A Security Model for Authentication on High-End Servers Using Digital Signatures

by

May K. Tse

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 11, 2000
[June 2000]

Author ________ [illegible]
Department of Electrical Engineering and Computer Science
March 2000

Certified by ___ [illegible]
Jerome H. Saltzer
Supervisor

Accepted by ________ [illegible]
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

A Security Model for Authentication on High-End Servers Using Digital Signatures
by
May K. Tse

Submitted to the
Department of Electrical Engineering and Computer Science

May 11, 2000

In Partial Fulfillment of the Requirements for the Degree of Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

This thesis applies systematic security engineering techniques to solve a practical business problem. In particular, this thesis will describe a security model that was implemented from July 1999 to January 2000 at SGI (Silicon Graphics Computer Systems), a company in Mountain View, CA that makes high-end servers and advanced graphics products. This security model involved remote authentication on components of the company's newest high-end servers. The motivation behind the security dealt with the problem of software theft due to improper software licensing, which stemmed from the unauthorized modification of system serial numbers. The authentication was put into place to restrict the access of those who wished to change the system serial numbers on the server components. The authentication was implemented using digital signatures created with the El Gamal variant of the Diffie-Hellman algorithm, and a public/private key pair system. The purely technical aspects were only half the problem, however; the design of the system as a whole proved to be the bigger challenge. In the corporate world, security is rarely the top priority in a company's agenda. Instead, ease of use for the average user and a seamless integration into the existing corporate infrastructure often take higher precedence than the level of security, making it difficult to design a security system that can still be effective and not too crippled by design trade-offs.

Thesis Supervisor: Jerome H. Saltzer
Title: Professor Emeritus/Senior Lecturer, Department of Electrical Engineering and Computer Science

TABLE OF CONTENTS

List of Figures

List of Tables

1.0 INTRODUCTION

This thesis describes an authentication system that was designed and implemented for the newest high-end servers at SGI (Silicon Graphics Computer Systems, a manufacturer of high-end servers and graphics products in Mountain View, CA). The high-end servers were essentially supercomputers made up of several processors connected together to perform extremely expensive computations (such as those related to computer graphics). The system focused on authentication for system serial number integrity.

1.1 Background

Nowadays, software theft is quite easy to do. There are not too many safeguards against someone copying some files or passing around a disk with a special software package on it. One way of protecting against this is to license each copy of the software package. System serial numbers are used to enforce the licensing of certain software packages. As software can often be readily reproduced and distributed, it is important to guarantee that the system serial numbers themselves are not vulnerable to duplication or modification; else, unlicensed software theft can run rampant. For high-end servers that are composed of multiple components, large software packages will be licensed to a specific system serial number for the entire server. Therefore, each server component (called a "brick") will then bear the same system serial number to enforce licensing. This system serial number will be the same no matter what type of component it is, whether it is used for processing, routing, or as an input/output module.

1.2 Problem Statement

As further protection against changes to the system serial number, in some prior designs of servers, the system serial number has been safeguarded as part of the hardware on the EEPROM. An EEPROM is a writable memory element, but the only way to change the data on it is to manipulate it through the hardware. Therefore, the EEPROM could not be modified through the software in the field – to actually change the system serial number on a particular brick, the part would need to be shipped back to the plant for modification. However, in the upcoming new server design in question, the system serial number will instead be devised as a member of the more easily rewritable NVRAM data record. An NVRAM is also a writable memory element, but it differs from an EEPROM in its method of changing the data on it – with an NVRAM, the data can be changed through software, not hardware. Therefore, this allows the system serial number to be changed through the software on-site, rather than at the plant. This allows for a more efficient and less expensive means of replacing broken or malfunctioning bricks – a spare brick can just be reconfigured on-site and swapped in rather than shipped back to the plant first to have the system serial number changed.

This new flexibility comes at a price, however: it makes the system serial number much more vulnerable to unauthorized modification because of the switch from hardware to software, which could potentially lead to improper software licensing. Since the system serial number can now be easily changed through the software, security becomes an issue; safeguards must be created to protect access to the program that actually modifies the system serial number. Extra precautions must be taken to ensure that this access is granted if and only if certain conditions are met, including the obvious constraint that the system serial number can only be modified by authorized company personnel. Therefore, in order to meet this need, a *security system for authentication* is desired to make sure the user is a member of the group of personnel authorized to make modifications to the system serial number.

2.0 DESIGN TRADE-OFFS

In designing any system, there will be certain design trade-offs that will have to be made between conflicting goals. It is therefore important to be explicit regarding certain priorities. In this particular security system for authentication, there are some special requirements to note:

- The authentication should be valid only within a predetermined time window.
- The brick at the customer site should have no secrets.
- The system must be easily exportable according to international export laws.
- The design should use as little of the limited NVRAM and EEPROM space on a brick as possible.
- The level of security should be modest -- functional but not extreme. The threat model here is from customers in the field thinking of improper software licensing if it is easy to do – *the threat model is not of seasoned hackers* looking to break into the system for the information it holds.
- The security system should not hinder the service personnel from conducting the usual number of tests at the plant or repair site.
- The system should be able to handle a manufacturing capacity of approximately 25,000 bricks per year (generous estimate).
- The system should be able to last for the life of the machines, and should be easily maintained and upgraded.

2.1 Security Versus Available Resources

When designing an actual system for the corporate world, sometimes it is not possible to meet all of the best conditions as they would be recommended in theory. Limited resources (such as a fixed budget) lead to certain design allowances and "tweaks" to deal with this.

2.1.1 Expiring Authenticator

An expiring authenticator is necessary because a static authenticator (such as a "root password") is easily leaked to unauthorized individuals who might need to use an authenticator once, or who happen to get it from an unknowing service person. In addition, a service person might also leave the company, taking the company's secret static authenticators with him.

A non-static authenticator can be achieved in two ways: either have the changing authenticator be produced externally to the system (via "safeword cards"), or to have the authenticator be produced internally, and set it to expire within a predetermined time window.

2.1.2 Producing the Non-Static Authenticator

One possible implementation of expiring authenticators would be to use "safeword cards," which are small, credit cards outfitted with a microchip for continual authenticator generation. The cards produce new authenticators at a constant rate, and a program that is run on the computers would be synchronized to match and verify the same changing authenticator.

The problem with this solution is that it is expensive. A card would have to be purchased for every member of the company who is authorized to modify the system serial numbers on the bricks. In addition, there is the added complexity of dealing with the distribution of the cards to all these people, as well as the forced retrieval of these cards if any personnel in possession of these cards leaves the company. This would be hard to do in terms of scope and scalability; it would be difficult to deal with the sheer number of cards to be distributed to the many field personnel in all parts of the world. In addition, the size and varied locale of the personnel would also make it hard to track every card in the system, so that it would be conceivable that a card could get into the wrong hands or would not be returned if an employee left the company. Therefore, it is recommended that the authenticator be produced internally in the system rather than rely on safeword cards.

2.1.3 A Public Key System

To produce an expiring authenticator internally in the system, a cryptosystem involving digital signatures is recommended. This cryptosystem can be designed in one of two ways: through a symmetric or an asymmetric (public key) system. A symmetric system works on the premise that the same key is used to both sign and verify a message, whereas two separate keys are used for a public key system. The private key is kept secret while the public key is widely distributed and publicly known [6]. The brick can have no secrets because we can not guarantee the privacy of the brick, so if a symmetric system were used, the key would be vulnerable to discovery, possibly leading to a known-key attack. Therefore, a public key system is preferred.

Now, applying a public key cryptosystem to this particular problem means that there will be a public key assigned to each brick. The public key can be hard-wired into the brick's EEPROM so that *it can not be modified*. It is important to note that it is the public key and not the private key that is kept on the brick, because the hard-wired information on the brick can be visible to everyone. The corresponding private key will be put on a server (along with all the other private keys) at the company's support center. The private key will be used to sign messages that only the public key on the corresponding brick can properly verify [8]. It is important to note that the company's internal support center server has the capability to be more heavily guarded than any brick that is in the field. This is appropriate, because breaking into the server would give the hacker access to several private keys, whereas breaking into a single brick on-site in the field would only give the hacker access to a single public key, which is useless without the corresponding private key.

Public Key System versus Symmetric Key System

Symmetric Key System with One Key, K_s

- To sign a message **M**: $\{\mathbf{M}\}\mathbf{K_s}$
- To verify the signature: $\{\{\mathbf{M}\}\mathbf{K_s}\}\mathbf{K_s}$ should recover the message **M**

Public Key System with a Public Key K_{pub} and a Private Key K_{priv}

- To sign a message **M**: $\{\mathbf{M}\}\mathbf{K_{priv}}$
- To verify the signature: $\{\{\mathbf{M}\}\mathbf{K_{priv}}\}\mathbf{K_{pub}}$ should recover the message **M**

***Figure 1.0:** Explanation of signing and verifying a digital signature with a public key system versus a symmetric key system.*

2.1.4 Choosing Algorithms for Authentication

Getting more to the mechanics of the security model, authentication with a public and private key system is based on an asymmetric algorithm. There were several algorithms proposed for this authentication model. Some possible choices included the Diffie-Hellman and RSA (Rivest-Shamir-Adleman) algorithms [5].

Algorithm	Patent Number	Date of Patent Expiration
Diffie-Hellman	4,200,770	4/29/97
Merkle-Hellman	4,218,582	8/19/97
Rivest-Shamir-Adelman (RSA)	4,405,829	9/20/00
Digital Signature Standard (DSA)	4,995,082	2/19/08

Table 1.0: Public key cryptography algorithms and their patent expiration dates. Data source for table is from [3].

The algorithms were all appropriate for public key cryptosystems, although they were each slightly different in nature. For example, the RSA algorithm relies on the difficulty of factoring large numbers. Based on two large prime numbers, two keys are then generated, a public key and a corresponding private key. The two keys can be used interchangeably: either the public key can sign and the private key can verify, or vice-versa [7].

The final algorithm chosen was the El Gamal variant of Diffie-Hellman. Unlike the RSA algorithm which depended on the difficulty of factoring products of large primes, the El Gamal algorithm relies on the difficulty of calculating discrete logarithms in a finite field [7]. The El Gamal algorithm was eventually chosen over RSA because it was the first to have its patent expire; RSA is not set to expire until September 2000 [3], which would have been too late for both the design phase of this system, as well as the actual implementation at SGI. The patent expiration was a deciding factor because licensing issues would have added complexity and slowed down the development process for this security system.

2.1.5 Limited Bit Size

There is only a limited amount of space available on the brick's EEPROM, which is where the public key will be stored. The space on the EEPROM allocated for the brick's public key is set at 512 bits. This limits the security of the system, as the more bits

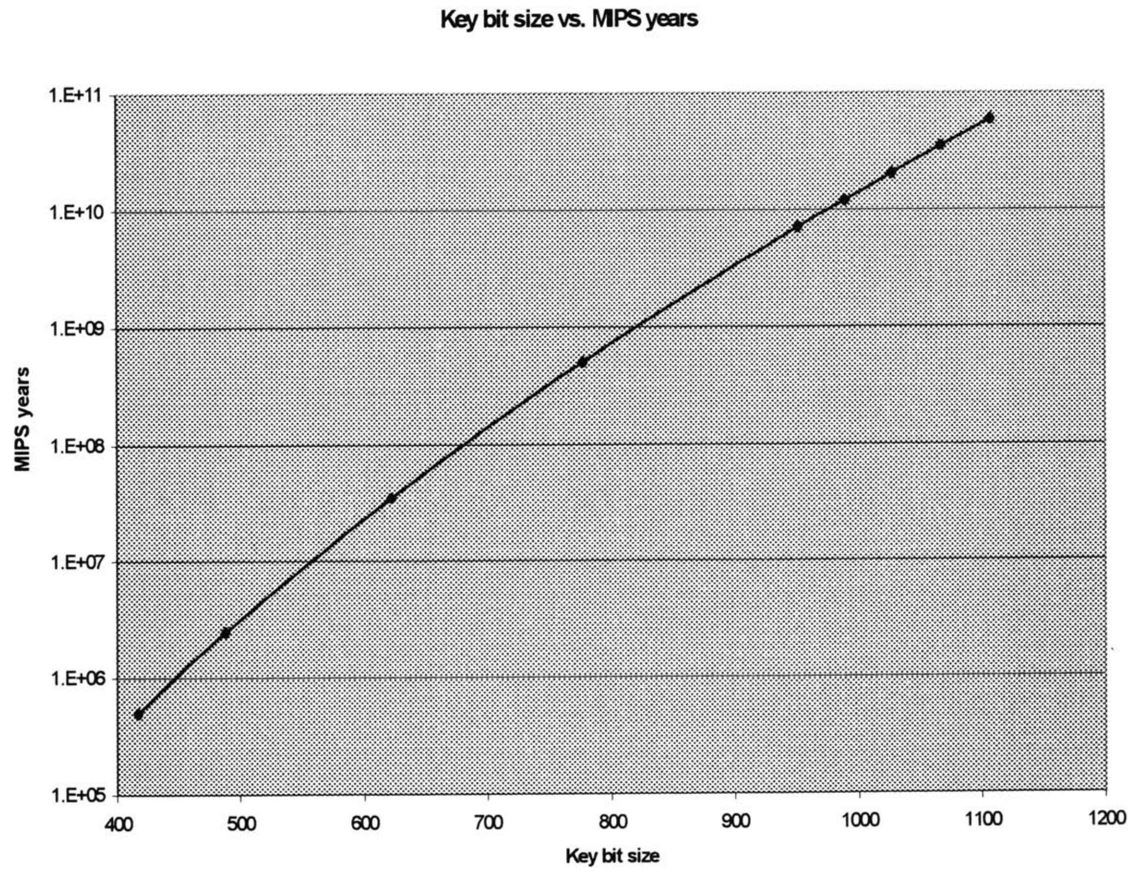


Figure 2.0: Graph of key bit size versus the number of estimated MIPS years needed to crack it for an asymmetric key algorithm. The larger the key size, the more MIPS years it will take to crack it. Data source for graph is from [4].

in the key, the more secure the system is. A key of 512 bits is considered pretty standard, although commercial software is now moving towards 1024 bits per key. (A key size of 512 bits is estimated to take something on the order of 10^7 MIPS years to crack, whereas a key size of 1024 bits is estimated to take something on the order of 10^{10} MIPS years to crack [4]).

The 512 bit key size is enough to suit the purposes of this project, however. It is a necessary trade-off because extra space on the brick's EEPROM is scarce, and allowing a larger public key on the EEPROM would be sacrificing another necessary field of data on the brick. In addition, it is also important to remember that the threat model here is not the typical "hacker" who is extremely knowledgeable and skilled at breaking into systems, but rather the owner of the system who might try to modify the system serial number so that he does not have to pay another licensing fee for expensive software -- *if it is not too difficult to do so*. Therefore, whether the public key is 512 bits or 1024 bits does not matter so much, since the typical person as identified in our threat model would not be able to understand how to crack a cryptosystem at any rate, regardless of the key size.

2.2 Security Versus Ease of Use

In designing any system, the intended audience must be considered. This is especially true for a security system, in which unexpected errors accidentally introduced by authorized users can compromise the security of the system. Security systems need to be designed to be easy to use and operate, as well as robust enough to tolerate errors from the average user [2]. Thus a simple approach is often desired.

2.2.1 Resurrecting Duckling Model

This security model will be approached with a slant on the "resurrecting duckling model" [1]. In the resurrecting duckling model, the first thing a duck sees becomes its mother, in its mind. This is called imprinting. In a process similar to imprinting, the first thing an item connects to will be identified as its owner. To change ownership, the item would need to "commit suicide" and then subsequently be "resurrected" -- ready to imprint -- when it is connected to a new owner for the first time. In this case, the system serial number of the brick will be analogous to this model of ownership.

This model is an elegant solution to the current problem. For one thing, it makes the process of brick replacement a very simple, seamless job on the part of the service person. The service person first enters the authenticator together with a request to the brick to set itself into an "imprintable state." Then, he merely needs to hook up the brick to the server with the new system serial number, and it should self-set the new system serial number by gleaning the information from its new connection. Besides its ease of use and installation, imprinting can also help to prevent errors that come about when incorrectly typing in the wrong system serial number.

2.2.2 Test Mode

A "test mode" for turning the security off is also important from a manufacturing and repair site point of view. At a repair or manufacturing site, a brick can go through a process in which it is connected to several test machines. When connected to these machines for testing, it is a hindrance on the part of the personnel to be required to attain an authenticator and remember it for each brick at each different test station. Therefore, it would be more convenient and efficient if the brick could go into a test mode at the beginning of the repair or manufacturing process, and then turn off the test mode at the end of the cycle, when it is about to ship out to the customer. The test mode would allow personnel to turn off the security on the brick so that it can be plugged and unplugged from multiple test machines without wasting time or causing difficulties for the personnel. This test mode helps to ensure that the process does not become too complicated from extra steps at the repair or manufacturing site, and can be achieved with a bit set in the hardware.

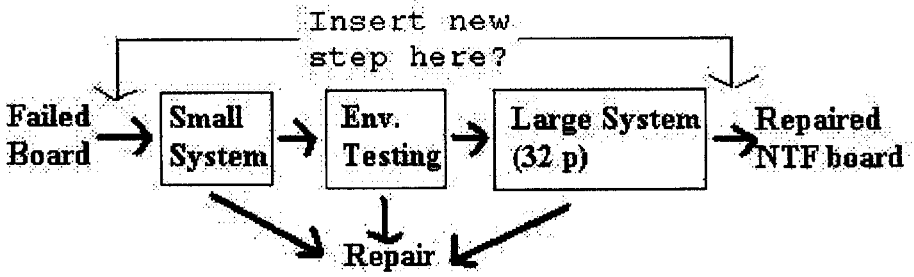


Figure 3.0: Sample flow diagram of the repair cycle. The proposed test mode would only add a "turn on" and "turn off" step at the beginning and end of the cycle, rather than adding a "get authenticator" step at the beginning of each block.

2.3 Security Versus Ease of Implementation and Integration

Another aspect in designing a security system for a corporate audience is that often times, the system can not be designed from scratch. This constrains the number of possible solutions to the problem at hand. There will usually be some corporate computing infrastructure already in place, so rather than scrap it and design a completely new system, the final design has to deal with plugging up the security holes of the existing system and building on top of it. The design of the system must take into account certain factors like ease of integration that would not be a problem for a design built from the ground up.

2.3.1 Easy for Manufacturers to Distribute

One thing to think about in regards to the integration of the system would be, is it feasible from a manufacturing standpoint? In this case, the system should be able to handle approximately 25,000 newly manufactured bricks a year. (This is a generous estimate). The system should be easily maintained, available for upgrades, and durable enough to last for the lifetime of the system.

The system must be simple for people in manufacturing to use. It must also be fault tolerant and able to handle the maximum capacity of bricks per year. The proposed solution to this would be to annually supply the manufacturer (Celestica, an electronics manufacturing company headquartered in Toronto, Canada) with CD's containing blocks of approximately 25,000 brick id (identification) numbers and corresponding pre-generated public keys. This can be done in the same fashion as Ethernet addresses are currently supplied to manufacturers today, as both deal with the placement of a unique number on each server component. This solution reduces some complexity for the manufacturer, because they do not need to worry about learning how to use the algorithms to generate key pairs at the plant, nor do they need to deal with sending the corresponding private keys to the support center -- which they would have been forced to do had they generated the key pairs at the plant. (The key pairs will be generated at the central server, and only the public key list will be sent out to the manufacturer). The manufacturers will just get their blocks of public keys from the CD's shipped to them, and they can then use this information in the same manner that they are already accustomed to as with the Ethernet addresses.

2.3.2 Easy to Maintain, Update, and Repair

The system needs to be fairly robust, but on the chance that it breaks down, it needs to be easy to repair, upgrade, and maintain. This objective is reached by making the system modular in design. This allows certain aspects of the system to be changed without affecting other things. Therefore, this helps with the fault tolerance of the system.

2.3.2.1 Modularity in the Algorithm Choice

The algorithm used for the current system is the El Gamal variant of the Diffie-Hellman algorithm. However, if in the future, the algorithm needs to be changed (for example, if a new one comes along that is better, or if the patent on something like RSA expires) then the code is such that it is easily extensible to allow for a different algorithm to be used to create any future keys at that point. This can be integrated into the system by adding a field in the central server database that includes the algorithm used, or by logging system serial number batches with specific algorithms.

2.3.2.2 Maintenance of Keys

The keys are unique from one another. One key pair's private key does not work for another key pair's public key. Therefore, the key pairs do not affect each other. If the key pair for a particular brick had to be changed for some reason, it would not affect the other bricks. The system can generate a new key pair at any time. This pair would then need to be updated in the support center. The most difficult part of this task would be re-setting the public key on the brick, which would have to be done at a repair center or at the plant since the public key is on the brick's non-rewritable EEPROM.

2.3.3 Easy for Field Personnel to Attain Authenticator

The security of the actual system should also be relatively good, but not so impenetrable that it becomes a problem for service personnel to use. The trade-off for ease of use might mean diminished security, but as long as it is still reasonably secure, then it is better to try to tailor this model for the service personnel -- who are more concerned for speed and efficiency rather than security. The system has to be easy for personnel to attain an authenticator, otherwise they will ignore the system and try to keep the field system perpetually in "test mode."

2.3.3.1 Accessing the Central Support Server

It is this usability factor that also dictates that the system should have capabilities for telephone access as well as through the Web, in case service personnel are working out in the field and do not have Web access readily available. Access through the Web is built on top of the company's existing Websafe password mechanism for authenticating company personnel online. This provides the users with a familiar interface that does not add extra complexity to the picture because they already use the Websafe system and have existing accounts and passwords for it. For those without Web access, they can instead call the support center to get the authenticator from an operator logged onto the system.

2.3.3.2 Determining the Time Window

The time window for the authenticator expiration has been set at 10 hours. This time window was chosen for a few reasons. Although security would dictate that it should expire within as short a time period as possible (such as within only one hour, for example), the perspective of the service personnel who would be using the authenticators had to be considered. Some of them would not have Web access while at the customer site; they would just attain the authenticators in the morning before they arrived at the customer site. Therefore, although the actual usage time of the authenticator would not be very time-consuming in terms of the time it would take to actually input it into the brick, the authenticator can not expire immediately if it will take the service person considerable time before he actually handles the brick. Therefore, the time window of 10 hours was chosen to cover the duration of an average workday, plus some travel time to the customer site. Thus the time window is set such that a service person would need to get a new authenticator every day, but not several times per day.

2.3.4 Difficulty of Conforming with Export Laws

Although the U.S. is getting more lenient in their export law policy, any system that has any involvement with encryption algorithms is subject to possible problems with export or import control departments from various countries, an important thing to note since SGI has sites around the world. There are a few things that can be done to lessen the chance of this, however. The most important thing is to emphasize that this is strictly an authentication system for verifying a valid system serial number modification request, rather than an encryption system used for private data transmissions; confidentiality and privacy are not issues here.

It is also important to note the difference in terminology between "encryption" and "decryption" and "signing" and "verifying." Encryption and decryption can deal with data transmission sent in an encrypted form, whereas signing and verifying deal strictly with authentication, to ensure the message authenticity. This is important in terms of the export policy; it is easier to export something that is merely an authentication tool than something that is used for encryption purposes.

3.0 LESSONS LEARNED

The researching of existing systems and the designing of the specifications for this security system were half of the solution. The other half hinged on the actual implementation of the system. This involved coding the different modules, working with the users and future managers of the system to ensure that it would work properly within the confines of the infrastructure, and reworking any elements that proved problematic. In short, after working out the entire system design in theory, certain lessons were to be learned from actually trying to implement this design in practice.

3.1 Implementation Details

This system was originally implemented as a small beta test program at SGI. The system was first sketched out in pseudocode, then implemented as a set of programs running on different modules. The different modules were used together to model the interactions of the different components of the system.

It should be noted that the first test run of the system did not actually involve bricks themselves because it was not possible to individually manufacture the bricks with individual "test" system serial numbers and public keys. Rather, the bricks were simulated by a separate shell on the computer that was running the verification code that would eventually reside on the bricks.

3.1.1 Computing Environment

The system was developed on the Microsoft Windows NT 4.0 operating system. The system was developed in NT rather than IRIX, which is SGI's "flavor" of the UNIX operating system; this was because the system was developed on a Dell Latitude CPi laptop computer running NT, similar to those that SGI personnel would operate in the field. As described in more detail later, the final design for the system involves using the Web, but when the problem first arose, different types of solutions were explored, including non-Web-based models which might run locally on the SGI service person's machine itself. Therefore, it was for matters of uniformity in platform and computers that the development was done on this particular type of computer and its associated operating system.

The programs were developed using the Microsoft Visual C++ Professional package for NT. However, the code itself was actually written in the more efficient C programming language, which is standard in most of SGI's software applications. The reasons for using this particular package, rather than using some open source C compiler were twofold: partly because of the particular operating system of NT instead of UNIX, and partly because of the corporate environment. Since this system would actually be implemented in the company's infrastructure, it was advised to go with a supported package like Visual C++ rather than some unsupported open source freeware compiler. This fact, coupled with the fact that the actual ordering and shipping of the software package considerably slowed the initial development of the project, illustrate the difference between developing something in a corporate environment versus a purely academic one. A large corporate environment with "red tape" is rarely as efficient during the software development process as an academic environment is during the exact same process.

3.1.2 System Requirements

This system will not add too many new elements, as it is designed to take advantage of the existing infrastructure instead. There will be an authenticator generator on the support center server, which is where the service person will log in to obtain the signed authenticator. The company already has such a support center with a high availability server, so this is just a case of adding more responsibilities, rather than adding an entirely new department. Also, as mentioned before, no new Web authentication is necessary; the existing Websafe method can be used in conjunction with this system. On the brick, there will be a microcontroller and some new information. The microcontroller will verify the authenticator to allow access to modify the system serial number. The new information on the brick will be limited to the public key in the EEPROM. As described in more detail later, the brick also stores the brick id number in the EEPROM and the system serial number on the NVRAM, but this information would have been stored on the brick regardless of the authentication system, so the only new addition is the public key.

There is nothing secret in the message, so it is not necessary that the transmission system provide privacy, nor do we require privacy of the information on the brick itself. What is required is that the integrity of the modification request be assured, which is indeed provided via the digital signature. The message returned from the server is useless to anyone else because it is signed with the brick's private key – only the corresponding public key can verify this, and only for a short time within the predetermined time window. The most important thing to protect is the database of private keys located at the support center server, and this should already be protected because of the other information on the server.

3.1.3 Protocol

Outside of the initial setting of the system serial number in the manufacturing plant or at a repair site, the only other times the system serial number would need to be reset would be in the field when a certified service person must swap in a brick to replace a broken brick on another system. The protocol necessary to gain access to do this modification is as follows:

1. The service person will log into the server at the support center where all the private keys are located. The login menu will only allow those service personnel on the access control list with the proper company network usernames and Websafe passwords to log into the system. This is the first checkpoint.
2. Once logged in, the service person sends a system serial number modification request to the server at the support center. The request will contain the brick id number. There will then be a check to ensure that the user is authorized to work on the brick with that particular brick id number, based on the access control lists built into the system.
3. The server at the support center will search an extensive database for the corresponding private key given a particular brick id number. The request will be logged and then a GMT (Greenwich Mean Time) time-stamp, signed with the brick's private key will be sent back; this will be the authenticator that is displayed to the

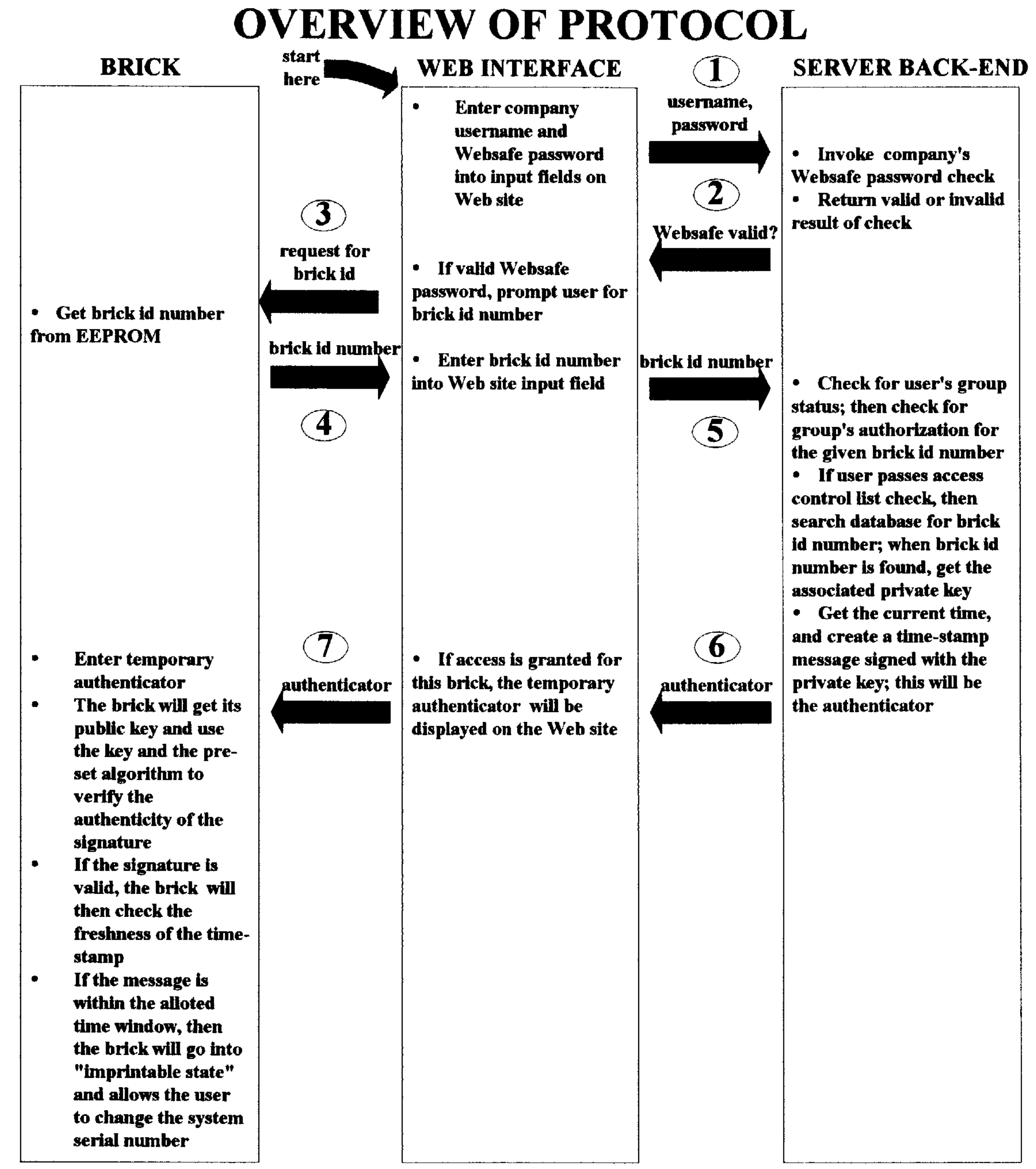

OVERVIEW OF PROTOCOL
BRICK
start here
WEB INTERFACE
SERVER BACK-END
1
username, password
• Enter company username and Websafe password into input fields on Web site
• Invoke company's Websafe password check
• Return valid or invalid result of check
2
Websafe valid?
3
request for brick id
• If valid Websafe password, prompt user for brick id number
• Get brick id number from EEPROM
brick id number
4
• Enter brick id number into Web site input field
brick id number
5
• Check for user's group status; then check for group's authorization for the given brick id number
• If user passes access control list check, then search database for brick id number; when brick id number is found, get the associated private key
• Get the current time, and create a time-stamp message signed with the private key; this will be the authenticator
6
authenticator
• If access is granted for this brick, the temporary authenticator will be displayed on the Web site
7
authenticator
• Enter temporary authenticator
• The brick will get its public key and use the key and the pre-set algorithm to verify the authenticity of the signature
• If the signature is valid, the brick will then check the freshness of the time-stamp
• If the message is within the alloted time window, then the brick will go into "imprintable state" and allows the user to change the system serial number

Figure 4.0: Overview of the protocol for attaining an authenticator.

service person. Note that the brick's private key itself is not sent back. (The authenticator will be in the form of numbers instead of letters. This is because the system will be used internationally, and foreign users will be more familiar with numbers than with letters, thus cutting down on errors in remembering and properly entering in the authenticator.)

4. The service person will input the authenticator to the brick in question. The brick will then use its public key to verify the signature. Once that is verified, the local time on the brick will be converted to GMT, and then this will be compared with the signed time-stamp message. If it is correctly authenticated, and the brick time is within the time window of 10 hours from the time-stamp (indicating that it has not yet expired), then the brick will allow access to the superuser menu on the brick which commences "imprintable state." In the future, other options might also be added to the superuser menu.

3.1.4 Architecture of the System

The system has thus far been described only in terms of how it will function and interact with the user. Although it is important to be explicit early on regarding these particular design goals, in designing a system it is also imperative to be explicit about how it will be composed, and to describe how each component will actually work in practice alongside the other components. The architecture of the system should be clear, modularized, and easily extensible. These concrete details are necessary for the final specification of the system before it is put into the implementation stage.

3.1.4.1 Modules

This system is comprised of several components. First, there is the protected central company server. This server is the central authority responsible for many things.

First, it will house the key generation program. Its operators will be in charge of generating batches of key pairs every year, and of updating the internal database of new brick id numbers with the new private keys, while also distributing the new public keys to the manufacturers of the hardware. This server will also run the back-end for the authentication Web site. Server operators will also be responsible for updating access control lists for certain bricks. The server will use its database of brick id numbers and private keys to generate time-stamps with digital signatures for authentication. Of course, it should be noted that when the system is deployed on a large-scale internationally, this "central" support server will actually be composed of several corporate high-availability servers in different geographic regions, rather than only just one.

A second component of this security system is the Web interface. This will also be the Web interface that a phone operator will use to access system serial number authenticators at company call sites, if a service person does not have access to the Web in the field. The Web interface will be the interface between the authentication requests and the subsequent temporary authenticators for SGI personnel in the field who are repairing or swapping bricks. The Web interface will incorporate the company's pre-existing Websafe online employee identification system, before the employee is allowed to enter any requests for specific brick authenticators. The Web interface will also connect to the program running on the back-end of the support center server, and this is the interface that will display the temporary authenticator to be used within the time window by the employee for a particular brick.

The final component can not be overlooked; it is the brick itself. The brick houses the system serial number, brick id number, and public key. It also will contain microcode

MODULAR DEPENDENCY DIAGRAM

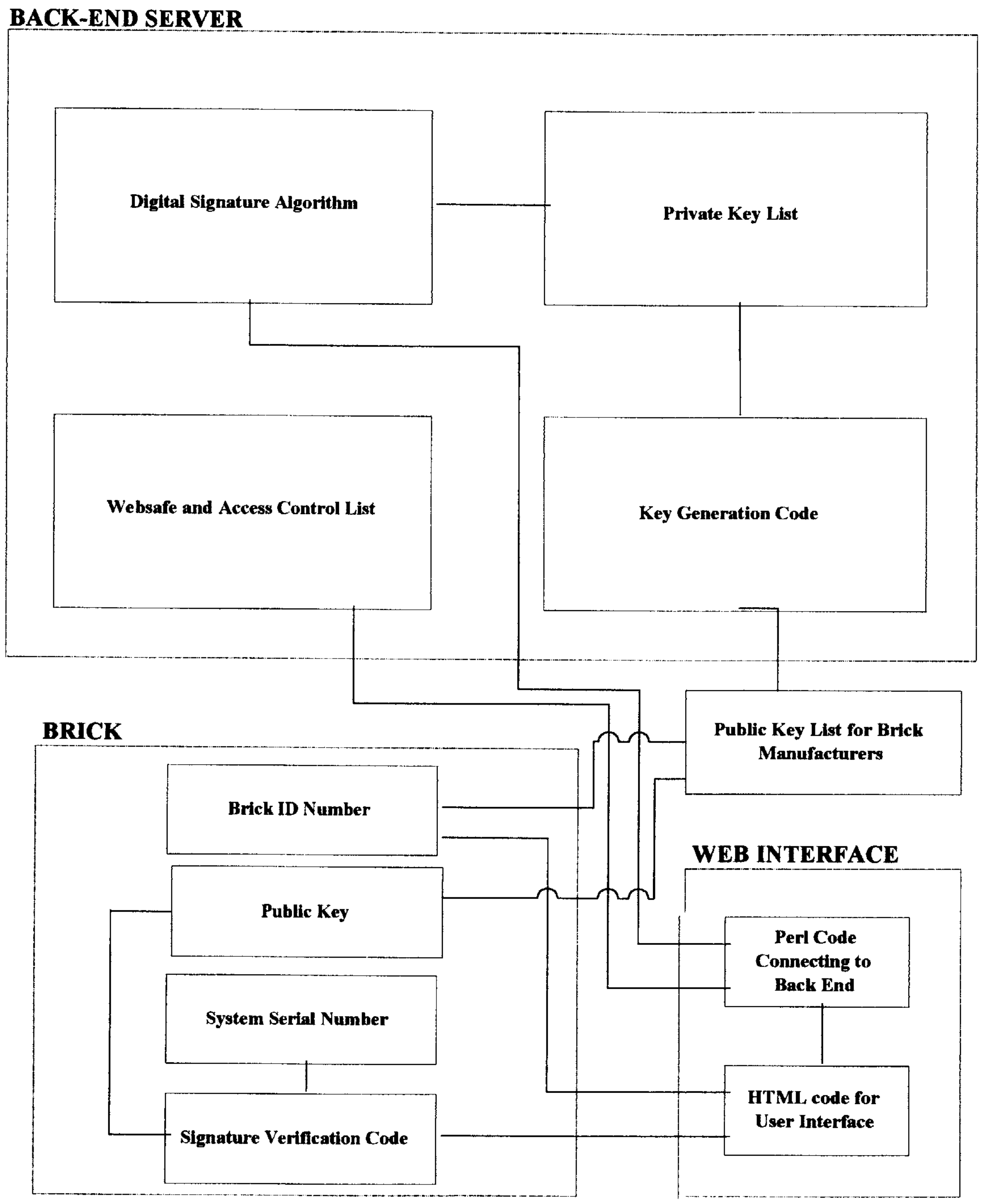

BACK-END SERVER
Digital Signature Algorithm
Private Key List
Websafe and Access Control List
Key Generation Code
Public Key List for Brick Manufacturers
BRICK
Brick ID Number
Public Key
System Serial Number
Signature Verification Code
WEB INTERFACE
Perl Code Connecting to Back End
HTML code for User Interface

Figure 5.0: Modular dependency diagram for this authentication system.

that uses the public key to verify the authenticator, as well as the local clock. Lastly, it will have a small "resurrecting duckling" program running on it that will cause the brick to go into "imprintable state" once the authenticator has been approved.

3.1.4.2 Algorithm Details

As stated earlier, the algorithm used for the authentication is a widely-known variant of the Diffie-Hellman public key algorithm known as El Gamal. The algorithm is dependent on the *difficulty of calculating discrete logarithms*. The algorithm can be used

Key generation

- Generate a large random prime $\boldsymbol{p}$ and a generator α of the multiplicative group Z^*_p
- Select a random integer $\boldsymbol{a}$, such that $1 \leq \boldsymbol{a} \leq \boldsymbol{p} - 2$
- **Compute** $y = \alpha^a$ ***mod p***
- A's public key is $(\boldsymbol{p}, \alpha, \boldsymbol{y})$; A's private key is $\boldsymbol{a}$.

Signing

- Select a random secret integer $\boldsymbol{k}$, such that $1 \leq \boldsymbol{k} \leq \boldsymbol{p} - 2$, and $gcd(\boldsymbol{k}, \boldsymbol{p} - 1) = 1$
- Compute $\boldsymbol{r} = \alpha^k \bmod \boldsymbol{p}$
- Compute $\boldsymbol{k}^{-1} \bmod (\boldsymbol{p} - 1)$
- Compute $\boldsymbol{s} = \boldsymbol{k}^{-1} \{h(\boldsymbol{m}) - \boldsymbol{ar}\} \bmod (\boldsymbol{p} - 1)$
- A's signature for the message $\boldsymbol{m}$ is $(\boldsymbol{r}, \boldsymbol{s})$

Verification

- Receive message $\boldsymbol{m}$ signed with $(\boldsymbol{r}, \boldsymbol{s})$
- Obtain A's public key $(\boldsymbol{p}, \alpha, \boldsymbol{y})$
- Verify that $1 \leq \boldsymbol{r} \leq \boldsymbol{p} - 1$
- Compute $\boldsymbol{v_1} = \boldsymbol{y^r r^s} \bmod \boldsymbol{p}$
- Compute $h(\boldsymbol{m})$
- Compute $\boldsymbol{v_2} = \alpha^{h(\boldsymbol{m})} \bmod \boldsymbol{p}$
- Verify that $\boldsymbol{v_1} = \boldsymbol{v_2}$

Figure 6.0: Explanation of signing, verifying, and generation of keys with the El Gamal algorithm. Source for the algorithm from [5].

for message encryption and decryption, but since authentication is the purpose here and not message confidentiality, the algorithm will only be used for digital signature generation and verification. The message, which will be a simple time-stamp, can be sent "in the clear." If the time-stamp is altered while in transmission, the digital signature – which is calculated with the time-stamp message – will not work.

3.1.4.3 Code

For proprietary reasons, the code is not included, but it is described in detail here. The code is split up into six "black box" abstractions. First, there are three pieces of code for the El Gamal algorithm. These include the key generation code, the code for the creation of the digital signature, and the code for the signature verification on the brick. The key generation code utilizes a random number generator to create unique, non-deterministic key pairs. The verification code then takes the existing public key that is present on the brick, and uses it to verify the digital signature added to a message, using the published mathematical algorithm steps outlined above. The code for the digital signature creation utilizes a list search mechanism to identify the private key associated with the input brick id number. Once the private key has been found, the code then retrieves a time-stamp to create a digital signature, again using the published algorithms above. Following this, there is also code to allow the brick to enter "state 0." It also has to query its new neighbors to glean its new system serial number.

In addition, there are two more pieces of code that make up the access control lists module; one is for adding new users to specific authorization groups, and the other is for adding group permissions for specific individual bricks or blocks of bricks. These two

Code Abstractions

Server Back-End

Key Generation

- **Input: number of new keys to be generated;**
- **Use random number generator to seed new keys;**
- **Use El Gamal key generation algorithm to create new keys;**
- **Output: list of new keys;**

Access Control

- **Input: username, group number, brick ID number;**
- **Check Websafe username and password;**
- **Search for group associated with username;**
- **Search for brick ID number range associated with group;**
- **Check for input brick ID number in this range;**
- **Output: authorization status -- ok to create authenticator;**

Digital Signature Generation

- **Input: username, brick ID number;**
- **Search for brick ID number in internal database;**
- **Get private key associated with brick ID number;**
- **Get current time for time-stamp message;**
- **Use El Gamal signature algorithm to sign message with private key;**
- **Log current time, brick ID number, and username;**
- **Output: time-stamp message signed with private key (authenticator);**

Web Interface

Perl Code to Connect Back-End to the Web

- **Input: username, Websafe password, brick ID number;**
- **Invoke back-end server functions with the above arguments;**
- **Get server response;**
- **Output: authenticator;**

Brick

Digital Signature Verification

- **Input: authenticator;**
- **Get public key from EEPROM;**
- **Use El Gamal verification algorithm with public key to verify authenticator;**
- **Get current time and check freshness of time-stamp message;**
- **Output: verification status of signature;**

Access to Change System Serial Number

- **Input: state change command;**
- **Change to imprintable state;**
- **Output: ready to receive new system serial number;**

Figure 7.0: Code abstractions for the different modules of the system.

sets of code probably will not be utilized very often in the early stages of the deployment of the system. In the beginning, it might not be essential to assign group permissions to certain types of bricks, so initially, the group permission check will probably be a transparent, unnoticed security check in the system that grants permission to all users with the standard company Websafe access. However, if needs change in the future, then these authorization groups can be used to allow access to certain restricted bricks or other commands. The idea behind this was that it would be hard to build access control lists into the system at a later stage when something like this is needed; it is easier and much simpler to just build it into the infrastructure now so that this system is modular and easily extensible.

Lastly, there is the HTML and Perl code for the Web page interface. This code simply acts as a simple and easily accessible way for SGI employees in the field to be able to connect to the support center server in a secure remote fashion. The code basically takes in the inputs for the Websafe company identification system. Then, it invokes the access control list code to check the employee's group status and group authorization for the brick in which authentication has been requested. If this check is also passed, the back-end digital signature code will then be called upon to generate a temporary authenticator. The Web site will then output this authenticator to the user for entering into the brick.

3.2 Design Changes and Surprises

When designing a system, it is virtually impossible to get everything completely correct the first time around. Almost always, there will be latent factors that become manifested in the actual implementation, although they were originally unforeseen when the system was first being designed. It is therefore valuable to allow time for troubleshooting and slight revisions of the original design during the testing phase of a system design cycle.

3.2.1 Feedback from People Involved

The system received feedback from all sorts of personnel who would be affected by the new design. The system serial number was a relatively small component of the overall design of the new high-end server, but even such a small item as this had a huge "ripple effect," since the new server was so complex and far-reaching. There was feedback from both those who would be implementing the new design to those who would be using the new design in every day practice.

First, the manufacturers were agreeable with the change and the protocol for its implementation on their end. They were complacent as long as the responsibility of the key generation would be up to the company and not up to them. This was a modular issue to them; to the manufacturers, it was just another component that required a unique identifier supplied to them by the company, such as Ethernet addresses. Therefore, they had no major gripes about it.

The firmware engineers in charge of the system controller had other issues that concerned them. They were interested in how much space the public key would take on the brick, because the brick's non-rewritable EEPROM had limited space. Initially, the space allotted for the public key was much less, about 128 bits. When some concern was expressed to them regarding the low security of this, they rewrote the EEPROM specifications to allow for up to 512 bits for the public key, a much greater improvement. The firmware engineers were also in charge of the system controller microcode, and they were generally happy with the brevity of the verification code earmarked for this, since that would have to be added to the brick's limited space too.

The support center representatives were the most vocal in their concerns over the new system. First of all, the support center consisted of three different groups: field personnel, brick repair site technicians, and the actual support center operators. The field personnel had a lot of concerns over the ease of use for the system, and suggested some ways to make the system less confusing for personnel in the field. They were also concerned about the privacy of their Websafe passwords if they were unable to access the Web on-site and were forced to talk to a call center operator instead.

The repair site technicians had other problems with the design. Their main complaint was that they did not like the idea of having to obtain and enter in the authenticator for each brick that was being repaired at their center, *for each time the brick was connected to one of many test systems*. It would simply be too cumbersome and time-consuming for the repair personnel to have to add more steps to the repair and test process. These steps would be particularly extraneous because this authentication system was not prompted by a need to protect the bricks from the repair center personnel at their own private repair sites; there would be no threat of software theft at this repair location, and thus the security on the bricks at this point would be superfluous. Therefore, this feedback prompted the "test mode" addition to the design, in which the check for a system serial number different from the one on a brick's NVRAM would be disabled.

This way, the brick would still be able to function properly while being connected to another component with a different system serial number.

Lastly, the support center operators had their own worries about the new system. This new system added complexity for them because it increased the number of things they had to update and maintain as part of the support center. However, other than scrapping the entire system – which was not an option – this was an inevitable aftereffect. Once this was made clear, the support center operators focused on details to minimize their responsibilities and time constraints involving this system. This meant that certain considerations had to be made to ensure that the system would be easy to update; this was accomplished by the modular design of the system. The system was also designed to be self-service; anyone could log into the Web site and obtain an authenticator without a middleman. The only exception to this would be in the case that a field employee did not have Web access, in which case the call center could be utilized. Moreover, no call center operator would have to be specially trained to operate the authenticator generation program; the Web site itself has a familiar interface and a simple, clear, and explicit input/output mechanism (Appendix A shows screen shots of the Web site user interface for the login process, as well as for key generation and for group management of the access control lists).

Surprisingly, all of the concerns that were expressed dealt with non-security issues. Although there were some requests to lessen the security efforts for purposes of ease of use, no one requested even greater security measures than what was already proposed here.

3.3 Evaluation of Final Product

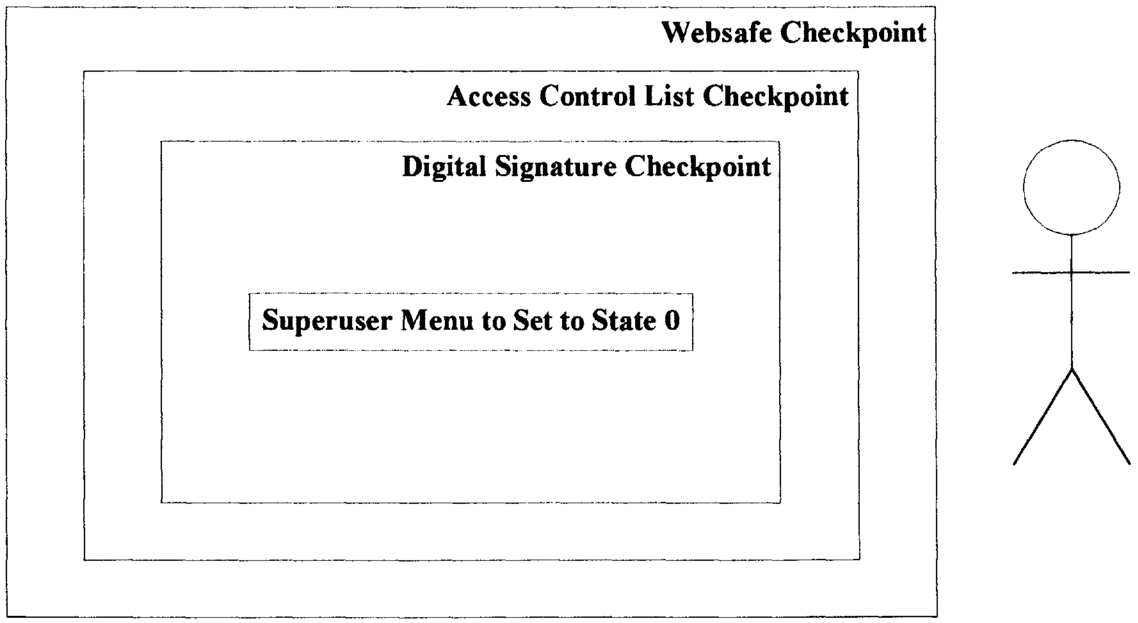
After some scrutiny and testing, the final product was handed off to the next level in the production cycle. The final product described here is a culmination of much input from the personnel involved. Nevertheless, despite any allowances and changes made for the convenience of the personnel, it can not be forgotten that the reason the system exists in the first place is for security purposes. Therefore, it is necessary to evaluate the final product in terms of its actual security.

3.3.1 Security Checkpoints

There are basically three distinct security checkpoints in this system. The first is the login menu on the server side. The identity of the service person is authenticated via their normal company Websafe password. This first checkpoint makes use of a pre-existing system that is already in place for company employees who login to internal Web sites with restricted access. The risks involved are the same as with any other Websafe Web site, and are therefore acceptable for this system.

The second checkpoint involves the access control list for each specific brick. This allows for certain users to have more power than others, and to limit the authority of users who should only have access to a select group of bricks. Only those on the access control list for a particular brick are authorized to access the system and generate the signed message for that particular brick. This is a necessary step for the future if newer versions of this system will require different access control lists not only for different bricks, but for different functions as well.

The last, most involved checkpoint is with regard to the access of the system serial number modification on a particular brick, or more specifically, the actual signing and verifying involved to set a brick to "imprintable state." The user is authenticated if he passes the first checkpoint login menu and successfully logs into the system on the company server, thus meaning he is one of the certified service personnel. At this point, the service person will then be authorized in the ACL to obtain an expiring authenticator for a particular brick. The process of signing a "go to imprintable state" message with an asymmetric algorithm is handled by the support center server. The digital signature from the support center tells the brick that the request to change the system serial number is an authentic one.


Websafe Checkpoint
Access Control List Checkpoint
Digital Signature Checkpoint
Superuser Menu to Set to State 0
Figure 8.0: Security checkpoints for this system.

In designing a security system, the overall security of the system is limited by the weakest of its components. Therefore, Websafe passwords should be kept confidential.

The access control lists should be protected from tampering. In addition, the database of private keys that lies on the support center server should also be properly safeguarded. A break in the security of a system needs only be at one small point, even if the rest of the system is secure, so care must be taken at the weakest links. In this case, that would probably be the last check, not because it would be easy to crack the algorithm, but because it is the most complex step, and the added complexity introduces more possibilities for attack. Thus, the security of the system falls mostly on ensuring that this step is properly executed.

3.4 Suggestions for Future Versions

In the future versions of this authentication system, there are a few changes which can be implemented to make the system more robust and multipurpose. For instance, more options can be added to the superuser menu that is accessed once a user is authenticated. This means that if there are other restricted commands that need to be added to the system, instead of adding new security protocols to the system, the new options can just be added to the superuser menu, and accessed in the same way that has already been established by this system for the "imprintable state."

3.4.1 Key Size and Algorithm

The size of the public keys are limited because they reside on a brick's EEPROM, but larger keys might be possible in the future, which would make the system more difficult to crack. In addition, the quality of the algorithm is also limited if the systems are used abroad, as there are the aforementioned export regulations. At this point it does indeed seem that this system will be used in servers both abroad as well as domestically. However, this might end up not being a very important concern, as the U.S. government is supposedly relaxing its export policy. So, if the time came that export regulations in all relevant countries were significantly reduced, or if patents on certain algorithms expired, then the system's modular design will be able to support the change to new algorithm implementations. Even if the key size limit was not changed, implementation of different types of algorithms might allow for a more efficient use of the space, since asymmetric algorithms require the largest keys [4].

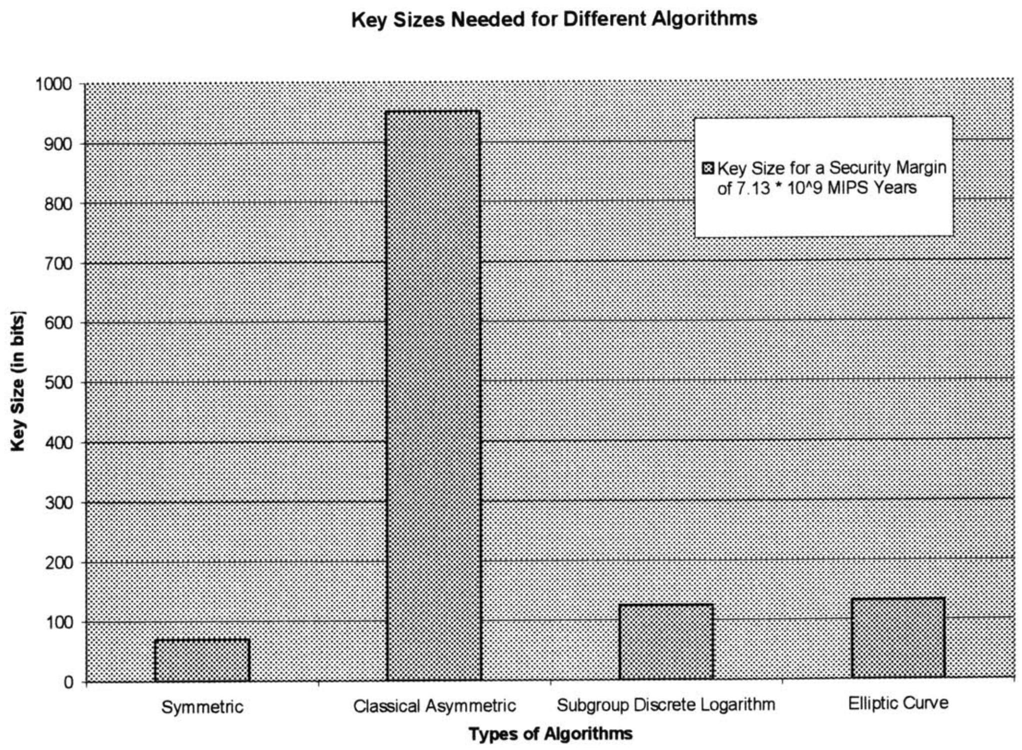


Figure 9.0 Graph of algorithm type versus key size, for an estimated "cracking time" of 7.13 x 10^9 MIPS years. Data source for graph is from [4].

3.4.2 Better Schemes for Randomness

Another point still under contention is the problem of making the keys random, but easy enough to manufacture on a large scale. Randomness is important because it means that the keys can be created in a non-deterministic manner that is difficult to reproduce or reverse-engineer. Currently, the keys are created using the library `random()` function, seeded with a combination of the current time as well as the batch number of the key series.

In addition, another problem that has arisen is that if there are different manufacturing sites, how does one determine if a public key has not been repeated between the various sites? How would random, non-repeating keys be generated on a large scale between different international manufacturing sites? The answer to this is that this really isn't a significant problem, the likelihood of repetition would be so small, that even if the two keys were exactly identical, the two systems with the identical keys would probably not be located at the same site. More importantly, just because it is known that there is a key that is sometimes reused, this does not make it easier for an attacker to figure out which key it is, or what its corresponding private key is; this does not make the system open for a known-key attack.

3.4.3 Arbitrary Precision Arithmetic Libraries

Another issue which is particularly problematic is the implementation of the algorithm for large numbers. This is because mathematical operations must be performed on numbers which exceed the usual 32 or 64 bits. To deal with this, arbitrary precision arithmetic libraries must be used.

In the initial implementation of the security model, artificially small numbers were used to deal with the fact that licenses for the arbitrary precision mathematics libraries were not yet obtained. Rather than create new arbitrary precision mathematics libraries and introduce the possibility of more errors into the system, it made more sense to just use existing libraries for the system. However, the licenses for the libraries could not be obtained in time for the purposes of the original prototype development cycle, and will be added in a later cycle.

3.4.4 Proxies

The authentication of personnel calling into the support center rather than logging into the Web site has posed the aforementioned problem of the security of Websafe passwords. Since some service people are uncomfortable with the idea of telling their Websafe password to a support center operator, the issue of proxies arises. One quick way of resolving this might be to have those service people set up a "phone-safe" password ahead of time to deal with this, but this issue probably needs to be explored more closely to deal with the cases where a phone-safe password was not pre-determined. In addition, this proxy usage could be reflected in the log, so that both the operator and the field personnel are recorded as having obtained authorization to modify the system serial number.

3.4.5 Direct Authentication from Web to the Brick

As already stated, the ease of use for the actual every day users of the system is an important factor to consider. One suggestion for making the system even more easy to use for SGI field personnel would be to eliminate the step of entering the authenticator into the brick -- that is, to go from the Web page displaying the authenticator straight to the verification input field for the brick without having to write it down and manually enter it. This might be accomplished a few ways. First of all, it is relatively easy to just make the system allow for the cutting and pasting of the authenticator from one field to the next. However, the brick itself might not have Web access. In this case, it might be more difficult to come up with a way for the Web interface (with the authenticator) to connect to the brick interface. Any modifications for this might first involve a check for the type of connection between the interfaces, and then some sort of "push" technology, depending on the constraints of the connection. Ultimately, though, this might not be possible if the brick and system with Web access are completely separated (due to firewalls, connection ports, etc.).

3.4.6 Clock Security

One specific area of concern is the security of the clocks that will be running on the bricks and the servers. The time window on the authenticators will not matter if the time on the clocks can be easily changed. If that were possible, then all a hacker would need to do would be to turn back the clock a few hours and continue using the otherwise expired authenticator, essentially making it a static authenticator. A few suggestions to fix this would be to make the time unchangeable on the bricks, or to limit the allowed time change to a very small amount, such as an hour or less. Another suggestion would be to have all the clocks synchronized to GMT, or to each other (although these are two features that are currently unimplemented). Lastly, one other idea would be to just require authorization; the service person tells the central system what time to use.

4.0 CONCLUDING REMARKS

The move from hardware interlocks to software interlocks is a wise choice both economically as well as for efficiency, but it is one that must be properly safeguarded and protected. The need for system serial number integrity is necessary for software licensing, and ultimately, it is important for the prevention of software theft. However, the problem is not a trivial one to solve.

The idea of a one-time use expiring authenticator in conjunction with digital signatures created by a public key cryptosystem seems to be a good fix for the problem at hand. It is fairly safe, yet not so stringent that it will tax the existing system infrastructure, and it will be a fairly simple system to use, operate, and maintain. The proposed security system will do away with the idea of static authenticators, but it will not make the generation of a new batch of authenticators a great chore for the service person.

The proposed plan also nicely makes use of the resurrecting duckling model. The brick will be owned at different points in time by different individuals. The different levels of permissions and the system of temporary ownership are both ideal for this type of situation, since the spare bricks can be swapped between different systems as needed. The imprinting feature will also allow for a more seamless approach to system serial number modification, and will reduce errors from typos and unlicensed serial number theft.

Lastly, the test mode will also make the system easy to implement in manufacturing and repair sites where our threat model is absent, and the Web interface makes the system easy for field personnel to use.

The system is currently being implemented in the newest breed of high-end servers coming out of SGI. Besides some initial concerns as mentioned early, there have yet to be any more major problems with the system design of the authentication system at the company; everything seems to be going well without fail.

5.0 ACKNOWLEDGMENTS

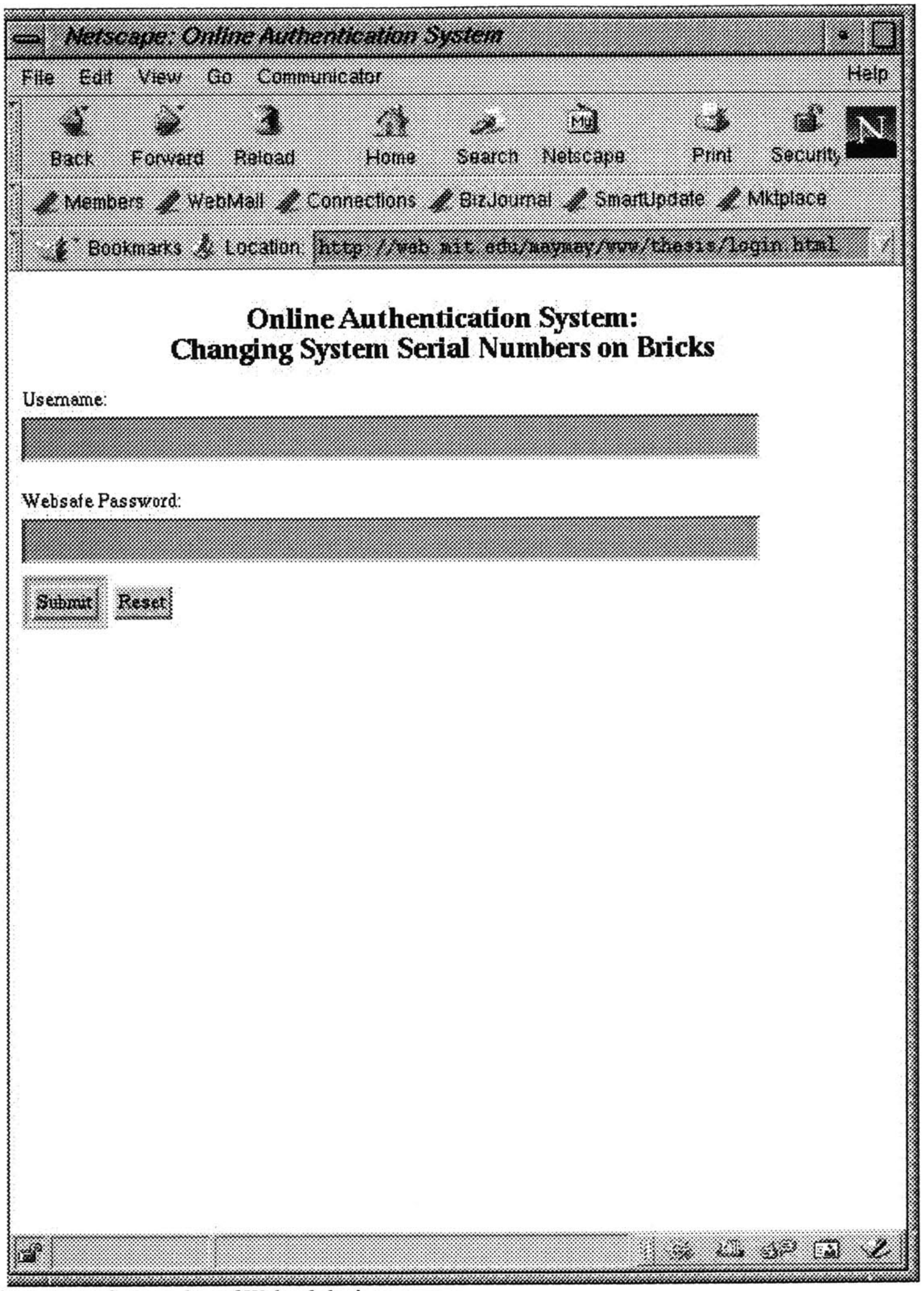
There are some people whose guidance, support, encouragement and inspiration helped to make this thesis a reality. First and foremost, I have to thank my thesis supervisor, Jerome H. Saltzer, who managed to impart good advice and guidance, no matter if he was miles away in Idaho or in the same 6.033 classroom!

Next I would like to thank Marty Deneroff and all the people involved in VI-A at SGI. As my manager as well as the VI-A company representative for SGI, Marty was the one who helped me find a thesis topic, and he provided the resources to accomplish this momentous task.

On a more personal note, I'd also like to thank those who encouraged me and kept my spirits up throughout the long process. I have to thank my parents, Bill and Lily Tse, for providing inspiration and always believing in me, as well as my sister, April Tse, for her support and "sympathy pains" as she struggled through the Masters thesis process at the same time alongside me. I also have to thank Jeremy Lin and Joyce Lee for providing some perspective and useful advice as past MEngers, and I'm especially grateful for their efforts to ensure that my life in CA did not consist entirely of my research at SGI! Lastly, I'd just like to thank all my friends, both in CA and at MIT, who patiently listened to me vent about my thesis and who encouraged me to keep going – there definitely is a light at the end of the tunnel!

6.0 APPENDICES

Appendix A: Screen shots of Web site user interface

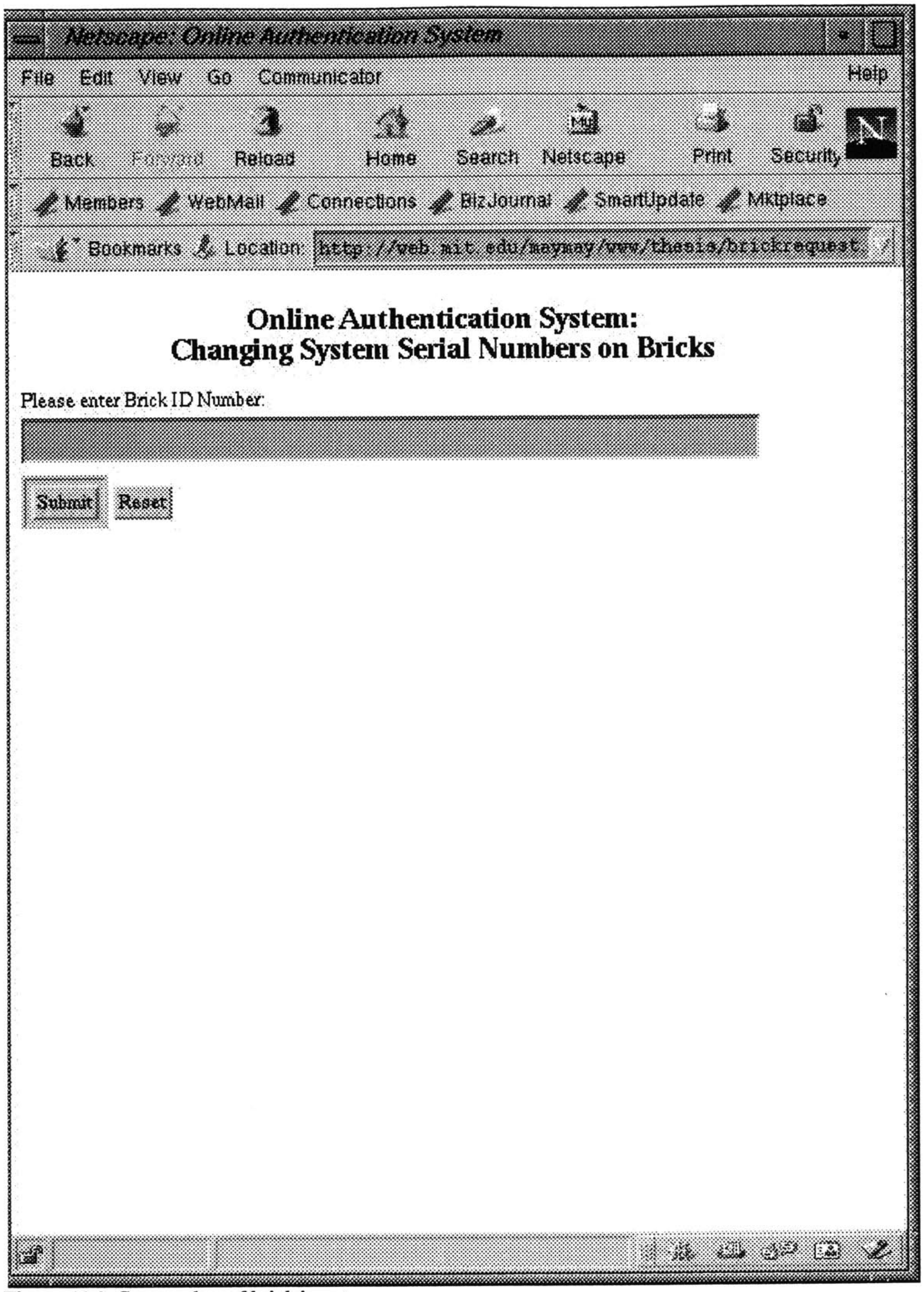

Netscape: Online Authentication System
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Members WebMail Connections BizJournal SmartUpdate Mktplace
Bookmarks Location: http://web.mit.edu/maymay/www/thesis/login.html

Online Authentication System:
Changing System Serial Numbers on Bricks

Username:

Websafe Password:

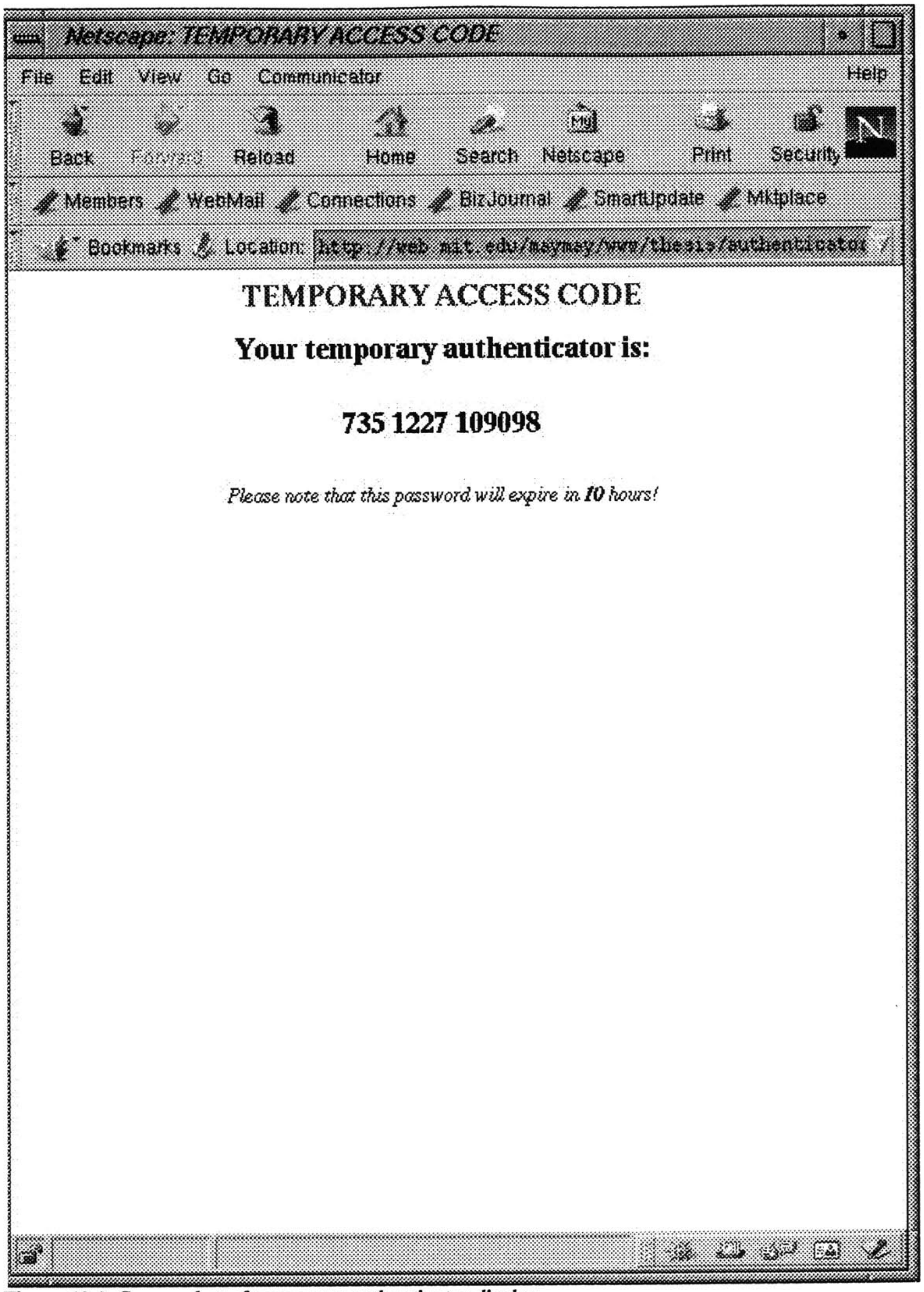
Figure 10.0: Screen shot of Websafe login menu.


Netscape: Online Authentication System
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Members WebMail Connections BizJournal SmartUpdate Mktplace
Bookmarks Location: http://web.mit.edu/maymay/www/thesis/brickrequest
Submit
Reset

Online Authentication System:
Changing System Serial Numbers on Bricks

Please enter Brick ID Number:

Figure 11.0: Screen shot of brick input menu.


Netscape: TEMPORARY ACCESS CODE
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Members WebMail Connections BizJournal SmartUpdate Mktplace
Bookmarks Location: http://web.mit.edu/maymay/www/thesis/authenticato

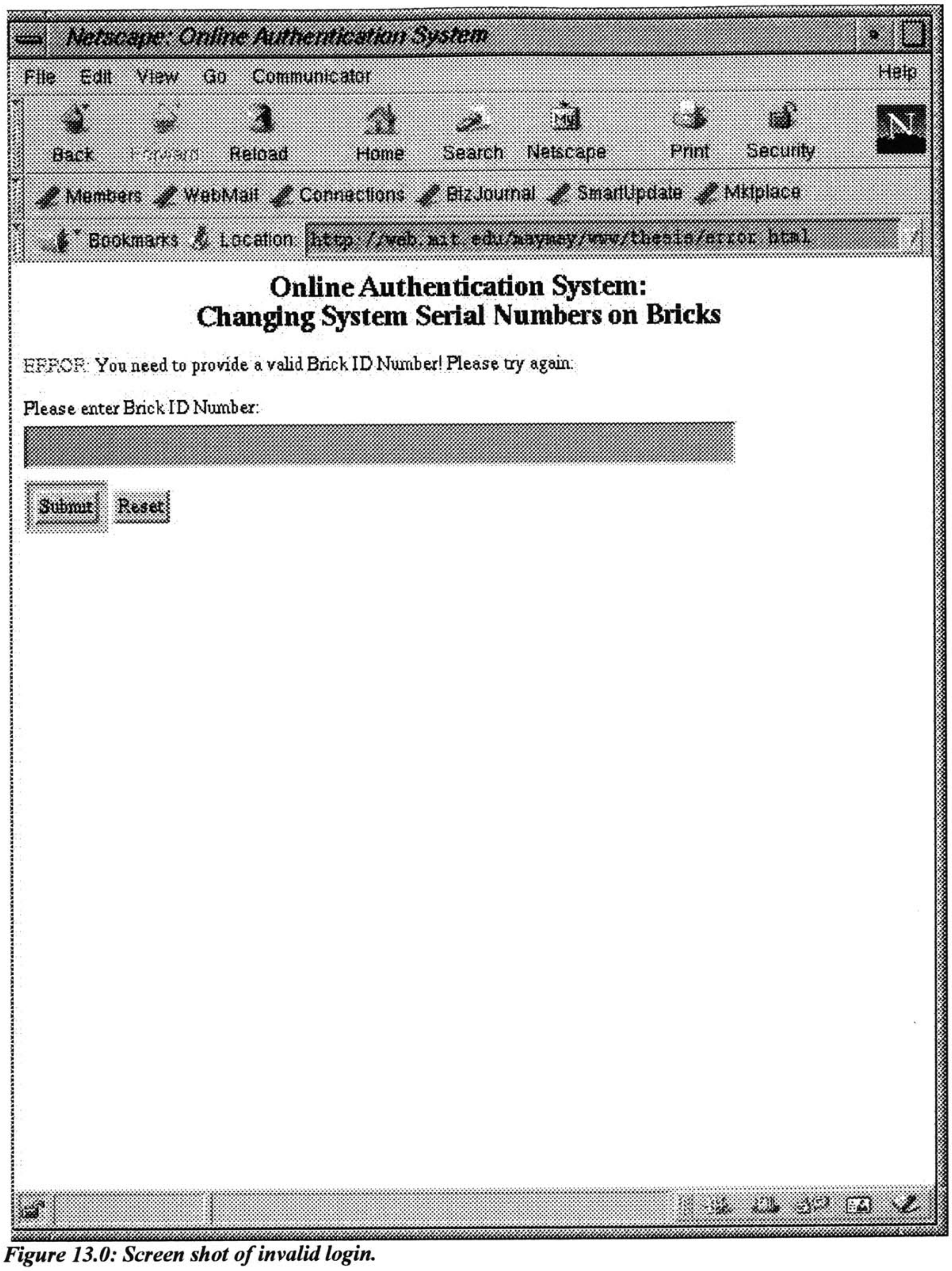
TEMPORARY ACCESS CODE

Your temporary authenticator is:

735 1227 109098

*Please note that this password will expire in **10** hours!*

Figure 12.0: Screen shot of temporary authenticator display.

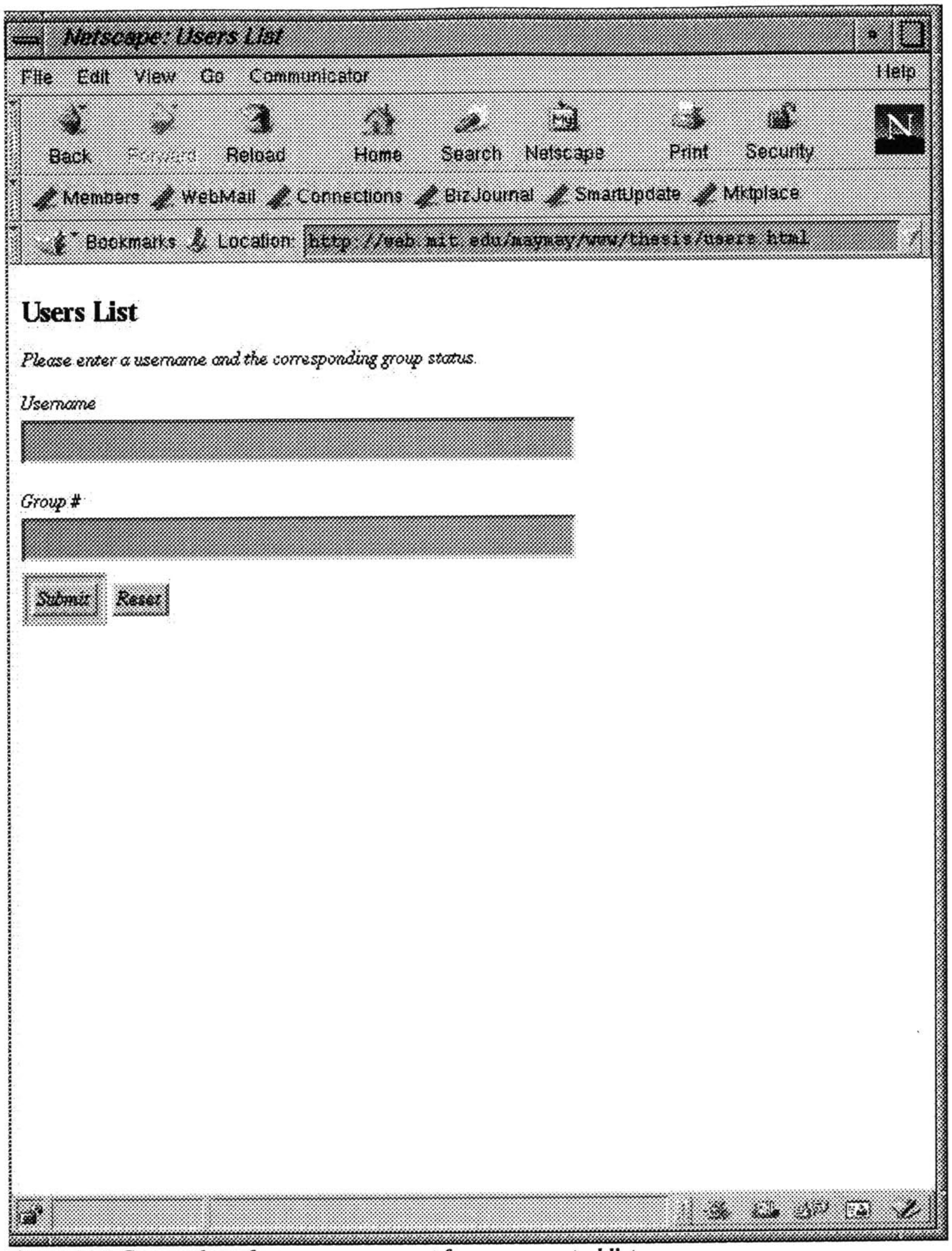

Netscape: Online Authentication System
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Members WebMail Connections BizJournal SmartUpdate Mktplace
Bookmarks Location: http://web.mit.edu/maymay/www/thesis/error.html

Online Authentication System:
Changing System Serial Numbers on Bricks

ERROR: You need to provide a valid Brick ID Number! Please try again:

Please enter Brick ID Number:

Figure 13.0: Screen shot of invalid login.


Netscape: Users List
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Members WebMail Connections BizJournal SmartUpdate Mktplace
Bookmarks Location: http://web.mit.edu/maymey/www/thesis/users.html

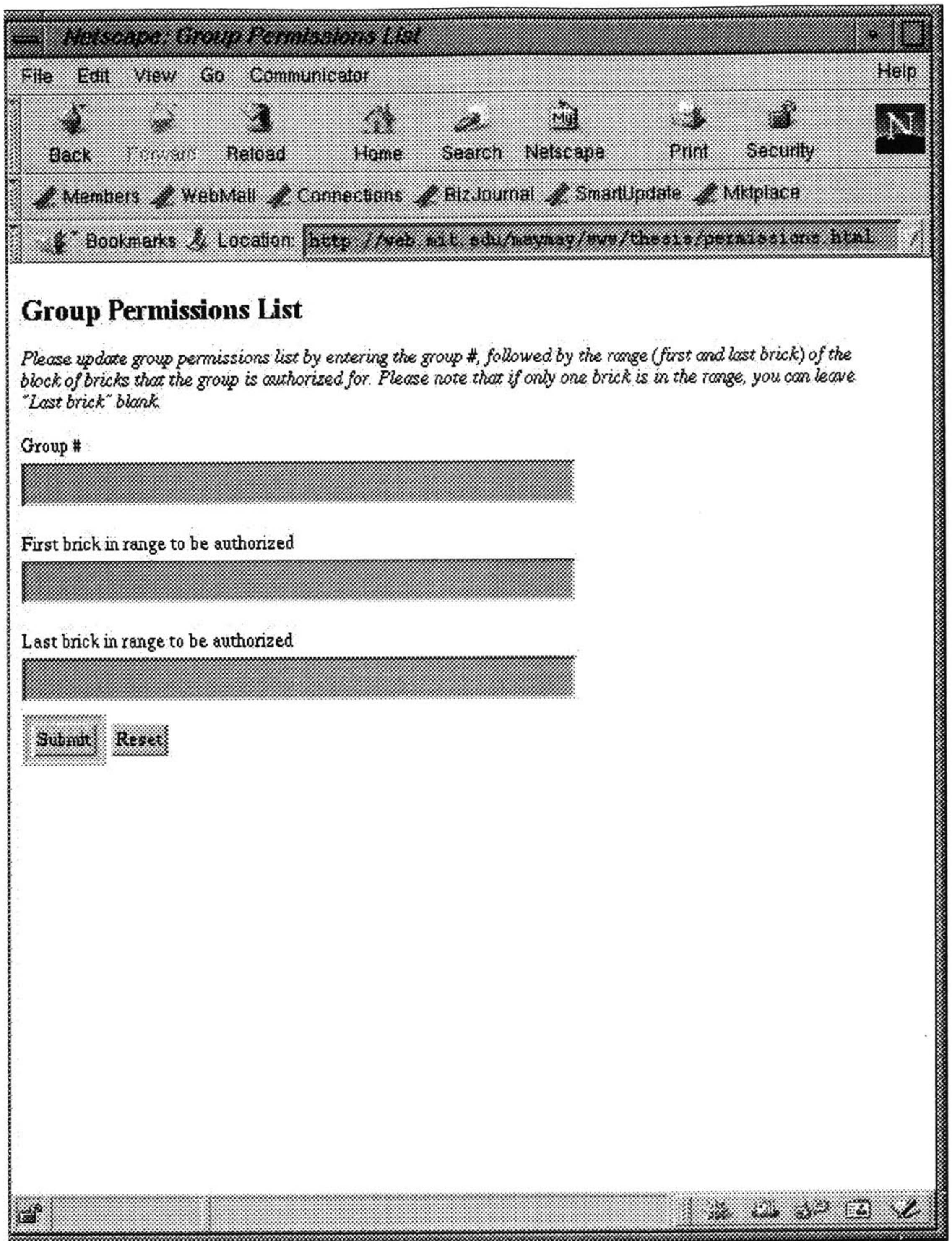
Users List

Please enter a username and the corresponding group status.

Username

Group #

Figure 14.0: Screen shot of group management for access control lists.


Netscape: Group Permissions List
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Members WebMail Connections BizJournal SmartUpdate Mktplace
Bookmarks Location: http://web.mit.edu/maymay/www/thesis/permissions.html

Group Permissions List

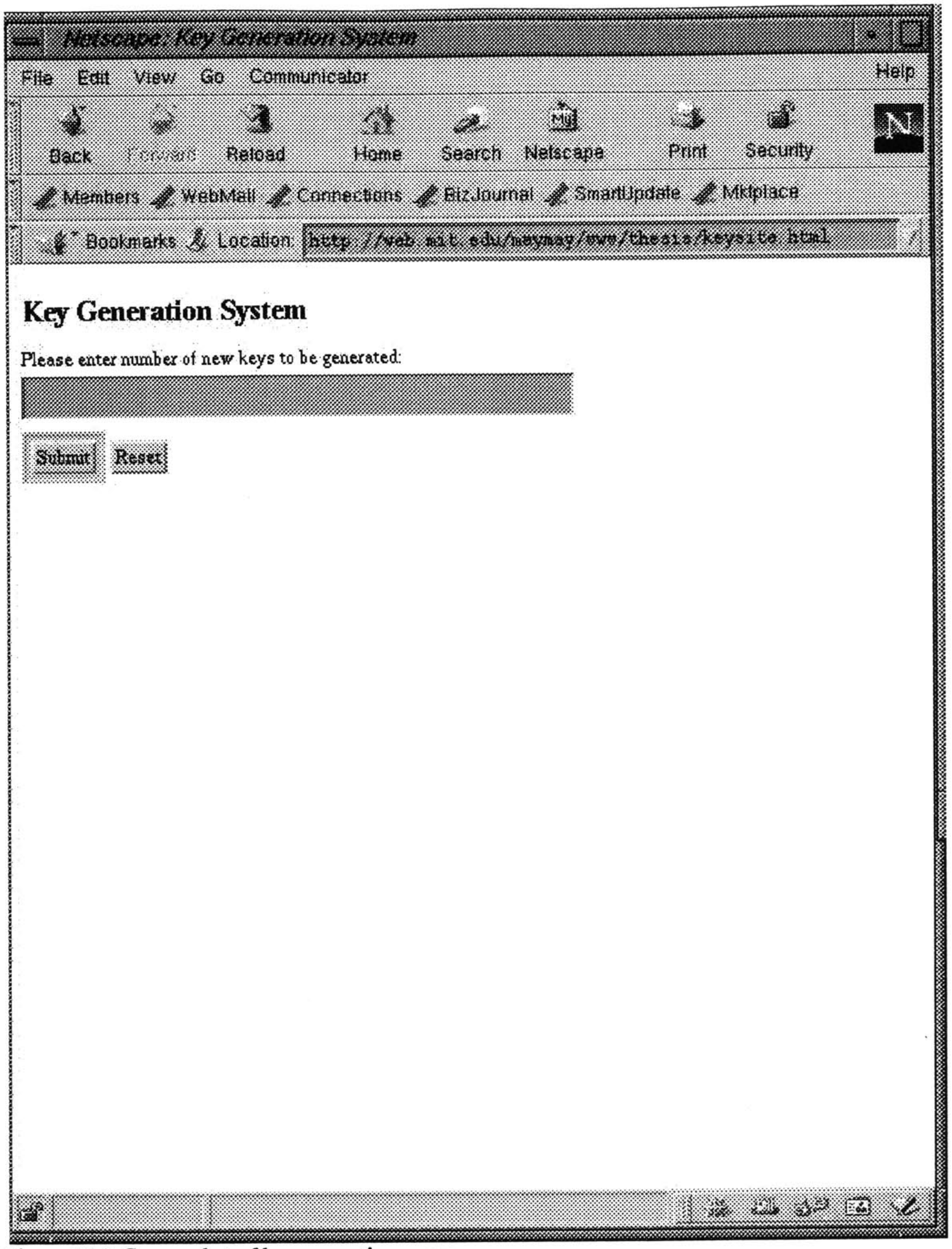
Please update group permissions list by entering the group #, followed by the range (first and last brick) of the block of bricks that the group is authorized for. Please note that if only one brick is in the range, you can leave "Last brick" blank.

Group #

First brick in range to be authorized

Last brick in range to be authorized

Figure 15.0: Screen shot of group permissions management for access control lists.


Netscape: Key Generation System
File Edit View Go Communicator Help
Back Forward Reload Home Search Netscape Print Security
Members WebMail Connections BizJournal SmartUpdate Mktplace
Bookmarks Location: http://web.mit.edu/maymay/www/thesis/keysite.html
Submit
Reset

Key Generation System

Please enter number of new keys to be generated:

Figure 16.0: Screen shot of key generation menu.

7.0 REFERENCES

[1] Anderson, Ross J. and Frank Stajano. "The Resurrecting Duckling." in Christianson, B. et al., editors. *Security Protocols: 7th International Workshop Proceedings* published as *Lecture Notes in Computer Science 1796,* Springer-Verlag, 2000.

[2] Anderson, Ross J. "Why Cryptosystems Fail." *Communications of the ACM 37, 11*: November 1994, pages 32-40.

[3] Garfinkel, Simson. *PGP: Pretty Good Privacy*. O'Reilly & Associates, Inc.: 1995.

[4] Lenstra, Arjen and Eric Verheul. "Selecting Cryptographic Key Sizes." *Public Key Cryptography Conference*: 2000.

[5] Menezes, Alfred J., et. al. *Handbook of Applied Cryptography*. CRC Press: 1997.

[6] Pfleeger, Charles P. *Security in Computing*. Prentice-Hall: 1989.

[7] Schneier, Bruce. *Applied Cryptography*. John Wiley & Sons, Inc.: 1996.

[8] Stallings, William. *Cryptography and Network Security*. Prentice-Hall: 1999.