorization Grantor and Directory Service are distributed services. In order to have local autonomy, the local equivalent of the Grantor and the Directory Service may be implemented at the local site. These two agencies will work in conjunction with the global NII agencies to allow external principals to communicate with them. At the same time, the principals under the domain of these local agencies may be autonomously managed.

ACCESSING SERVICES

The counterparts of the Proxy Agent in the user world are the Principal Agents and the Public Agents. This section will describe these agents, as well as describe how services are obtained through them.

Principal Agents (PrA)

Each principal is represented by a corresponding Principal Agent in the system. When a host has successfully authenticated a human user at the terminal, a Principal Agent is started at the access terminal to act on the user's behalf.

The Principal Agent has access to the principal's Private Key. Thus the agent is able to perform tasks such as answering authentication requests on the behalf of the user. This Principal Agent is also capable of performing other authentication and authorization functions such as generating a signature or obtaining a time stamp.

A Principal Agent is aware of the roles that the principal may act in. Therefore, at log on time, the Principal Agent will ask the principal which role he intends to assume for the session. The set of authorization tickets and billing information is different for each role that a principal may play. The selection of the role will thus tell the Principal Agent which set of attributes to use for the current session.

Public Agents (PuA)

Public Agents is used in place of the Principal Agent in the situation where the host machine is unable to authenticate the principal. This situation may arise for example with a public access terminal. A person may walk up to a terminal to make a general enquiry about the movies currently being screened. Such requests do not require the stringent authentication and authorization procedure. We thus have a Public Agent to represent the principal and to perform tasks on behalf of the principal. The basic difference between the Principal Agent and the Public Agent is that the latter is never presented with sensitive information, both from the user and the service provider.

Requesting A Service

A principal that wishes to use a service will contact the Directory Service to make a request for the service. The Directory Service will perform the look-up for the service

requested to obtain its handle. If the service is available to the general public, the handle is returned to the principal. However, if further authorization is required, the Directory Service will pass the principal's role certificates to the Authorization Grantor to request for an authorization ticket for the principal (explained further in the section on *Authorization*). The ticket may contain additional information such as the credit limit of the principal.

The Grantor will verify internally whether the principal is qualified to use the service based on the role that the principal has assumed and the service access restrictions determined during the registration of services. If a principal does not know which of his roles may access a service, he may make a different request to the Grantor. The Grantor will present to the principal the list of the roles and ask the principal to make a choice of which role(s) he wishes to use to access the service. If there is no role under which the principal may access the service, the request for the service is denied. Figure 6 illustrates the requesting of a service.

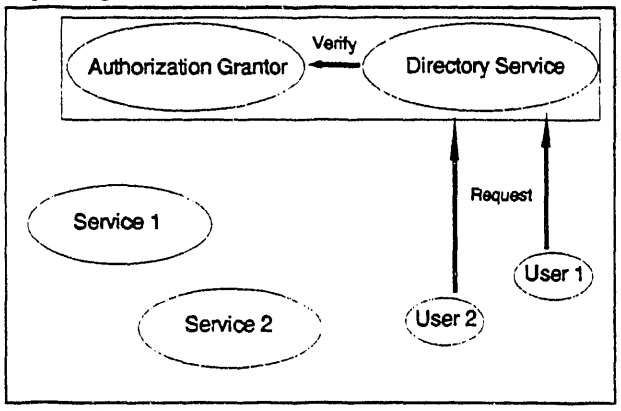


Figure 6. Requesting a service

Service Request Response

If the Grantor is satisfied that the principal may use the service, it will generate an authorization *authTicket* that contains the principal name, the role name, a time-stamp and a pointer to the limit information. This is signed by the Grantor and encrypted with the service's Public Key. The inclusion of the principal's name in the ticket ensures that only the principal may use the ticket. It is not possible for the principal to give the ticket to another. The ticket is signed by the Grantor for non-repudiation of the ticket. Since the ticket is encrypted with the service's Public Key, the ticket is valid only for the specific service that can decrypt the ticket. The authorization ticket is returned to the principal via the Directory Service together with the handle to the service. Figure 7 illustrates the response to the service request.

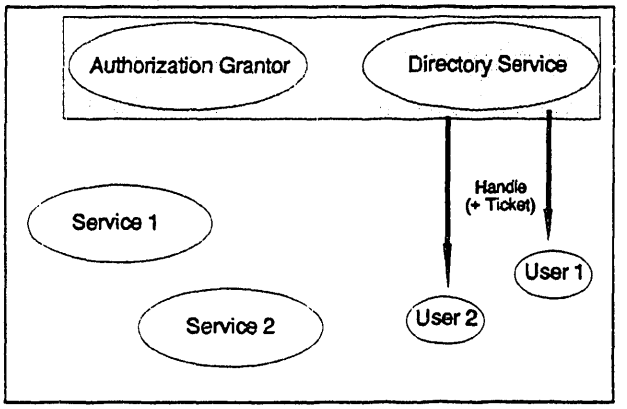


Figure 7. Request response

It is not necessary for the principal to make a request to the Directory Service every time he wishes to access the service. He may store locally the handle (and the ticket where necessary) for the service for subsequent access. A new request needs to be made to the Directory Service only when the ticket for the service has expired (eg. the credit has been consumed, time-stamp has expired or the service has been terminated). Thus for services whose accesses need to be tightly controlled, the time period for each ticket's validity can be set to be a short one.

Utilizing a Service

A principal will use the service's handle to contact the service and if necessary, present its authorization ticket *authTicket*. The service can decrypt the ticket to obtain the principal name, role, time-stamp and the credit limit information pointer. The service will only allow the principal access if the principal name and role in the ticket match the name and role of the principal making the request. The service has to verify through the Billing and Accounting Agency that the limit for usage (described later in the section on *Authorization*) has not been exceeded. Figure 8 illustrates the utilization of a service.

The following is done by a Service when access is made available to a principal:

- Verify the limit status of the principal.
- Log the usage and the resources consumed.
- Update the limit status of the principal after usage.

These tasks are generally done by the Proxy Agent on behalf of the Service. However, Services that wish to

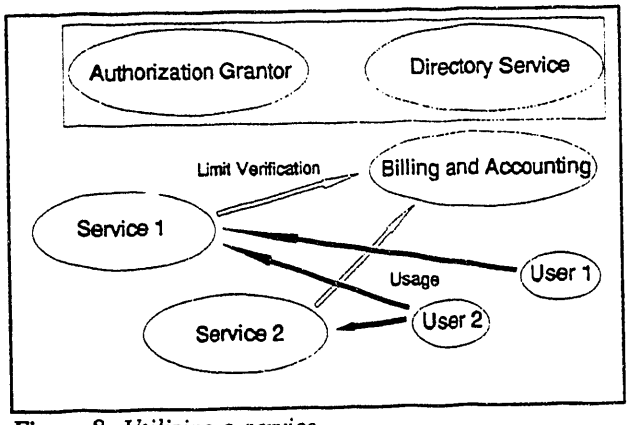


Figure 8. Utilizing a service

maintain their own autonomy may perform these functions themselves as well.

AUTHORIZATION

The preceding sections have described the basic components for providing security and privacy in the NII, as well as the mechanism by which services are provided and accessed. The next few sections will describe in greater detail the mechanisms for performing authorization and authentication, as well as how services may be refused. They form the basis of our thinking of the NII security and privacy system.

Authorization is the determination of whether a principal is entitled to use a protected resource [8]. The Authorization Procedure in the NII involves the use of an Authorization Grantor, a Directory Service, and the issue of authorization tickets. The earlier sections on providing and accessing services have described the authorization procedure for principals requesting services. The steps required for the refusal of services is discussed in the next section.

This system of authorization provides both capability list and access control list protection. The capability list protection is offered by the authorization ticket that is issued by the Authorization Grantor. The Grantor decides based on the principal's characteristics and attributes whether to confer a capability to use a particular service on the principal. A service provider that requires more stringent access control list protection may require the principal to use the authorization ticket as an initial establishment of a communication channel. The service provider can subsequently impose more checks against its own access control list to determine if the principal may use its services.

Service Credit System

The Service Credit System exists to check that principals do not use services beyond their ability to pay. Each principal has two types of credit. He has an overall limit which is the upper bound on his total usage of services, and a service credit limit, which is the upper limit on the usage of a particular service. If a service credit has been totally consumed, the principal will have to request for a new ticket from the Authorization Grantor through the Directory Service. Each new ticket from the Grantor has a new limit pointer. If the total credit has been consumed, then a ticket for a new overall credit is also obtained from the Grantor.

We have a number of ways of implementing these limits. One option for storing the value of the limit is with the authorization ticket itself. The problem with this scheme is that updating the ticket requires the principal to store a new ticket reflecting his new limit each time he accesses a service. The system may be abused by a principal who makes a copy of the original ticket and always submits the original ticket with the full credit to the Service Provider. Concurrent access of multiple instances of the same service may also pose a problem as there is only one copy of the ticket to use. Each instance of the Service will be operating based on the original credit issued to the principal instead of concurrently deducting from the credit value. The credit limit will not serve its purpose in this situation.

Since we cannot rely on the principal to store an updated ticket, a separate Billing and Accounting Agency is required. The limit value stored in the authorization ticket is a pointer that can be used by the Billing and Accounting Agency to identify a particular credit account associated with the principal. The first time the ticket is used, the total credit value is noted by the Accounting Agency. Each subsequent access of the service with the given pointer will result in the Accounting Agency decrementing the limit available to the principal. When the entire limit is consumed, the service is denied to the principal. It is the responsibility of the service to contact the Accounting Agency with the limit pointer to determine the actual remaining limit of usage before providing a service. The problem of concurrent access of the same limit is resolved by having this credit tracking agency.

There are situations where a service may be provided without requiring the acquisition of an authorization ticket. Billing however is still required for such a service. Without the authorization ticket and the corresponding limit pointer, each time the principal accesses the service, he

will need to digitally sign for the usage. This will prevent any dispute during the accounting and billing later.

Billing and Accounting Agency

The Billing and Accounting Agency, provides consolidated accounting of the resources consumed by a principal. The service provides the capability for the system to bill the principal for his usage at a later time.

The Proxy Agent submits to the Accounting Service details concerning the usage of each principal on the service that the agent represents. Summary information is also compiled from the logs maintained by the Proxy Agent for regular auditing.

Security of Credit System

With the use of the Billing and Accounting Agency to keep track of the credit limit of service usage, another problem that arises is the security of the credit limit. The system must not allow any malicious user from being able to change the credit limit of another principal. This problem is two-fold. The system must prevent another principal from pretending to be the Service Provider and making a request to decrement the credit of the service usage. The system must also prevent a malicious Service Provider from decreasing the limit even when the principal is not using the service.

The solution to prevent a malicious user from pretending to be a legitimate service provider is to give only the Service Provider and its corresponding Proxy Agent the authority to make the change. The Accounting Authority must therefore authenticate the source of the update message from the service provider before acting on it.

The second problem of preventing a service provider from decreasing the credit limit of the principal even when he is not using the service is achieved through the Proxy Agents. The Proxy Agents are supplied by the NII and they act as the auditor for the Service Provider. It is responsible for keeping logs of the transactions made through it, as well as reporting transactions to the Accounting Agency. A service may not charge the principal for services he did not consume as the Proxy Agent can detect a billing to the principal that did not originate from a corresponding request by the principal.

The credit system must also allow the establishment of a new credit limit when a new authorization has been obtained by a principal for a service whose credit limit has been fully consumed. The Authorization Grantor issues a new limit pointer for each new authorization, so that the Accounting Agency will never have to increase an existing limit. The only operations allowed will be decrementing the limit and establishing of a new limit. Old limit accounts that have been consumed and accounted for are purged so that the Authorization Grantor may reuse the credit limit pointer.

Abuse of the Authorization System

The service is responsible for checking each principal for his authority to access its services. The verification may be done by the Proxy Agent on behalf of the service. The service itself must however establish its own policies for when to check the principal's credit limit and when to deduct from the principal's limit. Thus it is possible for the system to be abused if the service is not stringent in its authorization checking.

Removal of Authorization

Since there is a mechanism for granting authority to use services, there must be a corresponding mechanism for removing authority. The removal of authority occurs at two levels - generation of the role certificate and the generation of the authorization certificate.

The Granting of Authority requires the principal to have a role certificate to prove that he has the prerequisite role to get the authorization ticket for service access. The role certificate is time-stamp and stored at the Directory Service or with the principal. The Directory Service presents the certificate to the Grantor during the process of obtaining a ticket for the principal. If the role certificate has expired, the Grantor will determine if the principal may be issued with a new role certificate.

Since the role certificate is dated, an authorization to assume a role can be removed by informing the Grantor that a particular principal can no longer obtain a role certificate for that role. Correspondingly, when a principal wishes to access a service that requires the use of the expired role, the Grantor will not generate a ticket for the principal when it knows that the principal is no longer authorized to act in that role. Effectively, the authorization of the principal for the service and the role is removed.

For services whose accesses have to be tightly controlled, we are experimenting with a time-stamp valid for 8 hours for authorization tickets and 24 hours for role certificates. This will allow the disenrollment of a principal from the use of a service to within a day. Other services whose access restrictions are not as stringent may set a longer time-stamp period of perhaps a month.

REFUSAL OF SERVICES

As reflected earlier, the system described so far provides only for the first part of privacy - protection of what information is shared. The second component of privacy involves the protection of a principal from receiving unwanted services. For example of such a situation is where a parent checking into a hotel may wish to refuse service to any adult movies if his children are travelling with him.

The general problem of refusing a service is a difficult one. However, in the NII, since we have a requirement that all services register themselves with an Authorization Grantor, the same Grantor may thus enforce the rejection or refusal of service.

General Refusal of Services

The above scenario of Refusal of Specific Service works only in the situation when the principal knows specifically what services he wishes to refuse. A more complete solution will be to provide a mechanism whereby a principal can define the *type* of service that he wishes to refuse. This may help to alleviate the problem where the principal may not know exactly what services to refuse.

Since all services are registered with the Authorization Grantor, when each new principal is created, he can be presented with a list of the services available, and he must specify for each service one of the following:

- *Rejection.*

  If this was selected, then there is an absolute rejection of the service. The principal will never be presented with this service.

- *Acceptance.*

  If this was selected, the service is considered as acceptable, and the principal will be allowed access if he is authorized to do so.

- *No Decision.*

  If this was selected, it means the principal cannot make a decision on whether to accept or reject the service at the start up time. The service will be required to prompt the principal at a later time when the service is being offered to determine whether the principal wishes to receive the service.

Scalability Problem

The above solution gives rise to another problem, and that is as the number of services increases in number, it is not feasible for the principal to go through the list of services exhaustively and make a decision on each service. The services will therefore have to be broken down into general wide categories. The principal may make a decision on a category as a whole, or he may elect to go into a specific category to make his selection in greater detail.

After a principal has been created and the general decision on the acceptance of services has been made, the principal is informed incrementally of additions to the list of services provided. This task is done by the Directory Service. The notification of new services is yet another service which the principal may choose to refuse.

As the NII grows, the classification of services will become more detailed and complex. Profiles on the types of principals may be generated and specific groups of services that are particular to the type of principals may be associated with the principal. This will reduce the amount of information the principal will have to wade through. The problem of how to divide services into categories is a separate problem on its own and is beyond the scope of this paper.

Refusal of Junk Mail

Another scenario where a principal may wish to use rejection of services is with junk mail. A principal may want to make a request to reject all correspondence of a certain type. The principal may also have a black list of all addressees whom he does not want to hear from, or he may have some conditions on the type of messages that he receives. The mail system in the NII is also based on an access control list and capability list system. A principal may therefore specify the addressees to reject mail from. Alternatively, a special mail filter agent may be employed to pre-process the messages that are received by the principal and remove the undesired messages.

AUTHENTICATION

Authentication is the process of verifying that a principal is who he says he is [8]. This section will discuss how a local access terminal and a remote service provider authenticates a principal.

Principal Authentication with Smart Card

As mentioned earlier, if a principal has a Smart Card, his

Private Key is stored in an encrypted form on the card. If a principal uses his smart card at an access terminal, he will still be required to enter a password (or biometric identification) as part of the authentication process. This will prevent someone from using a stolen Smart Card. The smart card reader obtains the password from the principal and this is sent to the smart card for the decryption of the Private Key. The access terminal will subsequently attempt to determine the authenticity of the principal by making a series of challenges to the smart card.

For transactions that follow after the authentication, all data that require the use of the Private Key will be sent to the smart card for encryption and decryption. The Private Key of the principal will not leave the Smart Card at any time. Figure 9 illustrates authentication with the use of a Smart Card.

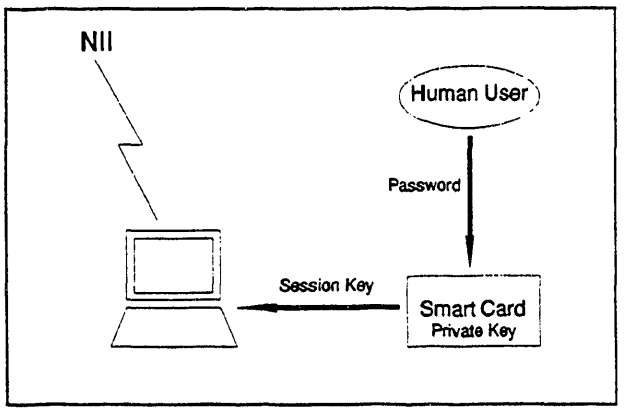


Figure 9. Authentication with smart card

There is a potential problem of the Smart Card being the bottleneck of the processing speed. However, this is the only means of ensuring complete secrecy of the Principal's Private Key. To partly rectify this problem, if the local access terminal has processing power, the smart card can negotiate the random session key with the remote party, and pass the task of using the session key on to the access terminal for encryption with a Single Key System. For an access terminal without any processing power, the smart card will have to bear the entire burden of encryption and decryption if the principal wishes to access sensitive data.

Principal Authentication without Smart Card

In the situation where no Smart Card is available, to access sensitive data, the principal must make his Private Key known to the access terminal so that it may perform the encryption on his behalf. The access terminal will obtain the password from the principal and use it to decrypt the encrypted Private Key. The criteria of the principal being able to trust the access terminal fully in this situation is crucial because the terminal will obtain knowledge of the Private Key. If the machine is not fully trusted, a Trojan Horse process that captures Private Keys may be present in the machine. Security of the system will thus be compromised. Figure 10 illustrates authentication without the use of a Smart Card.

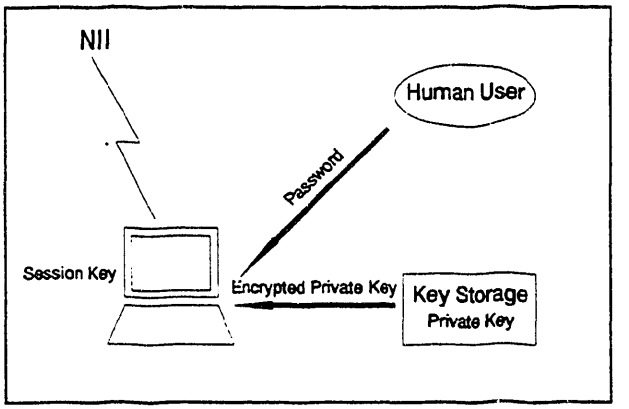


Figure 10. Authentication without smart card

Remote Requests

Ordinarily, when a principal makes a request, the request is sent to a service. To process the request, the service may be downloaded to the local machine, or the request may be sent to a service sitting somewhere at a remote site on the network. The latter case is likely to be a common case where requests are sent via the network to the service and the responses sent back. In such situations, the remote service requires a means of authenticating the principal before providing the service to him. We have designed a scheme for this authentication process.

Remote Authentication for Online Requests

Remote Authentication allows a remote application to authenticate a principal that is not at the application's own host. The principal's password or Private Key must never be transmitted over any link. Thus remote authentication employs a different scheme from local authentication.

Each machine on the network is also considered a principal and is assigned with a pair of Public and Private Key. When an application makes a request to authenticate a remote human user, the machine which the principal is logged on at is first authenticated using the standard Public-Private Key Authentication Protocol. If host A is trying to authenticate host B, A will first generate a

random number N_R, and encrypt it with its own Private Key. A will send the encrypted number to the receiving host. Host B upon reception of the message will decrypt the message with A's Public Key to extract N_R. Host B will then encrypt N_R with its own Private Key and send it back to A. A will verify the identity of B by decrypting the received number with B's Public Key and checking that the number is N_R. Note that at this point only A has authenticated B. If B wishes to be certain of A, it may repeat the above procedure. Figure 11 illustrates the online remote authentication.

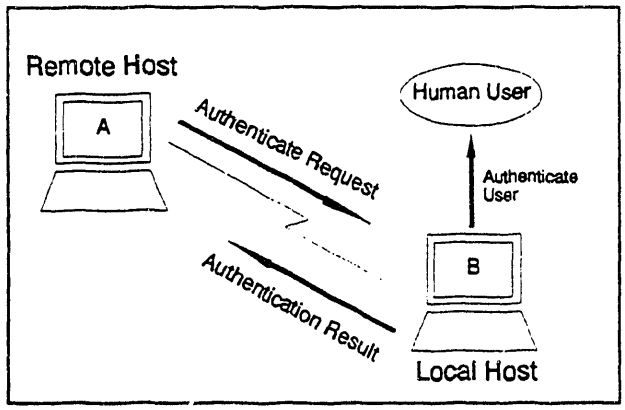


Figure 11. Online remote authentication

After the two hosts have authenticated each other, the application program may then complete the remote authentication by requesting the remote host to authenticate the principal and return the result. The actual host-to-user authentication procedure will depend on the host platform type. The same random number N_R is now encrypted with the principal's Private Key. The encrypted string is then returned to the application program and the similar verification procedure (by decrypting with the principal's Public Key) is carried out.

The additional step of authenticating the access terminal is for added security. If an application can ascertain that the terminal that the principal is using is not to be completely trusted, it can refuse to send any sensitive information to the principal. Since the information that principal wishes to view is available in the clear at access terminal, a breach of security may still result.

The capability to trust the principal but not the terminal is an important one. A service provider can trust the *requests* made by the principal, but it does not necessarily have to send any confidential *replies* to the principal. For example, a principal may wish to make a ticket reservation at a public access terminal. In order for the reservation system to accept the principal's request, it must be able to authenticate the source of the request. If the service can authenticate the principal, then it may accept the request submitted by the principal. Such a reservation would not be allowed if the system cannot ascertain the identity of the principal. However, the same principal at the public access terminal cannot request for the display of confidential information such as company project information, even if the terminal was equipped with a smart card reader for authenticating him. This is because such information will be displayed in the clear at the public access terminal and may be intercepted by some other malicious user.

Remote Authentication for Offline Requests

The procedure described in the previous subsection assumes online information access by the principal. If this is an offline request (ie. the principal is no longer at the access terminal by the time the request is processed), there is an additional complication of the service not being to authenticate the principal, since its Private Key is no longer available.

A solution to the offline request is possible if the principal has complete trust in his access terminal. A running process (eg. the Principal Agent) may be present at the access terminal to act on behalf of the principal. This process will respond to the authentication request from the application. The process may be terminated once the necessary authentication procedure is completed. Figure 12 illustrates the offline remote authentication.

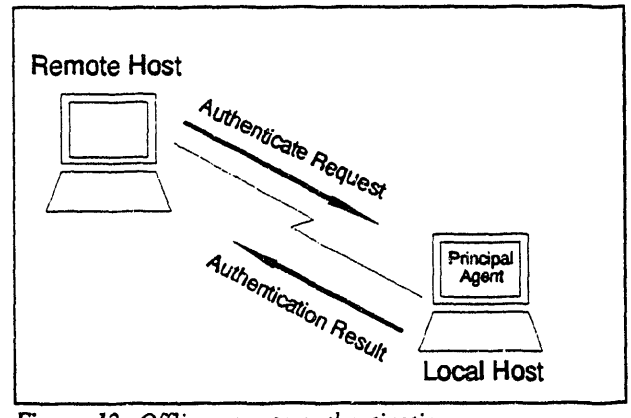


Figure 12. Offline remote authentication

Alternatively, a signed authentication string may be included with the principal's request. The Service may use the signed string to verify the authenticity of the request. The string must be based on the request, time-stamped and include a sequence number. To prevent anyone from

capturing the entire request message and replaying it at a later time, the Service will not process the request if the time-stamp has exceeded some time allowance, or if the same request (ie. the sequence number is equal or smaller than the last sequence number received) has been recently made.

The corresponding problem of the access terminal intercepting information as it is being displayed is removed, since the result information is no longer sent to the access terminal. Results of such offline processing are usually returned to the principal via electronic mail. The usual security mechanisms using encryption can be used to provide security for electronic mail.

FUTURE WORK

The Security and Management Framework for the NII is still very much at an early stage of development. This section describes some important areas that need to be worked on in the near future.

Secure Proxy Agents

The Proxy Agents that accept requests from principals and log the requests can perform intruder detection and report any abnormal behavior in the principal's operation.

Intruder detection involves the determination of the presence of an unauthorized principal in the system. Since requests are generally invoked through the Proxy Agent, it can detect processes running on the machine that are not authorized. Such processes can be identified as intruders and they can be suspended and reported to the system administrator. This will actively prevent any malicious user from finding a loophole in the system and bypassing the authentication and authorization procedure enforced by the Proxy Agent.

Abnormal behavior reporting involves the monitoring of the normal behavior of the principals of the system over a period of time. The behavioral information is extracted from the logs maintained by the Proxy Agents. This behavior is collected as statistics. Any significant deviation from the normal operating behavior is reported to the system administrator. This is still a relatively unexplored area of system management. Refer to [10,12] for some discussion on techniques of collecting statistics and analyzing them.

Role Relationships

The role handling mechanism described here does not provide for relationships between different roles. In particular, there are likely to be inheritance relationships between roles. There needs to be a way of representing relationships such as that between a manager and his assistant. The assistant may do some routine transactions on the manager's behalf, but he would refer important tasks back to the manager.

When a person assumes a particular role, the charging and billing mechanisms may differ. For example, the billing rates for a service to a business user may be higher than to an ordinary household user. Further work needs to be done to explore how these additional features of roles can be integrated into the current design.

Removal of Service Access Authorization

The current solution proposed in this paper of removing authorizations assigned to principals based on expiring tickets is incomplete. There is a time window within which the time-stamp on the ticket is still valid even though the decision to deny access has been made. There are a number of ways where this can be corrected (eg. informing the services of the invalidated tickets), but they are generally inefficient in large systems. A cleaner solution may be to develop a scheme to create a ticket that cannot be duplicated, or if duplicated, it can be detected as a duplicate. If such a scheme were possible, then the Authorization Grantor may revoke an authorization by asking the principal to return the ticket issued. Providing tickets that cannot be duplicated is a difficult issue (especially since all forms of data communication and transfer requires some form of copying). It is an area for possible further investigation.

Copyright Protection

The Digital Signature Mechanism only proves that the signer has intended the contents of the signed message. It proves to the receiver that the message did come from the sender and the sender cannot refute the fact that he sent the message. However, it does not prove to the receiver that the sender is the *original author* of the message. It is possible for one valid receiver of the signed message to remove the author's signature from the message, perhaps make some changes to the message, attach his own signature and claim to an unsuspecting third party that he is the author of the message. This flaw can open the door to copyright infringement of many kinds. Literary work copyright infringement is just one fraction of a much larger implication that one can easily copy what another person has said or written and claim to be his own.

There does not appear to be any clear solution to this problem at the present time. This may be an important consideration in the design of a large information network.

CONCLUSION

This paper describes some of the security, privacy and management issues that the IT2000 project team is examining during the ongoing design and implementation of the NII. The paper reflects some of the technical problems that we have encountered, the design choices that we are making and the rationale behind them. It is our aim to share our current thinking in the domain of managing a national network.

FOR MORE INFORMATION

Further information regarding the IT2000 and the NII may be obtained from the authors at the following address:

Planning and Infrastructure Department
National Computer Board of Singapore
71 Science Park Drive
Singapore 0511
Republic of Singapore

Electronic mail:

shgoh@cs.stanford.edu	(Seow-Hiong Goh)
yeowmeng@iti.gov.sg	(Yeow Meng Chee)
michael@iti.gov.sg	(Michael Yap)

ACKNOWLEDGEMENT

The authors would like to thank the NII Software Architecture Group, particularly Jin-Ho Tan, Chin-Chau Low, Richard Tan, Surjatini Widjojo, Kin-Yee Ng and Siew-Siew Lim for their valuable discussions and review that help shape our design.

BIBLIOGRAPHY

1 "The IT2000 Report: A Vision of an Intelligent Island", SNP Publisher Pte Ltd, March 1992.

2 Motiwalla, J., Yap M., "Building The Intelligent Island", *12th World Computer Congress, From Research to Practice*, September 7-11, Madrid, Spain.

3 Rivest, R., Shamir, A., Adleman, L., "A Method for obtaining Digital Signatures and Public-key Cryptosystems", *Communications of the ACM* **21** (1978) 120-128.

4 National Bureau of Standards, "Data Encryption Standard", *Federal Information Processing Standards Publication 81*, Washington DC, 25 September 1980.

5 Rivest, R., Dusse, S.R., "The MD5 Message-Digest Algorithm", preprint.

6 RSA Data Security Inc, "Public-Key Cryptography Standards", 3 June 1992.

7 Feige, U., Fiat, A., Shamir, A., "Zero-knowledge proofs of identity", *Journal of Cryptology* 1 (1988) 77-94.

8 Hoffman, L.J., "Modern Methods for Computer Security and Privacy," Prentice Hall, 1977.

9 Steiner, J.G., Neuman, B.C, Schiller, J,I.. "Kerberos, an Authentication Service for Open Network Systems", *USENIX Association* (February 1988).

10 Javitz, H.S., Valdes, A., "The SRI IDES Statistical Anomaly Detector", *IEEE Symposium on Research in Security and Privacy*, 1991.

11 Diffie, W., Hellman, M.E., "New directions in cryptograph", *IEEE Transactions on Information Theory IT-22 (1976)* 644-654.

12 Shieh, S.W., Gilgor, V.D., "A Pattern-Oriented Intrusion-Detection Model and Its Applications", *IEEE Symposium on Research in Security and Privacy*, 1991.

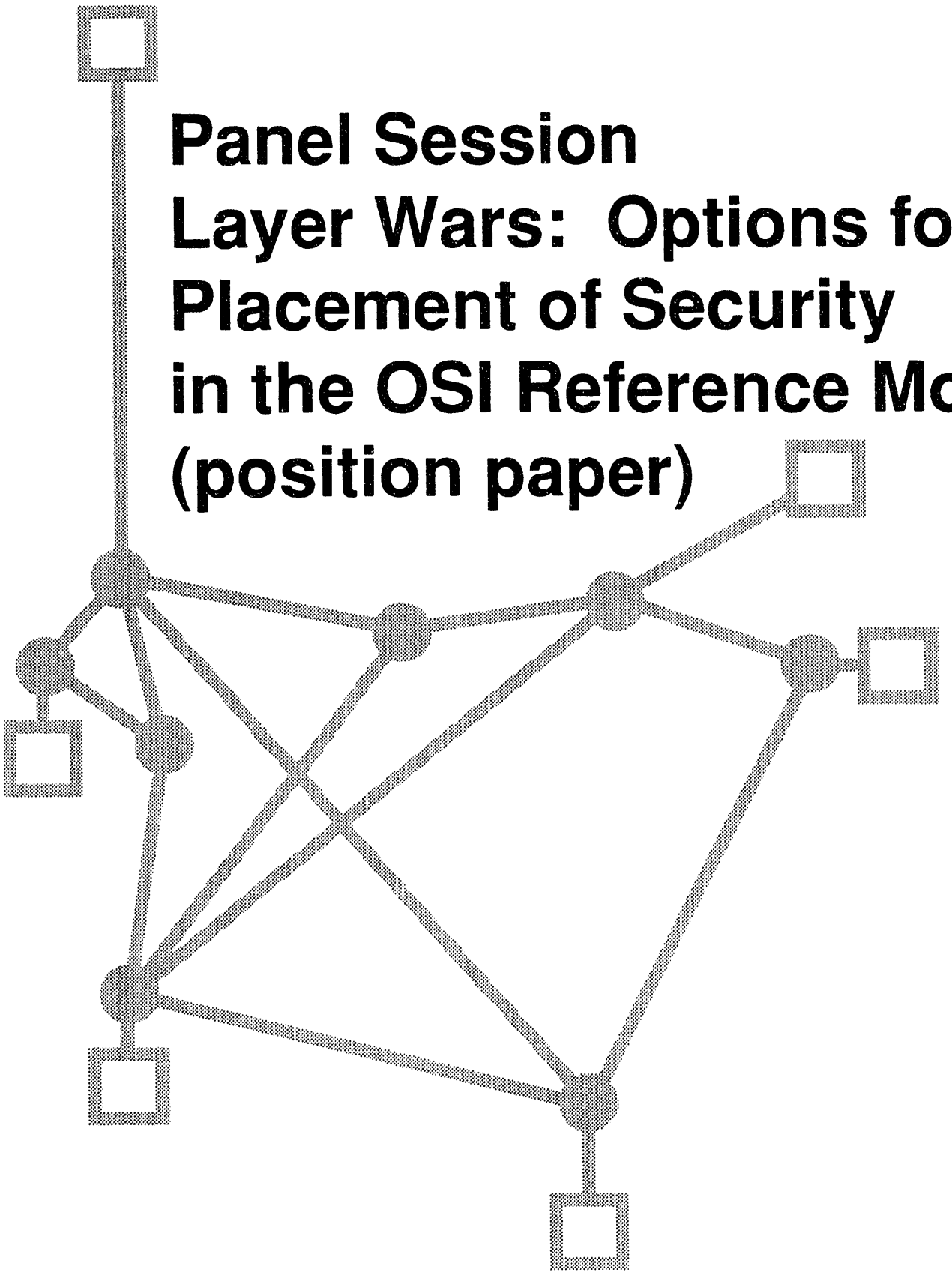
13 Woll, H., "Zero Knowledge Proofs and Secret Sharing Problems", University of Washington, Dept of Comp Sc, Technical Report 88-10-02 (1988).

14 Needham, R.M., Schroeder, M.D., "Using Encryption for Authentication in Large Networks of Computers", *Communications of the ACM* **21**, No 12, 993-999 (December 1978).

15 National Bureau of Standards and Association for Computing Machinery, "Executive Guide to Computer Security," 1974.

16 Diffie, W., "The first ten years of public-key cryptography", *Proceedings of the IEEE* **76** (1988) 560-577.

Panel Session
Layer Wars: Options for Placement of Security in the OSI Reference Model (position paper)



Layer Wars:
Protect the Internet with Network Layer Security

Paul A. Lambert

Motorola, Inc.
Secure Telecommunications

ABSTRACT

Rapid advances in communication technology have accentuated the need for security in distributed processing systems. The broad interconnectivity provided by these technologies amplify both the capabilities of a computer network and the security risks. New developments in communication protocols promise to alleviate security problems by the application of standard security mechanisms. This paper examines significant work in the recent development of lower layer security protocols. Protocols for link, network, and transport layer security are examined and the architectural tradeoffs inherent in these mechanisms are contrasted.

INTRODUCTION

The Organization of International Standardization's (ISO) Open System Interconnection (OSI) reference model divides data communication functionality into seven layers. Communication protocols within these layers describe the format and sequencing of data transfers.

Security protocols can provide strong cryptographic-based mechanisms for protection of these communications.

The use of cryptography to protect data communications is defined by ISO as one of several mechanisms to provide security services. Other mechanisms include physical isolation, audit trails, and trusted functionality. The various categories of security services include: confidentiality, integrity, access control, authentication, and non-repudiation. Confidentiality service prevents the unauthorized disclosure of information and is the service most often associated with encryption. Integrity checks detect the unauthorized modification of data. Access control provides the means to grant or deny access to information. Authentication verifies the identity of an entity. Non-repudiation prevents a sender from falsely denying that data was sent, or a receiver from falsely denying that data was received.

The services supported by a security protocol depend on the protocol's location in the OSI reference model. The layering of the security also determines the medias that can be protected and the extent of the protection in an architecture. Security protocols placed in or above the Network layer provide "end-to-end" protection by encrypting the user data and leaving unencrypted the headers that allow the data to be delivered. In contrast, Link and Physical layer security protocols are not able to extend protection across heterogeneous networks.

The "lower layers" of the OSI framework consist of the Physical, Data Link, Network, and Transport layers. Three notable proposals for security in the lower layers of the OSI reference model are shown in Figure 1. The Transport layer (TLSP) and Network layer (NLSP) protocols are currently the subject of work in ISO subcommittees. The local area network (LAN) secure data exchange (SDE) protocol is being defined by a working group within the Institute of Electrical and Electronic Engineers (IEEE).

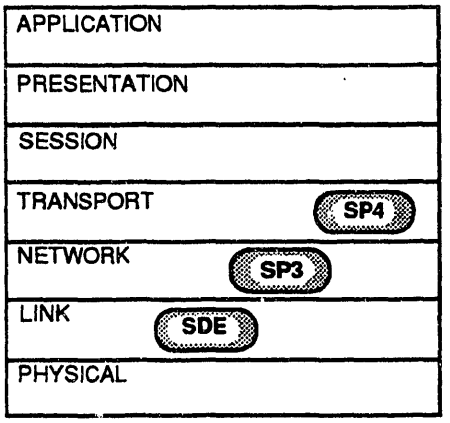


Figure 1. Lower Layer Security Protocols in the OSI Reference Model

UPPER & LOWER LAYER SECURITY

Security services in the Application or Presentation layers depend on the utilization of specific applications or application service elements. This typically means that the security mechanisms in the upper layers must be built directly into every application program. The benefits of this close integration of security include the capability to selectively protect specific fields of information, and the ability to provide non-repudiation services. Examples of upper layer security include secure electronic messaging

(X.400 series), directory security (X.509), authentication protocols, Privacy Enhanced Mail (PEM - RFCs xxx, xxx,xxx), and key management.

Lower layer security is relatively independent of the user application. Existing applications are readily protected by lower layer security protocols that simply encapsulate the user traffic. Networking protocols are typically shared by computer applications allowing a single lower layer security protocol to provide uniform protection within a computer system. Some lower layer security protocols are also amenable to implementations in front end communication devices, bridges, or routers. These stand-alone embodiments are an attractive approach for protecting existing computer systems. The bridge or router configurations can also protect the communications of multiple collocated computer systems.

TRANSPORT LAYER SECURITY

The Security Protocol at Layer 4 (TLSP) was initially developed by the Secure Data Network System (SDNS) project. The SDNS project developed a security architecture within the framework of the OSI reference model that included specifications for network and transport layer security. This program started in the summer of 1986 and completed an initial draft of the security protocols in 1987. These specifications have since been published by the National Institute of Standards and Technology (NIST), and have progressed through the American National Standards Institute (ANSI) into ISO as technical contributions.

The TLSP protocol specifies optional extensions to the ISO connection-oriented transport service (ISO 8073) and connectionless mode transport service (ISO 8602). The security protocol is supported by either connectionless network service (CLNS) or connection-oriented network service (CONS). Figure 2 illustrates the relationship of TLSP to the Transport and Network layer services. In this layered model, TLSP is logically at the bottom of the Transport layer and encapsulates the existing Transport protocols.

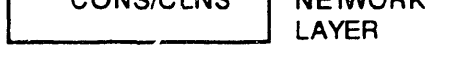


Figure 2. TLSP - Transport Layer Security

Transport service provides transparent and reliable delivery of data between end systems. The extensions provided by TLSP add the security services of: peer entity authentication, data origin authentication, access control, connection-oriented confidentiality, connectionless confidentiality, connection-oriented integrity, and connectionless integrity.

The format of the TLSP protocol consists of a "clear header" followed by protected header information and data. The first two fields of the TLSP PDU indicate the length (LI) and PDU type (SE). This format provides compatibility with the ISO transport protocol (ISO 8073). The next field in the clear header is the key identifier (KEY-ID). The KEY-ID establishes the cryptographic key used to protect the transport protocol data unit (TPDU). This mechanism assumes that the cryptographic key has been pre-established. The KEY-ID also determines the algorithm used to protect the data, security service options, protocol options, and a set of acceptable security labels. Within the SDNS security framework, the establishment of the KEY-ID and its associated attributes are handled by the SDNS Key Management Protocol.

The protected portion of the PDU contains a "protected header", data, and an optional Integrity Check Value (ICV) field. The protected header includes an optional label field and optional final sequence number field (FSN). The ICV field provides protection from the modification of data. The ICV is a checksum or hash calculated over the protected header and data. This effectively provides the basic integrity security service.

Connection-oriented integrity furnishes the ability to detect the replay or reordering of PDUs. This service is supported by sequence numbers in subclass of TLSP called TLSPC (C for connection-oriented). The basic connectionless integrity service is supported by the TLSPE subclass (E for Encapsulation).

The TLSP protocol utilizes the sequence numbers in the ISO transport protocol (only classes 2, 3, or 4 of ISO 8073). These sequence numbers are augmented by a Final Sequence Number (FSN) that is used only when DR, DC, or ER TPDUs are encapsulated. This allows the deletion of the final TPDU of a connection to be detected. The TLSPC protocol also requires that a separate cryptographic key be used for each end system pair and security level set to support connection-oriented integrity.

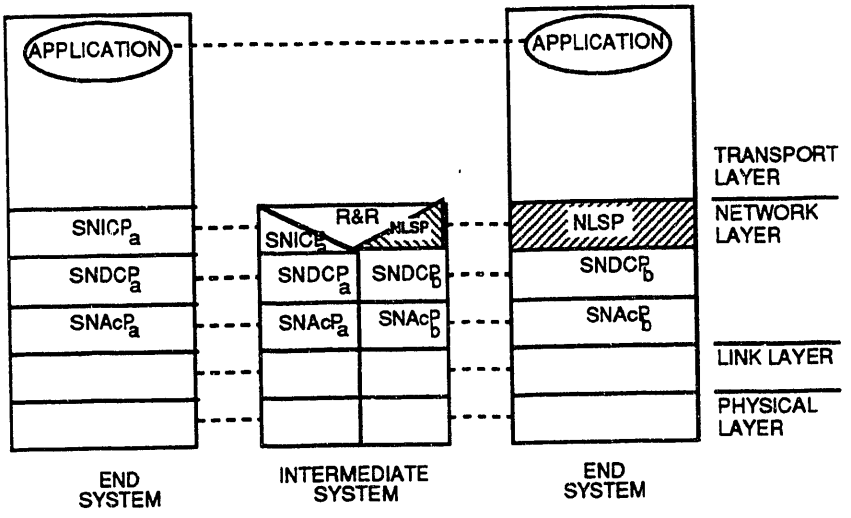


Figure 3. Network Layer Security in Intermediate System and an End System

NETWORK LAYER SECURITY

The Security Protocol at Layer 3 (NLSP) was, like TLSP, initially developed by the Secure Data Network System program. The specification for NLSP has been published by NIST and has progressed through ANSI as a technical contribution to ISO. NLSP directly provides all of the security services of TLSP except connection-oriented confidentiality and connection-oriented integrity.

The NLSP protocol is defined as a "sublayer" of the OSI network layer. To facilitate interoperation, the OSI network layer model provides considerable flexibility in the layering of the network protocols. NLSP is considered to be a SubNetwork Independent Convergence Protocol (SNICP). This places NLSP at the top of the network layer and allows NLSP to be installed in either "end systems" or in "intermediate systems."

A communication model for NLSP is shown in Figure 3. This figure illustrates security placed both in an "intermediate system" and in an "end system." The Subnetwork Dependant Convergence Protocol (SNDCP) and Subnetwork Access Protocols in this figure are intended to illustrate the flexibility in network media and protocols. The NLSP security in this model can protect communications carried over most local area networks or wide area networking technologies.

The NLSP and TLSP protocols are very similar in both format and services provided. The "clear header" portion of the NLSP PDU is identical to the header defined for TLSP. Both contain a length field, a PDU type field, and a key identifier. The "protected fields" in NLSP provide optional fields that are not available in TLSP. These fields support the intermediate system implementations by carrying the addresses of systems located behind a "router-like" security device.

The similarity of NLSP and TLSP is due partly to the positioning of these protocols in the OSI reference model. NLSP is considered to be at the "top" of the network layer, and TLSP encapsulates data at the "bottom" of the transport layer. The protocols actually share a mode of operation where the protocol formats and services are identical. In this common mode of operation only the minimum services are provided. Communication with intermediate systems is not possible and connection-oriented integrity is not supported.

Currently, NLSP supports only connectionless network layer services. Several proposals in ISO have recently been documented that will add "connection-oriented" network layer services to NLSP. A "connection-oriented" NLSP is required to protect systems that do not utilize the ISO Connectionless Network Layer Protocol (CLNP). The utilization of TP0 over X.25 is an example of this scenario. In addition to supporting many communication environments, a connection-oriented security protocol would also provide replay protection (connection oriented integrity).

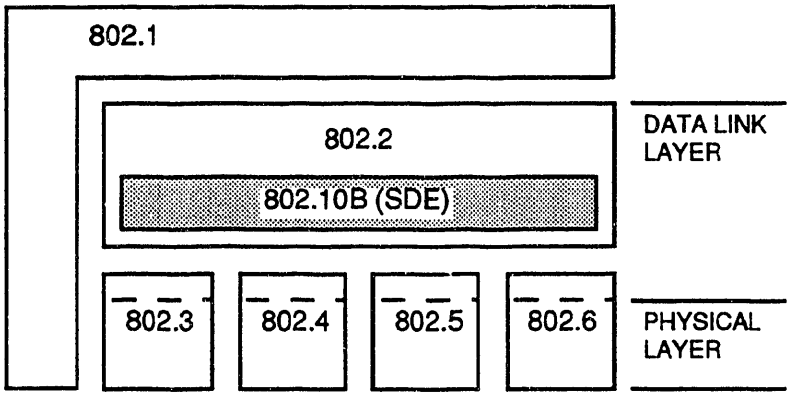


Figure 4. Relationship of SDE to Other LAN Standards

IEEE 802.10B (SDE)

In 1988, a LAN Security Working Group was formed under the auspices of the Institute of Electrical and Electronic Engineers (IEEE) to develop a Standard for Interoperable LAN Security (SILS). The LAN Security Working Group is unique in that it is jointly sponsored by the IEEE Technical Committee on Security and Privacy and by the IEEE 802 Standards Committee. The joint sponsorship and balloting should insure that SILS will meet both the security and communication requirements of LANs.

The scope of this working group (IEEE 802.10) includes the standardization of the secure exchange of data at the Data Link Layer, the management of cryptographic keys at the Application Layer, and the specification of associated network management objects. IEEE 802.10 has produced a stable definition of the Secure Data Exchange (SDE) protocol that should be balloted as an IEEE standard in 1990.

The SDE is a Layer 2 protocol that provides connectionless service as a sublayer of the Logical Link Control (LLC). Figure 4 illustrates the relationship of the SDE to the 802 reference model. The charter of IEEE 802 limited the protocol layering alternatives for the SDE to the Physical or Link layers. Physical layer security was not desirable since it would be specific to each of the many LAN media. The specification of SDE as a sublayer of LLC allows SDE to protect any 802 LAN medium, and also supports implementations of "bridge-like" security devices.

The SDE protocol supports the security services for: connectionless confidentiality, connectionless integrity, data origin authentication, and access control. These services are provided "transparently". The transparency goal of SDE assures that systems protected by SDE will not interfere with the operation of existing unprotected systems. This is a particularly important goal for LANs since many systems share the same broadcast media.

In the principal mode of SDE operation, the protocol consists of a "clear header" followed by protected data. The clear header contains octets that are used to discriminate SDE PDUs from LLC PDUs and is followed by a Security Association Identifier (SAID). The security association is an important aspect of the SDE protocol. The security association is defined as a cooperative relationship between communicating entities formed by the sharing of cryptographic keying information and security management objects. The SAID indicates how the PDU is protected, including what algorithm and key to use, what optional fields are present, and the security attributes of the association.

The utilization of the SAID field requires that some other secure mechanism has pre-established a security association. In IEEE 802.10, this is intended to be some form of manual or automatic key distribution. The security of cryptographic mechanisms depends largely on the distribution cryptographic keys. The companion standard to SDE, IEEE 802.10C, will provide an application layer mechanism for the distribution and management of cryptographic keys. The keys, security association identifiers, protocol options, and access control restrictions are all security management attributes that are established by the key management protocol and shared among entities to form the security association.

TRADEOFFS IN SECURITY LAYERING

The protocols described herein (SDE, NLSP, and TLSP) all provide cryptographic security services in the lower layers of the OSI reference model. The differences in the capabilities of these protocols are due primarily to the limitations imposed at each of these layers. A summary table

comparing the Link, Network, and Transport layer security protocols is shown below in Figure 5. The table includes the security services recommended by ISO 7498/2 for each layer of the reference model. It should be noted in this comparison that the services actually provided to the user depend on the complete protocol profile.

The security protocols have many similar characteristics. Each protocol has a "clear" and a "protected" header, a clear designation that distinguishes the protected PDU's from other unprotected traffic, and an optional integrity check field. The protocols are all algorithm-independent and each relies on a separate key management facility. These basic mechanisms provide confidentiality, integrity, access control, and data origin authentication for all three protocols.

	ISO 7498/2 Service Recomendations by Reference Model Layer							Lower Layer Security Protocol		
ISO 7498/2 Service	1	2	3	4	5	6	7	Layer 2 SDE	Layer 3 NLSP	Layer 4 TLSP
Peer Entity Authentication	•	•	Y	Y	•	◦	Y	• (1)	• (1,2)	• (1)
Data Origin Authentication	•	•	Y	Y	•	•	Y	Y	Y	Y
Access Control Service	•	•	Y	Y	•	•	Y	Y	Y	Y
Connectionless Confidentiality	Y	Y	Y	Y	•	Y	Y	Y	Y	Y
Connection Confidentiality	•	Y	Y	Y	•	Y	Y	•	• (2)	Y
Selective Field Confidentiality	•	•	•	Y	•	Y	Y	•	•	•
Traffic Flow Confidentiality	Y	•	Y	Y	•	•	Y	•	• (2)	•
Connection Integrity with Recovery	•	•	•	Y	•	•	Y	• (3)	• (3)	Y
Connection Integrity without Recovery	•	•	Y	Y	•	•	Y	•	Y	Y
Selective Field Connection Integrity	•	•	•	•	•	•	Y	•	•	•
Connectionless Integrity	•	•	Y	Y	•	•	Y	Y	Y	Y
Selective Field Connectionless Integrity	•	•	•	•	•	•	Y	•	•	•
Non-repudiation, Origin	•	•	•	•	•	•	Y	•	•	•
Non-repudiation, Delivery	•	•	•	•	•	•	Y	•	•	•
Communication Capabilities										
"End-to-End" Security								•	Y	Y
Intermediate System Security: bridge or router like security systems								Y	Y	•
Labels for Access Control								•	Y	Y
Security for non-ISO Communications (TCP/IP)								Y	Y	•
Protection for Routing Protocols: ES-IS, IDRP, ARP, EGP, etc.								Y	Y	•

Y Capability is provided by the protocol.

• Capability is not directly provided by the protocol.

(1) This service is not supported by the protocol, but can be provided in conjunction with key management.

(2) This service is supplied in recent proposals to upgrade NLSP.

(3) Recovery is not directly part of the security protocol, but can be provided by the utilization of TP4.

Figure 5. Comparison of ISO 7498/2 Security Service Recommendations and SDE, NLSP, and TLSP Protocol Services and Capabilities

Discretionary access control in security protocols is largely based on the address information available to the protocol. In this context, the granularity of access control for the SDE protocol is by Media Access Control (MAC) address. NLSP has Network Service Access Points (NSAPs), and TLSP has Transport Service Access Points (TSAPs) to support access control decisions. These mechanisms are not the ideal granularity for developing security policies. Because of this, each specification implies that "trusted" paths may allow mapping of the protocol services to attributes of an application program. To support these additional access control capabilities, and specifically to provide mandatory access control, both NLSP and TLSP allow optional security labels that are carried in the protected header of the PDU.

The only difference in security services between NLSP and TLSP is TLSP's support of connection-oriented integrity with recovery. The TLSP protocol can detect integrity attacks and the functionality of TP4 will retransmit the information. The extension of NLSP to include connection-oriented services provides connection-oriented integrity without recovery. If recovery is required in a system, TP4 could be utilized above NLSP.

The IEEE 802.10 Secure Data Exchange protocol can protect communication services only over LANs. The SDE protocol is not "end-to-end" because it encrypts the network layer information required to carry the traffic across heterogeneous networks.

A strong advantage of network layer security is the ability to install security in intermediate systems. Intermediate system security allows stand-alone security devices to be placed in front of one computer, or a network of many computers. Co-located computers are then readily protected by a single "router like" security device. The SDE protocol is also able to protect multiple systems in a "bridge-like" configuration. An architecture based on SDE bridges would be viable as long as no routers were used in the backbone network.

The TLSP protocol is very closely tied to the ISO Transport Protocol (TP). This limits the applicability of TLSP to ISO environments. The SDE and NLSP protocol are both able to protect non-ISO traffic. The SDE protocol encapsulates all traffic above the MAC layer. The NLSP protocol encapsulates the transport layer and is independent of the lower network layer protocol.

Traffic between user applications are not the only communications that need to be protected. Network routing protocols exchange information that is sensitive and subject to attack. The SDE protocol and NLSP both are able to protect routing protocols.

SUMMARY

No one security mechanism will meet the requirements of all possible environments. Different perspectives on these requirements have lead to the development of link (SDE), network (NLSP), and transport layer (TLSP) security protocols. Ideally, only one of these mechanisms should be necessary to protect a data communications system. Three security protocols in the lower layers of the OSI reference model may contribute to the security of systems, but not to their ability to interoperate securely.

The IEEE 802.10 Secure Data Exchange protocol is useful in protecting environments that utilize proprietary protocols. The link layer security provided by SDE will readily encapsulate any data above the MAC layer. This link layer encapsulation is also the principal disadvantage of this protocol. SDE's encapsulation of the network layer prevents protected traffic from being carried across routers.

Transport layer security (TLSP) has been advocated as the mechanism of choice for embedment in workstations. This perspective is based on the support of "connection-oriented" services by TLSP. The readily available transport layer interfaces in operating systems promise the ability to closely couple the lower layer security with the application processes. The advantages of embedded implementations of TLSP are offset by the difficulties in supporting these services in front-end implementations. The TLSP services protect ISO applications that utilize ISO TP4, but provided reduced services for lower TP classes (TP0, TP2). TLSP does not protect non-ISO protocols (e.g. TCP/IP) or routing protocols.

Network layer security promises the capability to protect the broadest range of systems. NLSP supports end-to-end security for ISO, non-ISO, and network routing protocols. The ability to place NLSP in an intermediate system allows stand-alone security devices to protect collocated groups of computers. NLSP can also provide services directly embedded in workstations. The integrated security will provide improved granularity of access control services to the user. The services provided by NLSP are identical to those provided by TLSP only when both use TP4 to support connection integrity with recovery.

DISCLAIMER

All three of the security protocols described in this paper are subject to change. While it assumed that the documents used for this review are stable, none of the protocols has yet been subjected to national or international balloting.

BIBLIOGRAPHY

1. ISO 7498, Information Processing Systems - Open Systems Interconnect - Basic Reference Model, 15 October 1984.

2. ISO 7498-2-1988(E), "Information Processing Systems - Open System Interconnection Reference Model - Part 2 Security Architecture.".

   IEEE 802.10 LAN Security Working Group, "Standard for Interoperable LAN Security (SILS)," P802.10B/D3, May 1990.

   NISTIR 90-4250, "Secure Data Network System (SDNS) Network, Transport and Message Security Protocols", U.S. Department of Commerce, February 1990.

ISO DIS 8648 Information Processing Systems - Data Communications - Internal Organization of the Network Layer.

ISO DP 10028.3, Information Processing Systems - Data Communications - Definition of the Relaying Functions of a Network Layer Intermediate System, 30 May 1990.

ISO 8072, Information Processing Systems - Open Systems Interconnection - Transport Service Definition.

ISO 8072/AD1, Information Processing Systems - Open Systems Interconnection Transport - Service Definition Covering Connectionless Mode Transmission.

ISO 8073, Information Processing Systems - Open Systems Interconnection - Transport Protocol Specification.

D. Branstad, R. Housley, K. Kirkpatrick, P. Lambert, "Shootout at the OSI Security Corral," *Proc. of the Fifth Annual Computer Security Applications Conference*, December 1989.

W.C. Birnbaum, "SP3 Peer Identification," Proceedings, *1990 IEEE Symposium on Research in Security and Privacy*, pp. 41-48.

W.C. Birnbaum, "SP3 Peer Identification," Masters Project, New Jersey Institute of Technology, December 1989.

D. Branstad, J. Dorman, R. Housley, J. Randall, "SP4: A Transport Encapsulation Security Protocol," Proceedings, *Third Aerospace Computer Security Conference*, December 1987.

L. K. Barker, "The Impact of Security Service Selection for LANs", March 12, 1989.

P. Lambert, "SDNS Network Layer Security," Proceedings, *Fourth Annual Symposium on Physical/Electronic Security*, August 1988.

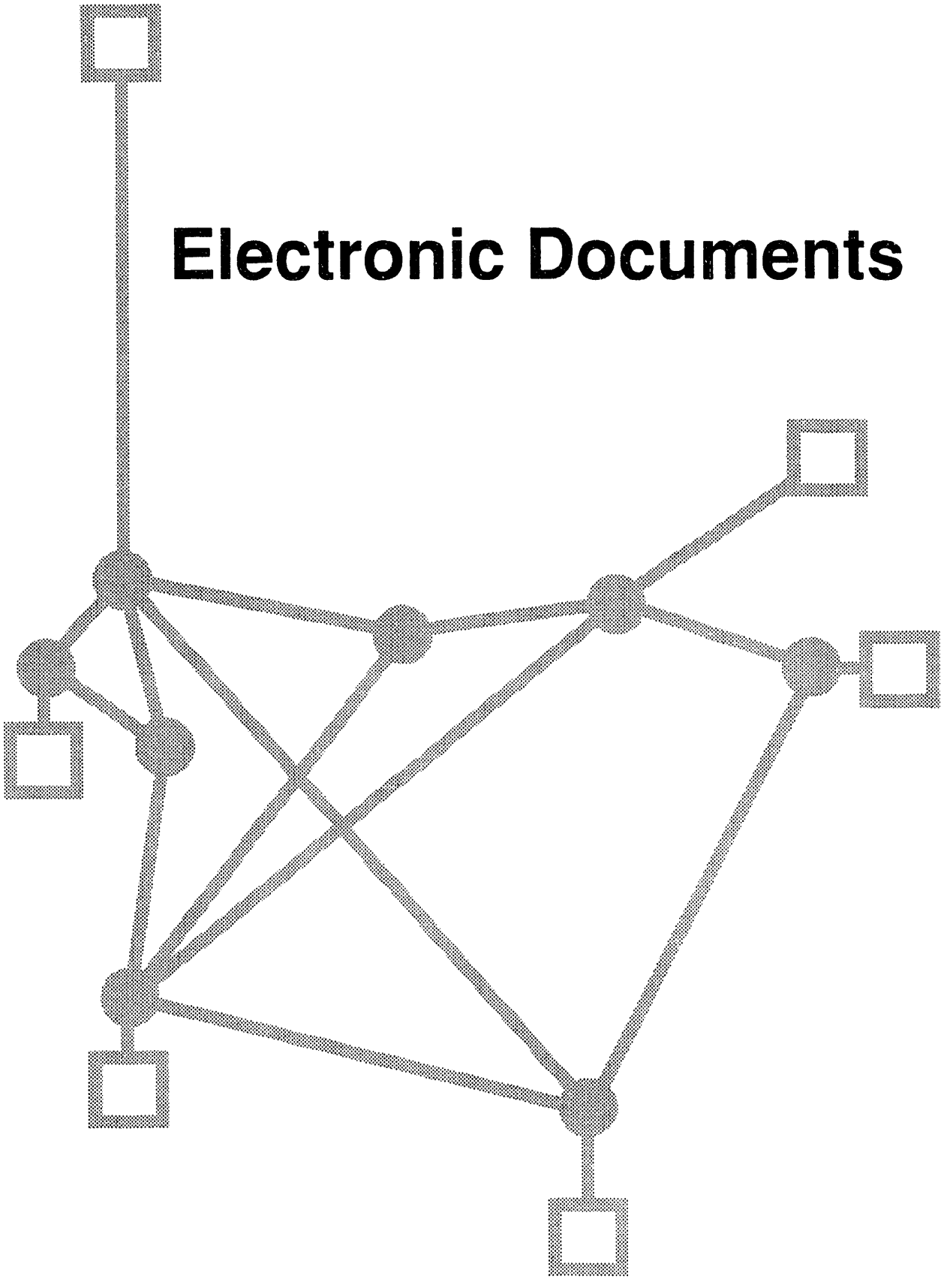
G. Tater, E. Kerut, "The Secure Data Network System: An Overview," *Tenth National Computer Security Conference*, September 1987.

R. Nelson, "SDNS Services and Architecture," Tenth National Computer Security Conference, September 1987.

Trusted Network Interpretation, NCSC-TG-005 Version-1, National Computer Security Center, 31 July 1987.

V.L. Voydock and S.T. Kent, Security in Higher Level Protocols: Approaches, Alternatives and Recommendations, Report No. ICST/HLNP-81-19, Nation Bureau of Standards, September 1981.

Electronic Documents



ELECTRONICAL COMMISSION MANAGEMENT

Vesna Ristic
Peter Lipp
Reinhard Posch

Institut fuer Angewandte Informationsverarbeitung und Kommunikationstechnologie
Technische Universitaet Graz
Graz, Austria

ABSTRACT

In this paper a secure application entitled "Electronic Commission" is presented. Its purpose is the automation of formal commission management using a decentralised organisation. In particular, commission work from all levels of a university administration is observed. Though forced to transmit their votes over an insecure network, participants can rely on the basic principles of democratic electoral systems. Special attention is paid to observing regulations and standing orders available from a local database. The commission chairperson can, according to circumstances, select a service from the set of available security services and, in this way, determine the commission proceeding. Because the purpose of this application is to provide an useful tool rather than to enforce observing specific regulations, it could easily be modified for use in other fields of administration and management communications.

INTRODUCTION

Computer applications supporting various kinds of meetings and working groups is a relatively new and therefore poorly explored area. Some authors [1] have previously studied the impact of electronic systems on the effectiveness, efficiency and satisfaction of working groups. For this purpose they created a "meeting environment" intended to make group meetings more productive. To achieve this goal, they investigated the mechanisms of group work and developed tools and techniques for the creation of information systems. They found that the new technology could significantly improve group processes and outcomes, although some effects were dependent upon the situation.

The implementation of security services based on cryptographic protocols has a significant impact on the field of computer supported group work. The set of possible modes of computer supported communication between two or more parties has grown enabling many forms of human communication to be performed over the computer network. Moreover, it is now possible to implement some new communication facilities which are not practically or easily implementable without computer support.

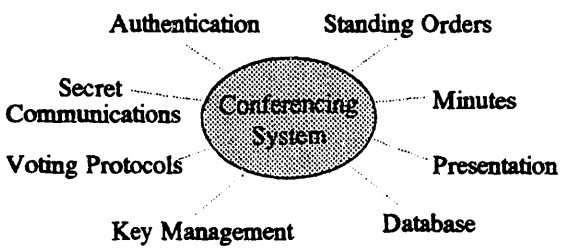


Figure 1. The conferencing system's components

Our aim was to create an application enabling the automation of formal commission management. There are certain standard procedures which are present in every commission proceeding; for example, agenda reviewing, action points processing, voting upon important decisions, collecting proposals, etc. All of these procedures are always performed according to certain rules and therefore can be formally described.

The scenario which we use was made by observing university commissions meetings. It is now possible for commission members to be seated in their offices at their terminals while participating in a meeting over an insecure computer network. This application supplies a set of available security services to be used during the commission execution. This means that the secrecy of discussions, voting strategies and created documents as well as authenticity of attendees are provided. The set of regulations which must be

observed is also available and conformed by "electronic" rules so that the commission work can proceed in concordance with the law. It is nevertheless possible to change the established protocol, if all attendees agree.

The primary purpose of the application is to provide an efficient software toolkit for commission meetings. It consists of the "working" elements common to all kinds of similar group works (Fig. 1). Although our aim was not to study the influence which such kinds of tools might have on the quality of the group outputs, we bore in mind the positive influence such an application could have on the efficiency of commission work.

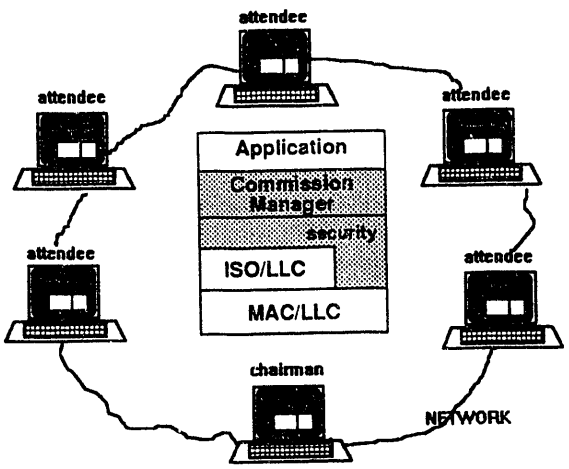


Figure 2. The conferencing system in the network

COMMISSION MEETING

In this section we give a general outline of a commission meeting. The given description corresponds to the usual university commission session.

In order to establish a particular commission, a constitutive meeting of the commission must first be held. During this meeting the commission members and their mandates and rights are to be determined. After the constitutive meeting, the general commission meetings can be held. If the commission has to be dismissed, the closing commission meeting terminates the commission activity.

The general meeting usually begins with a roll call of the members' presence (i.e. finding out who are the present attendees). The actual number of the participants is also important as the commission meeting cannot be held if the number of the present commission members is smaller than the required quorum. The commission member who are not able to participate may send their deputies to represent them at the meeting, i.e. to eventually participate in voting in the name of the absent member. If the session is not closed, there may also be some attendees who are not in fact members of a particular commission but simply interested in attending the meeting. These attendees are informants and do not actively participate in the meeting unless asked for an opinion.

The meeting agenda can be determined beforehand or the attendees can make proposals for the action points. The first action point in the agenda is usually called "Review of Agenda", if the agenda has been determined in advance, so that the members may vote upon the approval of the proposed agenda.

The further proceeding of the meeting is based on the action points, which are processed one by one. The commission chairperson manages the meeting in concordance with the standing orders.

During the processing of the particular action points different situations can occur:

- The chairperson may collect proposals about some actual topic, giving to every attendee who is an active member of the commission an equal opportunity to discuss and make proposals.
- The attendees may vote upon some important decisions or documents.
- A group of attendees may hold a short secret discussion and then return to the meeting.

The official record of the meeting is written in the form of minutes. Important documents are signed by all attendees who have taken part in the creation of the document.

SECURITY SERVICES

In this section the security requirements are described, i.e. what is to be secured. We stress some crucial

points to which special attention must be paid as they require special treatment from a security point of view. This refers to the security services available to the commission which may but need not be applied to a particular meeting.

The proceeding of meeting must be in concordance with the standing orders. The observation of the prescribed regulations must be provided and supported by "electronic" rules (i.e. enforced by security mechanisms). For example, these mechanisms can protect fairness in discussions (distribution: every attendee has equal opportunity to discuss, timing: every attendee has equal time interval for discussion), equal information level for the members of the same security group, etc.. On-line information about the regulations is also available.

The security services we need are the following:

- First of all, the commission members must be identified and authenticated (Authentication Service). It is also necessary to determine the scope of their activities before the meeting actually begins, i.e. to authorise them. There can be many different group of attendees. A particular group is defined according to the members' permissions/rights. For example, the commission chairperson may have a greater scope of activities than a "normal" attendee, who in turn has more rights than an informant.

- Although forced to transmit their votes over an insecure network, attendees can rely on the basic principles of democratic electoral systems (Voting Service). The voting security services enable the voting procedure to be performed in different modes; which one to apply depends upon the demanded security level. In some cases it is not necessary to keep the voting strategy secret. The voting can be either public (the votes and/or voting strategies) or secret.

- It is often necessary that some subgroup of the attendees hold a short secret mutual consultation (Secret Discussion Service). They receive permission from the rest of the members and conduct the consultations within a given time interval. The content of this discussion remains unknown to the other commission members.

- Meeting attendees may sometimes want to exchange some private (and therefore secret) information without interrupting the meeting proceeding (Secret Message Service). Only the sender and the receiver are involved in the conversation and the other members are not privy to any information about it.

- The minutes of the meeting present the record for the commission proceedings and can contain some parts which should remain secret or known only to some closed group of people (Minutes Service). Therefore it must be stored in a protected database so that the access can be controlled. The attendees decide during the meeting which parts of the minutes must remain confidential if it is not already determined by previous regulations.

SECURITY MECHANISMS

In this section the methods are described by which the security services in the previous section can be implemented.

Authentication

Authentication is a two-way process: the application authenticates the attendees and the attendees authenticate the commission server. This security service is called peer-to-peer authentication [2].

The attendee is authenticated by means of his/her Personal Identification Number (PIN). One possible solution is Personal Secure Environment [13], which is a software implementation of the SmartCard.

After the authentication phase the authorisation of the attendee is performed, in which the set of the attendee's access rights is determined based on the information obtained in the authentication phase.

Voting

There are three types of voting protocol: Normal Voting, Secret Voting and Public Voting.

a) Normal Voting

The following requirements are to be satisfied:

1. Only legitimate voters are allowed to vote and each of them only once.

2. The voting authority can read the votes and publish them to other voters during the voting phase.

3. Only the voter and the voting authority know which strategy any given voter adopted.

4. After publishing the vote, a voter can check if her vote has been properly counted.

We have chosen the simple voting scheme proposed in [4]. The scheme is an application of multiple key ciphers and has two useful properties:

- no interactive behaviour is required between the voting authority (the voting server) and the voters
- no secret key is required from the users.

The voting server issues the voting slips to the voters encrypted with their public keys. The voters send their votes back to the voting server encrypted with the server's public key. The RSA encryption scheme is applied.

The voting server produces n+1 keys, if n is the number of possible voting strategies. These keys, k_0, k_1,..., k_n must satisfy the condition

$$k_0 k_1 ... k_n = 1 \bmod \phi(m)$$

ϕ(m) being the Euler Totient Function [10]. The key k_0 is kept secret by the voting server and the other keys are made public.

The server issues a voting slip V to each voter:

V = (*random number*, *redundancy component*) .

Random numbering is used to ensure that the slip is not used more than once and the redundancy component is used to avoid forgery. The redundancy component can be changed for each voting session so that the same keys can be used more than once.

The voting slip is issued to the voter as

$$V^{k_0} \bmod m,$$

additionally encrypted with the each voter's RSA public key. The voter chooses the strategy i and forms:

$$V' = (V^{k_0})^{(k_1 k_2 ... k_{i-1} k_{i+1} ... k_n)} \bmod m$$

and sends V' (encrypted with the voting server's RSA public key) to the voting server. The voting server then validates each vote V' by forming and evaluating if

$$V'^{\,k_i} \bmod m = V$$

and checking for the redundancy condition. It is possible to reduce processing by sending the claimed value of the voting strategy with V' (in the worst case scenario the voting server must check for n values).

b) Secret Voting

The following requirements are to be satisfied:

1. Only legitimate voters may vote, and each of them only once.

2. Only the voter knows her voting strategy.

3. After publishing the outcome of the election, a voter may check if her vote has been properly counted. If not, she can complain without jeopardising the ballot secrecy.

4. (Optionally) Each voter can change her mind (cancel and recast her vote), also without jeopardising the ballot secrecy.

The chosen voting scheme is from [5,6]. We assume that the voting server (VS) sends to each legitimate voter her specific identification tag and then destroys the information which could reveal the identity of the voter having the specific identification tag. After this information has been made inaccessible, the second phase of the voting protocol can begin.

Let B be an individual voter with the tag t_B and voting strategy v_B. The voting protocol is then as follows:

1. B chooses a cryptographic hash function $h_B(x,y)$ and sends VS the pair $(t_B, h_B(t_B, v_B))$.

2. VS acknowledges the receipt by publishing the value $h_B(t_B, v_B)$.

3. B sends VS the pair (t_B, h_B^{-1}). VS can now compute v_B from $h_B(t_B, v_B)$, t_B and h_B^{-1}.

4. When the deadline for casting ballots is over, VS announces the outcome of the election by publishing, for each voting strategy v, the list of all numbers $h_B(t_B,v_B)$ such that $v_B=v$.

5. If B observes that her vote is not properly counted, she protests by sending VS the triple $(t_B,h_B(t_B,v_B),h_B^{-1})$.

6. If B wants to recast her ballot, she sends VS the triple $(t_B,h_B(t_B,v_B),v_B')$, v_B' being the new voting strategy. When the deadline for recasting is over, VS publishes the modified election results, where the numbers $h_B(t_B,v_B)$ have been reallocated in the list. The voter can also now check that her new vote has been properly counted. In this way the recasting of the ballot can be done only once.

c) Public Voting

The following requirements are to be satisfied:

1. Only legitimate voters may vote, and each of them only once.

2. The voting order is determined before the voting begins.

3. Every voter knows which strategy the other voters, who have already voted, adopted.

The voting order can be, for example, in alphabetical order. The voting protocol is as follows:

1. The voting server (VS) publishes the voting ordering list with n voters and the list of voting strategies.

2. The following steps are repeated for the each voter i on the list, in the order determined by the list.

3. VS sends the voting slip V=(*random number, redundancy component*) encrypted with the RSA public key of the voter with the position i on the voting list (voter v_i) to v_i.

4. The voter v_i creates a block (V, *voting strategy*), encrypts it with her RSA private key, and sends the encrypted block to the VS.

5. VS publishes the name and the voting strategy of the voter v_i .

Secret Discussion

The security service of secret discussion is defined in the following way:

1. The subgroup of attendees wishing to perform a short secret consultation asks the commission chairperson for the permission to do so and propose some duration.

2. The commission chairperson decides (or the commission members vote upon) whether the group may hold secret consultations. If yes, the maximal duration is determined.

The members of the secret consultation group requests a new session key (DES key) from the key management authority. The new session key is sent to the each secret consultation's participant in the form of the signed certificate (see [10]):

$$\text{Private\_RSA\_Key}_{Authority}(\text{Receiver\_Name}, \text{Time\_Stamp}, \text{Public\_RSA\_Key}_{Receiver}(\text{New\_Session\_Key}))$$

Secret Messages

The mechanism which enables the exchanging of secret messages between two attendees is based on the Privacy Enhanced Mail system [9]. The sender sends the message encrypted with the secret DES key encrypted with the receiver's public RSA key which the receiver can decrypt with her private RSA key.
The exchange of the secret messages does not affect the meeting proceeding.

Minutes

The organisation of the meeting record is based on the meeting agenda, as it is in the case of the proceeding. Every action point is observed separately.

Everything should be noted: proposals, discussion contributions, voting results, decisions, breaches of the standing orders, etc., so that no one (not even the chairperson) can prevent the recording of some event. The names of the attendees are also recorded in the

minutes and it is made impossible to eventually delete or add a name to the list.

Important documents are signed (digital signature, see [8]) by all competent attendees, either with agreement or disagreement. As only the particular attendee can generate her digital signature, she cannot later deny the fact of signing the document (non-repudiation service, see [2]).

The attendees who have signed a document determine the accessibility of this document. They decide whether this document should be kept confidential and determine the time when the document can be made public. This is important for the organisation of the database into which the complete record of the meeting will be stored.

The access to the information in the minutes database is controlled by capabilities (see, for example [3]). When a subject has access rights to an object (information), he gets the (object, access) pair, which is called the capability of the subject. The capabilities are dynamically managed. For example, if a document *D* can be made public, every user of the database gets the capability (*D*, read).

SOFTWARE ORGANISATION

The principal software organisation is based on the client-server model (Fig.3). The end users (attendees of the commission meeting) communicate with the application via the user interface. The users chooses one of the menu options and the application formulates, together with the necessary parameters, the task for the commission manager. The commission manager then performs the task by assigning various jobs to servers, i.e. sending requests to servers to perform some actions.

The basic elements of the software organisation are the following servers:

timing server
key server
access server
voting server
X500 server
regulations server
minutes server.

Timing server

Timing is very important for all security services. The role of the timing server is to provide the precise time information for the security protocols.

The rights owned by some subject are valid if certified by the certification authority. The certificate is always a temporary assignment of rights: it expires with the given date.

In the authentication phase it is also necessary to define the time-out interval (authentication deadline) so that after this interval has expired the session can begin.

In voting protocols the precise timing is of crucial importance. A deadline is determined for the duration of secret consultations.

Key server

The key generation is performed by the key server. The distribution of the RSA key pairs is accomplished prior to the session: these keys are stored in the Personal Secure Environment (a directory, for example). The key server generates keys for voting protocols, secret session keys for the service of the secret (closed group) consultations and similar purposes.

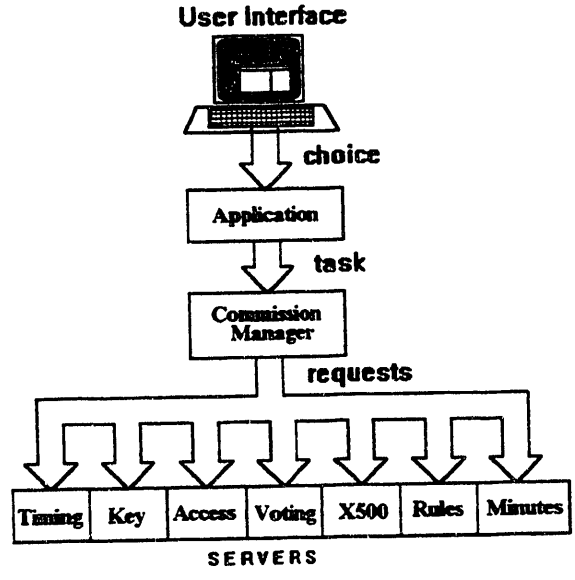


Figure 3. The software components

Access server

This server performs the authentication of the attendees and determines their set of rights, i.e. authorises them. The access server relies on the information obtained from the timing server.

If it is necessary to examine somebody's set of access rights during the meeting execution (after the initial authentication phase), the request is sent to the access server. This need may, for example, arise from the voting protocol, if only the subgroup of the attendees may vote. Because of this and similar situations, the authentification is performed on many levels.

Voting server

This server performs the voting protocols. It relies on the timing server and communicates with the proceedings server.

X500 server

This server is a Directory User Agent and enables the application's accessing the Directory Information Base [11].

Regulations server

The regulations server communicates with the local regulations database and ensures the validity of the meeting proceeding. In other words, the regulations server ensures that every official procedure is in concordance with the regulations. For every procedure certain conditions must be satisfied and the execution must follow the predetermined order. It is possible to neglect regulations, if the commission chairperson decides to do so and the attendees agree, but they are warned against breaking rules and informed that this event will be noticed in the minutes and submitted for arbitration.

Minutes server

The minutes server gets "reports" from all other servers and create an official report (i.e. stores the collected information into the minutes database). The data organisation is based on the time order, so that every information unit has a time stamp provided by the timing server.

Example 1: Voting service

The example of the communication between user and software components is shown in Figure 4.

The commission chairperson chooses the option "Voting" from the Menu with the available security services. She is then asked for parameters necessary for the voting procedure, i.e. what is the mode of voting (normal, secret or public), what is to be voted upon, who are the voters and what are the possible voting strategies.

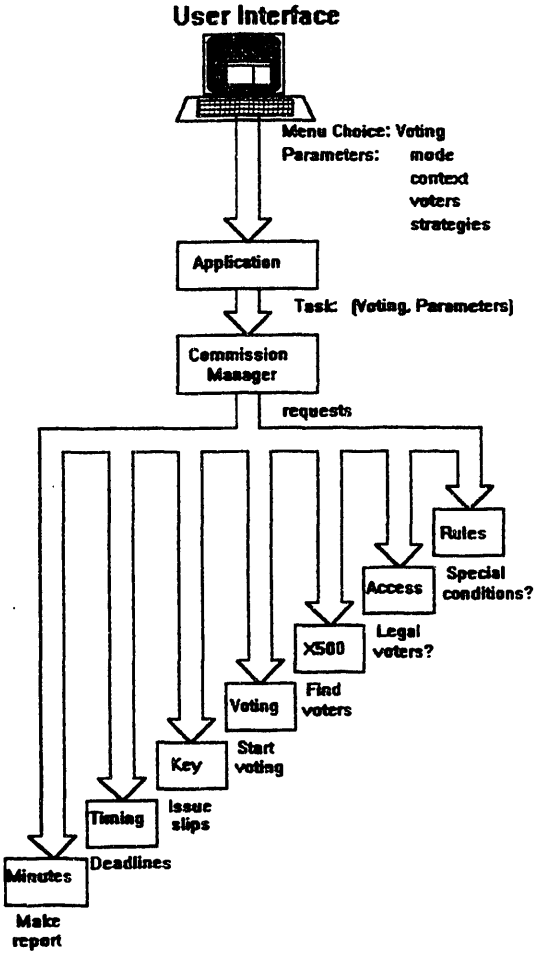


Figure 4. The voting service

From these parameters the application formulates the task and sends it to the session manager who co-ordinates the work of the servers. The manager sends the request to the rules server to examine the legal conditions of the voting. For example, if not all attendees are members of some very important commission they are, therefore, not allowed to vote upon the current subject. The rules server determines

if some further security measures are to be taken - for example, to examine the access rights of the proposed voters. This request is accomplished by the access server.

After all legal conditions have been satisfied, the voting protocol can begin. The request is sent to the voting server which in turn requests keys and timing service from the key server and the timing server, respectively. After the voting has been completed, the minutes servers makes a report which is appended to the meeting minutes.

Example 2: The commission data structure

The following example illustrates a data structure representing an university commission (Fig. 5).

In the section *Commission meeting* the types and ordering of commission meetings are mentioned. The first meeting to be held is a constitutive meeting and the last one is the closing meeting. After the commission has been constituted, the general meetings can be held. Every general meeting has an agenda with action points to be processed. The results of every activity (discussions, voting, important decisions) are stored into the database according to the time order.

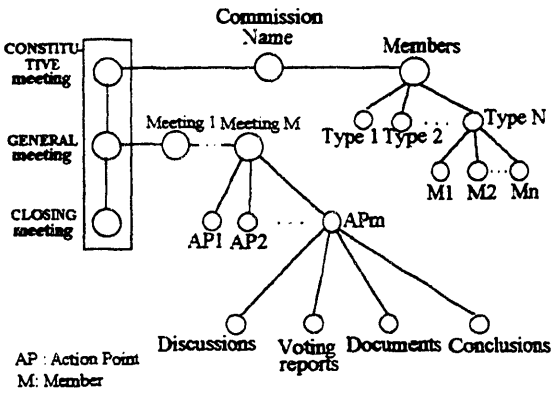


Figure 5. The commission data structure

SUMMARY

In this paper a model of the secure application for the automation of the commission work is presented. The purpose of the application is to supply an appropriate working environment comprising all important elements of the commission proceeding. This working environment enables collaboration over a computer network.

Special attention is paid to legal and security aspects. The commission proceedings consist of prescribed elements which require a certain security level of processing. These elements are defined as security services and create a set of tools available to the attendees. Attendees and commission chairperson may choose the security service. However, the chosen service must be in concordance with the legal regulations.

The mechanisms used to implement the security services are described. The application is organised based on the client-server model. The client in this scheme is application which proceeds the user's request to the commission manager. The commission manager co-ordinates the servers' activities.

The model described has not yet been implemented. It will be implemented in the Security Development Environment [13]. Our aim is to make an experimental system which could assist in the better understanding of the processes mentioned.

BIBLIOGRAPHY

1. Nunamaker, J.F. et.al., "Electronic Meeting Systems to Support Group Work ," *Communications of the ACM*, Vol. 34, July 1991, pp.41-61

2. Muftic, S., *Security Mechanisms For Computer Networks*, Chichester: John Wiley & Sons, 1989

3. Denning, D.E., *Cryptography and Data Security*, Addison-Wesley, 1982

4. Boyd, C., "Some Applications of Multiple Key Ciphers," *Lecture Notes in Computer Science* , V330, 1988, pp.455-467

5. Salomaa, A., "Verifying and Recasting Secret Ballots in Computer Networks," *EATCS Bulletin*, 44, 1991

6. Nurmi,H., A. Salomaa, and L. Santean, "Secret Ballot Elections in Computer Networks," *Computers & Security*, 10 (1991), pp.553-560

7. Diffie, W., and M.E. Hellman, "New Directions in Cryptography," *IEEE Trans.Inf.Theory* , Vol. IT-22, No.6, November 1976

8. Rivest, R.L., A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public Key Cryptosystems," *Communications of the ACM* , Vol. 21, February 1978, pp.120-1261.

9. Linn, J., Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures (RFC 1113), August 1989

10. Denning, D.E., "Protecting Public Keys and Signature Keys," *Computer*, February 1983, pp. 27-35

11. CCITT, "Recommendation X.500: The Directory - Overview of Concepts, Models and Services," Melbourne, 1988

12. Akl, S.G., "Digital Signatures: A Tutorial Survey," *Computer*, February 1983, pp. 15-24

13. Schneider, W., "SecuDE: Overview (Version 3.0)", Institut fuer TeleKooperationsTechnik, Darmstadt, March 1992

14. Desmedt, Y., "Society and Group Oriented Cryptography," *Advances in Cryptology - CRYPTO '87, Lecture Notes in Computer Science* , 293, 1988, pp.120-127

WORKFLOW.2000 — ELECTRONIC DOCUMENT AUTHORIZATION IN PRACTICE

Addison Fischer
Fischer International Systems Corporation

ABSTRACT

WorkFlow.2000 is a product designed in conjunction with the Electronic Document Authorization (EDA) protocol to enable EDA to be easily and seamlessly exploited in "real-world" business. Its mandate extends even beyond this — to providing a feasible vehicle for providing the true "paperless office."

This paper describes some of the philosophy behind both WorkFlow.2000 and EDA, and the technology developed to implement them as a commercially viable product.

THE MOTIVATION

To unfold the philosophy and direction of both WorkFlow.2000 and EDA, we start by posing the fundamental question:

How does one really create the truly "paperless office?"

We use as an underlying premise that "paper" serves three basic functions: 1) to accept information (writing); 2) to store information (filing); and 3) to transmit information (mailing).

A fourth — and equally important — function is served by paper, namely: 4) To seal a commitment.

This last function is derived from paper's ability to record an ink signature which is taken to reflect the signer's acknowledgement, approval or commitment concerning the other information recorded on the paper.

Before the advent of electronic media, paper was unquestionably the dominant means for performing these functions; it probably still is. But there are well-recognized business advantages gained by the eliminating paper business documents in favor of electronic media.

It is technically possible, especially with the invention of digital signatures as part of public key cryptography, to perform all four of the above functions electronically using computers.

Some of the important advantages of doing this include:

(a) Greatly improved communication speed. Electronic business messages can be sent instantly and cheaply over telecommunication networks. (FAX, which sends paper images electronically, while fast, has the disadvantage that it actually increases the total amount of overall paper consumption, and is generally not suitable for further subsequent computer analysis.)

It is also possible to send a bulk of messages cheaply by loading them onto (physically small) magnetic media for physical transport.

(b) Reduced storage cost. A single half-pound tape cartridge costing less than three dollars, suffices to archivally hold the equivalent of over 7,000 pounds of business documents. This means that a single 20' x 40' room suffices to archive electronically the same information that would require 4,000,000 sq ft of paper information. At a cost,say of $10/sq. ft./yr., this represents a (real estate) savings of $40,000,000/yr.

(c) Improved searching capability. Searching with computers is immeasurably faster than searching paper by hand. This is especially true if the search criteria are not identically the same criteria under which paper was filed. Electronic records can be multiply indexed with little additional overhead; and an exhaustive search, if required, is thousands of times more efficient than similar exhaustive search through comparable paper — which may not even be feasible.

(d) Electronic data can be quickly and easily copied for backup. A fire, or other disaster can easily wipe out paper archives. However, a tape cartridge can be easily copied in a few minutes and stored off-site. Copying the paper equivalent — 700,000 pages — is not so easy.

In the interests of being politically correct, we might note parenthetically that electronic business should save millions of acres trees each year.

In approaching full "electronification" of the office, it is important to appreciate that paper has enormous flexibility:

- Paper is a tangible medium: it is subject to being marked, transmitted or stored in a variety of ways. New information, including signatures, can be added as needed.
- Paper is always processed by human perception (with the relatively uncommon exception of OCR handling). Therefore unexpected deviations can be grasped with human comprehension.

Any electronic replacement must be sufficiently flexible to handle most exigencies, while compensating with sufficient additional incentives to make it attractive to both the end-userand

In addressing these needs, WorkFlow.2000 can be viewed in several different ways:

- It can be viewed as "Smart paper" where the replacement for a business document is actually a computer object that is easily integrated into computer applications, yet provides flexibility to accommodate the end-users. "Smart paper" knows its mission and how to accomplish it. If necessary, part of its mission might be to accept digital signatures on appropriate information.

- It can be viewed as a "travelling object oriented program" (TROOP). This transcends the notion of "electronic forms" by providing the ability to write applications that can do anything, even in a "fully distributed" environment. Each WorkFlow.2000 cell (to be described in more detail shortly), contains definitions for user interaction, coupled with a program for handling that data. Each WorkFlow.2000 cell is written in a high level language and can draw upon libraries of tools.
- It can be viewed as an "EDI vector" in two ways:

  - It can build, carry and display EDI information;
  - It provides a simple way for an organization to get started with EDI.

- WorkFlow.2000 can also be viewed as a general purpose tool for implementing EDA. Electronic Document Authorization, which can handle authorization for virtually any critical application, such as:
- Business documents — the most obvious and most universal.
- Design signoff — WorkFlow.2000 can manage any computer object, including CAD/CAM. WorkFlow.2000 is able to direct and interface with CAD/CAM processes.
- Program validation — In order to protect software against viruses and worms, especially software deployed acrossmany computers, it is useful to have it digitally signed by trusted uthorities. WorkFlow.2000 has the ability to move and install new software to any platform in a secure fashion.

WORKFLOW.2000 AS "SMART PAPER"

Going beyond the simple "electronic forms" paradigm, WorkFlow.2000 incorporates a number of additional ingredients to allow replacement of paper both within and across organizational boundaries.

WorkFlow.2000 borrows themes emerging from object-oriented paradigms and combines them with concepts evolved from "distributed computing" research.

The basic unit of WorkFlow.2000 is the "cell." Each WorkFlow.2000 "cell," which can be thought of, in a manner of speaking, as a "piece" of "smart paper," is actually instantiated as an " object" containing data, bound together with the full instructions with which to process it.

It is a "computer object" that "knows its mission and how to accomplish it."

The WorkFlow.2000 cell definition is incorporated into each "instance" cell from the original "prototype" cell at the time the instance is first launched. These definitions, the cell instructions, are specified in two languages. Logic, computation, control, and routing instructions are formulated in the high-level REXX language. This is coupled to the second language, a TeX derivative, that facilitates easy representation of the end-user displays. REXX logic has control over the entire object, including tailoring the TeX displays.

The WorkFlow.2000 driver has its own (built in) compiler and interpreter to ensure consistent behavior of WorkFlow.2000 objects regardless of the type of computer handling a particular piece of " smart paper."

REXX is sufficiently powerful to allow facile expression and computation of virtually any logic, including routing. "Smart paper" can be as rigid or as flexible as appropriate.

Internal Functions

As needed, the REXX control can invoke an assortment of internal ("built-in") functions which are supplied as part of the universalWorkFlow.2000 function suite. In addition to conventional REXX builtin functions, WorkFlow.2000 also supplies its own native functions to:

- specify the next destination(s) to which the WorkFlow.2000cell should be routed after completing the current execution phase. In general, WorkFlow.2000 can be routed through any electronic mail system.
- facilitate creation and validation of digital signatures.
- facilitate construction and parsing of EDI structures.
- allow creation of simple "spinoff" messages generated by the WorkFlow.2000 logic. These messages are (generally) not WorkFlow.2000 cells, but can be, for example:

  - simple acknowledgements
  - inform. tion to a central server that tracks and possibly coordinates the progress of WorkFlow.2000 cells
  - generated EDI transaction sets mailed to external organizations
  - audit information sent to a central location

- functions to access files and databases that may exist on the current platform.

External Functions

WorkFlow.2000 cells may also invoke "external" routines which are not inherently part of WorkFlow.2000. Examples include:

- Word processors
- Spreadsheets
- EDI processors - for specialized custom processing
- Functions accessing specialized types of databases
- CAD/CAM functions
- Custom business applications
- Electronic mail (Although WorkFlow.2000 is generally packaged with built-in access to electronic mail, it is possible that specialized or non-standard systems may require unique interfaces.)
- Batch processing
- Any other necessary function

It is the responsibility of the WorkFlow.2000 cell author to know (or create logic to determine) whether a particular external application is available on a platform before invoking it. In fact, since some functions may only exist on certain platforms, WorkFlow.2000 may sometimes contain logic to "move itself" to another platform especially in order to to access a special program. Typically, such programs reflect unique operations, such as data gathering, that have no user interaction; for example:

- Access to batch programs (WorkFlow.2000 might need to move to a "batch driver")
- Access to programs for using specialized databases
- Access to programs that perform special formatting (e.g., graphical manipulation, etc).

Digital Signatures

Another noteworthy ingredient contributing to the "smart paper" paradigm is provided by its inherent ability to perform digital signatures on arbitrary data.

This mimics the flexibility of (real) paper, where anything can be signed. Through use of built-in functions, WorkFlow.2000 is able to apply a digital signature on any material — from an internal "string" to an external file. The signature is created as a REXX variable — and may be combined with the data, stored separately— or treated in any other way as needed, since it is strictly under control of the cell 's logic.

As a tangible illustration, consider the flow of the archetypical Electronic Purchase Order. After starting as a "smart paper" Requisition, it collects enough information to construct an EDI Purchase Order X12 ("850") transaction set.

Whenever signatures are necessary, WorkFlow.2000 must apply them to the "850" transaction set itself — since this is what the EDI recipient will ultimately receive. Thus, through all stages of approval, WorkFlow.2000 must present the transaction in "visual" (translated form), but apply signatures to the "850" format.

The WorkFlow.2000 cell is therefore involved in all stages of EDI creation:

- coaching the creation of data with helpful displays,
- constructing the actual, final, digital representation of the corresponding 850 transaction set,
- interfacing with users for approval and digital signatures.

This reveals several important points in the application and use of digital signatures:

- In performing a digital signature, a user must be aware of exactly what is being signed.
- Since digital material stored in a computer cannot be directly perceived, the user must rely on software to render a faithful representation.
- Signed material must be in "final form" — i.e., it cannot be translated" in any way — a digital signature is valid only if the signed data is wholly identical — it cannot be modified or filtered before verification.

Similarly, in verifying a digital signature:

- The material cannot have been "translated" since the signature was applied.
- Any "conversion" must occur after verification.
- For signatures to have historical or archival significance, the original data must be stored. In this case, it is usually pointless to store a "converted" image, since it cannot participate in a validation, and could presumably be recreated if needed.
- When verification and processing occur together, the processing must be able to handle the material — as it was sent. If conversion is necessary, it must follow signature validation and precede the processing.

By guiding the flow and process of data WorkFlow.2000 can internally invoke conversion steps,as needed, to feed other embedded processes.

WORKFLOW.2000 AS A SELF-REFERENTIAL MAIL-ENABLED APPLICATION

A Mail Enabled Application (MEA) is a program which allows a user to electronically mail data while using the program.

WorkFlow.2000 goes further — not only is a WorkFlow.2000 application equipped to mail data, but it mails itself as well!

In fact, the transmission of itself, together with its latest generation of data, is an integral step of its distributed nature.

WORKFLOW.2000 AS A "TRAVELING OBJECT ORIENTED PROGRAM"

The "object" nature of WorkFlow.2000 stems from the fact that each instance of a cell contains a state; data; and instructions for processing the data.

Its "travelling" nature stems, of course, from the fact that it moves from destination to destination to satisfy whatever distributed task it is designed to solve. This includes, for example, moving to any destination where information must be collected or desseminated. This certainly includes users, but also encompasses platforms with special data, mainframe batch drivers, special servers,factory floor processes, etc.

An object can be programmed with any criteria at all in determining its next destination: from the use of fixed built-in constants, to being totally obedient to whatever a user supplies, to making a complex decision based on information collected during its

For applications where it is useful to centrally track a WorkFlow.2000 cell and know where it has traveled or presently resides, WorkFlow.2000 logic can dispatch electronic messages back to a central tracking focus destination. For example, it might be generated whenever a WorkFlow.2000 form isused and is ready to transfer to another platform.

If the usefulness of tracking varies, then a cell might generate it conditionally only under the appropriate conditions.

At the designated focus, anunattended task could automatically receive and digest the information into a log file.

This same technique applies as well to audit or archival information.

WORKFLOW.2000 AS A "DISTRIBUTED DATABASE"

Another useful way to view WorkFlow.2000 is as a new type of distributed database. In the same sense that Sun claims "the Network is the Computer," a sufficiently powerful TROOP can be viewed as a paradigm for a new class of distributed

Under this paradigm, WorkFlow.2000 cells themselves are viewed as "objects" integral to the amalgamated database. Each cell "knows" how to produce a desired result, including the locations to which it must travel. Concerns about where specific datum is located are thus "pushed into" the interior of WorkFlow.2000 cells.

For example, the "database object" may be a WorkFlow .2000 cell that compiles a monthly report from pieces lying across a number of personal computers and mainframes. Some parts of the aggregate database may actually be knowledge that must be solicited from users.

The data could reside anywhere in the organization.

As long as they can be reached, a WorkFlow.2000 cell can be designed to gather, assimilate and return the necessary information.

While the concept of "distributed" database spans a number of disparate notions, and the WorkFlow paradigm certainly does not cover all of them; viewing WorkFlow.2000 itself, the "glue" so to speak, as the "distributed database" brings a useful object-oriented flavor to the treatment of certain tasks — including many business needs that are likely to arise in a distributed corporate environment.

WORKFLOW.2000 AS AN "APPLICATION GENERATOR" FOR A DISTRIBUTED ENVIRONMENT

WorkFlow.2000 is an "application generator" which provides:

- A high level language in which to formulate solutions.
- Tools to converse with end-users.
- Means to interface to a variety of databases.
- Ability to interface with existing applications.

The addition of its "travelling" abilities and object oriented flavor make WorkFlow.2000 a "distributed application generator" capable of providing applications executing across a span of platforms as needed.

In a distributed environment, a TROOP seems ideally suited for creating applications.

WORKFLOW.2000 AS AN "EDI VECTOR"

To eliminate "paper" flowing between enterprises, WorkFlow.2000 provides tools to data into standard EDI

formats, and then both mail and receive these. It is sufficiently powerful to exploit any EDI format.

It can be viewed an EDI Vector in two ways:

- In the technical sense, WorkFlow.2000 provides the vehicle by which EDI can be constructed, digitally signed (as needed), mailed, received, viewed, and (optionally) transferred to other corporate systems for processing.
- In a business sense, WorkFlow.2000 can bring (e vector") widespread EDI capability into an organization immediately.

Even without any other electronic processing by an organization, WorkFlow.2000 allows full EDI construction and display on any computer, with no additional software requirements.

Consequently one could begin using EDI immediately without committing to, or developing, other processing systems. Once EDI is being used, migration into future systems can evolve gradually as business needs are defined.

By supporting the construction and receipt of all defined EDI formats, WorkFlow.2000 supplies the means for an organization to immediately handle all standard inbound EDI — if only to the extent that they are displayed on a screen, or printed out. Since no other computer processing is necessary, an organization does not need to analyze the particular "subset" of the standard they are willing to accept — they may accept them all. Similarly, outbound transactions can be composed using the full capability of the standard.

This allows immediate EDI capability — transactions that aren't processed by computer can be viewed through computer displays and acted on just as if they were paper.

Even if an organization never performs any special programmatic processing, a number of substantial benefits are still realized:

- The transmission is electronic, and is therefore usually faster by several orders of magnitude.
- The volume of (archived) material stored is substantially less. Electronic documents can be stored, by weight or volume, approximately three to five orders of magnitude more efficiently than paper. For example a single half-pound tape cartridge reflects the same information as roughly three and a half tons of paper business documents.
- Paper handling is reduced because EDI is already electronic, and can be routed electronically.
- Once an EDI document is archived, even if it was not specially processed on arrival, it can be processed (even years) later as needed by subsequent programmatic tools. Digital documents can be electronically audited and searched — saving magnitudes of time compared with manual paper operations ash as well as being more accurate and thorough.
- Of course, having EDI in digital format encourages more automation.

As long as EDI can be processed — even if only by viewing it on CRT displays as a substitute for paper — it invites the beginning of automation. The first step is identifying a "simple " subset for which automatic processing can be easily developed. Conforming material can then be programmatically filtered and handled, leaving the "residue" for manual handling. As time goes on, an organization will devise better ways to automate more, so that the manual residue shrinks.

Thus, WorkFlow.2000 allows the substantial benefits of EDI to be realized without demanding an "all or nothing approach."

PROTECTION AND VERIFICATION

WorkFlow.2000 is designed to operate as "intelligent paper" in a totally distributed manner — operating across multiple platforms of possible varying degrees of security.

The integrity and trustworthiness of WorkFlow.2000 cells are critical since they have the ability to

- potentially read and write file data,
- formulate the actual binary data on which digital signatures are applied,
- spin off generated electronic messages.

For these reasons and others, it is therefore crucial to insure that the WorkFlow.2000 instructions themselves cannot be corrupted or spoofed. Although WorkFlow.2000 cells represent "travelling programs" which might originate anywhere; as a practical matter, the underlying driver must be able to automatically distinguish (with minimal ongoing user decision) legitimate cells which conform to each user's acceptance criteria; yet automatically reject those that do not.

What is EDA?

EDA — Electronic Documentation Authorization — is a digital signature protocol designed to provide authority assurances within and across organizational boundaries.

One goal of EDA is to provide a framework allowing valid exercise of authority, such as spending authority, to be trusted across enterprises in such a way, ideally, that this authority can be verified entirely "electronically" without requiring human attention or decision. The archetypical example for this might is an Electronic Data Interchange (EDI) transaction representing a Purchase Order.

Currently, for example, paper purchase orders, which fundamentally represent contracts, are generally signed (or certainly should be). Current EDI is in its infancy, and is typically confined between trusting partners communicating across controlled channels.

In the future, as EDI becomes more widespread, one presumes EDI will be conducted across any and all channels. Just as paper documents are presently treated as mail, so one assumes that EDI will also eventually come to use any available communication medium.

Unlike electronic mail, however, electronic documents reflect business commitments, and it takes very little imagination to understand how their misuse, or forgery, could result in major losses.

EDA conforms with the PKCS.7 formats which were developed as an industry standard in 1991 under the auspices of a group representing a number of influential software organizations including: Apple, DEC, Fischer International, Lotus, Microsoft, Novell, RSADSI, Sun.

The PKCS specifications themselves are a protocol based on International standards such as ISO/CCITT X.500 and IS 9796.

EDA uses X.509 certificates for backbone identification together with standard PKCS structures for carrying authorization and enforcement specifications.

EDA provides that a digital signature that indicates the use of authority, such as the authority to spend a certain amount of money, is associated with an "Authorizing Certificate" showing that sufficient authorization has been granted by the signer's organization.

A chain of authority delegation can be traced through the hierarchy of Authorizing Certificates to a common certifier that is well-known and accepted as trustworthy.

To enforce the proper exercise of authority, EDA allows each delegator to stipulate that one or more digital co-signers, from an (indicated) list of possible candidates, must ratify any use of authority by the delegatee.

The use of co-signatures may be stipulated at any or all levels, including the very highest, to minimize the risk of corruption or misuse.

While the most obvious use of EDA is to control money transactions, it may also be used to control any other sensitive task for which the controlled exercise of authority is appropriate.

- Money
- CAD/CAM release
- Sending instructions to automated remote devices

Electronic Document Authorization:

- Uses standard X.509 certificates to carry identification specifications.
- Uses standard PKCS.7-compliant digital signature structures to carry authorization and enforcement specifications.
- Provides full authentication to all parties.
- Provides full and accountable authorization as part of digital signatures.
- Protects the issuing organization by insuring that digital documents are subject to at least as many checks and balances as currently exist with paper documents. Because this protection is built-in to a user's EDA (Authorization) Certificate, this arguably gives even stronger assurance.
- Protects the receiving organization by providing a full (machine-) verifiable authority audit trail. The use of authority at each level is demonstrated by explicit delegation and explicit associated safeguards.
- Protects all parties, by mitigating the possible damage that might occur on compromise of someone's (secret) private key. Co-signature requirements inhibit illegal use of a compromised key.
- Insures, by providing mandatory co-signatures, that no single individual (even a "Certification Authority") can deliberately (or accidentally) abuse their authority or corrupt the system.

By increasing the the number of co-signature requirements, protection can be extended to guard against collusion by any particular number of multiple parties.

This allows the risks to be reduced to whatever level is deemed appropriate to the situation. The exercise of greater authority can be protected with imposition of greater safeguards.

EDA facilitates abuse-resistant authorization which can be verified automatically without human intervention or decision.

The principal feature of EDA is the "Authorization Certificate" which defines the authorizations and constraints attributes associated with a particular public key. Broadly speaking, the "authorizations" define the powers which an individual is entitled to authorize through the use of their digital signature; while "limitations" define constraints.

An important constraint, which mirrors classical business practice,is that digital signatures performed by the owner of the public key are not to be considered valid (or "ratified") unless accompanied by other digital signatures.

This allows an organization to grant explicit authority to an individual, while also providing a means to prevent the in-

dividual from exercising the authority in a careless, capricious, or corrupt fashion.

Given that digital signatures are fundamentally exercised by individuals, one of EDA's goals is to recognize that authority is wielded by individuals on behalf of organizations, but that individuals are unpredictably fallible and occasionally corrupt.

In assigning authority, organizations are again faced with the prospect that the individuals within the organization who manage authority are themselves subject to the same concerns of fallibility and corruptibility. An individual who has the power to defi ne authority, has the power to assign authority to himself, for example.

Therefore the EDA paradigm — of controlling authority through explicit specification and co-signature requirements — is applied through the entire delegation hierarchy. At the very top, we have a panel of high level certifiers whose responsibility is to construct high-level Authorization Certifications for participating (member) organizations.

EDA provides that these high level authorities can (should) themselves be subject to co-signatures requirements — so that no one acting alone has the ability to corrupt the system.

WORKFLOW.2000 USES EDA TO SOLVE THESE MAJOR SECURITY ISSUES

EDA provides digital signature capability substantially equivalent, perhaps better, than that of signatures as they are currently used with paper.

(1) As described earlier, the logic within a WorkFlow.2000 cell can invoke built-in functions that apply EDA digital signatures to data, such as EDI transactions for example which may be used outside of WorkFlow.2000, for which a future recipient needs to know:

- that the data/document is unchanged since it was signed. (a fundamental characteristic of digital signatures).
- who signed the data/document (a characteristic of the X.509 framework underlying EDA), but also:
- that the data/document is authorized in full conformance with the rules set forth by the organization for whom the individual is acting. This important feature of EDA can be verified automatically without user decision.

Conversely, WorkFlow.2000 also has a built-in routines to verify previously created digital signatures.

- A WorkFlow.2000 cell can verify whether signatures use EDA, and whether they reflect the proper authorization and satisfy the enforcement constraints (e.g., necessary co-signatures are also present). If so, WorkFlow.2000 is able to take a quick path that does not require human decision or investigation.
- If a digital signature is present, but conforms only to basic X.500 or simple PKCS formats, then the WorkFlow.2000 cell could take an appropriate action — such as requesting a (human) decision to accept the document based on only the X.500 identification information.
- If no digital signature is present, then the cell can take whatever appropriate course of action makes good business sense, such as routing it for an "exception review" to be specially scrutinized, investigated, and evaluated.

(2) EDA insures that WorkFlow.2000 cells themselves cannot be corrupted and that "Trojan horse" cells cannot be introduced.

In the final analysis, WorkFlow.2000 cells are themselves simply data bits. The threat exists of interception and modification by someone sufficiently clever, who might be able to change the intended operation of the cell logic to induce erroneous results or data damage, or to incorrectly capture and reveal private data.

A similar "Trojan horse" attack exists where harmful processing is subtly embedded in a seemingly innocuous WorkFlow.2000 cell, or in one which mimics a normal cell.

To protect against such mischief, WorkFlow.2000 demands that each prototype cell — i.e., the underlying specification and logic — be signed with EDA. This is replicated in all derived "instances."

Before executing even the first instruction of any cell, WorkFlow.2000 insures that the cell logic is properly signed with EDA, and the EDA authorization hierarchy can be traced to a EDA signer trusted by the current user (see [1]). This is done in a few milliseconds, and insures that damaged or untrusted WorkFlow.2000 cells cannot execute.

When WorkFlow.2000 cells are executed under a batch driver, the EDA trust root is specified as a parameter as part of starting the driver.

(3) EDA insures that the data carried by WorkFlow.2000 cells cannot be corrupted.

The first type of validation provides EDA digital signatures to protect specific data constructs within the WorkFlow.2000 data set, such as EDI strings, that may have historic, archival or cross-enterprise significance. This is carried in a particular instance of a cell and is fundamentally tied to a particular piece of data.

The second type of validation provides EDA signatures to protect users against cell logic damage, and against "Trojan horse" or mischievous cells. This applies to the prototype

cell logic until it is retired and replaced with an enhanced definition. It is carried by every instance.

The third type of validation provides EDA signatures to the aggregate of the entire data set associated with a particular cell between the time it leaves one station and arrives at the next destination. Such data does not have long-term significance, as in the previous cases, but serves to protect against unexpected execution results that might arise from covert tampering of internal REXX "variables" between executions.

Since WorkFlow.2000 variables and state change with each execution, this protection is re-applied each "hop." It is computed as the WorkFlow.2000 cell writes itself for transmission, and is re-computed by the WorkFlow.2000 driver at the destination as the variables are accepted.

SUMMARY

In describing the genesis of WorkFlow.2000 as a vehicle for moving toward a truly "paperless office", we have seen how it can be interpreted under a variety of paradigms such as "smart paper" — a new breed of electronic forms; as a TRavelling Object Oriented Program; as an Application Generator; as a vector for EDI; and even as a distributed database.

Although EDA has a significant role in all of these, its special utility is most apparent as an authorization tool replacing hand signatures when WorkFlow.2000 is used to replace paper -- both within the office ("smart paper ") and across enterprises (as an "EDI vector").

The relationship between WorkFlow.2000 and EDA seems inherently symbiotic. It is difficult to imagine that EDA (or any digital signature protocol) could be used as a general substiitute for autographic signatures without a general-purpose underlying tool such as WorkFlow.2000. Similarly, to completely fulfill its mission of the paperlesss office,WorkFlow.2000 requires something with the power of EDA.

REFERENCES

1. Fischer, A., "Electronic Document Authorization, " Proceedings of the 13th National Computer Security Conference, 1990, also published as:

2. Message Handling Systems and Application Layer Communication Protocols; P. Schicker, E. Stefferud, eds.; Elsevier Science Publications B.V., 1991, "Electronic Document Authorization," p. 75.

3. ECMA, "Security In Open Systems: Data Elements and Service Definitions," ECMA-138, Dec. 1989.

4. Gasser, M., and E. McDermott, "An Architecture For Practical Delegation In a Distributed System," Proceedings of the 1990 IEEE Symposium on Security and Privacy.

5. Linn, J., "Practical Authentication For Distributed Computing," Proceedings of the 1990 IEEE Symposium on Security and Privacy.

6. SDNS Access Control WG, "SDN.802: Access Control Document," July 1989.

7. CCITT, "X.509: The Directory: Authentication Framework, " 1988.

8. RSA Data Security Inc., "PKCS.1: RSA Encryption Standard," 1991.

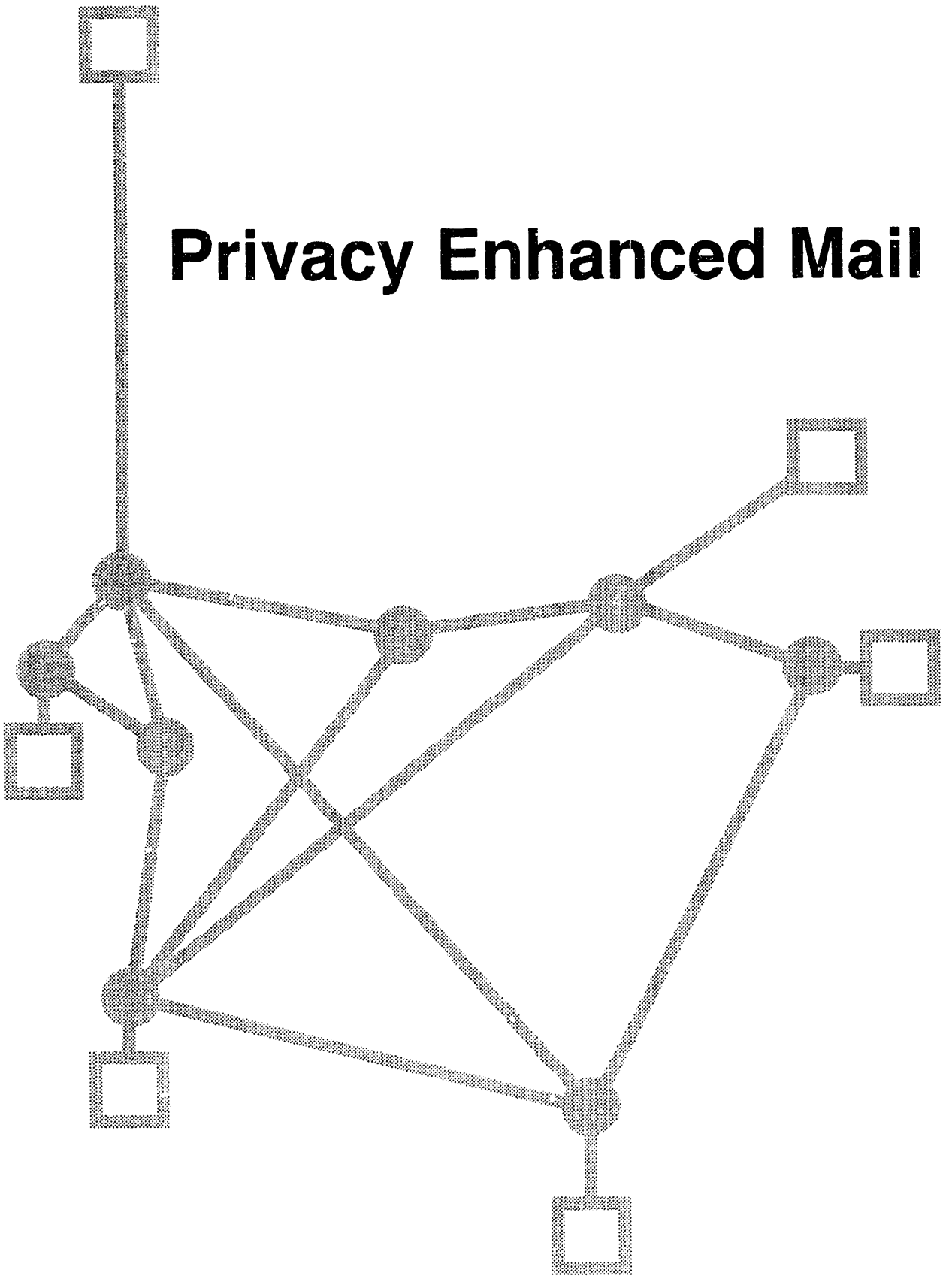
9. RSA Data Security Inc., "PKCS.7: Cryptographic Message Syntax," 1991.

10. RSA Data Security, Inc., "PKCS.9: Selected Attributes,e 1991.

11. Ankney, R., "Electronic document Authorization Using Public Key Cryptography," April 1992.

12. ISO/IEC, "IS 9796: Digital Signatures Scheme Giving Message Recovery," 1991.

Privacy Enhanced Mail



Security Issues of a UNIX PEM Implementation

James M. Galvin
David M. Balenson
Stephen D. Crocker
Paul C. Clark

Trusted Information Systems, Inc.
3060 Washington Road
Glenwood, MD 21738

Paper not received in time for publication in the Workshop Proceedings

Implementing Privacy Enhanced Mail on VMS

Michael Taylor

Digital Equipment Corporation
Littleton, Massachusetts

ABSTRACT

This paper presents the lessons learned from four years of building Privacy Enhanced Mail (PEM) prototypes on VMS. It describes the implementation of four prototypes, the evolution of the technology building blocks, and the successful deployment of the resulting technology.

INTRODUCTION

The Digital Equipment Corporation internal computer network supports more than 80,000 systems and 100,000 users. A significant proportion of the traffic carried by the network is electronic mail, generated by the VMS Mail Utility and other mail utilities. The increasing use of electronic mail for business and personal communication, together with the vulnerability of communication links, has resulted in the need for several kinds of security services for mail:

- o Confidentiality protects a message so only the intended recipient(s) can read it.
- o Message integrity assurance allows detection of any modification to the message.
- o Authentication assures that the originator named in the message is indeed the originator.
- o Non-repudiation assures the recipient of the ability to prove to a third party that the originator named in the message is indeed the originator.

While the need for secure electronic mail has been acknowledged for a long time, no such mechanism has gained widespread use. Since 1988, four prototypes of secure electronic mail have been built within Digital Equipment Corporation. Three of these prototypes have been based on a series of Request For Comments (RFC) released by the Internet Activities Board Internet Research Task Force's Privacy and Security Research Group (PSRG) and the Internet Engineering Task Force (IETF)'s Privacy-Enhanced Mail (PEM) Working Group. The security services were provided by the mail user agents, i.e., the software that interfaces with the users of mail. No special requirements were imposed at intermediate relay sites or the underlying mail transport protocol. This approach allowed the prototypes to be deployed on a site-by-site or user-by-user basis without impact on other mail services.

OVERVIEW OF THE FOUR PROTOTYPES

The following is a brief description of the prototypes. Each prototype is described in more detail in later sections.

The first prototype of a confidential mail system for users of the VMS Mail Utility was implemented in the second half of 1988. This prototype provided message confidentiality by encrypting the mail message using the Data Encryption Standard (DES). No cryptographic key management was provided.

In 1989, work was started to develop a Privacy Enhanced Mail (PEM) prototype for use with the VMS Mail Utility in conformance with the Internet RFC-1040 written by the Internet Activities Board Internet Research Task Force's Privacy and Security Research Group (PSRG). Conformance to the Internet RFC allowed secure electronic mail across corporate boundaries and provided a per-user key distribution mechanism. Interoperability testing with independent PEM development groups was started to guarantee correct PEM operation across network boundaries.

User comments from the initial deployment of PEM within the VMS community indicated that for PEM to gain widespread use and acceptance, a simplified user interface and full integration with the existing mail utility were required. Design of the third prototype to meet these requirements began in late 1989. Several versions of a prototype based on extending a VMS text editor to support mail, encryption, and symmetric and asymmetric key management, were implemented during 1991. User acceptance of PEM started to grow with this prototype.

RFC-1040 was made obsolete when it was replaced by a series of successor versions distributed as Internet-Drafts and by electronic mail. These versions contained, at different points in time, incompatible changes to: the certificate data elements, the PEM header format, and the sender/recipient identification convention. A fourth prototype was released in January, 1992 to include these changes.

FIRST PROTOTYPE

During 1988, a series of interviews were conducted to investigate the current level of security in applications and the need for security features in future development. The most important and often the only application mentioned during these interviews was mail; in particular, the people interviewed desired the ability to send and store mail messages such that the messages could not be read by others. As a result, the first prototype of a confidential mail system for users of the VMS Mail Utility was created over the next few months. This prototype was a stand-alone image integrating the VMS Mail Utility and VAX Encryption. Only symmetric encryption of the message using the DES algorithm was supported. Cryptographic key management was completely manual; the user had to create a DES encryption key for each message as well as communicate the key to the message recipients using an out-of-band communication channel, e.g., the telephone.

To send a confidential message, a user would have to perform the following steps:

1. Compose the message and store it as a file
2. Invoke the prototype from DCL (Digital Command Line, the VMS command line interface)
3. Provide the mail address of the recipient
4. Provide the encryption key for the message.

To read the message, the recipient would have to perform the following steps:

1. Extract the message from the VMS Mail Utility into a file
2. Contact the sender to obtain the encryption key using an out-of-band communication channel, i.e., not e-mail or the subject line of the message
3. Invoke the prototype from DCL
4. Provide the encryption key
5. Read the message from the resulting file.

This cumbersome procedure was, as might be expected, unacceptable to the potential user community. Three serious problems were identified.

The first problem was using the binary output of the encryption process as the message; it contained random bitstrings that were interpreted as characters such as Tab, Form-Feed, and escape sequences by the mail system. This could make reading the initial encrypted message an unpleasant experience for the recipient.

The second problem was that the typical user had to exit mail to use the stand-alone prototype to send or read a confidential message. A pseudo-editor program was created to act as an encryption filter to encrypt or decrypt messages whenever the VMS Mail Utility invoked an editor to read or send a message. Since the system manager was required to "install" this pseudo-editor program as a system-wide trusted image before it could be used, and some system managers would not install this image (for legitimate security reasons), the pseudo-editor was often not available and the need to frequently exit mail remained.

The third problem was the lack of an automated key management scheme. The sender was required to create an encryption key when the message was created, and the recipient had to remember the key each time the message was read.

SECOND PROTOTYPE

Having proven the ability to integrate encryption services with the VMS mail system to provide a form of secure mail, an advanced development project was started to develop a Privacy Enhanced Mail (PEM) prototype according to Internet RFC-1040. RFC-1040 solved two of the three major problems in the first prototype. The problem with sending the binary output of the encryption operation was solved by the specification of a encoding scheme that converts the binary output into a universally representable set of characters, e.g., a 64-character subset of ASCII. This encoding also allowed the user to send messages containing 8-bit characters through gateways that would modify the eighth bit. In addition, an automated key management scheme specified per-message encryption key generation and a key distribution mechanism. While RFC-1040 specified symmetric and asymmetric key management, only symmetric key management was supported in this prototype.

The same approach was implemented for this prototype as was used in the first prototype, i.e., DCL image invocation and a pseudo-editor encryption-filter program. This approach would provide the easiest, and therefore fastest, way to obtain an implementation that could be used to test interoperability with other PEM implementations. A couple of months later, during August 1989, PEM messages were successfully exchanged with another RFC-1040 implementation developed outside of the company.

The pseudo-editor in this prototype, when available, was not well received by the users. The VMS Mail Utility does not pass the message header information, i.e., recipient's e-mail address, when it passes the message body to the called editor. As a result, the user was required to provide the recipient's address twice: once to the mail system, and then

again to the pseudo-editor to compose the PEM recipient identification header. Providing the information twice was prone to error and users were unhappy having to provide address information a second time.

During the testing of the message integrity check (MIC) and encoding processing for this prototype, the developer created a "MIC-CLEAR" message type. This message type separated the message authentication and integrity services from the message body encryption and encoding services, allowing the sender to send messages which the recipient could read, even if there were a problem with PEM compatibility. This message type also allowed the sender to send the same message to a list of recipients where some people could process the PEM header information and encoding and some could not. For the PEM developer, this message type allowed easier debugging of the MIC processing without interaction with the message encryption services. This message type was adopted by the Internet working committee and added to the subsequent RFC-1113.

While user acceptance was marginally better than the original prototype, the lack of integration with the mail system required the use of two separate applications to send and read messages. Few potential users had sufficient need to justify learning and using another application.

PROTOTYPE THREE

By late 1989, the development team decided that integration with the user's mail system was critical to the acceptance and success of PEM within the VMS community. Two options were considered.

The first option was to obtain a copy of the VMS Mail Utility code and modify it to support PEM. This option was rejected because the code was under active development and changed with each release of VMS. As there was no commitment to include any work done by the PEM project in the released product, each new VMS release would have required that the PEM project obtain a new copy of the code and duplicate the changes made to previous releases. Ultimately, using this approach could mean the PEM prototype would cease to function with the first release of VMS after the conclusion of the PEM project.

The second option was to write a new User Interface (UI) based on the callable interface to the VMS Mail Utility. This option would require a significant amount of work outside of the area of PEM support, but would allow development to continue across releases of VMS and new versions of the VMS Mail Utility. This approach would also allow the maximum flexibility to test alternative user interfaces to the PEM functions. At this time, a new version of DEC TPU (a text editor provided with VMS) provided increased support for integration with other products, such as mail and encryption. While basing the UI on DEC TPU would require tracking DEC TPU versions, no modification of DEC TPU code was required.

While writing a new user interface was the best approach from the developer's point of view, users would need a reason to overcome the natural resistance to change. The usual engineer's approach was used: provide the user with desired features not otherwise available, while maintaining compatibility with the old way. Visual compatibility with improvements was achieved by creating an interface similar to one with which the users were already familiar, i.e., VAX Notes, while providing users with the often-requested ability to scroll the message body. The most important compatibility requirement was determined to be "finger compatibility," i.e., the users did not have to teach their fingers to type new commands or press new keys. As a result, it was decided to use the same command line interface and syntax used by the VMS Mail Utility.

To specify PEM processing of the message to be sent, the SEND, REPLY and FORWARD commands were extended to support the /ENCRYPT qualifier. In response to user requests, a /SIGN qualifier was added to specify the "MIC-CLEAR" message type. Initially, the message recipient used a DECRYPT command to process a PEM message. However, based on user feedback, this was changed and a PEM message was automatically decrypted whenever the message was read.

The "write a new user interface" approach proved workable to the developers, and acceptable to the users. The prototype provided full integration of the mail system and PEM services, and continues in use today. The new UI has proven to be sufficiently popular that people use the prototype without using or knowing about the PEM functions.

Who are you?

This was the first prototype that raised the awareness of names as an issue. A VMS Mail Utility address has the form <NODE> "::" <USERNAME>. The <NODE> is optional when the recipient is located on the system or VMS cluster. This gives a typical VMS user 3 valid mail addresses. For example, a user SMITH on the system A in the VMS Cluster Z will receive mail addressed to SMITH, A::SMITH, and Z::SMITH. In addition, the RFC-1040 format rules required the use of the Internet mail address format, e.g., <USERNAME> "@" <NODE> "." <ORGNAME> "." <ORGTYPE>.

To confuse matters further, forwarding mail to another system is common within the VMS community. There were many instances of people calling for assistance in decrypting mail sent to A::J_SMITH, which was then automatically forwarded without the sender's knowledge to Z::SMITH. The lesson learned was that people are very

mobile and frequently change systems, physical locations, and organizations. When these changes occur, users take their old mail to their new location and expect to be able to continue reading old confidential messages.

To solve this name problem for the recipient, the prototype would construct a table containing the three valid mail addresses discussed above. While this table could be extended to include user-defined addresses to accommodate forwarding, this mechanism was not available through the user interface. The recipient identifiers in the incoming messages were compared to the addresses in the table. Should a match not be found, e.g., in the case of unanticipated forwarded mail, the user could manually edit the PEM header in the message to correct the recipient identifier. A change in the PEM specification of the recipient identifier eliminated the need to check for multiple e-mail addresses, but the same mechanism is needed to check for recipients having multiple certificates. Upon verification of the MIC, the message originator's distinguished name was displayed. This display annoyed some users and a mechanism was created to suppress the display.

For out-going messages, the recipient's address was entered in the usual VMS Mail Utility format. A translation table was kept to map mail addresses to the correct certificate.

Asymmetric key management support

During 1990, a set of procedures for a public-key cryptographic algorithm and public-key certificates became available, and support for the asymmetric key management form of PEM was added. This support required the creation of a mechanism to create public key pairs, as well as the mechanisms and procedures to create and distribute certificates for the public key.

To ensure that the user was the only person with access to the user's private key, the design decision was made that the user's private key would never leave the user's system. The correctness of this decision was never challenged and remains a fundamental belief. However, this challenged the developers to make the asymmetric key generation and associated certification process as simple as possible. The following 5-step, 5-minute process was developed to allow any user of the prototype to successfully create a public key and obtain a certificate for that key.

1. The user generates a case-insensitive password with a minimum of 8 characters to limit access to the private key.

2. The user runs a DCL command file to create a public and private key pair. A checksum is computed and displayed to the user. The public key is automatically mailed to the certificate authority.

3. The user contacts the certificate authority to verify the request for a certificate. The user also provides their distinguished name, the checksum computed in the previous step, and their e-mail addresses for the translation table.

4. The certificate authority returns the public key certificate by mail and the user saves it in a local file.

5. The certificate authority includes the public key certificate into the certificate database and the local system administrator updates the local certificate database.

For purposes of this prototype, a centrally generated certificate database was copied to the individual systems. The mail address translation table mentioned above was part of the certificate database and was updated by the certificate authority on request by the user.

YET ANOTHER PROTOTYPE

During 1991, a new version of the prototype was developed. This version included the major changes to the PEM specifications in the areas of certificate data elements, the PEM header formats, and the sender /recipient identification convention. These changes made this prototype incompatible with the earlier prototype and any messages created with the earlier prototype could not be read. The changes in the applicable PEM specifications, progressing (for the case of the base message processing procedures specification) from RFC-1040 to RFC-1113, with subsequent revisions made to the intervening drafts, were a source of persistent problems to the development effort. Interoperability testing was also impacted, as the independent groups would sometimes be using different versions of the specifications in their implementations.

The last enhancement made to the third prototype was to add the ability to save confidential messages in cleartext. This enhancement, combined with early and frequent notification to the users of the upcoming incompatible change, allowed the conversion to the new prototype to occur without losing previous messages. However, approximately half of the user community did not create new asymmetric key pairs. This was attributed, by the users, to their lack of need of either confidentiality or integrity controls for their mail.

DEPLOYMENT

Deployment and use of PEM internally was difficult, even within the organization funding the project. While the prototype three version of key management and asymmetric

key pair generation was not as easy to use as had been hoped, the major stumbling block was a lack of perceived need. There were many prospects interested in the technology who were given demonstrations and a few used the system to send test messages, but they did not install or use PEM. The following section, LESSONS LEARNED, provides more details on why PEM was not used by these groups.

The first tests by an organization to use PEM to solve a business problem were started during late 1990. The organization was responsible for sending out a sensitive monthly report to a large group of managers. To ensure the confidentiality of the paper report, each copy was double wrapped before mailing and the recipients would return receipts to the sender. The decision to use the prototype was based of several factors. The two most important factors were: 1) distribution of the report was a one-to-many operation, and 2) the cost of the existing paper operation was known. The centralized one-to-many distribution operation allowed for easy slow incremental growth, i.e., users could be added one at a time. Knowing the actual cost of the paper operation made the cost savings of replacing the paper with PEM visible, and a cost-benefit analysis could be made. Several other groups had investigated the use of PEM, but their applications required a large user base before significant savings could be achieved.

LESSONS LEARNED

The last four years have provided several lessons about the barriers to the adoption of Privacy Enhanced Mail.

Infrastructure

There is an initial cost to set up an asymmetric key management infrastructure. The first group to use PEM incurs the largest cost of setting up and operating a certificate authority. As the early adopters of a new technology, they also incur part of the development cost of creating an easy-to-use mechanism for users to create asymmetric key pairs and creating procedures for updating certificate databases. Few groups want, or can afford, to incur the costs and risks of going first.

Corporate culture

Two cultural attributes inhibited widespread adoption of PEM.

1. Large corporations have a paper bias. Any organization that has existed for more than 20 years has institutionalized a set of operational procedures based on distributing confidential information on paper. That paper-based process continues to work, and the risk of change is greater than the perceived advantages of speed and reduced costs.

2. There is a naivete about the threat of mail impersonation and eavesdropping of network lines. The corporate culture grew from a small, friendly network with 40 nodes, 800 people, and limited known access points to 80,000 systems supporting 100,000 people with innumerable points of access. People cannot comprehend the scale of current networks and therefore underestimate the threats involved.

Therefore, the paradigm shift from sending confidential information on paper to using confidential electronic mail did not occur, not because of active resistance to the technology, but rather because of a lack of "energy of activation." Widespread acceptance will probably not occur until there is a "Tylenol" incident.

User perceptions of security

For many people, the distinctions between confidentiality and integrity, or authentication and authorization, are unknown. Their perception of security in mail is limited to "secret mail" and preventing someone from reading their mail. This perception makes "selling" PEM longer and more difficult, as time must be spent explaining the integrity benefits of PEM to someone with no need for secret mail. Few people have the requirement to be able to prove to a third party, e.g., a judge in a court of law, that an electronic message was written by the person cited as the author.

Separating authentication from authorization is harder, some "experts" fail to make the distinction, and it was, unsurprisingly, difficult to communicate the distinction to the user community. Proving that someone is the author of a message does not demonstrate that the person was authorized to make the claims stated in the message. For example, while it could be proved that I sent you the message giving you a free PC from my employer; it is doubtful that you or my employer would believe I was authorized to give you a PC. PEM was designed to provide authentication. Administrators seeking an authorization mechanism sought tighter controls, e.g., frequent certificate expiration, than was acceptable to users desiring a pure authentication service.

Easy to use

From the user point of view, security cannot be made too easy. Security often lacks value in the mind of the general user community; it is something they are forced to deal with and to work around to get their job done.

Standards

Tracking standards that are under active development is hard. Incompatible changes can lead to conversions that are opportunities for abandonment by the existing user

community. Such changes also increase the difficulty of interoperability testing, as each development group is often at different points in the conversion. However, interoperability testing is critical for a prototype developed to an Internet standard. Some problems will only be found by having multiple implementations. For example, shortly after testing with external sites began, byte-ordering incompatibilities were discovered that required modification of the specifications to remove the ambiguity.

Encryption export controls

Encryption products are export controlled by the United States government, and these controls work for honest people. However, these controls slow the growth of PEM for personal use by increasing the difficulty of installing encryption on systems throughout the network. In the US, system managers were often slow to install encryption due to the different distribution procedures used for encryption products. Expansion of the user base into Europe has been almost impossible due to the fear of violating export controls.

Several users, including several managers, liked being able to send confidential mail. Unfortunately, there was nothing in their work that required e-mail messages to be kept "secret". The use of PEM for personal e-mail was restricted due to the limited number of people with the technology to read the message. This "chicken and egg" difficulty was usually traced to problems with getting encryption installed on the system.

UNFULFILLED EXPECTATIONS

Throughout the development project, prospective users of PEM have been found to hold expectations of the technology that went, and will continue to go, unfulfilled. The most common is the expectation of protection against a message recipient reading the message when the message was incorrectly addressed prior to submission for PEM processing. For example, if a message intended for John Smith (smith@z) is incorrectly addressed to j_smith@z, then the message with be processed to allow only j_smith@z to read the message. When Joan Smith (j_smith@z) receives the message, she will be, and should be, able to read the mail message sent to her. One obvious solution to this problem, entry of separate PEM and e-mail addresses, was used in an early prototype and proved to be unacceptable to the user community.

CONCLUSION

Development of four prototypes has established that the technology for implementing privacy enhanced mail to provide users with confidentiality and integrity controls for mail messages is available. However, significant barriers remain before the technology gains widespread acceptance and use in traditional corporate environments.

ACKNOWLEDGMENTS

The author wishes to acknowledge the contributions of John Linn, Mary Ellen Zurko, Chris Walsh, August Reinig, and Walt Soltysik.

The author wishes to acknowledge the contribution of Richard Pitkin. In addition to developing the asymmetric encryption routines and certification authority program, his outstanding commitment to the success of the project was essential to the development of the prototypes.

DISTRIBUTED PUBLIC KEY CERTIFICATE MANAGEMENT

Charles W. Gardiner

Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

ABSTRACT

Distributed environments (such as network systems) require a high level of assurance in identifying and authenticating other entities in the system, either for access control or for general authentication services. Public key cryptography offers an effective mechanism for solving these problems as well as for supporting non-repudiation and confidentiality services. The use of public key cryptography for these purposes often relies upon certificates, as defined in CCITT X.509, which serve to bind users' public keys reliably to their names. Through a hierarchy of signed certificates, the users' certificates can be managed in a distributed fashion.

Privacy-Enhanced Mail, a secure electronic mail system with distributed certificate management, raises many of the same issues as are found in other distributed computer systems.

This paper discusses some of these issues and describes how the SafeKeyper(TM) Certificate Signing Unit helps to resolve them.

INTRODUCTION

A distributed certificate management infrastructure must meet a number of requirements, such as:

- Protection of the keying material of certifying authorities.
- Provision for invalidating certificates whenever needed.
- Management of the "name space" in the certificate hierarchy.
- Audit and authorization management.
- Reliable recovery procedures.

The SafeKeyper Certificate Signing Unit (CSU) has been developed as an important component of a certificate management infrastructure designed for Privacy Enhanced Electronic Mail (PEM). Work on PEM was initiated by the Privacy and Security Research Group and has been continued by the Privacy-Enhanced Mail Working Group of the Internet Engineering Task Force. The PEM system adds confidentiality and authentication services to the SMTP mail system by the use of the public key cryptosystem developed by RSA Data Security, Inc. [1] and of certificates that conform to the X.509 standard. PEM processing of mail messages relies on certificates to establish the identity of message originators and recipients. Validating these identities involves forming a certification path made up of certificates from the message itself or already in a local database. The certification path goes from the sender up to a certifying authority that can also reached by the receiver's certification path, what might be called a "common ancestor". PEM defines a "tree" hierarchy with a single root, the Internet Society. This single-rooted certification hierarchy is a critical element of a distributed certification environment to guarantee that any message recipient can find a valid certification path back to the originator.

The SafeKeyper CSU is a peripheral which can be connected to the RS-232 port of a workstation. The workstation interacts with the peripheral in a sequence of atomic transactions comprising a request and a response. For example, the workstation may send a certificate in a request for signature. The peripheral generates an appropriate signature and returns the entire signed certificate in the response.

KEY PROTECTION

The protection of keying material has three dimensions: protection from disclosure, protection from unauthorized use, and protection from loss. All three aspects are desirable for all users of a public key system, but they are particularly important for the entities which issue certificates. The certificates they sign are only as trustworthy as the protection of the signer's key. Furthermore, the "higher" an issuer is in the hierarchy, the more critical its key, because all certificates of "lower" authorities and the certificates they have issued are invalidated if the higher key is compromised.

Protection of keys from disclosure

The SafeKeyper CSU protects the private keys of issuing authorities by keeping them entirely within the unit at all times. (The terms "private key" and "public key" are used in this paper to refer to the private and public components of an RSA key pair in an asymmetric cryptosystem. The term "DES key" or "secret key" refer to a DES key in a symmetric cryptosystem.) The unit has a rugged, tamper-resistant, cast aluminum enclosure approximately 10 x 7 x 3 inches. It has been designed to meet various international EMI standards as well as NACSIM 5100A. Inside there is a Motorola 68340 processor, 2 kilobytes of battery-backed RAM, half a megabyte of ordinary RAM, 32 kilobytes of EEPROM, 128 kilobytes of PROM and a hardware random number

generator. There are also two slots to accommodate removable storage devices, one for a Datakey DK1000, the other for a Datakey PK64KB. The battery-backed RAM is the only place where cryptographically sensitive information is ever stored. Such data never leaves the unit. This memory is immediately erased if the case is opened or tampered with in any of several ways. This memory contains a DES key for the unit as well as the private key of the currently active issuing authority. The private keys of all issuing authorities (of which there may be more than one in a unit) are kept in EEPROM, each encrypted with the authority's DES key.

Protection of keys against misuse

Each authority's DES key is encrypted with the CSU's DES key and stored in the Datakey DK1000, which is called the Cryptographic Ignition Key (CIK). The CIK provides control of an authority's signing capabilities. The CIK must be inserted in the unit and the unit must be given a request to activate that authority. Only then will the contents of the CIK be decrypted and used to decrypt the authority's private key. If the CIK is removed at any time, the hardware notes the event so that the authority is de-activated on completion of the current transaction, even if the CIK is re-inserted.

For additional protection against misuse, an authority's encrypted secret key can be "split" among several CIKs using RSA's secret-sharing algorithm [2]. By this technique an authority can be created so that 2 out of 3, or 3 out 5, or any other combination of CIKs must be inserted in any order to activate that authority. (The CSU sends back an interim response and blinks one of its LEDs when it is ready to receive another CIK in activating an authority. When the CIK is removed under these circumstances, the next one must be inserted within a pre-defined time, usually 30 seconds.)

In some circumstances it may be desirable for two or more authorities on a CSU to share the same secret key and, therefore, the same CIKs. The SafeKeyper CSU can be configured to allow a new authority to use the secret key of the currently active authority if the request to create the new authority so specifies.

The physical control of CIKs is essential to the proper control of access to an authority's capabilities. If this control were lost and the contents of a CIK (or a sufficient number of them) were compromised, the private key of the authority would still be protected from disclosure but not from misuse. Rather than requiring the authority to get a new key pair (thus invalidating all the certificates it had signed), the SafeKeyper CSU has a transaction to change an authority's DES key. The authority must first be activated with the old CIK(s), and then protected with a new DES key (made with the CSU's random number generator) to be encrypted as usual and written to new CIKs. The authority's private key is encrypted with the new DES key and written to EEPROM.

By way of protection against accidental overwriting of CIKs, each CIK is checked to make sure it is empty before writing takes place. The contents of each CIK are verified after writing. There is a special transaction to clear a CIK.

Protection of keys from loss

The third type of protection, that against loss, must be provided in a fashion consistent with the other two types of protection. It is particularly important for the keying material of certificate issuers. If an authority's private key should become corrupted in EEPROM or if the unit should fail for any reason, there must be a graceful way for the authority to recover and continue in business with the same key pair. The design of the SafeKeyper anticipates such unexpected events by providing that an authority's data, including its encrypted private key, may either be returned to the workstation or written to the other removable storage device, called the fill device. The requests to create or activate an authority have options to ask that the EEPROM data be returned or written to the fill device or both. The contents of a fill device are verified by reading after writing, but the device is not checked for emptiness before writing.

The information thus obtained from EEPROM in whichever form cannot be used to create a new version of the authority on the same SafeKeyper CSU because duplicate authority names are rejected, but it can be used to install the authority on another CSU (or restore the authority on a repaired or replacement CSU) which has the same DES key. The procedure for creating such a unit without compromise is described below under Factory Procedures.

ISSUING CERTIFICATES

The primary operation of the SafeKeyper is signing, or issuing, certificates on behalf of an authority. Trusted certificates require that the use of an authority's private key be controlled as described above, and that certain parts of users' certificates be protected from inadvertent corruption and be checked for format.

In PEM the operator of the workstation receives applications for certificates, usually in the form of draft certificates sent by ordinary electronic mail. These drafts contain the issuer's name, a pair of dates defining the start and end of the validity interval, and the applicant's name and public key. An MD2 hash is appended for an integrity check. The applicant also sends the hash by some out-of-band means. The operator verifies the hash, makes whatever other checks are appropriate to validate the application, e.g. is the applicant a current employee or student, and then sends a request to the SafeKeyper CSU. That unit checks the hash and format of the application, makes sure the issuer name matches that of the currently active authority, and checks that the authority is authorized to issue more certificates. The unit then signs the certificate and returns it to the

workstation. The workstation software usually mails the certificate back to the applicant and also stores it in a local database.

Each certificate contains a serial number which is unique for the issuing authority. The primary function of these serial numbers is to provide a compact identifier for use in lists of revoked certificates. The SafeKeyper keeps a count of the numbers issued to each authority and assigns this as part of the serial number when each certificate is signed.

REVOKING CERTIFICATES

Any certificate-based security system must have a way to revoke or otherwise de-activate certificates. X.509-style certificates include starting and ending dates, but many things can occur between these times to invalidate a certificate. The subject's private key may be compromised, or his/her affiliation with the certifying authority may change or terminate, to cite two possibilities.

PEM handles these matters with Certificate Revocation Lists (CRLs), which are ASN.1-encoded lists of revoked serial numbers and the revocation dates, signed by the authority that originally certified them. PEM CRLs differ somewhat from X.509 CRLs in that a PEM CRL includes not only its own date of issue, but also the date by which the next list will be issued. A user can thus always tell if a CRL is up to date. X.509 CRLs also have a signature for for each item in the list, but a PEM CRL has only one signature over all. A revoked certificate remains on successive CRLs until the certificate's expiration date. CRLs are used by PEM applications as part of validating certification paths.

The role of the SafeKeyper CSU is to check the format of the CRL and the issuer's name, and then to sign the CRL for the currently active authority.

MANAGEMENT OF NEW AUTHORITIES

Whenever a new certifying authority is created, it must have a unique distinguished name to avoid confusion with other issuers. In PEM this is solved in two parts: For the top three levels of the certification tree a database is kept containing the names of all certifying authorities. Requests for new authorities in the second and third levels are checked against this list, and conflicts are resolved by mutual agreement before an authority is created. In levels below the third, which are expected to contain many more names, such centralized control might become impractical. Duplicates are prevented by requiring that authorities at the third level and below can only sign certificates for subjects (whether users or other authorities) that have names subordinate to the signer's name. Subordinate means that the subject's distinguished name begins with the entire issuer's name, followed by some distinguishing parts, e.g. the name "Country=US, Organization=Ajax Corporation, Organizational Unit=Widget Division" is subordinate to the name "Country=US, Organization=Ajax Corporation".

The SafeKeyper CSU can be configured to require that each subject name be subordinate to the issuer name. In addition, units can be configured to require that a new authority be sent a special signed message before it can actually sign any certificates. This is described more fully in the next section.

Some certifying authorities may need to have more than one instance of the authority in separate CSUs, either for reasons of geography or work load. The SafeKeyper CSU allows the transfer of an authority and its private key to another unit by the use of the fill device, as mentioned above. The new instance requires the same CIK (or CIKs) at installation time, but the CIK(s) may be subsequently changed, if needed. Since the issuer names are the same, the rules for certificate revocation require that the various instances not use the same serial numbers. In the SafeKeyper CSU this is guaranteed by combining the CSU's unique ID number with the authority's count to form a certificate's serial number. The least significant three bytes are the authority's count and the more significant bytes are the unit's ID.

Only one CRL must be issued by an authority. If an authority exists in more than one CSU, only one should generate CRLs. This is not enforced by the CSU.

Whenever a new authority is created for the first time, its name and public key are returned in the response. For added protection of integrity, these items are in a message signed by the unit itself. When an authority is transferred from another unit, a similar signed message is returned but without the public key. These messages can be used to notify appropriate entities of the creation of the authority, in case authorization is needed.

AUTHORIZATION, AUDIT AND CONFIGURATION MANAGEMENT

The certificate management infrastructure must be able to manage the relationships between certification authorities and to track the certificate issuing activity. In any new system with which there has not been any prior field experience such a management system should probably offer as much flexibility as possible to modify features as the system is used. At the same time such management must be protected against unauthorized use, lest the key elements in the system be corrupted.

For the SafeKeyper CSUs these management functions are performed by the use of "control messages", which are special ASN.1-encoded pieces of electronic mail signed by a special type of authority called a "control authority". Control authorities can only sign control messages, not certificates or CRLs. Conversely other authorities can not sign control messages. The control authority to which a unit

belongs is assigned at the factory as described below under Factory Procedures.

Each control message is addressed to an individual SafeKeyper CSU by its unique distinguished name. (Each CSU has a distinguished name assigned at the factory, as described below under Factory Procedures.) The message may refer to the unit as a whole or to a specific authority therein. A message can set or clear configuration flags or can set the value of any configured parameter described below.

Features that can be configured for a unit include: minimum and maximum modulus size for public/private key pairs; maximum validity period for certificates; ability to have only one authority, or extra authorities that are subordinate to the first, or any authority; ability to have a control authority; ability to sign certificates for non-subordinate subjects.

Configurable items for an individual authority include: the range of authorized serial numbers (within the unit's ID); the next serial number; ability to sign certificates for non-subordinate subjects (overriding the unit's configuration). The ability of a new authority to sign certificates can thus be managed by the control authority. There is also a control message to cancel an authority. (Initial plans called for a transaction by which the active authority could cancel itself, but it was later realized that a likely reason for canceling an authority might be the loss of its CIK(s). In that case there would be no way to activate it for cancelation! Control messages provide a secure means to do this.)

Each control message contains a timestamp which is checked by the SafeKeyper CSU to ensure that messages are submitted in order and that no message is submitted twice.

The response to a control message is always a report on the current configuration of the unit or the authority, depending on the type of control message. The response is signed by the unit with its own private key. In the United States, where the RSA cryptographic techniques are patented, this mechanism can be used to determine how many certificates have been signed by an authority and to charge an appropriate royalty.

Control messages thus provide a flexible, secure method of managing SafeKeyper CSUs and their certifying authorities in either a centralized or distributed fashion, whichever turns out in actual use to be more appropriate.

MAINTENANCE PHILOSOPHY

Since the SafeKeyper CSU is tamper-resistant, the basic maintenance philosophy is complete unit replacement. In such a circumstance it is clearly desirable to be able to re-instate in the replacement unit all the authorities that were on the failed unit, still using the same private keys. The technique for encrypting private keys demands that the replacement unit have the same DES key. The means for accomplishing this without revealing these secret keys are explained in the Factory Procedures section below. When an authority is thus installed in a new CSU, it requires new authorization via control messages. Control messages for the old CSU are of no use with new one because the new one has a different distinguished name.

FACTORY PROCEDURES

All SafeKeyper CSUs are configured to customer requirements and assigned to a control authority at the factory. Each unit initially creates a DES key for itself using its random number generator. Some units, however, must be given the same DES key as another unit. For security reasons such keys should not be revealed to anyone. The keys must, therefore, be both unknown and reproducible. Public key cryptography makes this possible.

At the factory there is a specially configured SafeKeyper CSU containing two authorities. The first is the authority which simply assigns IDs to all units in much the same way that a certificate-signing authority assigns sequential serial numbers; the second is a combined control and certficate-signing authority -- the only such authority allowed. The PROMs of all SafeKeyper CSUs contain the public key of the second authority.

As soon as a SafeKeyper CSU is assembled, it is connected to a workstation to which the special manufacturing unit is also connected. The operator gives the workstation the physical serial number stamped on the new unit. The workstation obtains the next ID number from the first authority in the manufacturing CSU and sends both items to the new unit in an initialization request. The new unit records its serial number and ID in EEPROM, generates its random DES key and its key pair, encrypts its private key with the DES key into EEPROM and prepares a response. The response contains the new unit's public key and also the new DES key encrypted with the public key of the second manufacturing authority. The workstation stores these in a database along with the serial number and the ID. The new unit can now be put in inventory with assurance that it cannot be tampered with. Once a unit has been thus initialized, it will reject any further attempt to initialize it. A unit that has not been initialized cannot perform any certificate-related transactions.

Some time later the new unit may be configured for a particular customer. It is again attached to the workstation, but now the second authority is activated. If the new unit is to have a different DES key, the workstation obtains both of the encrypted keys from the database and sends them in a special request to the manufacturing unit. The active authority there is able to decrypt both DES keys and to encrypt the desired DES key with the new unit's present key. The result is put in a control message and signed. When this is passed back to the workstation and over to the new unit,

that unit can decrypt the DES key and install it. At no time is any DES key transmitted outside of a SafeKeyper CSU in cleartext form.

In the third step the workstation prepares a control message to configure the new unit and to give it the public key of another control authority in accordance with requirements supplied by the operator. The manufacturing unit's control authority signs the message, and it is passed to the new unit. The signed response confirms the configuration, which is placed in the database. Once a unit has been thus assigned to a control authority, the unit's DES key can no longer be changed.

Finally, the workstation prepares a certificate for the new unit, containing the unit's serial number and ID as part of the subject name. The manufacturing control authority signs this, and it is sent to the control authority that now "owns" the new unit, thus providing that authority with the new unit's name to use in addressing control messages and the public key to use in confirming responses.

Whenever a unit is returned for repair, its PROM is replaced (in case the instructions there have been maliciously altered) and the entire initialization process is repeated. A single physical unit with a single physical serial number may thus have several different logical IDs in the course of its life, one for each time it is initialized.

This procedure allows two levels of protection: The first step, which takes up to five minutes to generate a key pair, uses the first authority. The later steps, which are more sensitive, require the second authority. (For convenience, the second authority can also execute the first step. It still gets the next ID from the first authority's counter in EEPROM. This is the only authority which is allowed to modify the counter of another authority.)

CONCLUSIONS

This paper has discussed some of the significant issues to be considered in creating a distributed certificate management system to provide identification, authentication, confidentiality and non-repudiation services. It has described the role of the SafeKeyper CSU in the Privacy-Enhanced Mail system and the techniques by which it deals with these issues.

ACKNOWLEDGEMENTS

I am grateful to my colleagues David Solo, Patrick Cain and John Lowry for their assistance in the SafeKeyper CSU project and in the preparation of this paper.

REFERENCES

1. Rivest, R. L., Shamir, A., and Adleman, L., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of the ACM* Vol. 21, Number 2, February 1978, pp. 120-126

2. Shamir, Adi, "How to Share a Secret", *Communications of the ACM* Vol. 22, Number 11, November 1979, pp 612-613.

PROTECTING THE INTEGRITY OF PRIVACY-ENHANCED ELECTRONIC MAIL WITH DES-BASED AUTHENTICATION CODES

Stuart G. Stubblebine [1] Virgil D. Gligor

Electrical Engineering Department
University of Maryland
College Park, Maryland

ABSTRACT

The Privacy-enhanced Electronic Mail supports integrity services with both symmetric and asymmetric keys. An option of the symmetric-key services is that of protecting message integrity with DES-based authentication codes. We discuss a vulnerability of this option to message-integrity attacks. We present a solution for the removal of this vulnerability that allows the retention of the DES-based authentication codes.

INTRODUCTION

The Privacy-enhanced Electronic Mail (PEM) [1,2] supports confidentiality, integrity, and authentication of electronic mail in the Internet. These services use end-to-end cryptography between sender and receiver User Agent processes, with both symmetric and asymmetric keys, and do not impose any special processing requirements on the underlying Mesage Transfer System. An option of the symmetric-key services is that of protecting message integrity with Message Authentication Codes (MACs) which are computed by using the Data Encryption Standard (DES) in Cipher Block Chaining (CBC) mode.

An early version of the DES-MAC option in PEM [3] was shown to be vulnerable to integrity attacks against multiple-receiver messages [4]. When these messages were used, a receiver could create a bogus message and have it accepted by other receivers, or by the sender, without detection. The current version of PEM eliminates this vulnerability (1) by using a different key for the DES-MAC computation from that used for the CBC encryption of the user data, and (2) by recommending that at most a single receiver be named as an addressee of DES-MAC messages [1,2]. However, another message-integrity vulnerability of the DES-MAC checksumed messages remains uncorrected. We present this vulnerability, and propose a solution for its removal that allows the retention of the DES-based authentication codes. (A discussion of whether such authentication codes should be retained by PEM is beyond the scope of this paper. We assume that, since the DES-MAC checksum is sufficiently strong for most applications, is an international standard [5,6], and is supported by commercially available hardware [7], its use is desirable.)

REPRESENTATION OF PEM MESSAGES

PEM services support both single- and multiple-receiver messages. The representation of a single-receiver, DES-MAC checksumed message, denoted by message type T1 in Figure 1, [2] consists of an Initialization Vector (IV) for data encryption and decryption, an address field for the sender and for the receiver, a key field, a MAC field, and a user data field. Both the key field and the MAC field are encrypted under an Interchange Key (IK), which is shared between the sender and receiver.

The key field contains a random Data Encryption Key (DEK), which is used for the CBC encryption of the user data with the initialization vector, IV. It is also used for computing the DES-MAC of the user data in the CBC mode with a zero initialization vector, IVo. When the DEK is used for computing the DES-MAC, the constant F0F0F0F0F0F0F0F0 is added (modulo two) to it. (This separation of the key used for DES-MAC computation from the key used to encrypt the user data is necessary to ensure that a bogus DES-MAC cannot be computed from message ciphertext.) A different DEK is used for each message. To enable detection of header modification, the user-data field must include the message header. (Since the header contains random fields, this requirement has the added benefit of guarding against chosen plaintext attacks.)

The representation of a two-receiver message, denoted by message type T2 in Figure 2, differs from messages of type T1 in that (1) separate pairs of DEK and MAC fields are included for each receiver, and are encrypted in the interchange key, IK, that is shared between the sender and each receiver (e.g., IK_{AB} and IK_{AC} in Figure 2); and (2) the MAC is the result of computing RSA-MDx over the user data, where RSA-MDx can be RSA-MD2, MD4 or MD5 [8,9]. If communication between pairs of principals includes both messages of type T1 and T2, the same IK is used.

---

[1] S. G. Stubblebine's current address is : USC Information Sciences Institute, 4676 Admirality Way, Marina del Rey, CA 90292-6695. (stubblebine@isi.edu, gligor@eng.umd.edu)

[2] The representation of both the single- and multiple-receiver messages omits details that are irrelevant to the subject of this note (viz., [1,2]).

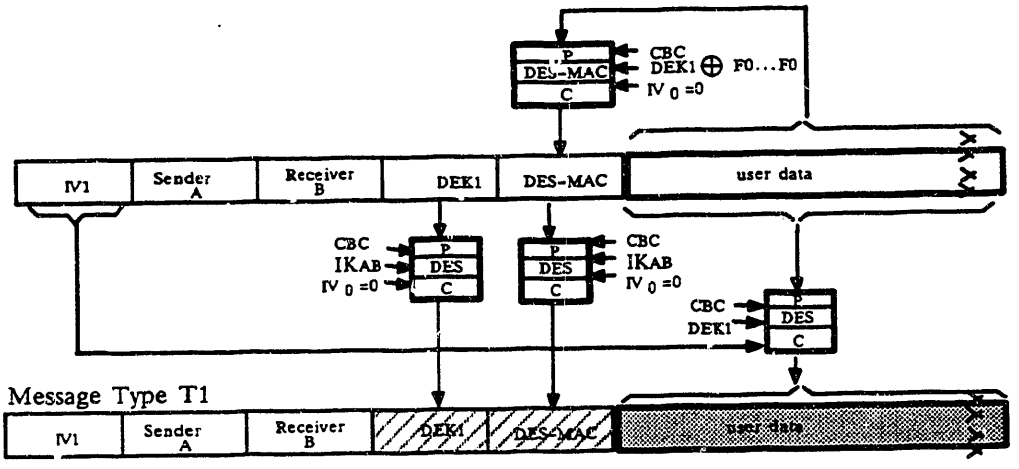


Figure 1. The Representation of PEM Messages using DES-MAC *

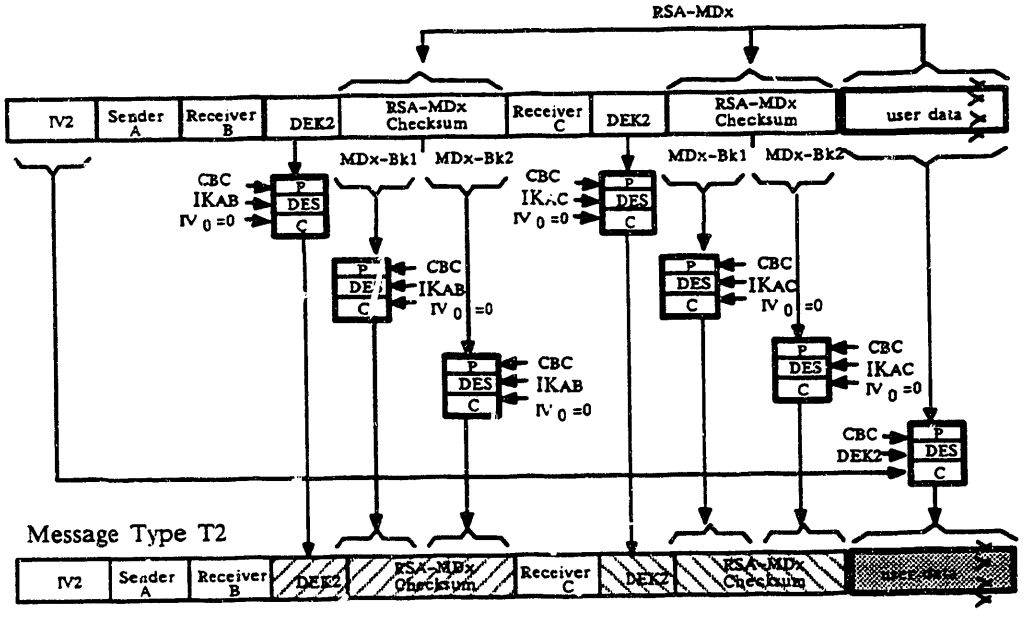


Figure 2. The Representation of PEM Messages using RSA-MDx*

Throughout this paper, the notation *ENC(key, IV; P1,...,Pn)* and *DEC(key, IV; C1,...,Cn)* represents the DES-CBC encryption of plaintext *P1,...,Pn* and decryption of ciphertext *C1,...,Cn* with the key *key* and initialization vector *IV*. The notation *DES-MAC(key, IV; P1,...,Pn)* represents the computation of the Message Authentication Code of *P1,...,Pn* with the key *key* and initialization vector *IV*, using the Data Encryption Standard (DES) in Cipher Block Chaining mode [5,6].

ATTACK SCENARIO

Suppose that principal P^A sends a type T2 message to principals P^B and P^C. Principal P^C intends to use this message to construct a bogus message of type T1 that would appear to be sent by principal P^A to P^B (or by principal P^B to principal P^A), as illustrated in Figure 3.

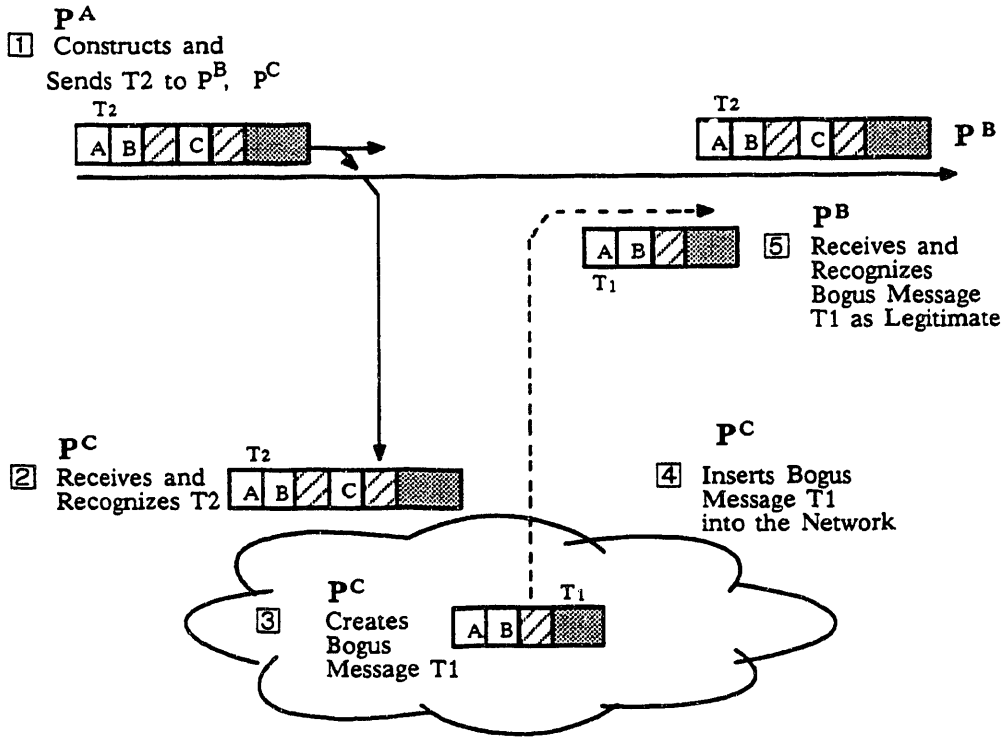


Figure 3. The Attack Scenario for the DES-MAC Option of PEM

The construction of the bogus message of type T1 is illustrated in Figure 4. To construct this message, principal P^C uses the encrypted key block, ENC(IK_{AB}, IVo; DEK_2), of the received type T2 message in the place of the encrypted key block, ENC(IK_{AB}, IVo; DEK_1), of a legitimate type T1 message sent by P^A to P^B; both blocks are encrypted under the interchange key shared by principals P^A and P^B, IK_{AB}, which remains unknown to P^C. Similarly, principal P^C uses the first block of the encrypted RSA-MDx checksum, ENC(IK_{AB}, IVo; MDx-Bk1), in the place of encrypted DES-MAC checksum, ENC(IK_{AB}, IVo; DES-MAC), of a legitimate type T1 message sent by P^A to P^B. Since principal P^C knows the plaintext block of the RSA-MDx checksum, P^C chooses the plaintext blocks $P1_1...P1_i$ for the bogus user data so that the result of the DES-MAC computation over these data equals the first plaintext block of the RSA-MDx checksum, MDx-Bk1; i.e., DES-MAC($DEK_2 \oplus$ F0...F0, IVo; $P1_1...P1_i$) = MDx-BK1

Principal P^C's choice of the first i-1 blocks, $P1_1 ... P1_{i-1}$, is unrestricted. However, to ensure that DES-MAC($DEK_2 \oplus$ F0...F0, IVo; $P1_1...P1_i$) = MDx-BK1, principal P^C must choose the last block $P1_i$ to equal $C2_{i-1} \oplus P1_i'$. To obtain $C2_{i-1}$, P^C encrypts the first i-1 blocks, $P1_1...P1_{i-1}$, of the bogus user data under the key $DEK_2 \oplus$ F0...F0 and IVo=0; and to obtain $P1_i'$, P^C decrypts MDx-Bk1 under key $DEK_2 \oplus$ F0...F0 and IVo=0. Principal P^C can compute $C2_{i-1}$ and $P1_i'$ because it knows both the key DEK_2 and the value of MDx-Bk1; both are decrypted by P^C from the type T2 message received from principal P^A under the interchange key IK_{AC} and IVo=0. However, as illustrated in Figure 4, the plaintext block $P1_i$ would appear garbled since it is defined as $C2_{i-1} \oplus P1_i'$. A receiver may or may not find this suspicious, depending upon the placement of that block in the message. By using similar choices of plaintext and ciphertext repeatedly, the placement of the garbled block within the bogus message becomes unrestricted.

The attack would be successful even if (1) a different block-cipher algorithm would be selected, not just DES, provided that the CBC mode would be used; (2) an initialization vector IVo $\neq$ 0 would be used; (3) any known plaintext-ciphertext block pair that is encrypted under the key shared by principals P^A and P^B, IK_{AB} would be used to construct the DES-MAC for the bogus type T1 message -- not just the

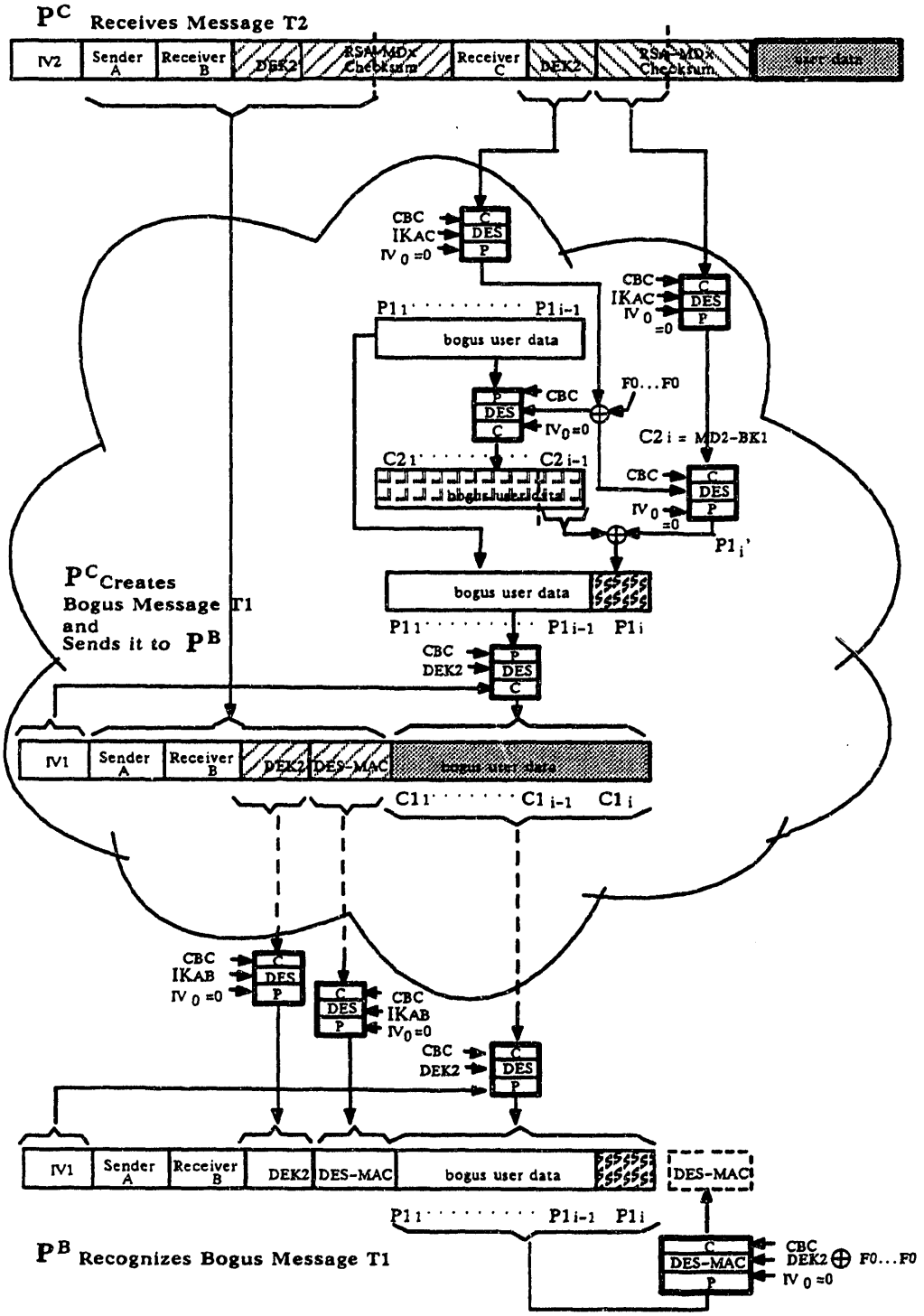


Figure 4. Creation and Recognition of a Bogus Message

pair <MDx-Bk1, ENC(IK_{AB}, IVo; MDx-Bk1)>; and (4) a separate key per receiver would be used to compute a different DES-MAC value per receiver.

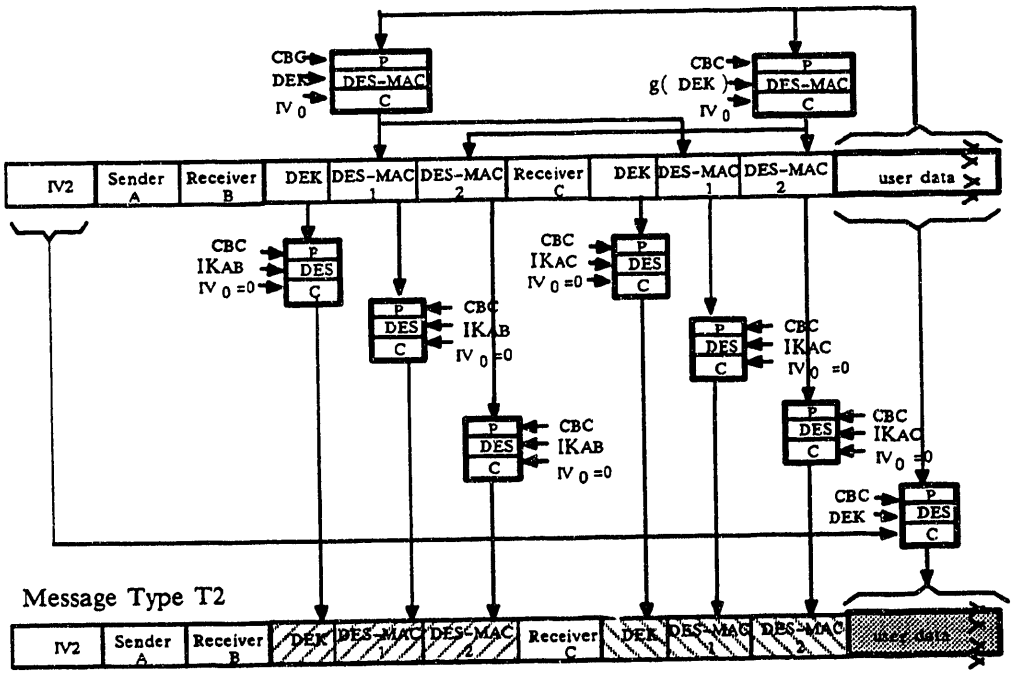


Figure 5. The Representation of Multi-receiver PEM Messages Using Double DES-MAC

REMOVING THE VULNERABILITY OF THE DES-MAC OPTION IN PEM

We suggest a two part solution for removing the PEM vulnerabilities posed by the use of DES-MAC checksums: the first part is proposed for single-receiver, DES-MAC checksumed messages, whereas the second part is proposed to allow use of the DES-MAC checksums for multiple-receiver messages.

In suggesting a solution, we make three desirable assumptions. First, we assume that the use of the same interchange key, IK, for both single- and multiple-receiver messages should be continued. This removes the task of acquiring an interchange key for each type of message. Second, we assume that a single data encryption key, DEK, is retained for each multiple-receiver checksumed message. This removes the task of re-encrypting a message for each message receiver. Third, we assume that the same checksum value (e.g., DES-MAC or RSA-MDx) is retained for each receiver of a multiple-receiver checksumed message. This eliminates the recomputation of a different DES-MAC for every receiver.

The three assumptions made above suggest that at least two known plaintext-ciphertext pairs would be available, namely <DEK, ENC(IK, IVo; DEK)> and <checksum, ENC(IK, IVo; checksum)> for every multi-receiver message. Thus, any solution must either eliminate these known pairs or must ensure that, despite the presence of known pairs such as those above, an attacker could not construct a bogus message that is recognizable within a probability threshold [10].

The proposed solution for single-receiver messages has the effect of eliminating known pairs. This solution requires that a variant of the interchange key, g(IK), be used to encrypt the DEK and DES-MAC value for type T1 messages. The function g(key) should be one-to-one, differ from the identity function, avoid weak or semi-weak keys, maintain key parity, change half of the key bits on the average, and neither weaken the cryptosystem nor unduly increase the probability of determining the secret key. The function g(key) = key ⊕ F0...F0 seems to be a reasonable choice for this purpose.

As illustrated in Figure 5, the proposed solution that scales well for multiple-receiver messages reduces the impact that the presence of known pairs under IK (from Message Type T2 from both that shown in Figure 5 and also from Figure 2) has on the probability that an attacker can construct a bogus message. This solution takes the approach that the DES-MAC function is applied to the user data twice, first with the key DEK and then with the key variant g(DEK), to obtain a *double DES-MAC*. The double DES-MAC and the DEK are then encrypted under the interchange key of each receiver in the same way the DEK and the checksum are encrypted in RSA-MDx messages.

This solution scales up well in the sense that its performance is independent of the number of message receivers. Also if IVo is set to be equal to the random IV_2, then the separate encryption of the first DES-MAC component (i.e., DES-MAC 1) is avoided since it is identical to the last encrypted block of the user data.

The alternate double DES-MAC solution, using the above basic format with the exception that DES-MAC 2 is obtained by computing DES-MAC(g(DEK), IVo; $C_1,...,C_n$) (i.e., the second DES-MAC is computed over the ciphertext of the user data), is also adequate. In contrast, the alternative where DES-MAC 1 = DES-MAC(DEK, IVo=0; $P_1,...,P_n$) and DES-MAC 2 = DES-MAC(DEK, IVo=0; $P_n,...,P_1$) (i.e., DES-MAC 2 is computed over the plaintext in the reverse direction), which is called the *bi-directional MAC* in an early version of PEM [11], is somewhat inadequate. This is the case because bogus messages consisting of plaintext blocks (i.e., user data) arranged in palindrome format would be recognized as legitimate by User Agents. Of course, users will probably find it suspicious that half of the message would be garbled.

CONCLUSION

We provide yet another example of the need for using systematic message-integrity analysis and design methods in two ways. Successful message-integrity attacks are still possible against protocol options that are only informally analyzed. We proposed a solution for the removal of a symmetric-key vulnerability of the DES-MAC option in PEM that allows the retention of the DES-based authentication codes for both single- and multi-receiver messages. Since the integrity protection provided by any message type is largely dependent upon other message types in the protocol [12], we caution that the security of these solutions must be re-evaluated should existing message types change or other message types be added.

ACKNOWLEDGEMENT

We thank John Linn and Dan Nessett for their comments on an earlier version of this paper. The work reported herein was supported by IBM Corporation under contracts YC313314 and MHVC2160. We are grateful to Tom Tamburo, Wen-Der Jiang, Marty Simmons, Curt Symes, Tom Russell, and Ping Lin for their continued support and encouragement.

REFERENCES

1. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures, Part II -- Certificate-Based Key Management, Part III -- Algorithms, Modes, and Identifiers," Internet Working Group, RFC 1113 - 1115, August, 1989.

2. M. Bishop, "Privacy-enhanced Electronic Mail," *Internetworking: Research and Experience*, Vol. 2, pp. 199-233, (1991).

3. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures," Internet Working Group, RFC-989, February, 1987.

4. C. Mitchell and M. Walker, "Solutions to the Multidestination Secure Electronic Mail Problem," *Computers & Security*, Vol. 7(5), pp. 483-488, 1988.

5. Federal Information Processing Standards Publication 113, Computer Data Authentication, May 1985 (also, see ISO DP 8730).

6. American National Standard X9.9-1986, American National Standard for Financial Institution Message Authentication (Wholesale), *American Bankers Association, Washington* (1986).

7. D. Abraham, G. Dolam, G. Double, J. Stevens,,"Transaction Security System," *IBM Systems Journal.*, Vol. 30, No. 2, pp. 206-229, 1991.

8. R. Rivest, "The MD4 Message Digest Algorithm," Technical Memorandum 434, Laboratory for Computer Science, M.I.T., October, 1990.

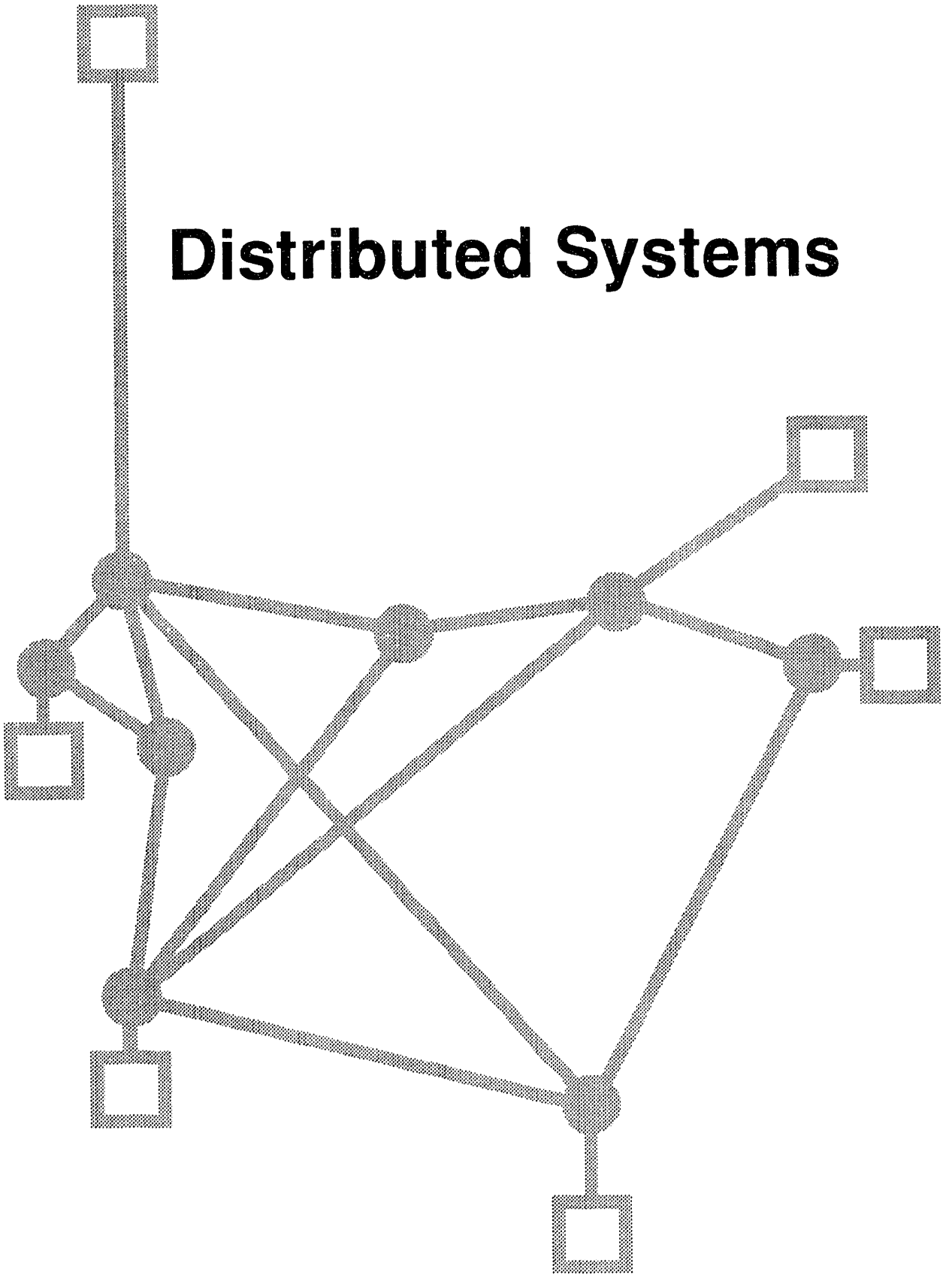
9. R. Rivest, "The MD5 Message Digest Algorithm," Internet Working Group, RFC 1321, April 1992.

10. S. G. Stubblebine and V. D. Gligor, "On Message Integrity in Cryptographic Protocols," IEEE Symp. on Research on Security and Privacy, Oakland, Calif., pp. 85 - 104, May 1992. (also technical report TR - 2843, University of Maryland, College Park, Maryland 20742, February 1992.)

11. J. Linn, "Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures," Internet Working Group, RFC-1040, January, 1988.

12. S. G. Stubblebine and V. D. Gligor, "Message Integrity Design," Draft.

Distributed Systems



Practical Authorization in Large Heterogeneous Distributed Systems

J.G. Fletcher and D.M. Nessett

Lawrence Livermore National Laboratory
Livermore, CA

ABSTRACT

Requirements for access control, especially authorization, in practical computing environments are listed and discussed. These are used as the basis for a critique of existing access control mechanisms, which are found to present difficulties. A new mechanism, free of many of these difficulties, is then described and critiqued.

INTRODUCTION

Over the past decade and a half, system researchers have thoroughly investigated distributed computing, analyzing its important issues and proposing various ways of treating them. However, the services they have developed sometimes poorly fit the problems arising in practical computing environments. We concentrate on how this is so for distributed access control.

Access control is implemented through two component services : 1) authentication and 2) authorization. The problem of authentication has received significant attention and we believe the mechanisms developed so far are adequate in most situations. Consequently, we concentrate here on distributed system authorization, a problem requiring more attention.

This paper is organized as follows. First, we analyze the characteristics of certain practical distributed computing environments and develop requirements for distributed system access control. We use these requirements to critique existing distributed system access control mechanisms, particularly those aspects related to authorization. We describe an authorization method that meets our criticisms, pointing out its strengths and weaknesses and providing a compromise containment analysis for it. We then describe a production application that uses our authorization scheme.

ACCESS CONTROL IN PRACTICAL COMPUTING ENVIRONMENTS

Researchers interested in distributed system security have extensively investigated the issue of access control. For the most part, they have concentrated on the problem of authentication, while on the whole limiting their investigations of authorization to the smaller sub field of distributed operating systems. With a few exceptions, architects of distributed systems other than distributed operating systems have relied on the existing non-distributed mechanisms of hosts to support authorization.

We believe that much of the previous work on distributed system authorization rests on assumptions that only rarely exist in practice. To support this claim, we analyze the characteristics of a typical distributed system supporting scientific and engineering applications and in section 3 discuss how existing distributed system access control techniques fail to operate correctly in the presence of these characteristics. While it would be appropriate to do so, we do not analyze systems that are used primarily for business applications, since we have little experience with them. However, our intuition suggests that many of the characteristics we describe are relevant for those systems as well.

The security environment of a distributed system supporting science and engineering

There are two classes of distributed application that use security services. The first class supports system level activity that is generally administered by system programmers and carried out to supply infrastructure services to distributed system customers. The second class involves computational activity initiated by non-privileged users, generally focused on solving some scientific, engineering or other customer related problem. These two classes of application possess contrasting security traits. Applications in the first class enjoy extraordinary security privileges, such as root access. Applications in the second class generally are not granted special security privileges.

Distributed applications supporting scientific or engineering work are initiated by customers rather than by system software or system programmers. Thus, they are an example of the second class of distributed application. They customarily grow from a central point and expand out into a distributed system. As with most distributed applications, their activity is organized around the client/server model. However, it is rare for the servers of these applications to exist prior to the initiation of an application run. Instead, servers are dynamically created when the application grows and are terminated when the application finishes. This pattern of behavior strongly influences which access control mechanisms are suitable for such applications. Generally, there must be an unprivileged server (see below) that permanently runs on hosts and that allows the creation of dynamic servers running in the context of a distributed

application user. It is the permanent server that makes the appropriate distributed system authorization decisions.

There is a very large investment in programs that analyze various scientific and engineering research problems. These programs use linear system solvers, implicit and explicit difference equation solvers and relaxation methods to solve partial differential and integral equations. It is far too expensive to rewrite this software for a particular distributed application. Instead, a distributed application must be able to incorporate this software without modification. Consequently, scientific and engineering distributed applications are not at liberty to change the way these programs do file I/O, terminal I/O or graphics I/O. While it is possible to write driver routines that call these programs and handle communications with other distributed application components, the underlying system service calls must not be disturbed.

Heterogeneity is an important characteristic of practical distributed systems [1, 2]. We are amazed at the number of designs that ignore this pivotal concern. Heterogeneity exists in the physical security environment of distributed system equipment, in the behavior of the organizations that administer this equipment, in the protocols used within a distributed system, in the level of vulnerability each host operating system experiences, and in the security mechanisms supported by hosts[1].

Previous work has dealt with security heterogeneity by organizing collections of similarly trusted hosts into pools known variously as Domains of Trust [3, 4], Authentication Domains [5], Inter-Organization Networks [6], Realms [7,8], and Administrative Domains [2]. Within these domains, security mechanisms may also display a certain amount of heterogeneity[2]. For example, a domain may support the Kerberos authentication mechanism [7, 8] on some hosts, while others may rely on the normal UNIX /etc/passwd file mechanism. Even within hosts, some applications may support Kerberos authentication (e.g., *rlogin*, *rcp*, *rexec*), while others may rely on /etc/passwd (e.g., Telnet, FTP).

Customer initiated distributed applications face considerable difficulties when run over resources located in multiple security domains. They do not have special privileges and therefore must use infrastructure security services provided by the domains. While there are authentication facilities available to accommodate multiple security domains [7, 8, 9], existing authorization mechanisms require either transmitting a user's password in the clear over a potentially hostile network, or the installation of software, such as a Kerberized or DASS-enhanced *rexec* daemon, that requires root privilege. Generally, system administrators are reluctant to install software provided by customers that require root access. Consequently, if systems on which the non-privileged distributed applications execute do not support the appropriate root privileged software, customers are forced to use dubious security practices, such as storing their passwords in files and passing them in the clear through vulnerable intermediate computing and switching equipment. A practical distributed system authorization method should eliminate these security hazards.

Most current distributed computing is what might best be described as network computing. Generally, hosts in the distributed system act as independent computing agents that retain a significant identity from an application's standpoint. While distributed operating systems may provide a more coherent and an ultimately superior performing base for distributed applications, so far, they have not been highly successful in the marketplace. Our own distributed operating system, LINCS [4, 10], failed not for technical reasons, but rather because we could not afford to support it as a unique LLNL specific product. Nothing is currently available from computer system vendors that provides its functionality.

Our experience with LINCS leads us to conclude that network computing will remain the predominant distributed computing model for some time to come. This means that distributed system support must be built on top of existing host operating systems, which today are largely some variation of UNIX™.

Given the ubiquity of UNIX, we are forced to consider its security properties. Most fielded UNIX operating system implementations contain significant security hazards. Moreover, there are few if any mainstream UNIX operating systems for state-of-the-art computing equipment evaluated according to the Trusted Computer System Evaluation Criteria [11][3]. We don't have much confidence in the idea that this situation is about to change. Consequently, we believe any distributed system security mechanism must operate in an environment in which the constituent hosts have intrinsic vulnerabilities. To be more precise, we believe that when a host compromise occurs, the security mechanisms should be architected to minimize the number of compromised resources and provide some kind of compromise containment support. Along these same lines, when system administrators discover a misbehaving user, they should be able to quickly and efficiently revoke that user's privileges to distributed system resources.

[1] Some may reject our thesis that a distributed system experiences heterogeneity in host security mechanisms, since we postulate the pervasive use of UNIX. However, variants of UNIX do not all support exactly the same security mechanism. For example, many versions of UNIX allow any user to obtain the contents of the etc/passwd file, while others hide its contents from public view.

[2] The work described in [2] argues against this practice. The definition of Administrative Domain given there insists that all constituent hosts use the same security mechanisms.

[3] Even if there were, we don't have a high regard for such evaluations, since they do not raise our confidence adequately to justify their cost. Furthermore, once evaluated systems are placed in the field, many of their handling constraints, such as the prohibition against customer operating system modifications, are impractical. We have other criticisms of the whole concept of evaluated systems, but this is a topic for another paper.

Requirements for distributed system access control mechanisms

We use the characteristics described above to develop requirements for distributed system access control. Specifically:

1) Access control facilities must not require existing scientific program modules and equations solvers to be modified. If these programs access stand-alone system resources, such as files, terminals, graphics equipment, etc., they must be able to do so in exactly the same way when they are integrated into a distributed application.

2) The support of customer initiated scientific distributed applications requires that the access control mechanisms operate without root privileges.

3) Distributed system access control must operate on systems running Unix.

4) Distributed system access control must operate in an environment of vulnerable hosts. When a host is compromised, the access control software must not allow the intruder to compromise the complete distributed system.

5) When system administrators discover a misbehaving user, the access control mechanisms must allow them quickly and efficiently to revoke his access to distributed system resources.

6) Access control facilities must not encourage users to engage in unsound practices such as storing unencrypted passwords in files or transmitting them in the clear over networks.

7) Access control must operate in a heterogeneous environment. It must work across multiple domains that may support different underlying access control methods.

A CRITIQUE OF EXISTING ACCESS CONTROL MECHANISMS

We investigate some popular distributed system access control mechanisms either in use or proposed to determine whether they meet our requirements. While our focus is authorization, some of our requirements are affected by the authentication service used for access control, so we briefly analyze several authentication schemes from this perspective. We concentrate on Kerberos [7, 8], DASS [9] and /etc/passwd based authentication.

Most distributed system access control schemes can incorporate any of the authentication mechanisms named above. However, /etc/passwd based authentication requires the transmission of a password in the clear from the client to the server, which violates requirement 6. Both Kerberos and DASS support authentication without transmitting cleartext passwords, so these authentication strategies are preferable for our applications.

Existing distributed system authorization mechanisms fall into one of two categories : 1) access control list based, or 2) capability based. Most distributed operating systems that have been developed so far have used capabilities. However, the majority of distributed system software used in practical computing environments uses access control lists, so in this critique we focus on that technology.

Access control list systems also fall into two categories : 1) those that hold the access list information in a file or database on each machine (per machine database authorization), or 2) those that hold all or part of this information on centralized servers (centralized database authorization). The most common approach to distributed authorization uses the authorization information maintained by host operating systems, which is a per machine database strategy.

Systems that use a centralized database for authorization data include Moira [8], the proxy-based ticket approach developed for Kerberos [12] and the DCE authorization mechanism [13]. The Moira approach, developed for Project Athena, keeps authorization information on a centralized server. This information is distributed to individual servers on a periodic basis. Servers use this data to make authorization decisions after a user has been authenticated by Kerberos.

The proxy-based ticket approach is based on the use of Kerberos tickets that are passed between principals. An authorization server, to which servers grant full access rights, creates restricted proxy tickets for principals according to authorization information it retains. Within the ticket may be information that restricts its use in some way. A principal proves its has obtained the ticket in a legitimate manner by carrying out a protocol with a server that uses the session key the ticket contains. This key is passed between principals when the ticket is passed.

Systems running DCE software from Open Systems Foundation authenticate the user using a Kerberos protocol exchange, the established identity being used for authorization decisions. DCE also supports a registry service that maintains the set of groups to which a principal belongs. This information is sealed in a Privilege Attribute Certificate and passed from client to server in support of authorization. Each DCE server is configured with DCE's access control list software that maintains full access lists for each resource. These lists contain entries that identify a user, a group and other information along with permission data for these identifiers. Since an access control list can contain multiple

entries, more fine grained control is supported than can be achieved with standard Unix permission bits. Moreover, a proposal to support access rights delegation is currently being studied as an enhancement to this scheme.

Layered authorization

Independent of where the access control information is stored, distributed system authorization services may be implemented in one of two ways. The first approach layers the distributed authorization mechanism over existing host authorization services. The second assumes all distributed system resources are managed and owned by servers, which multiplex their use among the server's clients.

Currently, most fielded authorization systems rely on the access list mechanisms supplied by host software. For example, a host authenticates a user through a service such as Kerberos, DASS, or by use of its own /etc/passwd file and from this obtains a local user identifier (uid). Then the authorization mechanism changes the security context of the executing process through the setuid system call, using the uid as input.

Layering distributed system access control over existing host authorization services allows program components such as existing system solvers to access stand-alone system resources without modifying their code. Thus, a layered approach satisfies requirement 1.

However, most layered authorization schemes require the software supplying distributed access control services to run as root. Thus, requirement 2 is not met by these approaches. Below we describe a layered authorization technique that does not use root privileges.

The layered approach meets requirement 3, since it utilizes distributed system authorization on each machine and we assume hosts support some variant of Unix. Its resistance to host compromise rests principally on the resistance of the authentication mechanism to this threat. Kerberos and DASS authentication mechanisms are relatively robust in the face of host compromise. Users that directly enter their Kerberos or DASS passwords on compromised machines are themselves compromised. The proxy-based ticket approach has the additional vulnerability that servers on compromised hosts possessing forwardable tickets allow them to be compromised. However, in a large distributed system these compromises give the intruder access to a small proportion of the total distributed system resources. Compromise of the Kerberos authentication and TGT servers compromises the whole distributed system, but these are special systems that may be strongly protected using high-grade physical and operational protection strategies. The use of /etc/passwd authentication is also fairly robust when a host is compromised, since users entering their passwords for other hosts are compromised on those hosts, but generally this does not compromise the whole distributed system. Consequently, requirement 4 is met by most of the popular authentication mechanisms.

If the authentication mechanism allows the quick removal of users from its databases, which is true for Kerberos and DASS, then requirement 5 is met. However, if the /etc/passwd mechanism is used, quick revocation is unlikely, especially in a large distributed system.

As specified above, only layered authorization mechanisms that rely on Kerberos or DASS satisfy requirement 6. Those that rely on /etc/passwd authentication fail in this regard.

Requirement 7 generally isn't met by most layered authorization schemes, because they do not interoperate with each other. For example, a user operating under an /etc/passwd based scheme cannot access resources in other domains controlled under a Kerberos based scheme. While there is an effort underway to harmonize Kerberos and DASS authentication, such a facility still will not interoperate with an /etc/passwd based facility.

Server-centric authorization

It is possible to design a distributed system authorization mechanism that does not rely on the authorization mechanism of hosts. Specifically, resources on the underlying machine can be owned and managed by a server, which multiplexes them among its clients (server-centric authorization).

Server-centric authorization doesn't meet requirement 1, since access to distributed system resources occurs not through system calls, but rather through server requests. This implies that existing libraries and programs must be modified to use resources managed by distributed system servers.

However, server-centric authorization does satisfy requirement 2. Servers multiplex access to stand-alone system resources, relying on the host operating system authorization mechanism to grant them access to the resources they own. This does not require root access privileges. Furthermore, this approach will operate on any Unix operating system, so requirement 3 is satisfied.

The compromise of one host may or may not compromise other distributed system hosts depending on how the authorization mechanism operates. It is possible to devise a server-centric authorization method that has good compromise containment properties. For example, the LINCS distributed operating system used the server-centric approach for its Unix guest file server. Since files were accessed through capabilities, the compromise of one host only compromised those files with capabilities on that host.

If the server-centric authorization mechanism relies on an appropriate authentication mechanism, such as Kerberos or DASS, then system administrators can quickly revoke a misbehaving user's access control rights. Consequently, requirement 5 can be met.

This approach gives the access control architect the flexibility to create a mechanism that does not encourage the user to engage in unsound security practices. For example, LINCS guest file server capabilities can be protected against both forgery and theft.

Finally, server-centric authorization can easily be made to work in a heterogeneous environment, since the difference in access control mechanisms are hidden by the server implementation. In effect, each server acts as an access control gateway, translating from the distributed system access control mechanism into the access control mechanism used by the host. Of course, if the server-centric mechanism is to operate between domains that use different authentication schemes, such as Kerberos or DASS, then either the servers must be instrumented to handle all such authentication mechanisms or there must be authentication gateways that translate from one scheme to another. This last approach is being taken in the effort to harmonize DASS and Kerberos.

Critique summary

Both the layered and server-centric approaches to authorization present difficulties when used in large practical distributed systems. Server-centric authorization imposes burdens on existing software, requiring it to be reimplemented for use in distributed applications. Most schemes that layer distributed system authorization on host authorization require servers to run at root and do not adequately cope with heterogeneity.

In the next section a layered authorization mechanisms is described that does not require root privileges and that accommodates heterogeneity by supporting several different authentication mechanisms concurrently. This is done in such a way that it also presents good compromise containment properties.

A PRACTICAL AUTHORIZATION SCHEME

The authorization technique described here is used by *Remoxe*, a *remo*te e*x*ecution service for Unix developed and in use at the Lawrence Livermore National Laboratory. A Remoxe server executes as a daemon on each computer where the service is provided. A client process on any computer can send to a server (generally on a different computer) a message asking that it execute some application. The client and the application may then communicate either through the server (in which case the application thinks that it is dealing with a controlling terminal) or directly (using sockets). The executing application has access to a *context* chosen by the client, where a context consists of the resources available to a particular user on the service computer. This choice of what constitutes a context is dictated by the nature of typical Unix systems; it could readily be modified for systems with other forms of local access control (such as capability-based systems). The access to a context is authorized without a password having to be typed.

Access lists and capabilities are frequently described as alternative means for authorizing access to resources. However, particularly in a distributed environment, the techniques are often complementary and are used together. For example, consider conventional remote access using such facilities as *telnet* or *ftp*. Access lists on the service (remote) computer (typically in the rather coarse-grained form of *owner*, *group*, and *world* access permissions) are used in connection with a user name and password provided from the client (local) computer. The user name and password together effectively constitute a capability, a coded record that establishes the client's relationship to the access lists (by defining and verifying the owner's identity).

Remoxe makes use of a capability we call a *xap* (execution *a*ccess *p*rotector, pronounced "zap"). It is a coded record that is originally generated by the server and sent to the client to be stored until needed. It is sent back to the server as a parameter in messages requesting remote execution or other action. It identifies and authorizes access to a context and includes the following information:

- the TCP/IP address of the Remoxe server for which the xap is valid,
- the local user name associated with the context on the service computer,
- permission bits,
- authentication information (e.g., a GSSAPI global name), and
- a DES encryption key for the local password associated with the user name.

One permission bit enables remote execution; the others enable various "housekeeping" actions in regard to the xaps themselves, such as issuing additional ones or revoking existing ones.

A xap should be kept in a safe place, such as in a file accessible only by the user (owner), on a client computer that has a secure operating system. This last condition especially is difficult to meet for all too many Unix systems. So there may be a problem of xaps being stolen, that is, illicitly copied. The purpose of the authentication information is essentially to provide a degree of protection against the theft of a xap by limiting the effectiveness of the xap to situations in which additional information is also supplied, authenticating that the client has the right to use the

xap. Each xap employs one of three authentication options (listed in order of increasing security):

- No authentication is required. So there is no protection against theft: a purloined xap may be used by the thief (or anyone else) from any client computer. This option is provided only as a last resort for situations (we hope that there are none) where the other options are infeasible or (would that it were so!) where there is no danger of theft.

- The use of the xap is limited to a particular client computer (more precisely, a particular client IP address). The thief cannot hide himself in a distant part of the network while he misuses the xap. This option is provided for use where the necessary infrastructure for the next option is not available.

- The use of the xap is limited to the user with a particular global name as defined by an authentication system based on GSSAPI (namely, Kerberos or DASS). The xap must be accompanied by the evidence (context token) required by that system for establishing that the user has that name, and the degree of protection depends on how secure that system is. This is the preferred option.

The 64-bit local password encryption key appears in a xap exclusive-or'd with a DES cryptographic digest computed using all the other information in the xap and a master key that is known only to the server. The xap thereby not only conceals the encryption key, but also is protected against forgery. Anyone trying to generate a xap (either out of whole cloth or by altering a few bits, such as permission bits, in a valid xap) has only one chance in 2^{64} of correctly rendering the encryption key (effectively only one in 2^{56} because of the way DES uses keys). When a user first establishes himself with the server, at which time he must supply his local user name and password for the service computer (but not a xap) in a secure manner, the server randomly generates an encryption key just for that user. The key is then used to encrypt the user's local password. The server stores the encrypted password (in association with the local user name) in its records with sufficient redundancy that it can with high confidence recognize an improperly decrypted password before attempting to use it. The server remembers *neither* the unencrypted password *nor* the password encryption key, but it includes the latter in a xap which it issues to the new user (Fig. 1).

Therefore the server can obtain the password only when a client provides a valid xap. This means that compromise of a user's password requires "breaking into" *both* the user's records on a client computer *and* the server's records on the service computer. It is our view that such is an obstacle sufficient to render Remoxe acceptably secure.

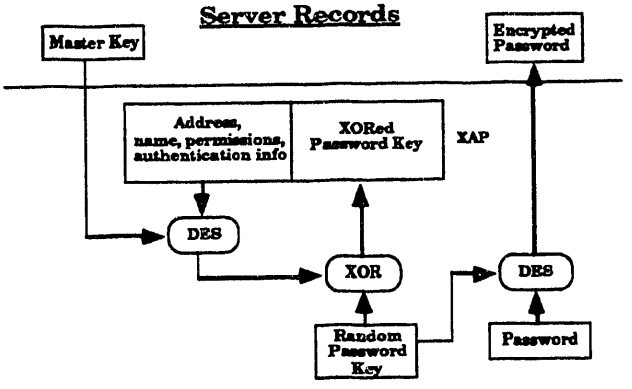


Figure 1. Concealment of password, using xap.

There are "housekeeping" chores in dealing with xaps. Remoxe provides for establishing a new user with the server and issuing the initial xap, for issuing additional xaps that may have reduced permissions and/or differing authentication information (in particular, allowing access from a different client computer), for changing the password on the service computer (both as known to Unix and as known to the server), for revoking all the user's existing xaps (by changing the password encryption key) and issuing a new xap to replace them, and for deleting the user from the server's records (which of course also effectively revokes all existing xaps). Note that, since the content of a xap does not depend on what the password is, changing the password does *not* affect the validity of existing xaps.

It is possible for each user to have a Remoxe server of his own, running on a service computer with its own "well known" TCP port (that is, a fixed port number that can be "built into" client programs). However, the user then assumes the burden of installing his server and assuring that it is always up and running. Also, there is an inefficiency in having many separate servers on a computer, all performing basically the same job. So the intent is that there be only a few Remoxe servers (often just one) on each service computer, each installed and maintained by a single user, its *sponsor*. This user owns and could access the files in which the server keeps its records. He therefore must be someone who can be trusted by the other users not to abuse his position and invade their privacy by either misusing the records himself or through carelessness letting them be accessed by others. That is, each server and its sponsor corresponds to a community of trust. The sponsor could be the "superuser", but we have not required this, because we want a user to be able to install and begin using Remoxe without waiting for an administrative bureaucracy to give its approval and then take action. The server's records are kept in files in a subdirectory of its sponsor's home directory; this directory is created when the sponsor installs the Remoxe server. The records are accessible only by the sponsor, who is their owner.

When a client requests the running of an application, it sends the server a xap accompanied by any necessary authentication information. The server verifies the authentication information (e.g., by calling the appropriate GSSAPI procedures) and also decrypts and verifies the password in the xap (Fig. 2). It then forks a process but does not directly "exec" the application. Instead, it "execs" the standard privileged Unix utility *su* and delivers the user's name and password to it as input. After *su* has converted the forked process into a shell associated with the user's id, additional input effects the "exec" of the desired application. In this way the application runs with the environment and access rights of the user, rather than with those of the sponsor.

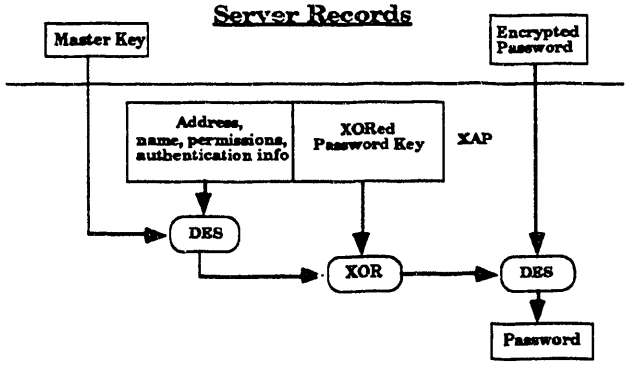


Figure 2. Recovery of password, using xap.

This rather roundabout, complex, and no doubt inefficient technique for establishing the proper context is necessitated by the peculiarities of Unix. Perhaps the designers of future operating systems will consider the following suggestions:

- The natural way for a program to interface to the operating system is through a privileged procedure (system call), rather than by forking and "exec'ing". So there should be a privileged procedure to which one can pass a user name and password (or other system specific access control information) and which will then set the user id to that of the user. Similarly, there should be a privileged procedure for changing a password.

- In fact, there should be a simple, direct way without administrative intervention for any user to establish a service and for other users to be able to grant that service such access to their resources as they choose. They should not have to grant this access in an "all or nothing" fashion, but should be able to adhere to the principle of least privilege. (A capability-based system would achieve this.)

An analysis of remoxe authorization

We briefly analyze the advantages and disadvantages of the Remoxe authorization method and then provide a crude compromise containment analysis. Servers, such as Remoxe, that utilize our authorization technique can be brought up immediately on any machine on which the user(s) have accounts. No system administrator help or approval is necessary. If the distributed system hosts are configured to accept a global password for local authentication, such as a Kerberos or DASS password, users need only remember one. This reduces the possibility of password compromise and the inconvenience associated with reissuing passwords that the user has forgotten. The Remoxe authorization scheme provides these advantages along with security that is at least as good, if not better, than other approaches.

One disadvantage of the Remoxe authorization scheme stems from one of its advantages, the lack of involvement of a system administrator when bringing up permanent servers. Without root privilege or system administrator help, keeping these servers up across system crashes is problematical. Normally, permanent servers are brought up at reboot based on an entry in the rc.local file. Since this file is owned by root, this technique is not available to the unprivileged user.

We are exploring use of the "At" utility to overcome this problem. In particular, a non-privileged server or a "persistence daemon" can periodically call "At" to schedule a check that the appropriate servers are still running. If the check fails, the server can be restarted. However, this approach requires that the activity scheduled by "At" survive across system crashes. This may or may not be true, depending on the particular variation of Unix on which the server runs.

The Remoxe authorization method has quite good compromise containment properties. If an intruder gains access to the files in which the Remoxe server master key and encrypted passwords are stored, these cannot be used, since the intruder lacks the Xaps that contain the keys to the encrypted passwords. If the intruder obtains a Xap, either by reading it from an improperly protected file on a client machine or by capturing it as it travels over a network, it cannot be used (assuming the GSSAPI authentication option is employed) since the intruder cannot manufacture the necessary time-limited credentials required by the supporting authentication technology.

If an intruder obtains root access on the server machine, he has compromised all resources on it. He need not take advantage of servers using the Remoxe authorization scheme. However, if he manages to compromise a sponsor, he gains access to the resources of all other users that have entrusted their access rights to the server the sponsor controls. This is a good reason for supporting several servers on a machine, one for each community of interest that runs there.

If an intruder obtains root access on a client machine, he can wait for its users to type their global (i.e., GSSAPI-based authentication) passwords and thereby compromise their resources in the distributed system. This would be true

whether Xaps are stored on the machine or not. Consequently, the Remoxe authorization scheme does not introduce any new vulnerabilities for this situation. In fact, since the intruder may not have access to all the users Xaps, i.e., those stored on other systems, the Remoxe scheme potentially can lessen the damage caused by a client machine compromise.

USE OF REMOXE

The need for Remoxe originally arose from the following typical situation: A user has a number of source files that he maintains and edits on his workstation, which offers him convenience, economy, and high interactivity. However, many of the sources are intended to be compiled and executed on a supercomputer, which offers power. After editing, the user transports the updated sources to the supercomputer and there compiles and executes them. He would like to have the required updating occur automatically in response to a single, simple typed command, such as "make".

The standard Unix utility *make*, used in conjunction with standard utilities that provide remote access, such as *ftp*, *telnet*, and *rsh*, would seem to provide the required facility. That is, *make* would invoke *ftp* to transfer the sources to the supercomputers and then invoke *telnet* or *rsh* to execute remotely the compiler, other utilities, and the compiled applications. However, there are two difficulties:

- *Make* makes decisions based on the exit status of the programs that it runs, which is not available for programs run remotely by *telnet* or *rsh*.

- Each time that *make* invokes a utility providing remote access, that utility prompts and waits for the input of a password, severely inconveniencing the user and requiring his continued attendance at the terminal; it would be difficult to view the activity as truly automated. Common means of circumventing this problem, such as identifying oneself as "anonymous" or "guest", making appropriate entries in the .netrc or .rhost files, and/or using the Network File System (NFS), open up privacy or security loopholes that are often unacceptable. These remarks also apply to use of the standard utility *rdist*.

A client utility has been provided for use with Remoxe that avoids both of these problems. In regard to the first problem, the protocol between the client and server is such that status information is returned after each remote execution; this status is in turn returned as the status of the client. To effect execution on a supercomputer, *make* invokes the client utility, which then (through the remote server) invokes the remote application. *Make* will then correctly interpret the returned status as that of the application.

There is no need for passwords, because authorization is effected using xaps appropriate to the remote servers. (In fact, the xaps specify which remote servers — which remote computers — are to be used.) These xaps are fetched from files specified in the commands to the client. The client also carries out any needed GSSAPI-based protocol. In addition, a file transfer utility has been provided to be run under the control of the server. Files may be transported between this utility and the client. A sample commented makefile for remote execution is displayed in Fig. 3. Note that *make* invokes a second, remote *make*.

```
#   This makefile effects the remote compilation and
# execution of the application "test", which tests a subroutine
# package.

#   The directory "shadow" contains empty shadow files,
# one for each file that is to be sent to the remote computer.
# These files "stand in" for their remote counterparts when
# "make" tests their ages. After a file is sent, its shadow is
# aged by using "touch".
#   The client utility is "clnt"; "-f sc" specifies that the xap
# is to be found in the file "sc", and "-d dir" specifies that
# remote execution is to occur in the directory "dir". The
# remote commands are "make" and "test", the former
# referring to a remote makefile that should effect the
# compilation of the three ".c" and ".h" files into the
# executable file "test".

update: shadow/test.c shadow/subrs.c shadow/subrs.h
        clnt -f sc -d dir make
        clnt -f sc -d dir test

#   Before the remote "make" is invoked, any updated ".c"
# and ".h" files are transported using the remote utility
# "rmx", which interprets "put xxx.x" as a request to have
# the file "xxx.x" sent from the client to the remote computer.

shadow/test.c: test.c
        clnt -f sc -d dir rmx put test.c
        touch shadow/test.c

shadow/subrs.c: subrs.c
        clnt -f sc -d dir rmx put subrs.c
        touch shadow/subrs.c

shadow/subrs.h: subrs.h
        clnt -f sc -d dir rmx put subrs.h
        touch shadow/subrs.h
```

Figure 3. A sample simple makefile.

ACKNOWLEDGMENTS

"Work performed under the auspices of the U.S. Department of Energy by the Lawrence Livermore National Laboratory under contract number W-7405-ENG-48."

BIBLIOGRAPHY

1. D. M. Nessett, "Factors affecting distributed system security," IEEE Transactions on Software Engineering, vol. SE-13, no. 2 Feb., 1987, pp. 233-248.

2. D. M. Nessett and G. M. Lee, "Terminal services in heterogeneous distributed systems," Computer Networks and ISDN Systems, Vo. 19, pp. 105-128, 1990, Elsevier Science Publishers B.V., Amsterdam, The Netherlands.

3. J.G. Fletcher, "Software Protection of Information Networks," Infotec State-of-the-Art Report, Future Network, vol. 2, 1978, pp. 149-164.

4. R.W. Watson and J.G. Fletcher, "An Architecture for Support of Network Operating System Services," Computer Networks, vol. 4, Feb., 1980, Elsevier Science Publishers B.V., Amsterdam, The Netherlands, pp. 33-49.

5. D.M. Nessett, "The Inter-Authentication Domain (IAD) logon protocol (Preliminary specification and implementation guide)," Lawrence Livermore Nat. Lab. Rep. UCID-30207, 1984.

6. D. Estrin, "Non-Discretionary Controls for Inter-Organization Networks," Proc. IEEE Symposium on Security and Privacy, IEEE, Los Alamitos, CA., April, 1985, pp. 56-61.

7. J.G. Steiner, C. Neuman and J.I. Schiller, "Kerberos: An authentication Service for Open Network Systems," Proc. Winter Usenix Conf., Usenix Association, Berkeley, CA., 1988, pp.191-202.

8. S.P. Miller, B.C. Neuman, J.I. Schiller, and J.H. Saltzer, "Kerberos Authentication and Authorization System," section E.2.1 of *Project Athena Technical Plan*, MIT, Dec. 1987.

9. J. Linn, "Practical Authentication for Distributed Computing," Proc. IEEE Symposium on Research in Security and Privacy, IEEE, Los Alamitos, CA., May, 1990, pp. 31-40.

10. J.G. Fletcher and R.W. Watson, "Service Support in a Network Operating System," CompCon 80, Spring, 1980.

11. "Department of Defense Trusted Compter System Evaluation Criteria," DOD 5200.28-STD, Department of Defense, Washington, DC, December, 1985.

12. B. Clifford Neuman, "Proxy-Based Authorization and Accounting for Distributed Systems," University of Washington Technical Report 91-02-01, Department of Computer Science and Engineering, University of Washington, FR-35, Seattle, Washington.

13. "OSF DCE 1.0 Application Development Guide; Revision 1," Dec. 27, 1991, Open Software Foundation, Cambridge, MA.

Extending the OSF DCE Authorization System to Support Practical Delegation

Marlena E. Erdos
Joseph N. Pato

Hewlett-Packard Co.
Chelmsford, MA

ABSTRACT

In a simple client/server distributed environment, two principals are involved in most transactions - the initiator and the target of the operation. The target of the operation can reasonably make authorization decisions based on the identity of the initiator. This model is insufficient, however, when the server performs operations on other components on behalf of the initiator as is common in distributed object oriented environments. This paper will describe the need for a delegation facility in distributed object oriented systems and then present some elements of the delegation system we've proposed for inclusion in OSF's Distributed Computing Environment (DCE).

INTRODUCTION

The need for a delegation facility

In a distributed object oriented environment, intermediate objects hide the details of complex system interactions. These intermediate objects receive high level requests from initiating clients and perform some series of low level operations on a number of other services. Unfortunately the interposition of the abstracting object prevents the target services from securely determining the identity of the initiator of the operation. All requests arriving at the target services appear to be the action of the intermediary rather than the true initiator.

The inability to determine the true initiator of a request has a chilling effect on the design of distributed systems. The designer of an intermediate service is forced into a set of unsatisfactory design choices. The service may be implemented as a local process that runs with the identity of the initiator, but loses the benefits of distribution. Alternatively, it may retain distribution but then it must run as a privileged principal that has full access to all services it abstracts. This solution has the disadvantage of forcing the abstracted services to trust that the privileged intermediary will make correct access control decisions on their behalf. A third unsatisfactory approach is the use of an alternate set of target service interfaces that allow an authorized principal to specify the principal on whose behalf the operation is really being performed. This solution comes at the cost of redundant interfaces that expose the details of privilege attributes to the application protocol. Finally, the service may be implemented in a way that impersonates the initiator - where the initiator transmits to the intermediary service the credentials (tickets and keys) necessary to be indistinguishable from the initiator. This final approach is much like a non-distributed application - but the mere fact of distribution (and in the DCE the high degree of location transparency) makes it so that this greatly increases the risk to the client of being compromised by a Trojan horse application.

To solve this problem adequately, some form of delegation is required. We've have proposed a delegation architecture and design to OSF for inclusion in the DCE [1]. This system is described below.

Delegation system

Our delegation architecture has three major components: First, we allow an intermediary to operate on other objects in a manner that reflects the initiator's identity as well as its own. A target server receiving such a chained request would see the privilege attributes of each participant in the chain.

Second, we extend the authorization model to allow target servers to make use of the distinction between initiators and intermediaries. Target servers may grant rights to principals acting as intermediaries on behalf of authorized initiators without granting rights for those prinicipals to act on their own.

Lastly, we allow clients performing operations to place restrictions on the uses of their identity in chained calls. A client may choose to entirely disallow delegation or to limit which principals may use the client identity in a delegated manner.

The delegation design uses composition of privilege attributes to realize identity chaining, additions of new ACL entry types to reflect the initiator/intermediary authorization distinction, and extensions to the security API to allow clients to control delegation of their identity. This paper will further discuss the delegation architecture and design, and then present some low-level extensions to existing DCE elements and mechanisms for accomplishing the implementation.

ASPECTS OF THE EXISTING DCE

In the basic DCE environment [2,3], access to a resource is managed by an application server which is the reference monitor for the resource. When a client attempts to perform an operation on the resource, the reference monitor examines the client's identity and compares it to control attributes associated with the resource. The client's identity is represented by a set of *privilege attributes* (PAs) and the control attributes are stored in an *access control list* (ACL).

Privilege attributes are the collection of information about a principal that is used by the authorization system when determining if access to a resource should be granted. These PAs are limited to the unique identifiers representing the initiating principal and the set of groups to which the principal belongs. A trusted system component, the *Privilege Server* (PS) [4], produces a tamper-proof *privilege attribute certificate* (PAC) that contains the PAs and is suitable for presentation by a client to a server. By relying on PACs for the identity of a caller, we place the same degree of trust in the Privilege Server that we have already placed in the authentication component of the distributed system. A compromised Privilege Server will be able to generate PACs that impersonate any legitimate user.

Collecting privilege attributes into a certificate that can be presented by the client to a server has the benefit of allowing servers to make authorization decisions locally without the need of contacting trusted system services to obtain privilege attributes for the client principal. In addition this model allows the client to choose the set of privileges to be used during a given session.

The emerging next generation of the DCE evolves the PAs supported to be closer to the capabilities found in ECMA [5,6]. This change allows for greater flexibility in the authorization models available to the distributed system, and of particular interest to this paper - provides a vehicle for recording delegation information.

ARCHITECTURE & DESIGN OF DELEGATION

Intermediaries and Chained identities

A server acts as an intermediary or delegate when in order to fulfill an operation a client made on it, it must perform one or more operations on other objects. We say that these subsequent operations are performed *on behalf of* the client as part of a *chained call*. All chained calls are performed with a *chained identity*.

We represent chained identities using the conventional notation [7,8,9]

foo **FOR** bar

This means that principal *foo* is acting on behalf of principal *bar*. When more than one intermediary is involved in the chained call, the identities of all participants are reflected in the chained identity. In general,

DelegateN **FOR** DelegateN-1 **FOR** ... **FOR** Initiator

means that principal *DelegateN* is acting on behalf of principal *DelegateN*-1 which is acting on behalf of principal *DelegateN*-1 extending back to the initiating client (aka *initiator*) which is acting on its own behalf.

Identity is represented by a set of privilege attributes (PAs). Logically, a chained identity is represented by an ordered array of PA sets, with the PA set of the initiator distinguished from those of intermediaries.

Each object acting as a client may choose whether or not it wishes to allow the immediate target to use its (the client's) PA set in a call chain. In other words, each client enables/disables delegation of its identity. However, the determination that a given call is part of a call chain is up to the intermediary and is dictated by the semantics of the situation. We'll discuss the consequences of a mismatch between what a client allows and what an intermediary needs to do in when discussing restrictions on the flow of identity below.

Authorization model at the target

Each server has one or more access control lists (ACLs). ACLs, as found in Posix or DCE [10], contain entries that identify the access rights granted to principals bearing certain PAs. To support delegation, the target server effectively grants one set of rights to initiators and another set to intermediaries.

We realize this distinction by extending the standard ACL entries for principals, groups, etc. with a corresponding set of entries that apply to principals and groups acting as intermediaries. These *delegate entries* grant *intermediary rights* i.e., the ability to act as an intermediary for an operation, but do not grant the ability to operate on the target object directly.

While authentication of an operation is done automatically by the security runtime at the server, authorization is only performed if the server application code explicitly invokes the authorization facilities. Whether a call is done with a chained identity or not is transparent to the server application code, however the authorization facilities go through additional checks when presented with a chained identity.

For both simple and chained identities the authorization facilities first determine whether or not the initiator is authorized to perform the operation. They do this by examining the standard entries - and only these entries - on the ACL when calculating the initiator's rights. Initiators

don't - and can't - expand their access through use of intermediaries; at best, intermediaries won't degrade access that the initiator has on its own. The initiator of a chained operation must have rights to perform the operation directly or the operation will be rejected.

If the initiator passes the authorization check and the call was chained, then the authorization facilities next check the PAs of each intermediary in the chain. Each delegate must have sufficient rights to act as an intermediary for the operation or the authorization facilities will return an authorization failure indication to the server application code. A delegate is deemed to be authorized if its PA set gives it either initiator rights or delegate rights for the operation. In other words, any principal that can perform an operation directly is implicitly authorized to be an intermediary for the operation. The access control algorithm is presented in Figure 1.

(i) *Check Initiator:*
Apply standard algorithm

IF access mode is denied **THEN**
Deny Access
ENDIF

(ii) *Check Each Intermediary:*
FOR EACH Privilege Attribute Set **IN** Extended PAC **DO**
Apply standard algorithm (allow delegate entries)
IF access mode is denied **THEN**
Deny Access
ENDIF
END

(iii) *Grant Access*

Figure 1: Access control algorithm for delegation

Note that the order of intermediaries, the topology of the call-chain, is not relevant in the access control decision.

Client restrictions on flow of identity

Before discussing client restrictions on identity flow we'd like to clarify some terminology. An object acts as a *client* when it sends an RPC to another object. We use the term *target* to refer to any object that is downstream in a call chain from a given client. *Immediate target* is the object a client performed an operation on directly. *Direct Requester* is the client that directly operates on a given target. As we've previously mentioned, *initiator* is the initial client in a call chain. *Final target* is the last object in a call chain.

In the absence of delegation it is simple to understand the flow of identity in the system. The identity of a client is projected to its immediate target. In this environment the client determines that it wants to perform an operation on the immediate target and its identity is not subject to further use by the target. In the presence of delegation, however, the immediate target gains the ability to project its caller's identity. In this environment we allow each client to protect itself by placing limitations on who may project its identity and to whom its identity may be projected.

As we've mentioned a client may simply allow or disallow use of its identity in a chained call. We also permit any object acting as a client, either as initiator or delegate, to place restrictions on the delegations it allows. The two types of delegation related restrictions are *target restrictions* and *delegate restrictions* .

Target restrictions set by a given client apply to all servers in a call chain that are down stream from the immediate client-intermediary pair. For example, in the following call chain

A -> B -> C -> D

target restrictions set by *A* apply to both *C* and *D*, but not to *B*. *C* and *D* are targets of a delegation through *B*. Though *B* is the immediate target of *A*'s operation, it is not a delegation target.

Delegate restrictions set by a given client limit who may act as an intermediary. They apply to all servers that are downstream from the client that wish to acts as intermediaries. Again, given the above call chain, delegate restrictions set by *A* apply to *B* and *C*. The delegate restrictions are irrelevant to *D* simply because *D* is not acting as an intermediary.

The effect of either type of restriction being violated is the same. The identity of the party that placed the violated restriction will be replaced with the anonymous identity. Other identities already present in the chain are not affected.

If the client doesn't allow its identity to be delegated, then the server it calls will receive its identity - allowing the server to make an appropriate authorization decision. Any subsequent objects called on the client's behalf, however, will not. see the client's identity These objects will still see a chained identity but the security system will substitute the *anonymous* identity where the identity of the client would have appeared.

Here is an example of a chained identity with anonymous entries:

foo **FOR** *anonymous* **FOR** bar **FOR** *anonymous*

This says that principal *foo* is acting on behalf of an anonymous intermediary which is acting on behalf of *bar* which is acting on behalf of an anonymous initiator.

While it may seem much less important to allow intermediaries to place delegation restrictions than initiators, we feel such functionality is important for an extensible system.

Extensible client restrictions

Our system has two types of extensible restrictions on privilege attributes. These restrictions allow applications to implement a variety of security models and policies beyond those expressible through the supplied PA/ACL system. An example of an extensible restriction that an application might define is a time-of-day restriction.

Required attributes limit the activities that a target server can perform. A server receiving a required restriction must be able to understand it. If the application is unable to decode a required restriction it must reject access.

Optional restrictions differ only from required restrictions in that applications that are unable to decode a given optional restriction are free to ignore its presence.

Example of the model

Figure 2 provides a frequently used example of a compound document. In it a user is accessing a document which contains a graph that obtains its data from a spreadsheet. When this document is implemented in a distributed object environment, each component may run as an independent process with a distinct principal identity. The document, graph and spreadsheet are each reference monitors for their data and grant access based on the contents of their associated ACL.

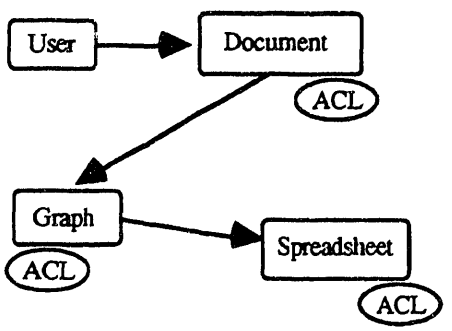


Figure 2. Compound Document Components

In this example let the User process run as principal *U*, the Document as principal *D*, the Graph as principal *G* and the Spreadsheet as principal *S*. The User process enables delegation and performs a view_document operation on the Document.

On receipt of the view_document operation, the Document consults its ACL and verifies that *U* has the rights necessary for the view_document operation. The Document proceeds to compose a delegated identity of *D* **FOR** *U* with delegation enabled and performs the view_graph operation on the Graph.

The Graph process receives the view_graph operation from the Document object. It consults the ACL shown in figure 3 and verifies that *U* is authorized to initiate a view_graph operation and also verifies that *D* is a legitimate delegate. As a component of completing the view_graph operation, the Graph composes the delegated identity *G* **FOR** *D* **FOR** *U* and performs the obtain_range_data operation on the Spreadsheet.

ACL Entry Type	PA Value	Permission
User:	*U*	view_graph
User_delegate:	*D*	view_graph

Figure 3. ACL for Graph Object

Finally the Spreadsheet process receives the obtain_range_data operation. Applying the ACL shown in figure 4 it verifies that *U* is a valid initiator matching the any_other entry and verifies that *G* and *D* are legitimate intermediaries since they also match the any_other entry in the ACL.

ACL Entry Type	PA Value	Permission
Any_other:	N/A	obtain_range_data

Figure 4. ACL for Spreadsheet Object

MECHANISM

The DCE provides strong mechanisms for trustworthy transmission of identity between client and server. Delegation introduces changes in these identity transmission mechanisms. We discuss the existing mechanisms for projecting identity prior to considering the changes for supporting delegation.

Note that the DCE is designed to allow a number of different authentication and key distribution protocols to be used. DCE 1.0, however, only includes a concrete specification and implementation using the Kerberos V5 [11] protocol suite. Consequently, we will restrict the following discussion to the mechanisms used for identity flow that are used by the DCE in conjunction with the Kerberos V5 protocols.

Overview of existing DCE system

Of the features of the DCE security protocols, two are fundamental - the ability to provide integrity and confidentiality protections to a communication session between a client and a server and the ability for the communicating agents to determine the identity of their

partner. Kerberos V5 protocols provide the mechanisms for accomplishing both tasks. Conventionally, however, the notion of identity in a Kerberos environment is limited to the name of a given principal. As described above, a DCE identity is a set of privilege attributes that are active for a given principal during a given session. The DCE leverages the authorization data field of a Kerberos V5 ticket to carry the additional privilege attributes.

When a user session is created, normally through some variant of the local system login sequence, the DCE security runtime acquires the principal's ticket granting ticket (TGT). The TGT may then be used to obtain tickets to other server principals. These new tickets may then be used to exchange keys with those targets and establish protected communication. These tickets, however, are not suitable carriers for privilege attributes since the client is free to request any data in a ticket's authorization data field.

For the security runtime code at a server to trust a given set of privilege attributes it must believe that an authorized system service has constructed (certified) the data. In the DCE, the trusted component is the cell's[1] Privilege Server. The server will trust a ticket bearing PAs and treat it as a PAC if the ticket is issued in the well-known name of the Privilege Server.

To get such a ticket, the DCE login code actually obtains two TGTs. The first TGT obtained in the client principal's name is generally only used to obtain a ticket to the cell's Privilege Server. Once this is obtained the runtime communicates with the Privilege Server to obtain a second TGT. The second TGT is issued in the name of the Privilege Server and contains the privilege attributes requested by and valid for the client principal in the ticket's authorization data area. This second TGT is referred to as the privilege TGT (PTGT). It is the PTGT that is then used by the client when obtaining tickets to target servers. The Kerberos V5 KDC will automatically transfer the PAs identifying the actual client from the PTGT into the authorization data field of the ticket for a given target server.

The Extended PAC

Our delegation model requires extensions to the contents of PACs. The content change is due to extending the notion of identity to include chained identities. In addition we have added delegation and extensible restrictions to the extended PAC.

Two other concerns have contributed to change in the DCE 1.0 PAC: performance and legal issues. The authorization field of a Kerberos V5 ticket is encrypted in the key of the target principal. This raises performance concerns as the number of privilege attributes is increased. In addition legal issues are raised with respect to the encryption of data. The extensible restrictions allow applications to provide arbitrary uncontrolled data in the EPAC. Encrypting this data may violate laws governing the export of encryption technology - and in some countries may violate laws controlling the transmission of encrypted data over public carriers.

The solution we have chosen to address these various issues was proposed by the European Sesame project [12]. The contents of a EPAC need not be confidential - therefore we have removed them from the authorization data field of the V5 ticket and simply placed a cryptographic hash of the EPAC in the PTGT. This seal[2] serves to connect the EPAC to the ticket and provides the same guarantees of authenticity of the EPAC to the target server.

Becoming a Delegate

When a server needs to perform an operation on another target on behalf of its client, it must become a delegate for that client. The security runtime at the server possesses the EPAC for its caller and it has a PAC representing itself, but these two PACs are not directly usable to represent the new chained identity. The server must obtain a new EPAC (and PTGT) that represents the chained identity from the Privilege Server.

The mechanism described so far does not provide the delegate with the necessary data to submit to the Privilege Server to acquire the new EPAC representing the chained identity. The delegate does not possess proof can be presented to a third party that the incoming EPAC is legitimate. It needs some form of delegation token [13] that may be submitted to the Privilege Server when requesting a new chained EPAC.

The extended PAC, obtained from the caller, contains all of the information needed by the Privilege Server to determine if a given principal should be allowed to chain its identity to that of the caller. The only missing data is a seal protecting the integrity and authenticity of the extended PAC. We have, therefore, added a signature field to the extended PAC allowing it to become a true certificate and thus serve as the delegation token. The signature field supports both a seal - using a key known to the Privilege Server - and a signature using public key technology. This allows the same certificate to be used by the DCE's existing Kerberos environment and by public key based facilities.

Compatibility with existing servers

Whenever there is the introduction of a new revision of system, interactions with the prior revisions must be

[1] The DCE's notion of an administrative domain, roughly comparable to a Kerberos realm.

[2] We use the ISO definition for seal indicating a cryptographic checksum using symmetric keys. This should not be confused with other uses of the term that indicated confidentiality of the data.

considered. In our case, it is possibly that a new intermediary will send an extended PAC to an old server that only understands the simple PAC format. To deal with this situation we provide three compatibility modes. We call refer to them as *initiator*, *direct requester* and *reject*.

When the intermediary requests initiator compatibility mode, the Privilege Server arranges the associated ticket so that it includes the PA set of the call chain's initiator in the authorization data field. Appended to this is the hash of the EPAC. In other words, the PA set of the initiator is placed where an old (DCE 1.0) server already expects a single PA set. The security runtime at old servers is set up to ignore extra bytes in a PAC, so the presence of EPAC hash does not have any untoward effect. With this compatibility mode, the current intermediary appears to the target server to actually have the identity of the initiator.

When the client requests direct requester mode, the PS arranges the ticket so that the intermediary's identity is in the authorization data field Again, the EPAC hash is ignored as extra bytes by old servers. In this mode, in order for a DCE 1.0 server to authorize the operation, the intermediary must have appropriate rights.

In reject mode, the intermediary effectively asks the PS to set up the ticket so that an old server will know that it is dealing with a new client and will reject the call.

The current intermediary requests a particular compatibility mode when it requests a new login context that reflects a chained identity. Likewise the initiator has specified whether or not it will permit initiator compatibility mode in the course of obtaining a login context appropriate for an initiator. If the intermediary requests initiator compatibility mode but the initiator has not allowed it, the intermediary will receive an error indication immediately. Because of the possibility of this conflict, we allow the intermediary to effectively say 'I want initiator mode, but if it isn't permitted I'll take direct requester mode'. This form of request will generally not fail.

Impersonation, aka Full Identity Forwarding

While many of today's distributed systems (such as the DCE) lack a delegation facility, the need for delegation has existed for some time. Often, to accomplish a delegation, a server acting as an intermediary assumes the client's identity when performing operations on the client's behalf. In other words, the intermediary *impersonates* the client.

We feel that impersonation is dangerous, and that most uses of impersonation are better modeled as true delegations with chained identities. However, we acknowledge that impersonation is necessary for compatibility with existing administrative setups and particular application sets; hence our delegation facility includes a means for clients to permit servers to impersonate them (and a means for a server to act as an impersonator). This facility is part of the client programming model and will be discussed in that section.

PROGRAMMING MODEL

The programming model decouples the manipulation of identity from the details of the security protocols. The interface is logically divided into a portion of interest to clients and a portion used by servers.

Clients are primarily concerned with establishing their identity and the necessary controls on how that identity is used. The model provides a *login context* as an abstraction of the client's identity. A login context is an application level opaque handle to the data, including the EPAC and tickets, needed by the underlying protocols. The security protocols are enabled in the RPC communication system by associating a particular login context with a communication session between a client and a server.

The details of how login contexts are shared by application processes are dependent on the operating system on which the application is running. For most DCE environments, however, a default login context for a given principal is created when a process is created for that user through the OS greeting function. Applications will inherit this default login context, but they are also free to create new contexts that reflect a different set of allowable privileges and/or controls. An application may also create new contexts for a different principal if the application has access to that principal's key.

Servers are the reference monitors for the data they manage. In general they are concerned with extracting the privilege attributes associated with a given remote request. These attributes are then generally passed on to the standard access control algorithm to determine if the client is authorized to perform the requested operation. The caller's privilege attributes may also be used for auditing the operation or otherwise recording information about the participants in the call chain.

Client Programming Model

A client must decide for each remote call that it makes whether it is performing the operation on its own behalf or on behalf of a caller. This ought to be obvious from program context. Additionally, a client that acts on behalf of a caller must decide on whether to chain its identity with that of its caller (i.e. be a delegate), or (try to) assume the identity of the caller (i.e. be an impersonator).

Operationally, the client must setup (or reuse) a login context that is appropriate for its role as either initiator, delegate, or impersonator, and perform the operation under that login context.

We provide three calls that setup login contexts: become_initiator, become_delegate, become_impersonater. All three calls allow the setting of delegation-related restrictions, extensible restrictions etc. Each creates a new login context as a return value. The calls differ only with regard to the identity information passed in. Become_initiator takes an existing login context as in input parameter. Become_delegate takes an existing login context (i.e. the delegates identity) plus a reference to the identity to chain with. Become_impersonater takes only the identity to impersonate, but needs no existing login context)

```
new_login_context = become_initiator (
        my_login_context,
        delegation_type_permitted,
        delegate_restrictions,
        target_restrictions,
        optional_restrictions,
        required_restrictions,
        permit_initiator_compat_mode,
        error_status );
```

The optional_restrictions and required_restrictions are lists of the respective additional restrictions to be applied to this call.

The *new_login_context* is an output argument that is the new login context that effectively refers to a new PTGT and EPAC that contains composite principal information plus the all the other relevant security attributes (e.g. delegation restrictions etc.)

The *delegation_type_permitted* parameter is an enumeration. The value *no delegation* means that this caller's identity may not be used in a call chain. With this value specified, the *delegate_restrictions* parameter is ignored. The value *delegate* means that the initiator allows its identity to be delegated but not impersonated. The *delegation_restrictions* do apply here. The value *impersonation* means that delegation or impersonation are permitted. It is up to the immediate target to choose which form of identity projection it wants (if any).

The *permit_initiator_compat_mode* comes into play if the initiator's identity is delegated in the call chain (as opposed to a chain of impersonations). The initiator may either permit or deny the use of initiator_compat_mode.

```
new_login_context = become_delegate (
        callers_identity,
        my_login_context,
        delegation_type_permitted,
        delegate_restrictions,
        target_restrictions,
        optional_restrictions,
        required_restrictions,
        compatibility_mode,
        error_status );
```

Note that when you use an existing login context in the *become_delegate* call only the base identity from the login context is used. The restrictions that were present in the login context are replaced by those explicitly passed as parameters.

The callers_identity argument refers to a caller's extended PAC. A value of NULL means 'use the caller information associated with this thread of execution'.

The compatibility_mode parameter is an enumeration with the following values: *initiator, direct_requester, initiator_if_possible, none .*

```
new_login_context =become_impersonater (
        callers_identity,
        delegation_type_permitted,
        delegate_restrictions,
        target_restrictions,
        optional_restrictions,
        required_restrictions,
        error_status );
```

Note that there is no compatibility mode argument. If the direct requester's identity was actually a chained identity, whatever compatibility mode was used there is retained.

Server Programming Model

The server-side API is extended to allow applications to extract the privilege attribute set for each participant in a chained identity. We include calls to extract the PAs of the initiator of the operation the PAs of each delegate the operation the delegation and extensible restrictions placed by each participant

COMPARISON OF MODEL AND MECHANISM WITH OTHER WORK

The model for delegation proposed here has been developed independently of, but bears a striking resemblance to, the model proposed by Gasser and McDermott [7]. In both models composition the privilege attributes for all principals involved in an operation is combined with extensions to the authorization model to allow the expression of the role of intermediaries in that operation. Significant differences exist in the details of the design given that the DCE uses shared secret key authentication and uses the Privilege Server [4] as a delegation server [14] while the Gasser and McDermott model uses public key authentication methods.

Other workers have concerned themselves with mechanisms for trustworthy transmission of delegated identities. Varadharajan et. al. [15] proposes a method for chaining certificates in a shared-secret key environment as well as a

mechanism for nesting delegation tokens in a public key environment. Karen Sollins [13] provides a mechanism for nesting shared-secret key delegation tokens. Both of these mechanisms for shared-secret key delegation tokens require target servers to contact the authentication service. This is inconsistent with the design goals of the DCE (as argued in [4]) which strive to reduce total system overhead by moving to a push model for privileges - thereby moving the collection of authorization data away from servers and to clients. Consequently we have developed our pushed token mechanism for nesting delegation information.

SUMMARY

We've presented a model and design for delegation in a distributed object-oriented environment. We were strongly motivated by the notions of protection of resources and protection of flow of identity:

We believe that each object in the system should have as much knowledge as is practically possible when making an access control decision in order to best protect its resources. Thus our delegation system permits a server to know all the participants in a chained call and to distinguish the rights granted to intermediaries from those granted to an operation's initiator.

We also believe that all principals in the system should have the ability to control the uses of their identities by objects that are not under their control. To that end, our system allows clients to place restrictions on the use of their identity in chained calls.

Our delegation system capitalizes on existing DCE trust mechanisms and is compatible with existing DCE applications. It is implementable with only modest changes to the DCE security system.

BIBLIOGRAPHY

1. Pato, J. "Extending the DCE Authorization Model to Support Practical Delegation," OSF DCE SIG RFC 3, June 1992.

2. _____, "OSF DCE 1.0 Application Development Guide," Open Software Foundation, Cambridge, MA., 1992.

3. _____, "OSF DCE 1.0 Introduction to DCE," Open Software Foundation, Cambridge, MA., 1992, Volume 2.

4. Pato, J., "DCE Authorization Services -- Privilege Server," OSF DCE Specifications, 1990.

5. _____, "Security in Open Systems: Security Frameworks for the Application Layer of Open Systems," ECMA TR/46.

6. _____, "Security in Open Systems: Data Elements and Service Definitions," ECMA-138.

7. Gasser, M. and E. McDermott, "An Architecture for Practical Delegation in a Distributed System," *Proceedings of the 1990 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1990.

8. Abadi M., M. Burrows, B. Lampson and G. Plotkin, "A Calculus for Access Control in Distributed Systems," Digital Equipment Corporation Systems Research Center report No. 70, February 1991.

9. Lampson, B., M. Abadi, M. Burrows, and E. Wobber, "Authentication in Distributed Systems: Theory and Practice," *Proceedings of the 13'th ACM Symposium on Operating System Principles*, October 1991.

10. Pato, J., "DCE Access Control Lists (ACL's)," OSF DCE Specifications, 1990.

11. Kohl, J. and B. C. Neuman, "The Kerberos Network Authentication Service," INTERNET-DRAFT RFC, revision 5, 17 April 1992.

12. Project SESAME (Bull, ICL and SNI), "Proposed Security Enhancements for DCE," OSF DCE SIG, 4 July 1992.

13. Sollins, K., "Cascaded Authentication," *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1988.

14. Neuman, B. C., "Proxy-Based Authorization and Accounting for Distributed Systems," Technical Report 91-02-01, Department of Computer Science and Engineering, University of Washington, 1991.

15. Varadharajan, V., P Allen, S. Black, "An Analysis of the Proxy Problem in Distributed Systems," *Proceedings of the 1991 IEEE Symposium on Security and Privacy*, IEEE Computer Society, 1991.

TRUFFLES — A SECURE SERVICE FOR WIDESPREAD FILE SHARING

Peter Reiher
Thomas Page, Jr.
Gerald Popek

UCLA
Los Angeles, CA

Jeff Cook

Trusted Information Systems
Los Angeles, CA

Stephen Crocker

Trusted Information Systems
Glenwood, MD

ABSTRACT

Truffles is a system meant to address some of the major issues that still make it difficult to share files between users at different sites. In particular, it addresses the problems associated with secure file sharing, and the problems of high administrative overhead. Truffles will combine facilities of the Ficus file system and TIS/PEM, a privacy enhanced mail system, to make file sharing considerably easier. Truffles must deal with several important security problems, including secure transport of data, authentication of the users sharing files, handling of different administrative domains, and permitting system administrators to control, flexibly, yet easily, what sorts of sharing are done. This paper describes these problems and the solutions Truffles will use.

INTRODUCTION

Users who share a single machine, or who share a single administrative domain over a local area network, are able to share files with each other very easily. Users can share source code for programs they are developing, work together on papers and other documents, and use common shared libraries and programs. Users who are not so closely connected physically currently cannot share files with the same ease. Although the complicated nature of the networks connecting such users used to be the primary impediment to sharing, today the lack of a common administrative domain and security concerns are the major unsolved problems.

Truffles (TRUsted Ficus FiLE System) is a system that attempts to make file sharing between users in different domains both simple and secure. The motivation for Truffles is the question: Why is it harder to share a file with an arbitrary remote user than it is to send him electronic mail? Sending electronic mail to another user is a relatively simple operation in today's networks. The sender need merely know the address of the receiver and have reasonable network connectivity. By invoking a single program, giving it only the receiver's address and the message, the sender can be reasonably certain that his mail will reach its destination, if possible. The sender need not worry about the path it takes, whether machines along that path fail, the hardware types involved, or any of the other complexities of the worldwide network of computers. It is really no harder than sending electronic mail to a user on the same machine, despite the fact that the operations that have to be performed to deliver the mail are much more complicated.

Sharing files between users on different machines, on the other hand, is not nearly as easy as sharing them with a local user. One can use electronic mail to ship text files back and forth, but many electronic mail systems do not handle non-text files well, propagating changes in the files must be done by hand, electronic mail will not coordinate access to the files, and, unless a secure form of mail is used, electronic mail offers little protection.

Some tools more specialized for file handling exist, such as ftp, telnet, and NFS [1]. Generally, however, these tools have significant disadvantages. None of them handle security issues well. Telnet and ftp also do not use the same user interface that normal local operations use, and do not easily permit users to mix local and remote operations. NFS and other network file system services do a better job of unifying the local and remote cases, but they require substantial setup service by the participating sites' system administrators, and they generally have poor availability in the face of failures. Further, since all operations have to

fetch their data across the network, these systems can be slow and/or expensive.

The Ficus file system has solved the problems of poor availability and performance by replicating files for users [2]. Since users at different sites have local versions of their shared files, they are not as affected by failures, and they get substantially the performance of local access. But Ficus only works well in shared administrative domains, and does nothing to address the security concerns.

Truffles is an attempt to solve the problem of permitting controlled file sharing between users in different administrative domains. Only those users and sites that are permitted to participate should be able to do so. Those who are not permitted to participate should not be able to eavesdrop on data belonging to the relationship, nor should their requests for data or updates be honored. Another important security aspect of the problem is that the sharing should be limited strictly to what the participants intend to share. A participant who only meant to share a few files should not be forced to grant access to his entire machine.

A file sharing service of this kind cannot be successful if it relies on constant system administrator intervention. Just as users do not typically ask their administrator if they can change the access permissions on their files to make them available to other local users, they should not necessarily have to consult their administrator before sharing them with remote users. If setting up every sharing relationship with an outside user requires positive action on the part of the system administrators of the machines involved, few relationships will be set up. On the other hand, system administrators must be able to exert some form of control on what their users can make available to the outside world.

Finally, Truffles must make the sharing easy and painless, both in the initialization phase and during ongoing operations. The operations to establish the relationships should be simple and straightforward. Truffles should require little or no human intervention to keep a sharing relationship going. Only when the relationship is being changed, such as adding or dropping participants, should any of the users be reminded that the shared files are any different than any of their other files. Even then, the reminder should be painless and easy to deal with.

This paper describes the Truffles approach to solving this problem, with particular emphasis on the security issues. The next section describes the general approach. The following two sections describe two major software components that will be used in building Truffles. The section after those describes the overall Truffles architecture, particularly details touching on security. The following section briefly surveys some related work. The final section describes the current state of Truffles, its future, and gives some conclusions.

THE TRUFFLES APPROACH

The Truffles approach to this file sharing problem is to provide a secure file sharing service usable over normal network connections. Setup will be done through an electronic mail interface. File sharing has normal Unix file semantics, once the relationship has been established. Because the connections between sites using Truffles may have high delay, and the networks or sites may fail, Truffles automatically supports keeping multiple copies of a file on different machines.

Truffles is meant to run in a UNIX environment, on sites running standard UNIX operating systems with minor modifications and some additional software. The security goals of the Truffles system are not to improve the existing security features of UNIX systems, but to extend the existing level of UNIX security to files that are shared across administrative boundaries over insecure networks.

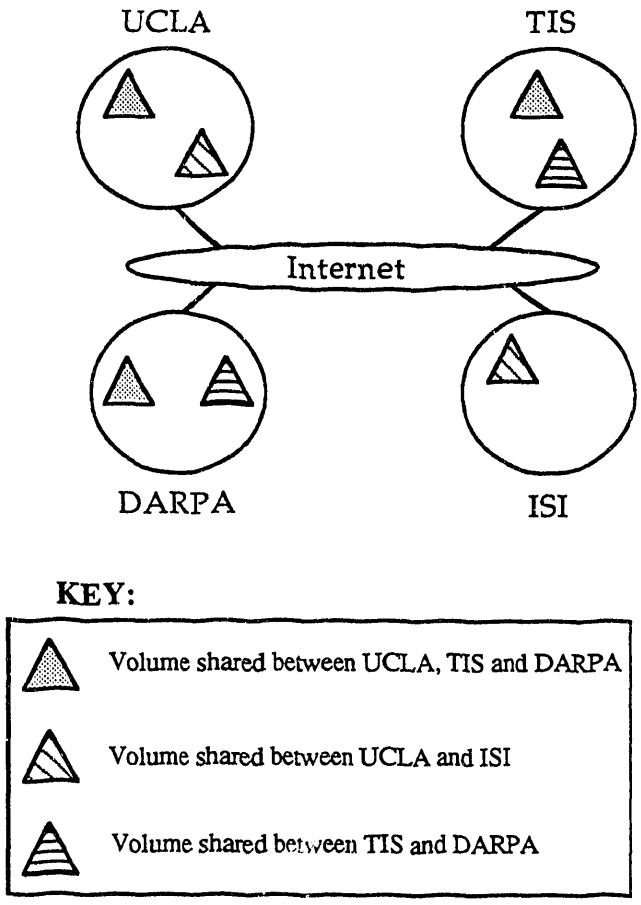


Figure 1. Sites sharing volumes via Truffles

Truffles file sharing

Truffles can provide file sharing either through replication or transparent remote access. Truffles provides file replication on a per-volume basis, rather than a per-file basis. A *volume* is similar to the concept of a Unix file system. It consists of a connected tree-like structure of directories and files, all stored on a single physical device. Any files that are to be replicated must be collected into a volume or set of volumes. Any files in those volumes will be shared, with normal Unix permission controls dealing with access to them. Truffles permits multiple sites to store replicas of a volume, and other sites to participate without storing a replica (at the cost of some performance degradation and inferior availability when sites fail).

Figure 1 shows how several sites might share Truffles volumes among themselves. In this example, four sites (UCLA, TIS, DARPA, and ISI) share three volumes (represented as triangles) among themselves, using the Internet to provide transport services. UCLA shares a volume with DARPA and TIS, TIS and DARPA share another volume, and UCLA and ISI share a volume. Different users might have set up and used each relationship. Each relationship is separate from the others, and does not depend upon them. Depending on how permissions are set and name spaces are organized, users at the four sites might or might not be able to access files whose volumes are not locally replicated.

Figure 2 shows another view of how Truffles volume sharing works. This figure shows the top levels of the file hierarchies on two sites, UCLA and TIS. Part of each site's file hierarchy contains files that are stored with normal UNIX file systems, such as the files under `/usr` and `/etc`. Another part of each site's hierarchy contains Truffles files. Only files in the Truffles parts of the hierarchies can be shared. Files in the non-Truffles parts of the hierarchies are completely inaccessible via Truffles. This design has an important security implication — any files not stored in the Truffles part of a site's namespace are completely shut off from sharing via Truffles, and are inaccessible to remote sites using Truffles.

Within the Truffles portion of the file hierarchy shown in figure 2, files are organized into volumes. The triangular shaded areas of the hierarchy show volume delimitations. The two sites do not necessarily share a common namespace. In this example, the root of the UCLA Truffles file system is called `/global`, while the root of the TIS Truffles file system is called `/g`. Some portions of the namespace are shared, though. The Truffles volume rooted at `/global/us/edu/ucla/reiher/shared` in the UCLA hierarchy has a replica at `/g/us/com/tis/cook/shared`. Despite the two replicas being stored at different places in the hierarchies, Truffles will keep all files in the two replicas consistent. Alternately, sites are permitted to completely share identical Truffles namespaces.

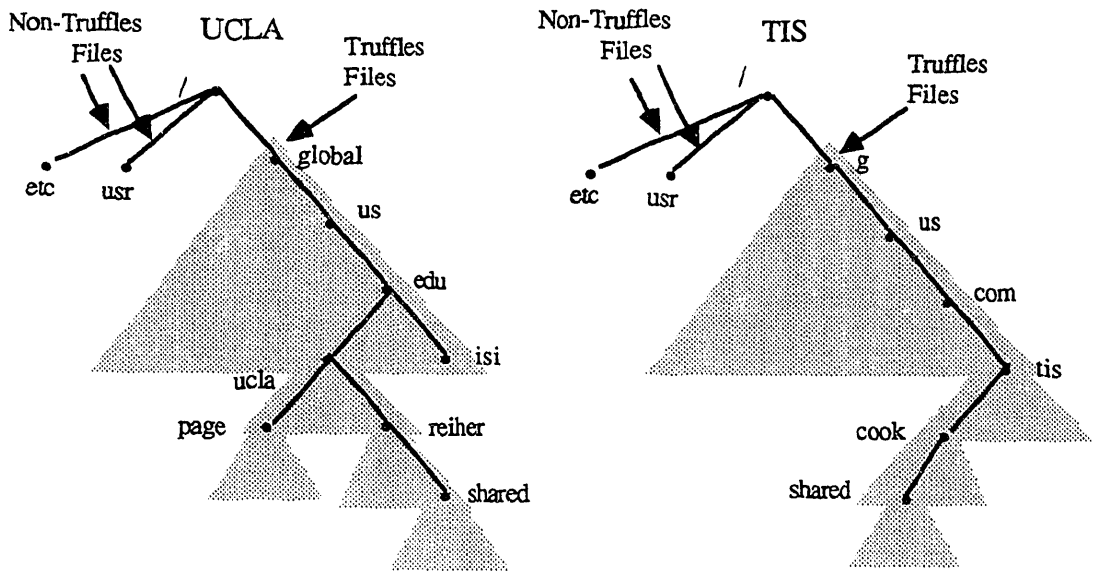


Figure 2. File hierarchies using Truffles

If another machine at TIS stored all of the TIS-local Truffles namespace up to, but not including, the volume shared between TIS and UCLA, users on that machine would be able to use Truffles to remotely access the shared volume, to the extent that normal access permission mechanisms allowed. Truffles provides not only replication services, but transparent remote access. However, the portion of the namespace not shared between TIS and UCLA is not accessible to remote sites through Truffles. For example, users at TIS are not able to see the volume `/global/us/edu/ucla/page`, and cannot use Truffles to get at that volume in any way, short of setting up an explicit relationship to share that volume. In fact, if the single shared volume shown in the diagram were the only volume shared between TIS and UCLA, only those files would be jointly accessible by both sites. UCLA could not use Truffles to examine any other TIS files, and TIS could not use Truffles to examine any other UCLA files.

Constructing Truffles

Truffles is being built from two existing pieces of software. The Ficus file system will provide file sharing and replication. The TIS implementation of Privacy Enhanced Mail (TIS/PEM) will provide a secure channel for the setup traffic and distributes the keys used for authentication and encryption. The full Truffles system will require a merging of these two components, with minor modifications, plus a reasonable amount of additional software. The components not directly provided by Ficus or TIS/PEM include

- the protocol for setting up a relationship
- daemons to handle most of the setup work without user intervention
- secure transport of data in an established relationship
- handling user identifiers between different administrative domains
- mechanisms and policies to control file sharing

Broadly, the Truffles approach is to use TIS/PEM, which can both authenticate and encrypt electronic mail, to send the messages between users to determine that a relationship is desired, and to authenticate each other. The use of electronic mail to establish the connection has certain advantages over other alternatives. The cooperating users need only know each other's electronic mail addresses. There is no need to request direct intervention of a system administrator to set up the connection via some other mechanism, like NFS. Electronic mail is able to handle issues like temporary failure of the destination site gracefully.

TIS/PEM is also used to establish what encryption keys will be used for this sharing relationship. Truffles daemons then take over the rest of the protocol to set up a shared set of files. This protocol consists of trading electronic mail messages between daemons that run Ficus utility programs in response to the messages. Since these utility programs permit remote users to gain access to local files, all messages in the protocol must be authenticated.

Once the relationship is established, all users involved in it will see the volume at the appropriate places in their file hierarchies. Ficus will ensure that all updates are seen by all replicas. Ficus will also deal with any problems arising from failures, recoveries, and partitions. From the users' points of view, the situation is little different than if they shared the files with each other on a single machine.

Truffles design problems

Truffles must deal with some other problems related to establishing and maintaining the sharing relationship. First, there are policy questions concerning which users can share which files with which sites. The answers to such questions are likely to vary greatly from site to site, depending on the importance of the data, the trust in the users, and the caution of the system administrator, among other factors. Therefore, Truffles will provide a mechanism for validating each request to set up a file sharing relationship, but will keep the policy well separated from the mechanism. Depending on circumstances, relationships might be permitted without any checking, or only between approved sites and users, or only for certain volumes, or only if the system administrator has previously granted permission, or only if the system administrator actually reviews and approves the request. The mechanism will be sufficiently flexible to permit these, and many other, policies to govern sharing. Policy will be under the control of local system administrators.

Another problem is that different sites have different low level names for their users. In the Unix world, each user on a site has a unique identification number, called a UID. But this UID is unique only to that site, or to that site and others sharing the same name assignments. In general, this UID might be used for a different user on different sites. But Truffles must not permit one of those other users to be mistaken for the local user, simply because they share a UID. Truffles must be careful to map remote users' UIDs without mistakes.

Truffles must also provide secure transport of the data. Ficus itself does not depend on secure data transport, as it was originally built to run on a trusted network. In the world of the Internet, however, Truffles messages could easily be read by eavesdroppers, or improper messages could be injected into the network. Truffles must protect its users from these dangers.

Another security concern is that Truffles must ensure that only data that is explicitly shared be made available to other sites. The non-Truffles portion of each site's file hierarchy must be unavailable to remote sites, and the Truffles portion of the file hierarchy must limit sharing to only those volumes that were meant to be shared.

of a certification hierarchy in the form of a tree, where each node is certified by a node above it, and the leaves of the tree are users, mailing lists, etc. The Internet Policy Registration Authority (IPRA) is at the highest level of this hierarchy. This authority will be managed by the Internet Society.

TIS/PEM is a reference implementation of the PEM standard, developed by Trusted Information Systems [7]. It is UNIX based, and runs on a variety of platforms. Figure 4 shows a view of TIS/PEM. The PEM library serves as the primary entry point to the system by electronic mail or other services. That library, certain PEM utilities, and key management programs communicate with the local key manager (LKM), which handles key management, independent of the particular application requesting its services. The LKM maintains a local database for certificates and private keys, enforces access control, and provides cryptographic services employing private keys. One of the private libraries attached to TIS/PEM is the crypto library, which has an algorithm independent interface, and handles key generation, message digest computation, encryption and decryption, and signature computation and verification for a variety of encryption schemes.

TIS/PEM's role in Truffles is to provide secure electronic mail services, which will be used to perform the setup of file sharing relationships. Also, TIS/PEM's key manager services are used to handle encryption keys related to the secure transport of data between sites sharing files through Truffles.

TIS/PEM is currently in use at a variety of sites, including three TIS sites spanning the country, UCLA, and others.

THE FICUS FILE SYSTEM

Ficus is a distributed file system designed to run on networks of Unix systems, ranging from portable units and workstations to large file servers [2]. Ficus provides high availability for read and update, utilizing an optimistic "one copy availability" policy. "One copy availability" permits access to a file even if a majority, quorum, or token are unavailable, as long as a single copy can be accessed. This policy maximizes availability, at the cost of permitting copies of a file to become conflicted, when different copies are updated simultaneously while not in communication. Ficus handles such conflicts by reliably detecting them. Many conflicts are automatically resolved by Ficus.

Ficus supports very high availability for both read and write, allowing uncoordinated updates when at least one replica of the file is available. No-lost-update semantics are guaranteed. Asynchronous update propagation is provided to accessible copies on a "best efforts" basis, but is not relied upon for correct operations. Rather, periodic reconciliation ensures that, over time, all replicas converge to a common state. This policy is more appropriate than serializability for the scale and failure modes of a very large distributed system.

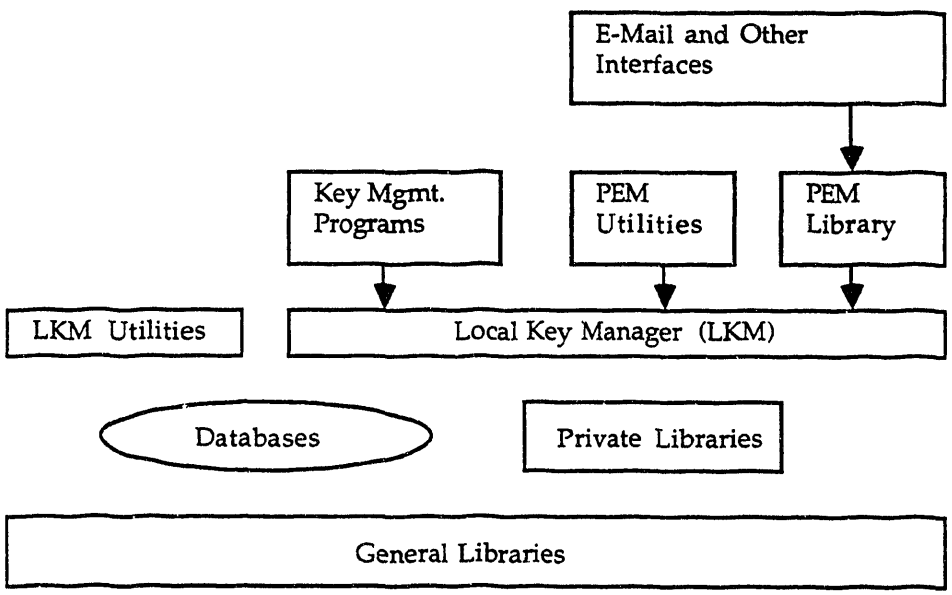


Figure 4. TIS/PEM architecture

of a certification hierarchy in the form of a tree, where each node is certified by a node above it, and the leaves of the tree are users, mailing lists, etc. The Internet Policy Registration Authority (IPRA) is at the highest level of this hierarchy. This authority will be managed by the Internet Society.

TIS/PEM is a reference implementation of the PEM standard, developed by Trusted Information Systems [7]. It is UNIX based, and runs on a variety of platforms. Figure 4 shows a view of TIS/PEM. The PEM library serves as the primary entry point to the system by electronic mail or other services. That library, certain PEM utilities, and key management programs communicate with the local key manager (LKM), which handles key management, independent of the particular application requesting its services. The LKM maintains a local database for certificates and private keys, enforces access control, and provides cryptographic services employing private keys. One of the private libraries attached to TIS/PEM is the crypto library, which has an algorithm independent interface, and handles key generation, message digest computation, encryption and decryption, and signature computation and verification for a variety of encryption schemes.

TIS/PEM's role in Truffles is to provide secure electronic mail services, which will be used to perform the setup of file sharing relationships. Also, TIS/PEM's key manager services are used to handle encryption keys related to the secure transport of data between sites sharing files through Truffles.

TIS/PEM is currently in use at a variety of sites, including three TIS sites spanning the country, UCLA, and others.

THE FICUS FILE SYSTEM

Ficus is a distributed file system designed to run on networks of Unix systems, ranging from portable units and workstations to large file servers [2]. Ficus provides high availability for read and update, utilizing an optimistic "one copy availability" policy. "One copy availability" permits access to a file even if a majority, quorum, or token are unavailable, as long as a single copy can be accessed. This policy maximizes availability, at the cost of permitting copies of a file to become conflicted, when different copies are updated simultaneously while not in communication. Ficus handles such conflicts by reliably detecting them. Many conflicts are automatically resolved by Ficus.

Ficus supports very high availability for both read and write, allowing uncoordinated updates when at least one replica of the file is available. No-lost-update semantics are guaranteed. Asynchronous update propagation is provided to accessible copies on a "best efforts" basis, but is not relied upon for correct operations. Rather, periodic reconciliation ensures that, over time, all replicas converge to a common state. This policy is more appropriate than serializability for the scale and failure modes of a very large distributed system.

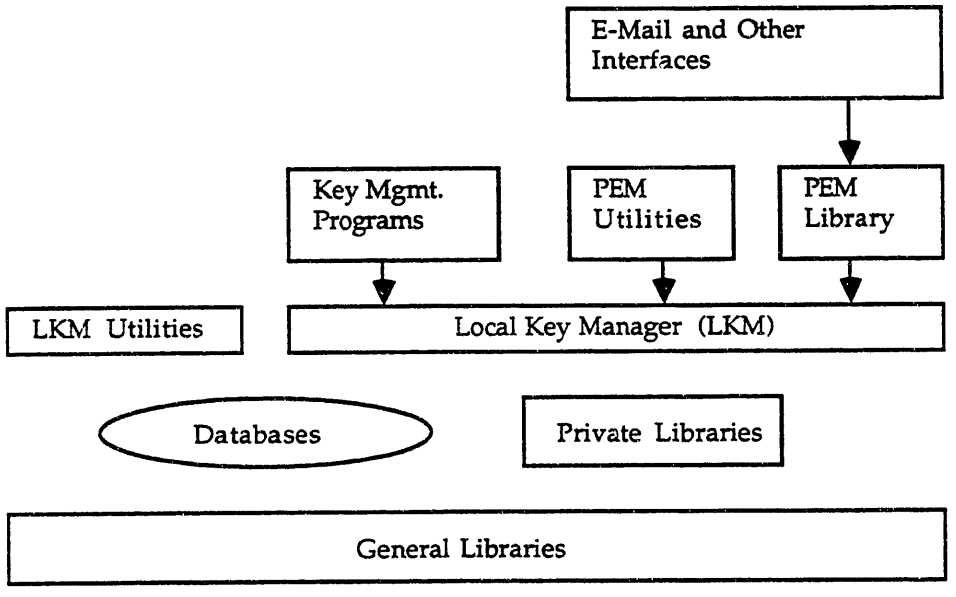


Figure 4. TIS/PEM architecture

Both because of the asynchronous update strategy and the one copy availability policy, different replicas of Ficus files can become in conflict. Conflicts occur when two or more replicas all receive updates without successfully propagating their updates to the other replicas. Conflicts are reliably detected and directory update conflicts automatically reconciled. Many other types of file conflicts are also automatically reconciled. Those conflicts that cannot be resolved automatically are brought to the attention of the owning user for resolution. Ficus provides tools for users to reconcile such conflicts by hand. Experience with Ficus has shown that conflicts are relatively rare events, and are generally easy for users to reconcile.

Ficus is built to run in a single administrative domain. It assumes that all sites and the connecting network are all trusted, so no special security is necessary. Moreover, it is not prepared to deal with sites that have different sets of users with conflicting user identifiers. With some effort, Ficus can work in this environment, but it requires substantial work by the various installations' system administrators, and is less than perfect in many other ways.

Ficus and stackable file systems

The replication service of Ficus is packaged so that it may be inserted above the base Unix filesystem on any machine running a stackable file system interface. This modular architecture permits replication to co-exist with other independently implemented extended filing features.

In addition to running on top of stackable file systems, Ficus is built using stackable layers [8]. The stackable layers approach to file system design permits adding functionality to an existing file system merely by writing the new functionality into a new layer of code. This code is placed on top of the existing layers, providing a compatible interface to users, while simultaneously making the new functionality available. The stackable layers approach does not require any changes to the existing code, so adding functionality is relatively easy.

Ficus itself consists of two layers that sit on top of the Unix file system (UFS) and the network file system (NFS). The Ficus Physical layer supports operations that deal with a single replica of a file. The Ficus Logical Layer supports operations that deal with all replicas of a file. The UFS provides actual storage of data on disk, and NFS is used as a transport layer to move Ficus requests from one site to another. Figure 5 shows a typical stack of Ficus layers on two sites.

A great advantage of the stackable layers technology is that other filing services can be used in conjunction with Ficus, merely by inserting another layer into the appropriate place. Encryption and compression of files are two examples of services that could be combined with Ficus via layers.

Ficus was built at UCLA, and is in daily use there, as the system on which further Ficus development work is done. Ficus has also been installed at several other sites, including TIS and ISI.

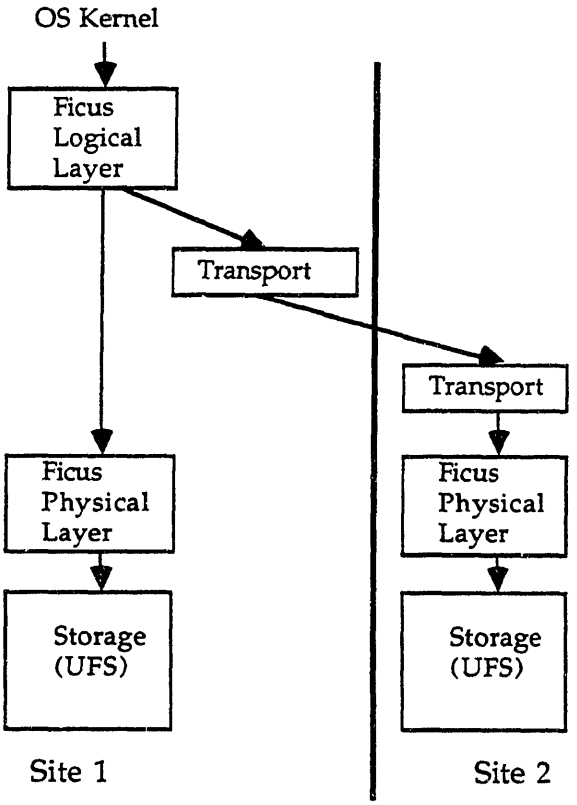


Figure 5. A Ficus stack

HANDLING SECURITY PROBLEMS

Truffles faces several security problems:

- Security of the setup procedures
- Security of the ongoing sharing relationship
- Proper handling of different administrative domains
- Enforcement of policies governing sharing

Secure setup of Truffles sharing relationships

There are several security problems relating to setting up sharing relationships. The users participating in the relationship want to be sure that the others are who they claim to be. Otherwise, an intruder could masquerade as another user to gain improper access to files he otherwise wouldn't be permitted to share. The protocol's various messages must each be authenticated, both to ensure that an intruder is not inserting protocol messages for his own purpose, and to ensure that the participants are not improperly trying to gain access to more than the files that were agreed upon.

When two users decide to share a file volume via Truffles, they must first authenticate their identities. They will do so with electronic mail messages sent under TIS/PEM containing their certificates. Each user will check the identity of the other by authenticating the certificate. The user who already has access to the volume (the originator) will then invoke a program to start up the protocol. This program will accept information about which volume is to be shared with which user on which site, to ensure that not only are protocol messages coming in from the proper sender, but that they refer to the proper volume.

The protocol could take any of several forms. One possible protocol would request a session key from TIS/PEM's key management facilities. This session key will be used to authenticate messages in this volume's setup protocol. The Truffles program will then encrypts this session key using the new user's public key, obtained from TIS/PEM's key management facility and send the result to a daemon at the new user's site.

The daemon at the new user's site will receive the message and start the process of creating the volume replica. After extracting the session key and authenticating the message, it will invoke a Ficus program that creates a volume replica. After this Ficus program runs, the replica is empty, and is not logically connected to the original replica, so further work is necessary. The Truffles daemon will get the replica ID of the new replica and send it to the originating site's Truffles daemon, appropriately authenticated.

The originating site's Truffles daemon will check the authentication, and check to see that the request concerns a volume that was supposed to be shared. It will then run a Truffles program to tell the originating site's replica of the existence of a new replica. Once this program completes, the originating site's daemon will send another message to the new site's Truffles daemon.

After checking authentication on this second message, the new site will run a Truffles reconciliation program that pulls the contents of the volume's files from the original replica to the new replica. This Truffles program will also add a line to a Ficus system file to ensure that this site will periodically run the reconciliation program on the new volume replica, to ensure that the replica remains up to date.

Later, other sites may join the relationship. The process of adding them will be substantially the same as adding the first site. A single existing member of the relationship will exchange messages with the new site using a protocol similar to that described above. The standard Ficus update propagation and reconciliation mechanisms will ensure that all other sites participating in the relationship are quickly informed of the presence of the new site.

The protocol to establish the relationship will probably use electronic mail and TIS/PEM for all its messages. TIS/PEM provides authentication and encryption, making it unnecessary to introduce another secure transport mechanism for these purposes. The ongoing activities of sites sharing files via Truffles will be handled largely by another secure transport mechanism, built into Ficus, since these activities are not suitable for electronic mail. They will continue to use TIS/PEM's key management facilities, however.

Security of the ongoing relationship

Once a Truffles relationship has been set up, the major security concern is that only legitimate participants be permitted to read or write data. The two sites involved in the relationship can use standard Unix and Ficus access control mechanisms to ensure that only the local users who have proper permissions can gain access to this data. But, since the data handled by Truffles might pass over an insecure medium, more is necessary.

Encryption of the data that Truffles sends from site to site is all that is really required. The sites participating in a sharing relationship must encrypt any Ficus requests they send to each other, using a key particular to the relationship. This key will be generated and distributed during the creation of the relationship. A separate key is used for each relationship to ensure that sites participating in one relationship with other sites cannot improperly eavesdrop on conversations about volumes they do not share.

Encryption between the sites in a relationship is done with DES. A unique initialization vector is attached to each message and encrypted, ensuring that two otherwise identical messages do not encrypt to the same value.

Optionally, for situations in which privacy is not a concern, but authentication is, Truffles allows a volume's traffic to be merely authenticated, rather than encrypted, thereby saving the cost of encryption.

Handling different administrative domains

In a Unix system, users are known by several names, two of which are of importance to Truffles. For login purposes, they are identified by a character string name. For purposes of saving file and process ownership information, they are assigned an integer identifier, commonly called the user identifier, or UID. The system maps between the character string login name and the numerical UID whenever necessary, using information stored in the password file or the NIS.

The mapping of login names to UIDs is only unique to a given administrative domain, which is made up of one or more machines closely connected together. Generally,

machines that wish to set up Truffles relationships are not in the same administrative domain. Different users might have the same login name and/or UID in the two domains. Since file ownership and access requests are tagged with the UID, and access permission is checked using the UID, there is a security risk in this situation. Unless Truffles can handle this problem, a user on one site might improperly be given access to files on another site simply because another user on that site has the same UID. The situation is intolerable, even when security isn't a concern, as the potential for user confusion is considerable. There is a similar problem when the system maps from a UID to a login name, as it does when a user wants to display the ownership of a file.

This problem has been recognized before in other distributed file services, such as RFS [9]. Their solution was to map UIDs from remote machines to UIDs on local machines. This method worked reasonably well in RFS, since an RFS file was stored on a single machine, and the ownership information for that file could be stored as the local version of the UID for the owning user on that machine. In a replicated file system like Truffles, replicas of the file might be stored at different sites with different UIDs for the owning user. This situation causes a certain complexity in replication control, as the ownership information from one replica must not be propagated to another under normal circumstances, yet must be propagated upon change of ownership. Also, this situation makes it difficult to move the physical storage for a replica from one site to another, since the UIDs associated with the replica's files might be map to different UIDs on the new site.

Truffles will handle this problem by mapping a user's UID to a globally unique identifier. Truffles will save file ownership information using this globally unique identifier, storing it as one of the file's attributes. Truffles will map from the UID to the globally unique identifier whenever a user process tries to access a file. The requesting user's globally unique identifier can then be compared to the file owner's identifier to determine if access should be granted. When the system needs to perform the reverse mapping, to display the ownership of a file, Truffles will map from globally unique identifier to login name. Since the file has a single globally unique owner at all replicas, it will be easy to handle update propagation, and the physical storage can be easily moved from one site to another without losing ownership information.

Many forms of globally unique identifiers could be used for this purpose. Truffles will initially use X.500 *distinguished names* (dnames). When a user makes a request for a file stored under Truffles, his local UID will be mapped to a dname, which is used to determine whether he can access the file. Should the request require remote access at the other site, the dname must be passed with the request across the net. On the opposite end, Truffles will compare the dname to the owner of the file in question, in the same manner as the local case.

This mapping of UIDs to dnames and access permission checking will be done via a Truffles file system layer that sits above the Ficus logical layer. All requests for Truffles volumes will go through this layer, and access permissions will be checked before the request is submitted to the lower layers of the file system. Those lower layers will never reject a request that has been approved by the Truffles layer, since access checking has already been done. Because Ficus has a layered file system available, no existing system or application code will need to be changed to make access checking via dnames work. All the new code will exist in a self contained layer and associated new utility programs.

Certain UIDs in Unix systems have special meanings, especially the root user ID. The root user on a system is permitted to perform many operations that could have disastrous consequences if done improperly. Also, the root user can effectively gain access to any files on the site. Generally, no remote user should ever be permitted to map to the root user on a site through Truffles mechanisms. In particular, the root user on site A should not be able to use a Truffles sharing relationship to gain root privileges on site B. Truffles will not permit any remote dname to be mapped to the local root user. Attempts to do so, even by privileged users, will be rejected by the Truffles software.

An analogous problem exists with group access permissions. Unix systems permit users to belong to several groups, and group membership can also allow access to files. Like UIDs, group IDs (GIDs) are numerical, and are not coordinated between different administrative domains, so a given GID in one domain might refer to an entirely different group in another domain. The Truffles solution to this problem will be similar to its solution for UIDs. Users will be permitted to establish groups in one domain, and users in other domains can map that foreign group to a local one. A common case is expected to be two newly cooperating users setting up a special group strictly to permit them to jointly access their shared files, while using Unix access control to lock others out. Truffles will include tools that make this common case simple to set up.

Truffles has not, as yet, dealt substantially with the issue of revocation of access. In the simplest, probably most common, case, a temporary sharing relationship will come to an end, and must be cleanly torn down. Less frequently, users who are currently able to use files in a sharing relationship must be prevented from doing so in the future. The former case will be adequately dealt with by standard Truffles methods of destroying volumes. The latter case is a subject of further study for Truffles.

Enforcement of policies governing Truffles sharing

The purpose of Truffles is to permit users at different sites to share files with each other without undue burden on system administrators. However, system administrators still need to exercise some control over which of their sites' files can be shared, by whom, with whom. Certain files may be sufficiently sensitive that no one should be permitted to share them outside the site. Others might only be sharable by certain trusted users, with other local users unable to set up sharing relationships on them. Yet others might be sufficiently insensitive that any user who can access them should be permitted to share them.

Generally, the sharing policies that different system administrators may want to enforce could be quite varied, ranging from freely permitting any sharing relationships to requiring explicit permission for all relationships. Truffles must be able to support the whole spectrum of possible policy decisions.

Truffles will support this spectrum of policies by separating the policy from the mechanism. When a user wants to share a volume with another site via Truffles, he will invoke a command to start up the relationship, as described in section 5.1. This command will consult a system file to determine which policy module should be run to determine if the system will permit the relationship to go forward. This system file will be set up by the system administrator, and will indicate what program should be run to determine whether to let the user proceed. Truffles will be distributed with a small set of programs for this purpose, and with instructions on how to write other programs for policies the distribution set does not support. The various programs will require different input information, including user identities, volume identifier, identity of the new site, and possibly certificates of various sorts.

The new user's site must also validate the sharing request, and will do so in a similar manner.

Part of the ongoing Truffles research is to examine what sharing policies are most important, and the best way of implementing them.

RELATED WORK

Truffles is basically a system for sharing files across machine boundaries. The primary related work is other file services with the same goal. One obvious effort is NFS [10]. In fact, the early version of Truffles uses a modified version of NFS as a transport layer. However, NFS has certain limitations that Truffles does not have. Setting up an NFS relationship is a heavyweight operation, requiring substantial system administrator intervention on both sides. Also, NFS currently provides little security (though its security will be improved in the near future). In its original version, NFS did not provide any form of replication service. A subsequent version has provided a form of replication through automounting, but this replication method does not automatically propagate updates [11], making it more suitable for read-only files (like manual pages) than more general file usage. NFS lacks a protocol for automatically setting up the sharing relationship, as well.

Other related systems include the Andrew File System and RFS. The Andrew File System [12] is meant to work in a rather different environment than Truffles. The Andrew File System consists of a distributed collection of servers (known as Vice) servicing a much larger numbers of workstations, each of which runs software known as Virtue. The files are stored permanently by the Vice servers, with extensive caching done by the Virtue clients. The Andrew File System authenticates a workstation and the Vice servers to each other, when they first communicate. Subsequent communications can be encrypted or merely authenticated. Since local copies of the file are cached only, the issue of replication at the client sites does not arise. A given Andrew installation uses a global name space for its users' identifiers, thus avoiding the problem of mapping disjoint identifier spaces. Only workstations that are members of the Andrew File System installation can share files. Thus, the Andrew File System cannot be used to assist arbitrary users at arbitrary sites to share files.

RFS offers a similar service to NFS, with the primary difference being that RFS maintains state for file operations, while NFS does not [9]. RFS has the same general set of limitations as does NFS, for the purpose of solving the Truffles problem. Setting up RFS remote mounts is an administratively heavyweight operation, RFS does not support replication, and RFS does not include a protocol to set up the relationship. RFS has addressed some security concerns that NFS does not, including allowing only specified users to mount file systems, and mapping user and group IDs from other administrative domains.

The Locus Operating System supported replicated files with automatic update and recovery mechanisms [13]. However, Locus ran in a single administrative domain, with all sites in close cooperation. While possible for a relatively small set of machines, this solution cannot apply to the broader case of sharing files with arbitrary users at other sites. Also, since Locus typically ran within a local area network, rather than across long haul lines, and since a single administrative authority controlled the entire system, the security issues that Truffles deals with were not considerations in the Locus system.

Kerberos [14] offers an authentication service that has some overlap with TIS/PEM. Kerberos is specifically designed to authenticate various entities to each other securely. In a Kerberos system, a Kerberos server stores a database of authentication information. Each entity that can be

authenticated (referred to as a principal) has a secret key known only to itself and the Kerberos server. Principals authenticate each other through the Kerberos server, which then assigns them a session key to use for encryption during that session. Kerberos names entities using a combination of a primary name, an instance, and a realm; for example, name.instance@realm. Kerberos currently uses its own form of names for principals, rather than X.500 distinguished names. Also, Kerberos itself does not provide for the secure transmission of electronic mail, though its services clearly could be used in a secure mail system. The secure connections provided by Kerberos could also be used to perform setup of Truffles volumes. Kerberos only provides services for authentication. It is not a file sharing or replication facility.

Project Athena makes use of Kerberos as part of its distributed services [15]. Unlike Kerberos itself, Athena is a distributed filing service. Athena uses a workstation/server model for its system, unlike Truffles, in which all sites are viewed as peers. Athena workstations are regarded as dataless nodes, which use their local hard disks to cache data to reduce network traffic. Users log into workstations, are authenticated via Kerberos, and get access to their files through file servers. Athena uses NFS to make remote files available to users. Replication is only supported for read-only system and library software. Athena is intended for use in a single (though possibly very large) administrative environment, like a university. It is not meant to support the more general sharing patterns Truffles supports.

CONCLUSIONS

The resources of the Internet are not well coordinated, largely because of the difficulty of cooperation between its users. The only relatively simple, robust service available is electronic mail. If the Internet, and all other world computer networks, are to reach their full potential, much work is necessary to make cooperation over the networks trivially easy for the average user.

Truffles is intended as a significant step in that direction. It makes one of the primary resources of the network's sites, their file systems, quite easy to share. Users will be able to share files with little more effort than is necessary to send each other electronic mail. The optimistic replication service offered by Truffles' Ficus component will give them high availability, excellent performance, automatic update propagation, and automatic recovery of many conflicts arising from the network environment.

A service like Truffles will never achieve any real popularity, however, unless it is secure. The world network is a dangerous place, and users are rightfully wary about putting too much trust in other sites or the interconnection media. Truffles will use TIS/PEM to reduce the number of entities that must be trusted to a reasonable number — just the certification authorities and the participating sites. By using encryption of data passed via Truffles, privacy of the ongoing relationship will be preserved.

The schedule for building Truffles is relatively short. The plan is dependent almost entirely on reuse of major software components, Ficus and TIS/PEM. While these two system seem entirely unrelated, they can be used together to provide most services required by Truffles. Some time and effort will be spent, of course, in customizing the components for the Truffles environment, and in merging them, but the amount of time saved by not having to re-implement their services is expected far outweigh any time lost merging them. One important result of the Truffles research will be to demonstrate how judicious reuse of existing software components can speed research.

Truffles is currently in an intermediate state of development. Much of the necessary software has been written, and much of the unwritten software has been designed. The major incomplete components are the daemons necessary to run the protocol and the layer of software to perform UID mapping. The basic system should be complete by late in 1993. Further research into the use of Truffles to facilitate sharing between remote users will continue, from that point.

ACKNOWLEDGEMENTS

This work is being performed under DARPA contract number N00174-92-C-0128, under the supervision of Brian Boesch.

REFERENCES

1. Satyanarayanan, M. "A Survey of Distributed File Systems", Annual Review of Computer Science, 1990.

2. Guy, R., Heidemann, J., Mak, W., Page, T., Popek, G., and Rothmeier, D., "Implementation of the Ficus Replicated File System", Proceedings of the Summer USENIX Conference, 1990.

3. Linn, J. "Privacy Enhancement for Internet Electronic Mail: Part I — Message Encryption and Authentication", DEC Technical Report, 1992.

4. Kent, S. "Privacy Enhancement for Internet Electronic Mail: Part II— Certificate-Based Key Management", BBN Communications Technical Report, 1992.

5. Balenson, D. "Privacy Enhancement for Internet Electronic Mail: Part III — Algorithms, Modes, and Identifiers", Trusted Information Systems Technical Report, 1992.

6. Kaliski, B. "Privacy Enhancement for Internet Electronic Mail: Part IV — Key Certification and Related Services", RSA Data Security, Inc. Technical Report, 1992.

7. Galvin, J. and Balenson, D. "Security Aspects of a UNIX PEM Implemenatation", Proceedings of the UNIX Security Symposium III, 1992.

8. Page, T., Popek, G., and Guy, R., "Stackable Layers: An Object-Oriented Approach to Distributed File System Architecture", IEEE Workshop on Object Orientation in Operating Systems, 1990.

9. Rifkin, A., Forbes, M., Hamilton, R., Sabrio, M., Suryakanta, S. and Yueh, K. "RFS Architectural Overview", Proceedings of the Summer USENIX Conference, 1986.

10. Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., and Lyon, B., "Design and Implementation of the Sun Network Filesystem", Usenix Conference Proceedings, Summer 1985.

11. Callaghan, B. and Lyon, T. "The Automounter", Proceedings of the Winter Usenix Conference, 1989.

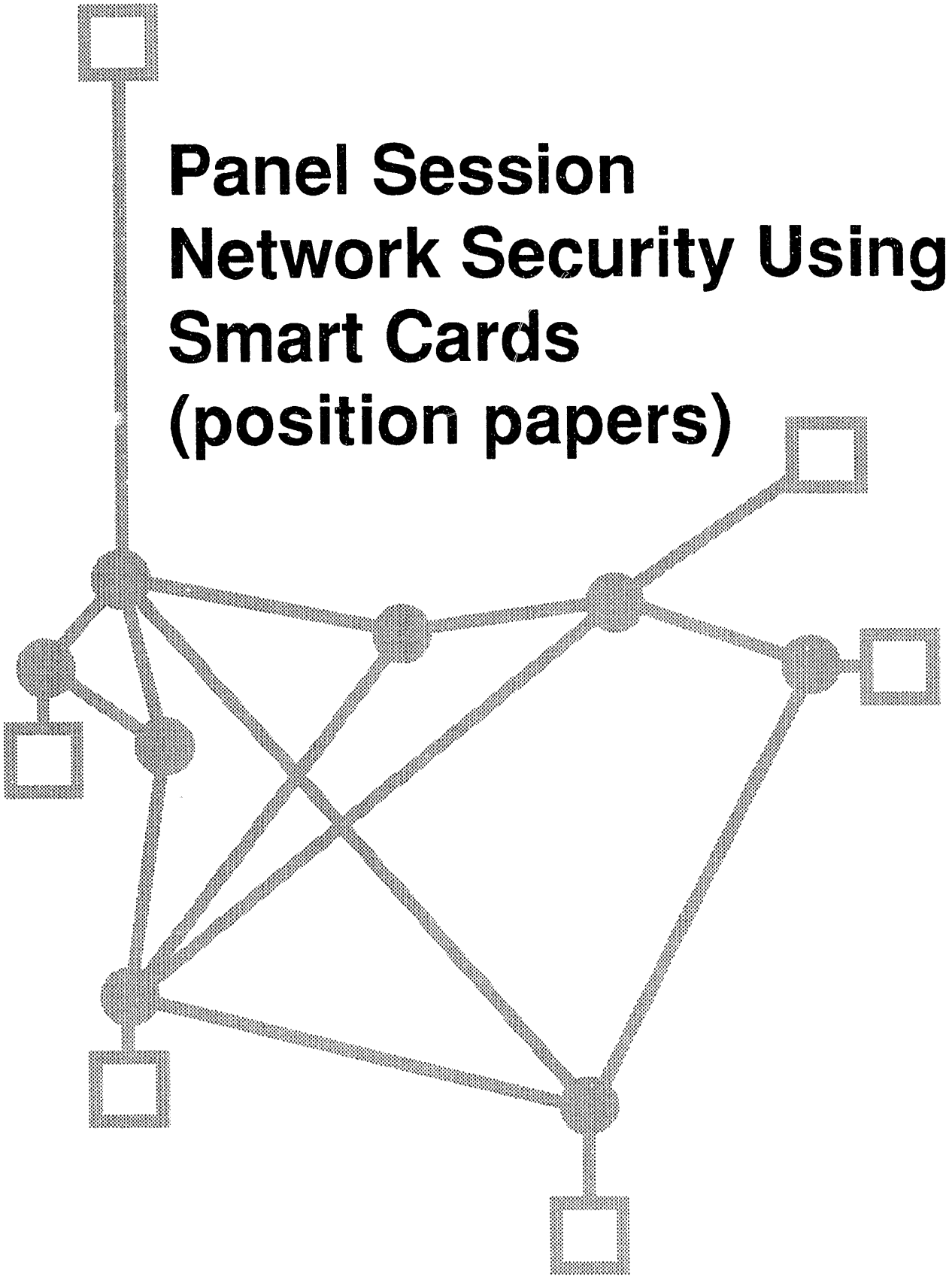
12. Satyanarayanan, M., "Integrating Security In a Large Distributed System", ACM Transactions on Computer Systems, Vol. 7, No. 3, August 1989.

13. Popek, G. and Walker, B., "The LOCUS Distributed System Architecture", The MIT Press, Cambridge, Massachusetts, 1985.

14. Steiner, J., Neuman, C., and Schiller, J., "Kerberos: An Authentication Service for Open Network Systems", Usenix Conference Proceedings, Winter 1988.

15. Champine, G., Geer, D., and Ruh, W. "Project Athena as a Distributed Computer System", IEEE Computer, Sept. 1990.

Panel Session Network Security Using Smart Cards (position papers)



Issues surrounding the use of Cryptographic Algorithms and Smart Card Applications

Jeffrey I. Schiller
Massachusetts Institute of Technology
Cambridge, MA

Introduction

One of the primary applications for Smart Card technology is to provide authentication of users in electronic transaction systems. By using cryptographic algorithms it is possible for users to identify themselves to electronic systems without revealing information that would enable an observer of the authentication process to subvert subsequent authentication transactions (e.g., take over the identity of the Smart Card user).

We will use the NIST developed Smart Card as an example of a typical Smart Card implementation.

NIST has been working on a Smart Card that can be used for various applications, including personal identification. This card is the size of a standard credit card and can be carried as easily as a credit card.

According to NIST, this card has been programmed to use at least three different cryptographic algorithms and protocols (not at once, i.e., the card can be downloaded with software to use one of the algorithms).

This paper will describe the NIST smart card and then discuss some issues that arise when it is used with particular cryptographic algorithms. We will specifically cover its use with DES, RSA and the proposed U.S. Federal Digital Signature Standard (DSS).

Background

The NIST smart card is a credit card sized processing system that uses a Hitachi H8/310 processor. It has 10K bytes of ROM, 256 bytes of RAM and 8K bytes of EEPROM as well as the necessary I/O hardware to allow it to be electronically queried. The card does not include a battery, instead it is only "on" when it is inserted in an appropriate reader, from which it draws its power.[1]

It is quite a challenge to fit processing power and memory into the form factor of a credit card. Significant effort must also be expended to ensure that the card will remain viable given its potentially hostile use and storage environment. The card must be able to get wet, withstand physical stress in various dimensions (i.e., bending). It must also be able to survive in the static rich environment of a wallet or pocket book. It is therefore quite impressive by itself that the card contains the processing power and memory described above.

However it is also important to mention what the card does NOT contain. The card does not contain either a battery nor (as a result) battery backed up memory. The card also does not contain a hardware random number generator. The lack of these facilities will become important to the discussion below.

General use of a Smart Card for Identification

The traditional approach to using a smart card for identification involves the storage of a secret quantity, typically an encryption key, within the smart card.

The user proves their identity by inserting the smart card into a reader. The reader challenges the card by providing a value and expecting the card to perform a calculation that only the possessor of the secret would be able to perform.

In many systems a Personal Identification Number (PIN) is supplied by the card user (through a keyboard which is part of the reader system) to the card as a means of authenticating the request to the card. In other words the user supplies the PIN to the reader, which forwards it to the card. In this fashion the user is authenticated to the card. The card then will use knowledge of its secret to prove identity to the challenging system.

Symmetric Encryption Use (DES)

The NIST smart card was originally programmed to use the U.S. Federal Data Encryption Standard (DES). DES is a typical symmetric cipher system. It enciphers and deciphers data using one key.

Using the DES both the smart card and the card's challenger have possession of the same DES key. The challenger provides a challenge value to the card and expects the card to encrypt (or decrypt) the challenge using the stored secret DES key. The challenger then performs the same computation and expects the results to compare successfully.

The primary disadvantage to using the DES in this fashion is that both the card and the card's challenger need to have access to the secret DES key used by the card. The

[1] This description is based on public presentations that have been made by members of the NIST technical staff. The author did not work directly on the NIST smart card project.

challenger is therefore in a position to forge additional cards with the same DES key and therefore can illegitimately claim to be the possessor of the original card to other challengers.

If smart cards are used by consumers for electronic business, this risk is significant as many banks and or merchants will need to have access to the DES key for each user's card. Of course in practice what will have to evolve is some centralized identification clearing center. However as this center will have access to all user's cards DES keys, it will represent a significant target for parties wishing to subvert the identity system.

These disadvantages can be bypassed by using Public Key Encryption

Public Key Encryption Use

Public Key Encryption systems use two keys, one for encryption and one for decryption. Keys are generated in pairs, one public and one private. If one key is used to perform an operation (either encryption or decryption) then the other can be used to perform the opposite function. Importantly, knowledge of the public key does not give away the private key.

In smart card systems a key pair is generated for each card. The private, confidential, key is loaded onto the card (or the key generation is performed on the card and the private key never leaves it) and the public key is used by challengers of the card.

When a user goes to use the smart card, the challenger sends the card a challenge which the card "signs" using its private key. The challenger then verifies the "signature" by using the public key. The challenger has access to the public key for the card and can challenge it. However because the challenger does not have the private key (which is on the card) the challenger cannot illegitimately forge additional cards.

Of course in order for this to work it MUST be impossible for the challenger to get the smart card to divulge the private key.

NIST has implemented two different public key algorithms on the smart card, RSA and the proposed DSS.

Using the Smart Card with RSA

NIST has ported the RSA algorithm suite to the card. With RSA the card can generate "signatures" as well as verify signatures of other cards (or other entities which have a key pair). However the NIST smart card as yet has not been programmed to generate key pairs. Key pair generation with RSA is a complex process which would consume significant memory resources on the card, but in normal use is performed but once.

When using RSA, the NIST smart card must have a "trusted" external agent generate the key pair and load it onto the card. The "trusted" agent must be trusted to NOT keep a copy of the generated private key. No legitimate interest should require it.

Using the Smart Card with DSS

The DSS has also been implemented on the NIST smart card. In addition to the sign and verify operations, with the DSS the card can also generate the private key locally because very little memory overhead is required to generate key pairs with the DSS.

However there are some significant issues and risks when using the DSS with a smart card, including the NIST smart card. For the most part they boil down to the requirement for a unique random value ('k' from the NIST specification) for each signature. Where this value comes from and how it is guaranteed to be unique is the problem.

Using RSA with the NIST smart card does not require the storage of any dynamic state on the card. The card need only hold its operating software and the public/private key pair.

However using DSS requires a random value 'k'. To get this value the card will need either a hardware random number generator (which it doesn't have and probably cannot have) or a cryptographically strong software pseudo-random number generator. The later is the most likely scenario.

However introducing a software pseudo random number generator to the card will require dynamic state (the state associated with the generator) which will change on every signature but which is kept between signatures, including during those periods when the card is not in a reader and therefore not powered.

The problem with this dynamic state is that NVRAMs typically have a limited re-write lifetime, and NVRAM is the only location that the card can store state between usages. This will result in the card being able to sign only a limited (though this value may be in the thousands) number of signatures before the card must be discarded. The software on the card must also be careful to ensure that the NVRAM has not failed, where failure in this context is defined as failing to store a new value into a memory location or otherwise having memory read a constant value. One approach might be to define a limit on the number of signatures, and therefore the number of writes to NVRAM, before the card administratively shuts down (hopefully with some warning to its user).

Bad alternatives

Below I itemize some "bad" alternatives to having dynamic state on the card.

Bad alternative 1: Have the reader provide the value of K. This is bad because if the challenger can provide the value of "K", then the challenger can derive the card's private key. This is a mathematical property of the DSS algorithm, if an attacker has a signature (and the message it signs) and the K value used to compute that signature, then the attacker can easily derive the private key.[2]

Bad alternative 2: Have the reader provide a seed for a software pseudo-random generator on the card. This generator will involve a secret value known only to the card so that knowledge of the generator's seed will not result in the knowledge of the computed K.

Turns out that this method is also weak. The reason is that an attacker can determine the card's private key if the attacker has access to two signatures (and their corresponding text) that were computed with the same value of K. Again, this is a result of the mathematical properties of the DSS.[3]

It would be a simple matter for a crooked challenger to issue two challenges and then provide the same random number seed for them. In fact the challenger could even drop power to the card between the challenges to assure that the card isn't keeping state in RAM memory.

A word on DSS vs. RSA timing

One of NISTs claimed reasons for preferring the DSS over the RSA system has to do with the speed of computing signatures. Often quoted are signature times of about 25 seconds for RSA and 0.05 seconds for the DSS. On the surface it appears that the DSS wins hands down. However closer examination of the DSS numbers shows that to achieve this 0.05 second signing time requires a pre-computation which requires about 25 seconds, or a time comparable to the time required to generate an RSA signature. What makes it pre-computation is that it can be performed before the receipt of a challenge, whereas the signature time itself is measure from the time of receipt of a challenge. However whether or not most of the overhead of DSS can be written off as pre-computation, it is computation that has to occur nonetheless. Under any circumstance the card will need to be inserted into a powered on reader for the time necessary for a pre-computation as well as the actual signature computation.

---

[2]Appendix A goes into the details of this attack.

[3]Appendix B goes into the details of this attack.

If the card has dynamic state which is saved in NVRAM across power down times, then precomputation can occur whenever the card has power. Multiple pre-computations can be performed (until memory for the results in full) and stored for later use. However if the card maintains no state across power downs, then pre-computation MUST occur between the time the card is inserted in the reader and the time it is removed (after a signature is issued). So regardless of how you account for the necessary 25 seconds of computation, the card needs to be in the reader for at least 25 seconds.

A word on Trust

One of the features of using the DSS with the NIST smart card is that the card can generate the private key. RSA implementations require that the key pair be generated off the card and then downloaded. This naturally leads to an argument that a private key chosen using the DSS is more secure because off card components do not need to be trusted.

However this argument is fallacious. With the DSS a private key is generated simply by choosing a random number of the appropriate length (the public key is then computed based on this random choice). However this again brings up the issue of where "random" numbers come from. Even in the case of a card which maintains state in NVRAM, the pseudo-random number generator needs to be initially seeded. This seeding needs to be from a trusted off card entity. If you have to trust an off card entity to initialize the random number generator, then you can also trust it to generate an RSA key pair.

Of course this argument hasn't even touched on the trust required of the system that initially downloads the operating software into the card.

Conclusion

Smart card technology may be used to engage in several different cryptographic authentication protocols, based both on symmetric (private) and asymmetric (public) cryptography. Asymmetric cryptography offers some decided advantages over symmetric systems because challengers (banks and merchants in a commercial system) need not be completely trusted.

Several asymmetric algorithms exist. The two most discussed being the RSA and proposed NIST DSS algorithms.

Given a choice between the NIST proposed DSS algorithm and the RSA algorithm, the RSA algorithm appears to offer less risk to the end user because its implementation is less complex. Without discussing the relative cryptographic strengths of the two algorithms, the DSS has the disadvantage that it requires cryptographically unique random numbers for each signature. Generating or acquiring

these values is a subtle process which may easily be mis-implemented, exposing the end user to the risk of private key disclosure. A secure implementation requires the continual re-write of the on-card NVRAM memory, something that NVRAM isn't well suited for.

Appendix A - Compromise of DSS with knowledge of "K"

Using the NIST DSS a signature is computed by the following equations:

$$(1) \qquad r = (g^k \bmod p) \bmod q$$
$$(2) \qquad s = (k^{-1} (H + xr)) \bmod q$$

r and s together comprise a signature. p, q, and g are public parameters used for both signature computation and verification. H is the hash that is being signed. k is the random value needed to compute a signature and k^{-1} is its multiplicative inverse mod q. Finally x is the private key being used to compute the signature.

k and x must only be known to the signer. Although x is the same for multiple signatures (it's the user's private key), k is unique to a particular signature.

To verify a signature one needs to know q, p, g, r, s, H and the public key y. y is equal to g^x mod p. To compute x given y requires solving the discrete log problem. However this requires an intractable amount of computation and is the strength of the DSS algorithm.

Typically q, p, and g are publicly known and shared by a community of users. y is obtained from a directory and H is computed from the document whose signature is being verified. r and s comprise the signature itself.

Lets rewrite equation (2) above to solve for x, the private key.

$$(3) \qquad x = \frac{s - k^{-1}H}{k^{-1}r} \bmod q$$

The only unknown that prevents this equation from being solved for x is k. If k is known then k^{-1} is trivially computed and x can be determined.

Appendix B - Compromise given two signatures using the same k

Appendix A demonstrated that if the k used to compute a signature is known to an adversary, then the private key x can be simply computed. In this section we consider the case where k is not known, but where two different signatures were computed with the same k.

From equation (1) if two signatures are generated with the same k, then r will also be the same. This implies that it is trivial to determine if two signatures were computed with the same k, for the same r will appear in each. Let's rewrite equation (2) below:

$$(2) \qquad s = (k^{-1} (H + xr)) \bmod q$$

Now for a second signature (using H' to represent the second hash and s' to represent the second s value from the signature) we get:

$$(4) \qquad s' = (k^{-1}(H' + xr)) \bmod q$$

Solving for k^{-1} we get:

From (2) $\quad k^{-1} = \frac{s}{H + xr} \bmod q$

From (4) $\quad k^{-1} = \frac{s'}{H' + xr} \bmod q$

So:

$$(5) \qquad \frac{s}{H + xr} = \frac{s'}{H' + xr} \bmod q$$

Solving for x yields:

$$(6) \qquad x = \frac{s'H - sH'}{r(s - s')} \bmod q$$

As can be seen, all the necessary values to make this computation are present and x is easily computed.

SMART CARD AUGMENTATION OF KERBEROS

Marjan Krajewski, Jr.
The MITRE Corporation
Bedford, MA

ABSTRACT

This paper addresses security issues associated with authenticating users to system services in distributed information systems. Its focus is the presentation of the need for and an approach toward augmenting the Kerberos distributed system identification and authentication protocol via the integration of emerging smart card technology.

INTRODUCTION

Two critical aspects of information system security are the application of access controls based on a user's authorizations and the creation of an audit trail based on a user's actions [1]. Both are dependent upon the accurate authentication of users to guard against the threat of intruders masquerading as valid users. Traditionally, a user is authenticated to a host upon presentation of a valid combination of userid and password. In a distributed processing environment, a user often needs to access resources located at multiple servers from multiple workstations interconnected via a communications network. Authentication to each host accessed is crucial, but presenting separate userid/password pairs can be both unwieldy and unsecure. What is needed is a mechanism which requires users to identify and authenticate themselves once to a trusted agent which then performs the necessary user identification and authentication to each accessed resource transparently (unitary login).

While much work has and is being done in this area, a solution suitable for a truly hostile environment (i.e., one subject to active attacks against both workstations/servers and the network) does not yet exist. Some unitary login protocols, designed for use in military environments where the network is physically protected from intruders and the users are trusted, do not use any form of encryption and can be easily defeated by any one of a number of commercially available network protocol analyzers capable of intercepting network transmissions. Other protocols protect against the threat of network eavesdropping through the use of various forms of encryption but still assume that workstations and servers are physically protected (e.g., by individual user ownership/control). The covert introduction of a Trojan Horse program into these workstations can easily "break" the authentication mechanism. Both Government and non-Government organizations could greatly ease the many problems associated with password management and the threat from masquerading on their increasingly distributed information systems with a unitary login capability which was secure from both a workstation/server and a network perspective.

The Kerberos protocol possesses many advantages as a basis for this capability [2]. Originally developed to provide user authentication for the distributed open computing environment of MIT's Project Athena, Kerberos is growing significantly in popularity (it has been adopted by the Open Software Foundation and Unix International as well as being offered in several commercial products). It also uses algorithm-independent conventional (private) key encryption to protect against network eavesdropping. This latter feature is especially important for military/intelligence applications in that the current Data Encryption Standard (DES) algorithm might be inadequate for certain environments. If so, it can easily be replaced with a stronger algorithm.

KERBEROS

Kerberos utilizes a trusted central authentication server, the Kerberos Authentication Server (KAS). The KAS contains a database of system entities (registered users and services) and their private cryptographic keys. These private keys, known only to the respective entity and the KAS, allow the KAS to communicate privately with the Kerberos agent of each system service (server Kerberos) and with the Kerberos agent of each registered user who wishes to be logged in (client Kerberos). The KAS also contains a ticket granting service to provide a trusted means for logged in users to prove their identity to system services. Finally, it contains a key generation service which supplies authorized pairs of entities with temporary cryptographic keys (session keys).

The Kerberos protocol is based on the concept of tickets and authenticators. A ticket is issued by the KAS for a single user and a specified service. It contains the serviceid, the userid, the user's (workstation) address, a timestamp, the ticket's lifetime and a randomly chosen session key to be used by this user and this service. This information is protected by encryption under the service's private key. Since this key is known only to the service and the KAS, the service is assured of the authenticity of the ticket. Once a ticket is issued, it can be used many times by the named user to gain access to the indicated service until the ticket expires.

Unlike the ticket, the authenticator is built by client Kerberos. A new one must be generated every time the user wishes to use a ticket. An authenticator contains the user's id, the user's (workstation) address, and a

timestamp. The authenticator is encrypted with the session key which is associated with the ticket. Encryption of the authenticator provides integrity of the authenticator and assures the service that the user is the system entity who received the original ticket. The further agreement of the user id in the authenticator with the one in the ticket and the address with the one from which the ticket arrived provides further assurance. Agreement of the timestamp with the current time assures the service that this is a fresh ticket/authenticator pair and not a replay of an old pair.

Security within Kerberos is enforced through the protection of session keys during transmission. The initial response from the KAS to the client, upon presentation of a userid, consists of a ticket to the ticket granting service and its associated session key. The ticket granting service ticket is encrypted with the ticket granting service's private key. The ticket granting service session key is encrypted with the client's private key (derived from a password). Any additional session keys associated with tickets to system services requested by that particular client are henceforth encrypted in the ticket granting service session key.

Kerberos has been analyzed from a general security perspective [3]. A number of vulnerabilities are associated with the initial login process. A significant vulnerability involves its manipulation of the user's Kerberos password and other sensitive authentication information (i.e., session keys) within the workstation, thereby making it vulnerable to Trojan Horse threats which could capture such data for later use by an intruder. Another vulnerability involves the threat of repeated attacks at the intruder's leisure following interception of the initial message from the central authentication server. This message contains the ticket granting service ticket and associated session key and is encrypted by a relatively weak password-derived key. A third vulnerability involves the inherent weakness of depending solely upon a single factor (i.e., a password) for the initial user authentication. Passwords can be easily borrowed or stolen (Figure 1).

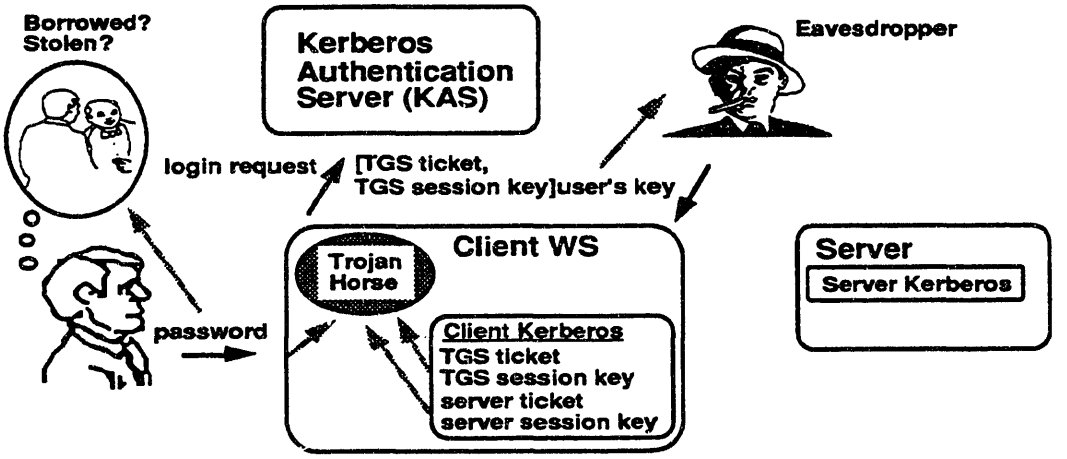


Figure 1. Kerberos login vulnerabilities

AUGMENTATION CONCEPT

Advances in smart card technology have reached the point where significant amounts of information can be stored and significant processing can be performed entirely within the isolated environment of the card itself [4]. When this technology is combined with user-unique information (e.g., a password) that is useless except when processed by the appropriate smart card, a significantly stronger authentication mechanism can be constructed than is available with "standard" (i.e., software-only) Kerberos. Kerberos itself has evolved to accommodate such augmentations [5].

The Kerberos augmentation concept advocated here is based upon moving all cryptographic processing from the workstation into a user-unique smart card and storing the user's private key in encrypted form on the smart card.

The user's private key would be encrypted in a key derived from a password. In this way, neither possession of the card alone nor knowledge of the password alone would be sufficient to authenticate a user. Encryption and decryption operations and the storage of unencrypted authentication information would occur only within a trusted processing environment (i.e., that of the smart card). The initial concept is detailed in [6].

In addition to enhancing security, it is important to maintain interoperability with existing Kerberos implementations in order to allow augmented Kerberos components to be gradually introduced into an

operational environment as time and resources permit. This constraint mandates that neither the Kerberos Authentication Server nor the server Kerberos implementations be affected in any way. It also mandates that the client Kerberos software be modified in a way that results in the least impact to the existing code. The approach selected was to replace the cryptographic routines in client Kerberos with a smart card driver that, together with the attached smart card, emulated the original code with one difference -- all session keys presented to the smart card driver were returned to the calling routine in encrypted form rather than unencrypted form. By doing this, the client Kerberos software is tricked into "believing" that it is handling red (unencrypted) session keys when in fact it is handling only black (encrypted) session keys. As stated in the previous paragraph, key decryption and unencrypted key handling occurs only within the smart card itself (Figure 2).

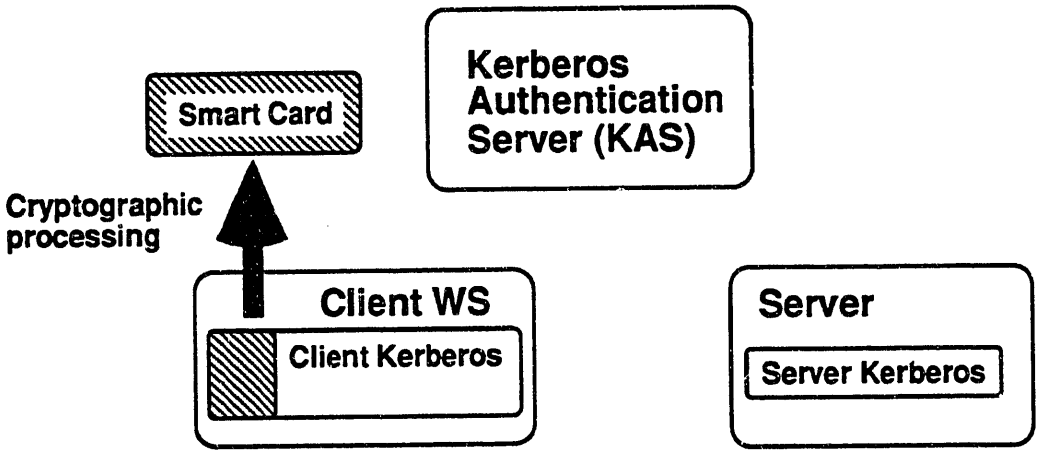


Figure 2. Kerberos augmentation concept

Issues that quickly arise in attempting to realize this concept include those related to feasibility, security, and performance. Regarding feasibility, it is not clear how well current smart card technology will support the required functional partitioning between inboard and outboard elements. Regarding security, the movement of all sensitive processing into a smart card mandates that the smart card provide a trusted environment which is incorruptible from the workstation and contains no sensitive data when not in use. It is not clear how well current smart card technology can provide the needed isolation. Regarding performance, the use of an outboard microprocessor will undoubtedly impact response time, but how severely is unknown. Implementation of a proof-of-concept prototype is necessary to definitively answer these questions. The details of this prototype are the focus of the remainder of this paper.

SMART CARD SELECTION

Card technology is advancing rapidly to support a vast number of applications in the financial, medical, telecommunications, and mass transit arenas. It is estimated that, by 1996, several hundred million cards will be sold worldwide. Of the three basic card types, magnetic stripe (the familiar credit card), optical, and integrated circuit, only the integrated circuit or "smart" card contains the required on-card data processing capabilities.

A typical smart card is approximately the size of a credit card and contains a microcontroller with a limited amount of on-chip volatile random access memory (RAM), a substantially larger amount of electrically erasable programmable read only memory (EEPROM), and an "operating system" residing in masked read only memory (ROM). Numerous vendors are currently producing such cards, each with slightly different RAM, EEPROM and programming characteristics and with varying levels of security (i.e., card protection).

For the proof-of-concept implementation, a minimum of 8 kilobytes of EEPROM and a highly flexible software development environment were paramount. At the time the implementation decision was made (mid-1991), the OmegaCard by Sota Electronics, Inc. appeared to best meet these criteria. The OmegaCard contains a custom processor configuration based upon the Intel 8051 family of microcontrollers, 8 kilobytes of EEPROM, and a mini operating system which manages the resources of the card. The EEPROM can be partitioned into program segments and data segments as required by the application and can readily be reprogrammed. Applications are developed in C using standard Intel 8051 software development tools. The card communicates via a standard RS-232C interface.

SMART CARD IMPLEMENTATION

The Kerberos augmentation concept was implemented using MIT Kerberos version 5 and the Sota Electronics OmegaCard described above. Sun Microsystems workstations (a Sun IPC and a Sun 3/50) served as the KAS/server and client hosts. The overall process flow is as follows (Figure 3).

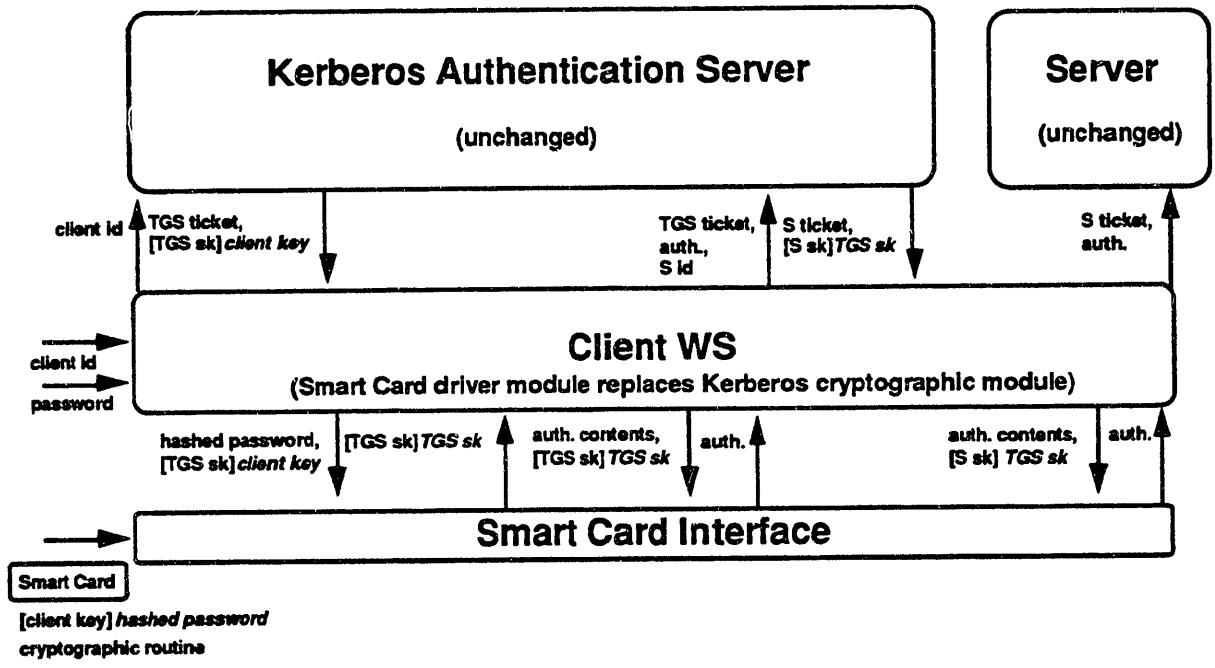


Figure 3. Process flow

Initial State: Client Kerberos holds no user-unique information. The smart card holds the user's private key encrypted in a key derived from a password.

Step 1: The user inserts his/her smart card into the card reader attached to the workstation and initiates the login process. Client Kerberos requests a userid.

Step 2: Client Kerberos sends the userid to the KAS.

Step 3: The KAS generates the user's ticket granting service (TGS) ticket and associated TGS session key, encrypts the TGS session key in the user's private key, and sends them to client Kerberos.

Step 4: Client Kerberos requests a password and derives a key from it. It then transfers this key and the encrypted TGS session key to the smart card.

Step 5: The smart card uses the key derived from the user's password to decrypt and obtain the user's private key. The smart card then uses the user's private key to decrypt the encrypted TGS session key. The smart card then stores the TGS session key in its volatile memory and destroys both the key derived from the user's password and the decrypted copy of the user's private key. Finally, it encrypts a copy of the TGS session key using the TGS session key itself and transfers the encrypted TGS session key back to client Kerberos. (The purpose of this last step is to ensure that all future data transfers to the smart card contain a session key encrypted in the TGS session key and data to be encrypted in the accompanying session key.)

Step 6: To access a system service, client Kerberos first determines if a ticket to that service is needed (one may have been obtained earlier and, if so, the process jumps to step 10). If a ticket is needed, client Kerberos transfers the encrypted TGS session key and the relevant authenticator data to the smart card.

Step 7: The smart card first decrypts the transferred session key using the TGS session key previously stored in volatile memory and then creates an authenticator by encrypting the transferred data with the decrypted session key. It then transfers the newly created authenticator to client Kerberos.

Step 8: Client Kerberos sends the service request, together with the TGS ticket and authenticator, to the KAS.

Step 9: The KAS generates the appropriate server ticket and associated server session key, encrypts the session

key in the user's TGS session key, and sends them to client Kerberos.

Step 10: Client Kerberos transfers the encrypted session key and relevant authenticator data to the smart card.

Step 11: The smart card first decrypts the transferred session key using the previously stored TGS session key and then creates an authenticator by encrypting the transferred data with the decrypted session key. It then transfers the newly created authenticator to client Kerberos. (The client Kerberos smart card driver returns the unchanged encrypted session key back to the calling routine when required to return a "decrypted" key.)

Step 12: Client Kerberos sends the server ticket and authenticator to the requested service.

CONCLUSIONS

The smart card Kerberos augmentation concept improves system security in three significant ways. It requires a user to provide both something he/she possesses (i.e., a smart card) as well as something he/she knows (i.e., a password). Either item alone is useless. This significantly reduces the risk from password borrowing/theft. It allows the initial message from the central authentication server to be encrypted in a truly random key rather than a password-derived key. A cryptographic attack on this message must therefore assume that the entire keyspace is available for use. This significantly reduces the risk from network eavesdropping. Finally, it ensures that only encrypted data is processed by a user's workstation. Software residing on a workstation can view only the same data a network eavesdropper can view (and a password tied to a specific smart card). This significantly reduces the risk from Trojan Horse programs.

The prototype implementation described in this paper has proven that the concept is feasible. The primary lessons learned indicate that performance and security are major factors in the selection of a suitable smart card. Regarding performance, the smart card used in the proof-of-concept prototype contained 8 kilobytes of EEPROM and 128 bytes of RAM. This amount of memory was found to be marginal for the application. In particular, there was not sufficient memory to store common cryptographic computational results and previously decrypted session keys, thereby forcing significant amounts of recomputation and increasing response time. In addition, some intermediate results that should, for security reasons, be stored in RAM, had to be stored in EEPROM. A minimum of 256 bytes of RAM appears necessary. Regarding security, it was found that the selected smart card must have protected areas that cannot be viewed or altered from the workstation. The smart card used in the prototype is potentially vulnerable to an attack whereby EEPROM-resident applications software is overwritten with malicious software.

Future work in this area will be directed toward porting the client Kerberos smart card driver software to the Open Software Foundation (OSF) Distributed Computing Environment (DCE) and porting the smart card cryptographic software to the DataKey Signature Card. It is hoped that this will provide a robust, high performance implementation directly applicable to tomorrow's distributed information systems.

REFERENCES

[1] National Computer Security Center, "Department of Defense Trusted Computer System Evaluation Criteria," DoD Standard 5200.28-STD, December 1985.

[2] Steiner, J., Neuman, C., and Schiller, J., "Kerberos: An Authentication Service for Open Network Systems," USENIX Conference, 1988.

[3] Bellovin, S. M. and Merritt, M., "Limitations of the Kerberos Authentication System," Computer Communications Review, October 1990.

[4] Smart Card Industry Association, and Personal Identification Newsletter, CardTech Conference Proceedings, April 7-9 1992.

[5] Kohl, John T., "The Evolution of the Kerberos Authentication Service," Spring EurOpen Conference, 1991.

[6] Krajewski, M., "Concept For a Smart Card Kerberos," 15th National Computer Security Conference, October 1992.

AN OVERVIEW OF THE ADVANCED SMARTCARD ACCESS CONTROL SYSTEM (ASACS)

Jim Dray <dray@st1.ncsl.nist.gov>
Computer Security Division / Computer Systems Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

David Balenson <balenson@tis.com>
Trusted Information Systems, Inc.
Glenwood, MD 21738

ABSTRACT

The Advanced Smartcard Access Control System (ASACS) was developed by the National Institute of Standards and Technology in conjunction with Datakey and Trusted Information Systems. The system includes a smartcard with public key capabilities and a portable reader/writer with computational capabilities, including a microprocessor, programmable memory, a keypad, and an LCD display. Through the use of a layered interface, ASACS was integrated into several demonstration programs and into the TIS Privacy Enhanced Mail (TIS/PEM) system. This paper provides a brief overview of the ASACS.

INTRODUCTION

Computer access control systems which rely solely on password-based authentication have proven to be inadequate in many environments, particularly where network systems are involved. The security of access control systems can be significantly strengthened if the authentication process is based on something the user possesses, such as a smartcard, in addition to a memorized password or Personal Identification Number (PIN). Modern smartcards have the ability to process as well as store information, and this capability has significant advantages over passive memory card technology for security applications. Smartcards can implement secure cryptographic authentication and automated key distribution protocols, provide secure data storage, and perform a variety of other functions which increase the security of an access control system. This increase in security can be realized while maintaining or even enhancing the level of convenience for the system user.

The Advanced Smartcard Access Control System (ASACS) has been developed by the National Institute of Standards and Technology in conjunction with Datakey and Trusted Information Systems. The primary goal of the project was to develop an advanced smartcard system which exploits recent advances in semiconductor and cryptographic technology for secure login authentication. ASACS also provides secure data storage, automated key management, and digital signature capabilities. The services supported by the ASACS implementation are designed for use within networking environments, including both local area networks and wide area networks such as the Internet.

The ASACS smartcard provides cryptographic capabilities based on standard cryptographic algorithms and techniques, in combination with software running on a host computer. Many of the underlying concepts applied to the design of ASACS have been successfully demonstrated in the NIST/Datakey Token Based Access Control System (TBACS) [1] as well as the Secure Access Control System (SACS) [2] projects. Each of these systems provides token-based secure access to a host computer through a cryptographic handshake protocol based on the Data Encryption Standard (DES) algorithm. However, the ASACS project involves the development of a smartcard with greater capabilities through the addition of public key cryptographic functions. A new smartcard reader/writer with significantly greater capabilities has also been developed for ASACS. The ASACS reader/writer has

This work partially sponsored by the U.S. Government Defense Advanced Research Projects Agenyc (DARPA) under contract number 8139 MOD 01 to the National Bureau of Standards and under contract number F30602-89-C-0125 to Trusted Information Systems.

computational capabilities, and includes a microprocessor, programmable memory, a keypad, and an LCD display. These features support the needs of mobile users who require a portable reader/writer for authentication from remote sites. To demonstrate the capabilities of ASACS, several applications have been developed, most notably a system maintenance program and several other useful demonstration programs. In addition, ASACS has been integrated with the TIS Privacy Enhanced Mail (TIS/PEM) system.

SYSTEM OVERVIEW

Figure 1 depicts the ASACS system components. A user possessing a smartcard inserts the card into the reader/writer which is attached to a local workstation. The workstation is connected to a local area network (LAN), which in turn may be connected to other networks. The smartcard may be used to control the user's access to both the local workstation as well as to other workstations and host computers on the attached networks.

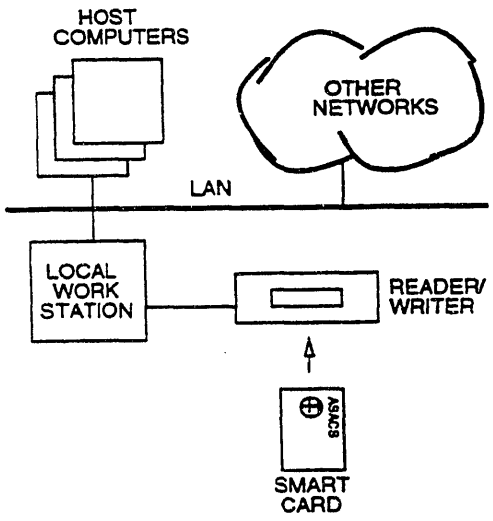


Figure 1: ASACS system components.

From an architectural standpoint, ASACS is divided into several different functional layers, comprising both the hardware and software components of the system (see Figure 2). The lowest layer consists of the ASACS hardware, including the public key smartcard and either the SACS reader/writer or the ASACS portable reader/writer. The next layer of ASACS is comprised of host system software, which is functionally divided into four layers. This software is used to provide to provide a convenient and standard method for integrating the ASACS public key smartcard into a wide variety of host system application software. The top layer is a Smartcard Application Program Interface (SCAPI) which is directly accessed by applications software to interface with the ASACS system. The other layers provide command set interfaces for the smartcard commands and the reader/writer commands, a smartcard communications protocol, and hardware-level I/O support.

Finally, the top layer of ASACS represents the various applications with which the ASACS system can be integrated. ASACS can be integrated into these applications using either the SCAPI or the command set interfaces. A security officer maintenance program and several demonstration programs, including a signature utility program and a login manager were developed as a part of the ASACS project. In addition, using the SCAPI, the ASACS system has been integrated into the TIS Privacy Enhanced Mail (TIS/PEM) system.

PUBLIC KEY SMARTCARD

The ASACS smartcard is based on the Smartcard-based Access Control System (SACS) developed by NIST under a previous DARPA sponsored contract. The SACS and ASACS smart cards contain a Hitachi H8/310 integrated circuit, designed specifically for smart card applications [3]. The H8 is configured with 256 bytes of RAM, 10K bytes of ROM, and 8K bytes of EEPROM. In order to meet ISO requirements for contact spacing and arrangement, the H8 die has pads for power (+5V), ground, clock (10MHz), reset, and serial I/O [4]. An ISO-standard micromodule is bonded to the H8 die, and this assembly is then mounted in a plastic card with the same dimensions as a standard credit card.

Smartcard Firmware

The ASACS public key smartcard firmware implements a set of commands which support card maintenance, key management, user authentication,

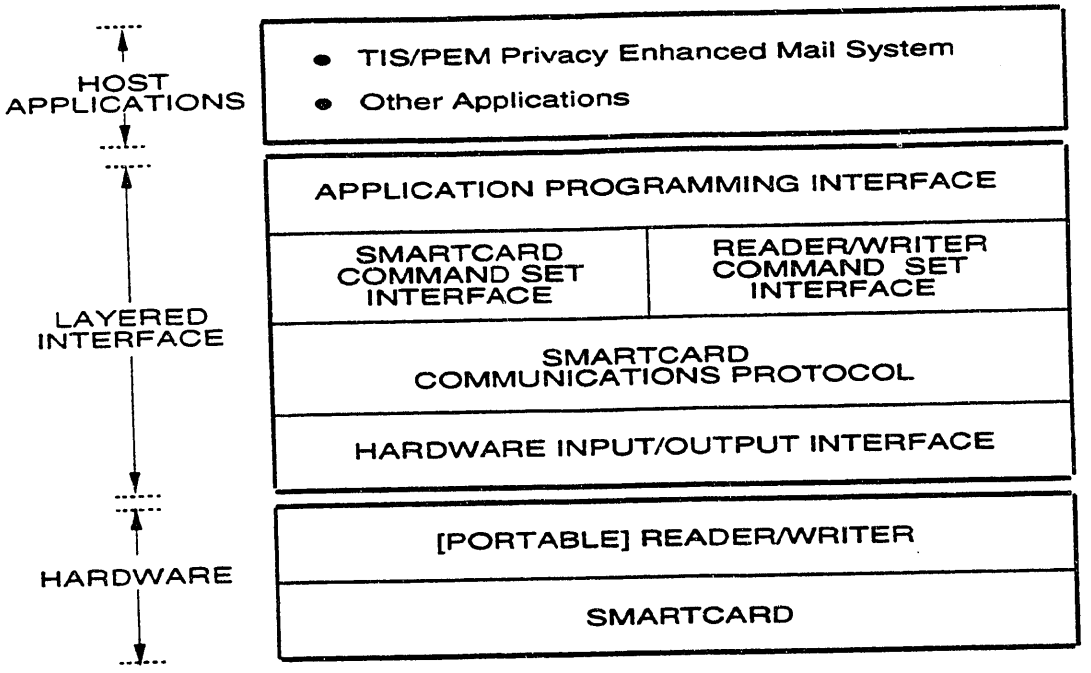


Figure 2: ASACS functional layers.

data storage, and data encryption and authentication. Access control software running on a host computer issues commands to the smartcard through the reader/writer interface. The firmware of the card then executes the requested function and returns the appropriate response to the host computer. It is the responsibility of the host access control software to mediate the authentications between the user, the user's smartcard, and the host computer.

The ASACS command set is the successor to the smartcard command set developed for the Smartcard-based Access Control System (SACS). The cost and time constraints of the ASACS project did not allow for the production of a new ROM mask. Therefore, the ROM mask developed for the SACS project was also used for the ASACS smartcard. ASACS retains the symmetric key capabilities of the original SACS system, since the authentication protocol is based on the Data Encryption Standard (DES) algorithm. This challenge-response authentication protocol provides a rapid and secure method for two parties to perform mutual identity verification based upon the possession of a shared secret key and the use of that key to encrypt randomly generated cryptographic challenges. This protocol is described in detail in NIST Special Publication 500-157 [5]. The ASACS smartcard is capable of accepting or generating the initial cryptographic challenge, and therefore complies with the requirements of ANSI X9.26 [6] for secure sign-on.

The principal difference between the ASACS and SACS command sets is the addition of public key cryptographic capabilities. There are certain arithmetic operations, such as modular exponentiation and modular multiplication, which are common to a variety of public key algorithms. These operations have been implemented in the ASACS firmware as distinct routines which can be used to support most of the currently available public key algorithms. The development and optimization of firmware which performs these modular operations is the most difficult aspect of implementing public key cryptography on a smartcard. A variety of public key algorithms can be realized in the ASACS smartcard firmware by calling the low-level arithmetic routines in the required sequence. Both the Digital Signature Algorithm (DSA), which has been proposed by NIST as a Digital Signature Standard (DSS) [7], and the

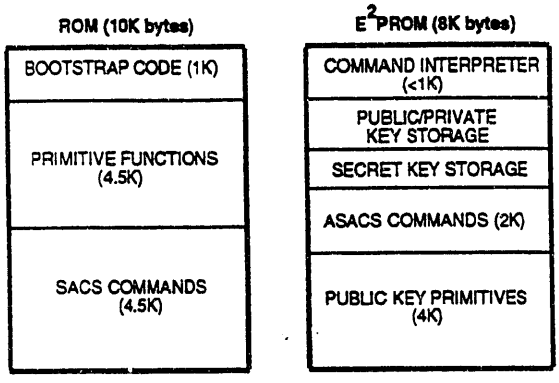


Figure 3: ASACS smartcard memory layout.

Rivest-Shamir-Adleman (RSA) [8] cryptographic algorithm have been implemented in the ASACS smartcard firmware.

Figure 3 depicts the layout of the ASACS smartcard memory from a high level perspective. The majority of the firmware is stored in ROM, including a bootstrap routine and code for the commands from the SACS smartcard. The Data Encryption Standard (DES) [9] algorithm is also located in ROM. The EEPROM contains the firmware for the public key algorithms, a command interpreter, and a jump table which points to the firmware routines associated with each command. Since the addresses in the jump table can be modified, new firmware routines can be loaded into EEPROM to replace existing routines and to add new functions. Specific locations in EEPROM are reserved for the storage of symmetric and asymmetric key components. In addition, a number of general purpose data storage zones are available in EEPROM.

See [10] for a more detailed description of the ASACS public key smartcard.

SMARTCARD READER/WRITER

The ASACS public key smartcard can be interfaced to a workstation using either the SACS reader/writer or the new ASACS portable reader/writer. Both the SACS and the ASACS reader/writers provide an RS-232 serial communications connection between the smartcard and the host computer. RS-232 was chosen because a serial port is standard equipment on the majority of computers. Therefore, the reader/writer can be connected to most computers without the need for a custom interface or hardware modifications.

SACS Reader/Writer

The SACS reader/writer is a relatively unsophisticated device which simply serves as a direct I/O interface between the smartcard and a host. It cannot perform any processing itself since it does not contain a microprocessor. Its main purpose is to provide power, ground, clock and I/O signals to a SACS or an ASACS smartcard. To interface the smartcard to the host, the reader/writer performs level conversion between the 12V RS-232 I/O signals used by the host and the 5V I/O signals used by the card. See [11] for a more detailed description of the SACS reader/writer.

The SACS reader/writer features an ISO standard smartcard receptacle, external power and data indicator lights, and an RS-232 port for connecting to a host. In addition, the SACS reader/writer's card receptacle features a locking mechanism which holds the card internally after insertion into the reader/writer, and an automatic ejection mechanism to remove the card from the reader/writer.

An RS-232 cable is required to attach the SACS reader/writer to a host, whereupon it functions as data communications equipment (DCE). Signals are sent by the reader/writer to the host which indicate that the reader/writer is powered-up and that a card is inserted. The SACS reader/writer is a rectangular box approximately 2 1/2 inches high, 5 inches deep, and 5 inches wide. An ISO smartcard receptacle and indicator lights are located on the front of the reader/writer, and the power cord and RS-232 jacks in the rear. The power supply for the SACS reader/writer is internal.

The SACS reader/writer is designed to accept a smartcard whose physical characteristics, dimensions and contact locations adhere to ISO International Standard 7816, Parts 1 and 2 [4,12]. The electrical signals that the SACS reader/writer supplies to the smartcard also meet most of the requirements specified in ISO International Standard 7816, Part 3 [13], with the exception of the initial clock (CLK)

frequency, which is 10MHz as opposed to 3.5795.

ASACS Portable Reader/Writer

The ASACS portable reader writer was built to provide functionality not offered by the earlier SACS reader/writer. As a portable device, it allows users the option to authenticate themselves using hosts not equipped with a smartcard reader/writer. Several significant improvements have been made to the design of the reader/writer. The overall size has been reduced to less than half that of the SACS reader/writer, so that the device can easily be carried for use at remote sites. The new reader/writer is powered by rechargeable batteries, and includes a transformer for use with 110V line power. The front panel has a keypad and liquid crystal display which allow the user to interact directly with the smartcard. This feature is useful in situations where the reader/writer cannot be connected to the user's workstation. A protocol has been developed which allows the user to perform authentications manually via the keypad and display. A remote host computer can then require manual ASACS authentication even if the user's workstation is a dumb terminal. In this case, all interactions with the card are through the keypad and display. After the user personal identification number (PIN) has been submitted to the card, the remote host will generate a random challenge and send this to the user's workstation. The user reads this challenge from the screen and types it on the reader/writer keypad. The smartcard encrypts the challenge and displays the encrypted result, so that the user can submit it to the remote host. When a serial connection to the workstation is available, the user still has the option of entering the PIN through the keypad on the reader/ writer. Since the user's PIN does not travel through the workstation, system security is enhanced.

The ASACS reader/writer has an 8-bit microprocessor with 256 bytes of internal RAM. In addition, the reader/writer has 256 bytes of EEPROM used for data and setup parameter storage, 32K bytes of RAM used for scratch pad and data buffering, and an industry standard 32K byte EPROM chip which holds firmware implementing the internal logic and external commands. The EPROM chip can be easily removed for custom firmware development. See [14] for detailed specifications for the ASACS portable reader/writer and firmware.

The reader/writer supports a set of commands that are executed directly on the reader/writer, as opposed to on the smartcard. These commands use the same protocol that is used for smartcard commands. Several of the reader/writer commands allow the host to load the default parameters into the reader/writer's non-volatile memory to control such things as baud rate, and the date/time. These same default values can also be specified manually from the keypad by pressing the F1 key to access the reader/writer's set-up menu. Another command can be used by the host to determine if a smartcard is inserted into the reader/writer. Two commands can be used to temporarily put the reader/writer in manual keypad entry mode. The first of these two commands, as discussed above, is used by the host to allow the user to enter their PIN to the smartcard via the reader/writer's keypad. The latter command can be called to allow the user to perform a manual challenge/response with a remote host. The remaining reader/writer commands can be used by the host to utilize the ASACS reader/writer's communications buffer for more efficient DES encryption, DES decryption or MAC calculation with the smartcard.

SMARTCARD LAYERED INTERFACE

The ASACS host system software is comprised of a set of four interface layers. Each layer corresponds to a specific set of functions needed to integrate the ASACS system into a software application on a host system (see Figure 2).

Smartcard Applications Program Interface

The Smartcard Application Program Interface (SCAPI) [15] was developed to provide a consistent, but robust interface designed to ease the integration of smartcard technology into applications. The SCAPI is intended to insulate applications from the differences among the various smartcards, as well as differences likely to appear as smartcard technology evolves. The SCAPI is not tied to specific smartcards or to specific capabilities (e.g., memory capacity) of smartcards. In fact, the SCAPI can be, and has been, completely implemented in software, thus providing a simple, but useful tool for integrating smartcard technology into applications. The functional capabilities of a particular smartcard

determines how much of the SCAPI functionality is implemented in software on the host computer and how much is performed on the smartcard. Thus, as technology advances, more of the SCAPI functionality may be directly implemented on the card or on the reader/writer while leaving applications unaffected.

The SCAPI currently defines four types of functions:

- Initialization Functions,
- Account Functions,
- Cryptographic Functions, and
- File and Directory Functions.

The SCAPI is intended to be consistent with the ANSI C standard. The file functions are designed to map directly upon those defined by Kernighan and Ritchie [16]. Since C is known for its portability, it makes sense to extend this platform independence to smartcard systems. Further, this flexibility and consistent feel for C programmers is likely to promote the use of the SCAPI. The directory functions reflect widely used operating system calls. Unfortunately, ANSI C does not address the cryptographic functionality to which smartcard technology is so well-suited. Therefore, the SCAPI defines a set of cryptographic functions which provide an algorithm-independent interface for cryptographic operations which may be implemented on a smartcard.

Smartcard and Reader/Writer Command Set Interfaces

The Command Set Interface Layer consists of C language object module libraries. The libraries each provide a set of C function calls, each directly corresponding to a command from the firmware command sets for the public key smartcard [17] and the portable reader/writer [18]. The function which represents a particular command is called with the appropriate input data for that command as arguments. The function returns the output data from the command and a status code. Status codes are mapped onto a set of error messages defined in a header file. This layer is called indirectly through the SCAPI, thus making the choice of reader/writer invisible to the application.

Communications Protocol and Hardware I/O Interface

The Smartcard Communications Protocol Layer transmits the data assembled by the Command Set Interface Layer to the ASACS portable reader/writer and the public key smartcard. The data is transmitted according to the communications protocol used by both the reader/writer and the smartcard. The Communications Protocol Layer interacts with the Hardware I/O Interface in order to send and receive each byte of the
data.

The Hardware I/O layer consists of a software driver which provides low-level input/output routines for communicating with the smartcards. Currently, the Hardware I/O Layer consists of a serial interface, since both the SACS and ASACS reader/writers employ serial interfaces. This layer can support other types of hardware interfaces for reader/writers that do not employ an RS-232 interface.

The Serial I/O Interface is written to be as portable as possible across a broad range of hardware/software platforms, such as SUNOS (Sun's UNIX Operating System) and MSDOS. However, some systems may require that this layer be customized. The interface to this layer is clearly defined, and can be modified with minimal effort.

APPLICATIONS SOFTWARE

Security Officer Maintenance Program

The Security Officer Maintenance (SOMAINT) Program [19] provides functions which are used by a security officer or system manager. These functions include the initialization of cards for new users, synchronization and maintenance of key databases stored on the cards and host computers, deactivation of cards, and reactivation of cards which have been inadvertently deactivated or corrupted. The programs which support the system management functions are restricted to use by authorized security managers through the standard UNIX operating system file protections.

Signature Utility Program

The DSS Signature Utility Program [20] was developed to demonstrate the generation and verification of digital signatures using the ASACS public key smartcard. The program utilizes the algorithm proposed by the Standard Hash Standard (SHS) [21] to calculate a hash value on a file of arbitrary size. The hash value is transmitted by the host computer to the smartcard, which applies the Digital Signature Algorithm (DSA) to this value to generate a digital signature with the cardholder's private key. The signature can then be verified by the host computer or the smartcard using the cardholder's public key.

Login Manager

The ASACS Login Manager [22] is a collection of programs which control login access to host computers. These programs manage the series of authentications between the user, the smartcard, and a host computer. When a user requests access to the host, the login manager establishes communications with the user's card through the reader/writer. The login manager prompts the user for the user PIN, and transmits it to the card in order to authenticate the user to the card. The card and host will then authenticate to each other using a random challenge-response protocol based on the Data Encryption Standard (DES). This protocol provides a means for rapid authentication of two parties with protection from wiretapping and playback attacks. If the authentications are successful, the user is granted a session on the host.

The login demonstration software also supports login authentication to remote host computers. When a system user wishes to access a remote computer, the user executes a program which communicates with the user's card to obtain a list of host computers with which the card shares authentication keys. This list of host computer names is displayed in a menu, so that the user can select the particular host to access. The software establishes a connection with the ASACS authentication server process running on the remote host selected by the user. The remote host then performs the challenge-response authentication with the user's card in order to verify the identity of the user.

Privacy Enhanced Mail

The Internet Privacy Enhanced Mail (PEM) protocols are an extension to the existing Internet electronic mail protocol (RFC 822) which provide simple end-to-end security services including optional message confidentiality, message integrity, and source authentication with non-repudiation. The protocols are specified in a 4 part series of specifications [23,24,25,26] which are currently published as Internet Drafts, and are targeted to be published as Internet Request For Comments (RFCs) with Proposed Standard status.

The PEM security services are provided through the use of standard cryptographic techniques, including message encryption using the DES in the Cipher Block Chaining (CBC) mode of operation to protect message text and the RSA algorithm to provide for distribution of DES keys, digital signatures using RSA algorithm in conjunction with either Message Authentication Code (MAC), Message Digest Algorithm MD2 [27], or the Message Digest Algorithm MD5 [28]. RSA public keys are managed as public key certificates using a distributed certification hierarchy based on CCITT X.509 [29].

The TIS Privacy Enhanced Mail (TIS/PEM) System is a UNIX-based implementation of PEM. At the core of the TIS/PEM system is the Local Key Manager (LKM), which, as its name implies, is responsible for all the local key management activities on a multi-user host system. This includes (1) maintaining a database for local users' private keys, (2) controlling the use of private keys to compute digital signatures and decrypt message tokens (encrypted message encryption keys), (3) maintaining a database for local and remote users' public key certificates, and (4) providing access to validated public key certificates. In addition, the LKM shares the responsibility for the registration of a local user, that is, the generation of a public/private key pair and the construction and digital signing of a certificate embodying the public key.

The ASACS system was integrated with the TIS/PEM system by integrating it with the LKM. In particular, a user's private key is generated by the LKM and then stored on the smartcard, where it remains in the protected confines of the smartcard. When called upon to perform the cryptographic operations involving the user's private key, the LKM, instead of

performing those operations directly, now invokes the functions of the smartcard via the SCAPI. The smartcard then performs the necessary computation of a digital signature or decryption of a message token, using the private key stored on the smartcard.

The storage of a user's private key provides added protection that cannot be achieved in a shared database. The inherent security features of the smart card allow for limiting access to the private key to the user, who must be authenticated to the card before the private key can be used.

ACKNOWLEDGEMENTS

Lots of people at NIST, Datakey, and TIS have contributed to the design and development of ASACS. Some of the developers deserving special thanks include Tom Cain, Paul Clark, Steve Crocker, Mike Indovina, Gary Ostrem, Miles Smid, and Robert Warnar.

REFERENCES

1. Dray, James F., Miles E. Smid and Robert B. J. Warnar, Implementing an Access Control System with Smart Token Technology, National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C., April 12, 1989.

2. NIST SACS Smartcard Specification, Datakey, Inc., Report #065-0097-000, July 11, 1991.

3. Hitachi H8/310 Single-Chip Microcomputer, Hitachi, Ltd.. Tokyo, Japan, 1989.

4. International Standard 7816-2, Identification Cards - Integrated Circuit(s) Cards with Contacts -- Part 2: Dimensions and Location of the Contacts, International Organization for Standardization, 1988.

5. Haykin, Martha E., and Robert B. J. Warnar, Smart Card Technology: New Methods for Computer Access Control, NIST Special Publication 500-157, National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C., September 1988.

6. American National Standard X9.26-1990, Financial Institution Sign-on Authentication for Wholesale Financial Systems, American Bankers Association, Washington, D.C., 1990.

7. Proposed Digital Signature Standard (DSS), National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C., August 30, 1991.

8. Ronald L. Rivest and Adi Shamir and Leonard M. Adleman, A Method for Obtaining Digital Signatures and Public Key Cryptosystems, Communications of the ACM, Volume 21, Number 2, February 1978, pp. 120-126.

9. Federal Information Processing Standard Publication (FIPS PUB) 46-1, Data Encryption Standard, National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C., Reaffirmed January 22, 1988 (Supersedes FIPS PUB 46, January 15, 1977).

10. ASACS Smartcard Specification, Datakey, Inc., Report #065-0130-000, April 24, 1992.

11. NIST SACS Reader/Writer Specification, Datakey, Inc., Report #065-0098-000, July 11, 1991.

12. International Standard 7816-1, Identification Cards - Integrated Circuit(s) Cards with Contacts -- Part 1: Physical Characteristics, International Organization for Standardization, 1987.

13. International Standard 7816-3, Identification Cards - Integrated Circuit(s) Cards with Contacts -- Part 3: Electronic Signals and Transmission Protocols, International Organization for Standardization, 1989.

14. ASACS Portable Reader/Writer Specification, Datakey, Inc., Report #065-0131-000, April 24, 1992.

15. Smartcard Application Program Interface for the Advanced Smartcard Access Control System (ASACS), Trusted Information Systems, Inc., Glenwood, MD, October 1992.

16. Kernigan, B. and D. Ritchie, The C Programming Language, 2nd Edition, Prentice Hall, 1988.

17. Advanced Smartcard Access Control System (ASACS): Smartcard Command Set Interface, National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C., 1992.

18. Advanced Smartcard Access Control System (ASACS): Reader/Writer Command Set Interface, National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C., 1992.

19. Security Officer Maintenance (SOMAINT) Program User's Manual, National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C., 1992.

20. Advanced Smartcard Access Control System (ASACS): The DSS Signature Utility Program Manual, National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C., 1992.

21. Proposed Secure Hash Standard (SHS), National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C., January 22, 1992.

22. Advanced Smartcard Access Control System (ASACS): UNIX Access Control Software Manual, National Institute of Standards and Technology, U.S. Department of Commerce, Washington, D.C., 1992.

23. John Linn, Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures, Internet Draft (draft-ietf-pem-msgproc-02.txt), Digital Equipment Corporation, July 23, 1992, (RFC in progress; will obsolete RFC 1113).

24. Stephen Kent, Privacy Enhancement for Internet Electronic Mail: Part II -- Certificate-Based Key Management, Internet Draft (draft-ietf-pem- keymgmt-01.txt), BBN Communications, August 6, 1992, (RFC in progress; will obsolete RFC 1114).

25. David Balenson, Privacy Enhancement for Internet Electronic Mail: Part III -- Algorithms, Modes, and Identifiers, Internet Draft (draft-ietf-pem- algorithms-01.txt), Trusted Information Systems, September 3, 1992, (RFC in progress; will obsolete RFC 1115).

26. Burt Kaliski, Privacy Enhancement for Internet Electronic Mail: Part IV -- Key certification and Related Services, Internet Draft (draft-ietf-pem- forms-01.txt), RSA Laboratories, September 1, 1992.

27. Kaliski, B., The MD2 Message-Digest Algorithm, Internet Request for Comments (RFC) 1319, April 1992.

28. Rivest, R., The MD5 Message-Digest Algorithm, Internet Request for Comments (RFC) 1321, April 1992.

29. CCITT Recommendation X.509, The Directory - Authentication Framework, The International Telegraph and Telephone Consultative Committee, November 1988.

END

DATE FILMED

7/7/93