UNIVERSITY OF CALIFORNIA
Santa Barbara

The use of Safe Areas of Computation (SAC) for secure computing

A Dissertation submitted in partial satisfaction
of the requirement for the degree of

Doctor of Philosophy

in

Computer Science

by

André Luiz Moura dos Santos

Committee in charge:

Professor Richard A. Kemmerer, Chairperson

Professor Terence R. Smith

Professor Alan G. Konheim

March 2000

The dissertation of André Luiz Moura dos Santos is approved

Committee Chairperson

February 2000

March 2000

VITA

July 24, 1965 – Born – Presidente Alves, Brazil

1988 – B.S., Instituto Tecnológico de Aeronáutica

1989-90 – Software Engineer, FUNCEME, Fortaleza, Brazil.

1990-91– Software Engineer, INPE, São José dos Campos, Brazil.

1991-94 – Research Assistant, Department of Atmospheric Science, University of Washington, Seattle.

1994 – M.S., University of Washington, Seattle.

1994-2000 – Research Assistant, Department of Computer Science, University of California, Santa Barbara.

PUBLICATIONS

André dos Santos and Richard Kemmerer, "Safe Areas of Computation for Secure Computing with Insecure Applications", *Proceedings of the Fifteenth Annual Computer Security Applications Conference*, pp. 35-44, December 1999.

Flavio de Paoli, A. dos Santos and R. Kemmerer, "Web Browsers and Security", in *Mobile Agents and Security*, Lecture Notes in Computer Science, vol. 1419, pp. 235-256, July 1998, Springer-Verlag Publishers.

Flavio de Paoli, A. dos Santos, and R. Kemmerer, "Vulnerability of 'Secure' Web Browsers", *Proceedings of the 20th National Information Systems Security Conference*, pp. 476-487, October 1997.

André dos Santos, "Variation of rainfall and OLR over the Amazon basin", master thesis, University of Washington, winter 1994.

André dos Santos et al., "Interface for on board data collection", Encontro de Físicos do Nordeste, 1989. (title translated from Portuguese)

André dos Santos, “Capacity planning of the computing system of the Air force Technological Center (CTA)”, graduation thesis, Instituto Tecnológico de Aeronáutica, December 1988. (title translated from Portuguese)

André dos Santos, “Laser system for gas identification”, 1º Encontro de Iniciação Científica do ITA, August 1987. (title translated from Portuguese)

FIELDS OF STUDY

Computer security. In particular:

- The security and use of tamper resistant devices, e.g. smart cards.
- The security of Internet technologies and applications.
- The security of online banking and online brokering systems.

Professor Richard A. Kemmerer.

ABSTRACT

The use of Safe Areas of Computation (SAC)

for secure computing

by

André Luiz Moura dos Santos

Currently the computer systems and software used by the average user offer virtually no security. Because of this many attacks, both simulated and real, have been described by the security community and have appeared in the popular press. This dissertation presents an approach to increase the level of security provided to users when interacting with otherwise unsafe applications and computing systems. The general approach, called Safe Areas of Computation (SAC), uses trusted devices, such as smart cards, to provide an area of secure processing and storage.

This dissertation describes the Safe Areas of Computation approach and the results of using the SAC approach to protect specific browsing applications. The intent is for protected browsers to be used to interact with institutions that have requirements for high security, such as digital libraries that provide restricted documents and financial institutions that enable users to perform sensitive operations for electronic commerce or online banking.

CONTENTS

FIGURES

TABLES

CHAPTER 1

Introduction

Currently the computer systems and software used by the average user offer virtually no security. In addition, the Internet and the World Wide Web (WWW) have had a phenomenal growth during the past few years, interconnecting average users and connecting the average user to expert users that are not always good neighbors. Because of this many attacks, both simulated and real, have been described by the security community and have appeared in the popular press. This research presents an approach to increase the level of security provided to users when interacting with otherwise unsafe applications and computing systems. The general approach is called Safe Areas of Computation (SAC) and uses trusted devices, such as smart cards, to provide an area for secure processing and storage.

The Safe Areas of Computation approach uses a collection of trusted devices that enforce the protection of users from the insecurity of specific applications. Each of these devices is called a Safe Area of Computation. The goal of such devices is to provide islands of security that interact with an ocean of insecurity.

The main goal of the SAC approach is to provide security for client-server applications. The approach can also be used to protect stand-alone applications.

However, this dissertation only describes its use for protecting client-server applications.

In a client-server configuration Safe Areas of Computation provide:

a) *Strong authentication*. A client SAC exchanges cryptographic messages with a server SAC in order to perform mutual authentication. At the end of the exchange the client SAC and the server SAC will have agreed on a secret key.

b) *Secure channels*. Both the client and the server SAC will use the secret key that was agreed on in the authentication step to encrypt and decrypt messages that are exchanged, thus providing a secure channel.

c) *Access control*. Every client SAC has an access control list that specifies for each application being secured, the types of data that the user can access. In addition, if the access is hierarchical, the access list entry also specifies the clearance level of the user for that type. In order to access data the user's access control list must contain an entry of the type specified by the data's security label and the user's clearance level must allow access to the data based on its security label.

An important goal of the SAC approach is to provide strong security even for users that are not concerned with security. The price paid for this security is that the SAC approach may require the user to deal with a hardware token, which is the client SAC. Although this requires an initial adaptation to its use, the benefits of the additional security provided outweigh the initial discomfort. The only requirement for a user is to insert the client SAC in a reader before performing a secure transaction, in the same way that ATM cards are currently used for interactions with bank ATM machines.

The SAC approach provides access control in a generic way. The approach uses generic hierarchical security labels and access control lists without predefining what the labels and clearance levels are. This enables contextual interpretation and implementation of more than one security policy concurrently. A bank can, for example, use security labels to specify what banking transactions a user can perform, while a digital library can use labels to specify what type of documents a user can access and if the documents have clearance levels what level is required. Although this dissertation uses documents to demonstrate the use of security labels, many different access policies can be modeled in the same way.

This approach is generic, being able to provide security for a wide variety of applications. However, the SAC approach can be better understood and analyzed with a test bed implementation. Therefore, this dissertation details how the SAC approach can be used to protect data distribution over the Internet, using off the shelf browsers.

In addition this work proposes to use the processing power of the client SAC to reduce the workload of central servers without compromising security. As a proof of concept the proposed system was implemented in an environment that requires secure Internet transactions. A prototype implementation for both the Alexandria Digital Library and a banking system were implemented.

This dissertation has four more chapters. The next chapter gives background information about the Internet and its insecurity, showing the reasons why Internet data transactions are a very good candidate for the SAC approach. In addition, it provides basic information about smart cards and cryptography. Chapter three describes the SAC approach, giving details of how it works and the features it offers. The last two chapters discuss the conclusions of this research and outline the future work to be done.

CHAPTER 2

Background

This chapter discusses some background information necessary to better understand the SAC approach and its choice of test bed implementation. The chapter is divided into five sections. Section 2.1 describes the Internet and the World Wide Web, along with the technologies used and flaws found. Section 2.2 introduces smart cards and presents basic information about them. Section 2.3 presents basic concepts of cryptography and cryptographic algorithms. Section 2.4 describes the container model used by the military message system. The last section provides a short overview of the Alexandria Digital Library.

2.1 Internet

The growth of the Internet and the World Wide Web (WWW) during the past few years has been phenomenal. The Internet is currently serving tens of millions of people connected through millions of computers. All these networked computers have a big impact on the lives of the people, facilitating communication among them and helping in dissemination of information. However, this also results in large security risks imposed on users that have little or no knowledge about the risks and damage that can be inflicted by using the Internet. Although this research proposes a solution to many different situations, the connection of unsafe computers and applications through the Internet was a large motivation for it.

Most every business and government institution has a web page, and the web and web browsing are fast becoming the primary source of information for people of all ages. The ease of use of web browsers currently available is what makes this type of interaction with the Internet so widely used.

Languages like Java and JavaScript have been developed to embed programs into HyperText Markup Language (HTML) documents (pages). Java applets, which are designed to be downloaded from the web and run directly by the Java virtual machine within a browser, are also increasingly being included in web pages to provide more sophisticated animation and other desirable features. Downloading and executing code from anywhere on the Internet entails security problems. That is, the host computer is open to a variety of attacks, ranging from attacks that simply monitor the environment to export information, to attacks that change the configuration or the behavior of the host (e.g., changing files or consuming resources), and finally to attacks that open a back door to let intruders get into the host.

Issues to be considered when investigating secure browsing on the Internet are the different platforms and operating systems used, and the user's education. Even without considering the existing and possible future flaws in the technologies available to Internet programmers, it is reasonably easy to convince a naive user to install files that he/she receives. Since most operating systems and platforms used do not provide secure areas, these files could corrupt the security model.

To solve these problems and to provide users with a high degree of security two steps should be taken:

1) Improve the security of the technologies used by the Internet.
2) Provide an area where security critical modules could execute using secure data.

The first of these steps is being pursued by improving the security model of the technologies used to browse the WWW and by providing fixes to identified flaws. One important security improvement is the use of public key cryptography to authenticate the server site to the user. This technology, which also establishes an encrypted channel between the client browser and the server site, is called secure socket layer (SSL).

An approach to satisfy the second step is the use of tamper resistant devices, such as smart cards. Using the existing smart card technology, smart cards can be used to carry the necessary cryptographic keys for authentication of specific sites that require a higher degree of security, such as banks and stock brokering agencies. The new generation of cards has more memory capacity and could have a Java Virtual Machine (VM) implemented on them. In this case and assuming Java can be accepted as implementing a secure language, mobile agents and alternative models of certification can be run in a secure environment on the card. Even if Java is not accepted as secure, the Java smart card provides a model for downloadable programs that is safer than the PC or workstation model.

The World Wide Web is in continuous growth, adding new technologies at a very high rate. This dissertation does not have the goal of describing in detail all these technologies and their flaws. However, it describes some of the technologies that have been used for a few years, in order to show the motivation for using the World Wide Web as the base for proof of concepts for this research. In addition, some flaws and attacks that affected these technologies are presented.

2.1.1 History of the Internet

The concepts for the Internet started in the beginning of the 1960’s as an answer to the cold war question of how US authorities would be able to talk to each other in the aftermath of a nuclear attack. The RAND Corporation, the leader in research for the US military, took the task of giving an answer to this question.

At this time all communications were done point to point, with a dependency of each link on the previous link. So, in this configuration, if any point was defeated, the whole network would become inoperative. This certainly was not an acceptable situation in the event of a war.

Paul Baran, from the RAND Corporation, conceived the concept of a network that would work its link as a fishnet, so if one point did not work anymore, the network would still continue operation. Although his work inspired the way the Internet would operate, it was shelved by the Pentagon. With the concept of a new network as a goal, in 1962 the RAND Corporation started its research into a robust, distributed network for the military to use as control centers and distribution of information.

In 1965, this network started to take shape when the US Department of Defense Advanced Research Project Agency (ARPA) sponsored research for a “cooperative network of time-sharing computers”. This network, called ARPANET, would be used for the sharing of super-computers among researchers. And in 1967 the first plans for the ARPANET were discussed in a meeting of the Association for Computing Machinery (ACM).

The first ARPANET sites, four US Universities, were linked in 1969. This network linked Stanford Research Institute, UC Los Angeles, UC Santa Barbara and the University of Utah. The ARPANET, which was first conceived to share computing among scientist, was a success, with the most used application being electronic mail (e-mail). In 1971, the ARPANET had 23 hosts in the USA. In 1972 the InterNetworking Working Group was created with the task of regulating the growth of the ARPANET. In 1973 the ARPANET went global connecting to University College in London and the Royal Radar Establishment in Norway.

At this time the network was directed and accessed by academic researchers and the military. But in 1974 the first commercial version of this network was offered by Bolt, Beranek & Newman, called Telenet.

In 1979 another service that became very popular started in the Internet. Two grad students from Duke University, Tom Truscott and Jim Ellis, and Steve Bellovin from the University of North Carolina established the first USENET newsgroup for sharing information. In 1981 ARPANET had 213 hosts, with a new host being added every 20 days.

The first time that the word Internet was used was in 1982, and by the same time the transmission control protocol/Internet protocol (TCP/IP) was used as the primary communication protocol on all of the Internet. By 1984 over 1,000 hosts were connected to the Internet. With the appearance in the market of personal computers and cheaper mini-computers, there were more companies willing to be connected to the Internet to exchange information among their peers. In 1987 the number of hosts exceeded 10,000.

In 1988, the community that used the Internet was forced to face the first real, widespread security issue: the “Internet Worm”. The Internet Worm was released in November of 1988, disabling 6,000 of the existing 60,000 hosts. In response the Computer Emergency Response Team (CERT) was created for addressing security concerns about Internet usage.

In 1990, with over 300,000 hosts, the ARPANET was decommissioned. The next year the first step in making the Internet an environment for commerce and wide use was taken by the National Science Foundation (NSF). It lifted its previous restriction of not allowing any commercial traffic in the NSFNET, which was the backbone of the Internet.

By 1991, the Internet was widely used by researchers and companies to exchange information by e-mail and discuss topics on newsgroups, but the everyday person was still not attracted to it. However, a team led by Mark MaCahill, from the University of Minnesota, released a friendly user Interface for information on the Internet, called “gopher”. In MaCahill’s words, this was “the first Internet application my mom can use”. The wide use of gopher was the first signal of the acceptance that the World Wide Web would have.

The most important step for the Internet to be partially known as the World Wide Web and for documents to be easily accessible to everybody that wanted to do more than just exchange e-mails was taken in 1989. At that time Tim Berners-Lee from the European Laboratory for Particle Physics (CERN) in Switzerland had a project for using the Internet to exchange information in more than just text form. He combined pictures and sound, using a new approach for linking related subjects (hypermedia). He called this Internet based environment for exchanging information, which would link several hosts and contain a significant amount of hypermedia documents, the World Wide Web (WWW). In November 1990, the first web client and server prototype was developed using a Next platform[APP97].

In 1991 Tim Berners-Lee posted his code, which combined text, sounds and pictures in the newsgroup alt.hypertext. The potential for the Internet to become a bigger medium of publishing information and commerce was readily noticed, and programmers started to develop software to interact with hypermedia. After this, and particularly after 1993, when the first clients to the World Wide Web, called web browsers, were released, the number of hosts connected to the Internet exploded. The most famous of the early browsers, which was called Mosaic, was developed by Marc Andreesen and a group of student programmers at the National Center for Supercomputing Applications (NCSA), located at the University of Illinois at Urbana-Champaign. At the same time other client browsers were also released: Midas, Erwise and Viola for X, and CERN Mac. There were around 50 web servers when these clients were released [APP97]. After these web clients were released the traffic on the Internet expanded to an annual growth rate of 341,634%. In September 1993 the WWW traffic measured at the NSF backbone accounted for 1% of the total traffic. Today there are over 147 million people connected to the Internet [TRE99].

After Mosaic was a big hit, in 1994, Marc Andreessen joined with Jim Clark to form Mosaic Communications Corp., now Netscape, with the primary goal of developing and commercializing web browsers. They have been developing and releasing Netscape web browsers since then. In its current version 4.7, it is one of the most used web browsers today.

Since web browsers are so popular and useful, Microsoft also entered into the competition. They launched Microsoft Internet Explorer, which in its version 5.0 is the other browser that competes for first place in the number of people using it.

With this explosion, the World Wide Web has become an attractive place to explore security flaws in Internet technologies. In 1995 the number of security incidents reported to the Computer Emergency Response Team was 2412. The number of sites affected by those incidents: 12,000, while in 1988 the number of security incidents was 6 [INVA97].

2.1.2 HTML

The World Wide Web is an innovative way of indexing information. The first person who idealized this was Vannevar Bush in 1945 in his article “As We May Think” published in “The Atlantic Monthly” [BUSH45]. For him the human mind didn’t work the way that indexes were used in order to find information. He proposed a machine called Memex that would store a large amount of information and the user would be able to create trails of information, or links among text and graphics. The WWW adopts this philosophy and uses hypermedia to provide information and to link correlated subjects. This information is shown to the user by web browser clients. Each of today’s web browser clients uses a graphical interface in a point and click manner. A pointer, usually an arrow, marks a position in the computer screen that can be changed by the user, and a “click”, the user pressing a button, results in an action by the browser. This graphical interface also uses the concept of windows, that are regions in the screen that are used by a program. Each set of hypermedia information that the web browser shows in its window is called a page. The browser paints pages, one at time, making use of scroll bars in case the page is bigger than its window.

Web servers must be able to provide pages when they are requested by a client application. The web server is a daemon program running in a computer linked to the Internet that understands a request for a page and sends it using the specified protocol. The concept of a universal resource locator (URL) is used to address the web server that has the requested page and to specify the protocol that should be used to transmit the page. URLs use the server name and the file name, possibly with subdirectory names, to do the addressing. The primary protocol used to send a page is the Hypertext Transfer Protocol (HTTP), but the page may specify that parts of it, for example an image embedded in the page, should be sent from other URLs which may be the addresses of other web servers and/or using other protocols.

The information shown in a page is derived from a language that describes how the page is laid out. This language is called HyperText Markup Language (HTML). Every page description consists of a text file in the web server that is sent to the web browser client when the URL describing that file is requested by the client.

The text files that describe pages describe the many elements that compose each of these pages. These elements are headers, tables, images paragraphs, lists, etc... HTML tags are used to mark elements of the file, so the client web browser will know how to paint a page when it is received. The elements may consist of other elements, plain text, or a mix of text with other elements.

2.1.3 Java

The Java language is a general-purpose object-oriented language that was released by Sun Microsystems in 1995 [GJS96a]. One of the major design goals for Java was portability. The result is that in addition to the Java source code, the binary code is also executable on all processors. This is accomplished by compiling the source code into platform independent bytecode and providing bytecode interpreters for every possible platform. The bytecode is executed by the Java virtual machine, which is an implementation of the Java bytecode interpreter on a particular platform.

Some of the features of the Java language that make it simpler and supposedly more secure are that it is strongly typed, there are no preprocessor statements (like C's #define and #include), there are no pointers, no global variables, and no global functions. By keeping the language simple and without many of the error prone features, such as multiple inheritance, it is expected that Java will be more secure.

A Java program is a collection of classes and instances of classes. Each class is compiled into bytecode, which is then interpreted to execute the program. As mentioned above, a major characteristic of Java is that pointers are not supported; object references are provided instead. Java supports dynamic creation of instances and bindings. When a class instance (an object) is needed, it is created explicitly and a reference to it is returned; when a method is invoked on an object, the interpreter selects the method to be executed according to the class hierarchy and method overloading. Object destruction is automatically handled by a garbage collector, so that memory management is completely under the control of the interpreter.

Another feature of Java is the support for concurrent programming via threads. Threads allow programmers to associate an independent execution flow with each class. A class with a thread can be started, stopped, and suspended independently from the execution of the rest of the system of which it is part. Synchronization among thread executions can be accomplished by class monitor's.

As mentioned above, Java code was designed to run on any client; therefore, compiled Java programs are network and platform independent. The absence of physical pointers and automatic memory management help achieve this independence. Moreover, the bytecode has been designed to fully support the typing mechanism of Java so that dynamic code verification can be performed. This is a safety and a security feature designed to prevent the execution of corrupted or malicious code.

The Java Virtual Machine is emulated in software and can run on numerous platforms [LY96]. It could also be compiled or implemented directly in microcode or hardware, but currently it is mostly emulated in software. The virtual machine deals with class files, which contain Java virtual machine instructions, a symbol table, and a few other necessary items. Java virtual machine instructions are all one byte long, and that is why they are called bytecodes. Bytecode can also be generated from other high level languages, such as Ada or C, or it could be generated manually.

When Java classes are downloaded from the network it is necessary to use a class loader. Java supplies an abstract ClassLoader class for this purpose. Because abstract classes cannot be used directly, each browser needs to declare a subclass of this class to be used by the browser for downloading classes. Each subclass must include a customized implementation of the loadClass() method to retrieve and download a class from the net. A class is downloaded as an array of bytes that must be converted to an instance of class Class. That is, the array of bytes must be translated to the structure of a class. The ClassLoader method that actually does the conversion is defineClass(). Every class object contains a reference to the class loader that defined it, so related classes can be downloaded by the same class loader. These features make Java suitable for writing programs in a distributed, heterogeneous environment such as the web.

Because Java was designed with security in mind, the language provides an abstract Security Manager class that, when defined by the application, implements the security policy that it will enforce. All system accesses, such as reads and writes to disk, can be protected by the Security Manager as much or as little as desired. An application can set only one Security Manager, and any attempt to set a new one will result in an exception being thrown.

The possibility of running Java on any machine is one of the main features that makes Java desirable. This enables applications to run on the user's computer, when a user is using a browser to navigate the World Wide Web. This explains Java's wide acceptance as a language for developing applications on the web. Java allows a programmer to develop an application, called an applet, that is sent to and runs on the user's machine. In contrast to the first days of the web when all information was static text and graphics, the use of applets makes the experience of navigating the web more colorful and enables interaction with the user.

Applets are embedded into web pages using the HTML language. A special tag in a downloaded HTML file tells the web server to download the bytecode for a Java applet. The most popular web browsers, such as Netscape Navigator and Microsoft Explorer, include support for the execution of applets. Applets can be used for implementing small stand-alone applications, as well as for implementing clients of client/server applications.

A page may have more than one applet embedded in it. In this case, all of the applets are downloaded together. Because a thread is associated with each applet, more than one applet can run concurrently within the same web-page context. The class Applet defines a set of methods to control an applet's behavior. These methods can be overridden by programmers to get the desired behavior from their applets. The most important methods are init, start and stop. When an applet is downloaded it is automatically created, initiated and started: the init and start methods are invoked and the applet is labeled active. When the page that embeds an applet is not displayed (the browser is iconified or shows another page), the execution of that applet is suspended (i.e., the stop method is invoked and the applet is labeled inactive). If the page is shown again, the applet execution is resumed (i.e., the start method is invoked again and the applet is labeled active). It is possible to keep an applet running even when it is inactive, however. This is accomplished by overriding the stop method so that it never returns.

2.1.4 JavaScript

JavaScript is an object-based, loosely-typed, scripting language that has been specifically designed by Netscape to include programs (scripts) into HTML pages. It gives the programmer a higher degree of access to the browser than Java does. In JavaScript functions can be declared that are not methods of a particular object and that can be used with any reference to an object.

JavaScript was designed to provide Internet developers with an environment that is easy to program, and it results in programs that run on both the client side, embedded in the HTML language, and on the server side, where it is called LiveWire. On the server side, LiveWire is used as common gateway interfaces (CGI) are. That is, they are accessed by a URL and execute a program in the server with access restrictions specified on the web server. The requirements for flexibility and for a friendly environment for the programmer subject JavaScript to many security flaws as is demonstrated in [DIGI97].

The following table shows how Netscape compares JavaScript and Java [NET97].

JavaScript	**Java**
Interpreted (not compiled) by client.	Compiled on server before execution on client.
Object-based. Code uses built-in, extensible objects, but no classes or inheritance.	Object-oriented. Applets consist of object classes with inheritance.
Code integrated with, and embedded in, HTML.	Applets distinct from HTML (accessed from HTML pages).
Variable data types not declared (loose typing).	Variable data types must be declared (strong typing).
Dynamic binding. Object references checked at run-time.	Static binding. Object references must exist at compile-time.
Cannot automatically write to hard disk.	Cannot automatically write to hard disk.

Table 1: Differences between Java and JavaScript

JavaScript provides the programmer with many pre-defined objects, such as history, text, and windows. Many of them refer to browser properties, such as location, history, navigator, and window, or to the HTML page being shown in the browser. This is where the language provides the programmer with a good deal of manipulation of the user's browser, resulting in very pleasing pages and/or in unsafe and perhaps annoying pages.

One particularly dangerous, although very useful, object is the form object. The form by itself is a feature of the HTML, but JavaScript can access it as a JavaScript object. The basic fields of a form declaration are as below:

```
<FORM
  NAME="formName"
  TARGET="windowName"
  ACTION="serverURL"
  METHOD=GET | POST
  ENCTYPE="encodingType"
  [onSubmit="handlerText"]>
</FORM>
```

The form specified as above is sent to a CGI or LiveWire in the web server specified in the ACTION field, and this program may send back an answer in Multipurpose Internet Mail Extensions (MIME) format that will be shown in the window or frame taken from the TARGET field. Another way to send a form is through e-mail, and for this it is only necessary to specify “mailto:” in the ACTION field along with a valid e-mail address. This is a very flexible and dangerous feature.

The fields METHOD and ENCTYPE are self explanatory. OnSubmit is a function that is called when the form is submitted.

To define the elements, or fields, of a form the objects button, checkbox, hidden, password, radio, reset, select, submit, text and textarea are used. Their behavior and appearance are clear from their names. These objects may also be manipulated as properties of the form.

A forms object is an array that can be used to refer to the defined forms in the document (HTML page). It is ordered in the order that the form fields appear in the source of the HTML page. Thus, if the page has three forms then document.forms[0] refers to the first form, document.forms[1] refers to the second, and document.forms[2] refers to the third. The object document is a way to refer to the page shown in the browser.

The form object may also make use of a submit object. The submit object is a button that is created in an HTML form, and when it is clicked it submits the form. The basic fields of a submit declaration are as below:

```
<INPUT
  TYPE="submit"
  NAME="submitName"
```

```
        VALUE="buttonText"
        [onClick="handlerText"]>
```

The fields are self explanatory, and the submitName can be used to access the submit object, which is always an element of some form.

The submit object has a pre-defined method click that simulates a click on the submit button, which results in submitting the form without the need for the user to actually click the submit button. This is potentially dangerous and can be used to collect e-mail addresses.

If any script running could access all objects of JavaScript, the privacy of the user could be compromised, such as in the case of accessing the history object. Another way to compromise the privacy of a user would be to have two browser windows opened and to allow one of the browser windows access to objects on the other window and to keep track of the user’s activity. To prevent this, the security policy of JavaScript that is implemented in Netscape Navigator prevents some of its builtin objects and methods from accessing privileged information. For instance, although JavaScript has a history object, an access to its fields returns the null string. Although this policy is safe, some programmers would like to be able to access these objects for legitimate operations with the knowledge of the user. In response to the desire of these programmers, Netscape introduced data tainting in Navigator 3.0 to provide access to scripts that need this information. With data tainting enabled a JavaScript script can access this privileged information but can not send it to a server without the consent of the user. The only requirement for data tainting to be enabled was for the user to set an environment variable called NS_ENABLE_TAINT [NET97b]. In Navigator 4.0 and above, which is part of Netscape Communicator, Netscape abandoned this model in favor of a much better model that uses digital signatures to give capabilities to JavaScript scripts [NET97c]. Using this capability model a JavaScript script may request only the particular capability that it requires for the completion of its task. By checking the digital signature associated with this script, a user can grant or deny the requested capability.

2.1.5 Browsers

The two most used web clients (browsers) are Netscape Navigator, currently in version 4.7 and Microsoft Internet Explorer, currently in version 5.0. Netscape has integrated its web browser in an environment called Netscape Communicator, which besides web browsing, supports e-mail, internet conference, newsgroups, composing and editing web pages and channel technology. Microsoft integrated its web browser in the last version of the Windows operating system (Windows 98) [MS97a]. Its browser supports the same technologies mentioned above.

2.1.5.1 Channel Technology

The channel technology, supported by Netscape's Netcaster and Microsoft Explorer Channels, is an automatic access to a site using http that delivers information periodically. The information delivered by Netcaster and Microsoft's channels use the same HTML technology that is used for web pages, and it is delivered in the form of pages. The difference is its ability to update itself without the need for interaction with the user.

2.1.5.2 Java and JavaScript

Both Netscape Navigator 4.7 and Microsoft Explorer 5.0 support the execution of Java applets and JavaScript. Their implementation for the support of the Java virtual machine and the JavaScript interpreter differ, but they both support the basic requirements of these languages. However, Netscape claims that Microsoft does not support Java or JavaScript fully [NET97e].

They both implement security restrictions for applets based on what is called "sandboxing". The idea behind this model is that an applet, or for that matter any code that is executed in the user machine, is allowed to do whatever it wants in a sandbox in the user's machine. The sandbox prevents the code from accessing potentially compromising resources, such as the file system and the network. In the Java sandbox model the sandbox prevents an applet from:

a) reading or writing files on the user's computer;
b) making a network connection, except to the host that the applet came from;
c) starting programs in the user's computer;
d) loading libraries;
e) defining native methods;
f) reading part or all of the system properties.

The sandbox model has been blurred with the release of Java 1.2. This version of the language enables applets to request capabilities outside the sandbox [GON97]. The user is supposed to grant or deny these capabilities, based on the digital signature of the applet.

2.1.5.3 Cookies

Cookies are a technology that was developed to make up for the stateless nature of web communications. The specification of this technology was introduced with Netscape Navigator 2.0 and was soon also implemented by Microsoft in Internet Explorer. A cookie is a small piece of text information in the form of a name/value pair. It can either be stored on the user's computer, providing persistence, or on the browser, in which case it will be lost at the end of a browsing session. It can be sent to the server that set it at any time. The first time a server receives a request for a page it sets the cookie(s), which will be sent back to the server at its request. Cookies are useful for maintaining related information during a browsing session or across multiple browsing sessions. Shopping baskets provide an example where cookies can be useful in a single browsing session. Configuration preferences is an example where cookies can be useful across browsing sessions. Although cookies have been introduced to improve the quality of the service provided by the web, they can also be exploited to invade the privacy of a user. An example where this happens is described in [DDK97].

2.1.5.4 ActiveX and Plug-in Technologies

Microsoft Explorer and Netscape Navigator both support ActiveX technology, which is a specification used by programs, like applets, that can be sent to a user's computer when he/she is browsing the web. ActiveX does not have a restrictive sandbox model like Java does. It gives the program more control over the user's environment, which gives it more functionality, but it can also pose a security threat. Microsoft uses a model of trust/non trust for ActiveX. Using this model, ActiveX programs, called controls, may use digital signatures to improve the degree to which a user trusts a control. In this model, if a user trusts a control, the control is downloaded to the user's computer and no restrictions are imposed on the resources it can access. The only restriction, in this case, are those provided by the operating system. If the user does not trust the control it is not downloaded.

Netscape's browser supports a similar technology called Plug-ins. As in the case of ActiveX, Plug-in applications are executables that are downloaded to the user's computer to extend the browser's functionality. As with ActiveX a trust/non-trust model is adopted for Plug-ins, where Plug-in applications may improve their trust levels by using digital signatures

2.1.5.5 LiveConnect

Netscape Communicator also supports LiveConnect, which is a technology developed by Netscape and incorporated in its browser. It was developed as an answer to page developers who wanted to be able to do inter-application communication on the client side [LEV96]. It enables communication between any two of Java applets, JavaScripts and Plug-ins [NET97d]. Microsoft Explorer does not support LiveConnect.

The communications that may occur are:

a) A JavaScript script may call methods from Java applets or Java builtin methods.

b) A JavaScript script may call Plug-in functions.

c) A Java applet may call JavaScript functions, both builtin and user defined.

d) A Java applet may call Plug-in functions.

e) A Plug-in may define Java classes.

f) A Plug-in may call Java and JavaScript functions.

The restriction imposed on this communication is that any communication must be between an applet and a script contained in the same page or Plug-in applications that were loaded by the page. Also, since one of the Java, JavaScript, or Plug-in can access builtin functions of the other, they should not be able to subvert the security features in place for the accessed technology.

The LiveConnect technology gives an Internet developer many opportunities for interaction with the user, which were not previously available. Applets can work together with JavaScript and Plug-ins, giving the user all the features available in all three technologies. Also they can open channels of communication on the client side, without the need for interaction with the server. This was a feature requested by many Web page developers.

The consequence of LiveConnect adding these additional features, however, is that they open new doors to security flaws. Concerns that were nonexistent before, like a JavaScript script being able to open a socket connection [GAR97], are real threats now, since a script can communicate with a Java applet and ask it to open the connection on the script's behalf. This connection could then be used to send the server information that previously was only able to leave the client as a form submission.

2.1.5.6 Security Options

Both browsers also give the user options to increase the security of their browser. These options allow an expert user to fine tune the security that he/she desires, taking into consideration the risks. On the other hand, the numerous options may cause a naïve user to allow malicious sites to compromise their computer systems. For example, Internet Explorer has an option to enable the download of an unsigned ActiveX control. This option can be set to *disable*, *enable* or *prompt*. While an expert user probably would set this to *disable* or *prompt*, a naïve user is likely to set this option to *enable* since he/she does not want to deal with pop up windows every time he/she wants to see an "interesting" web page. This setting enables the download of malicious ActiveX controls without requiring the user's interaction.

Both Netscape and Internet Explorer support the options to turn on or off the execution of Java and/or JavaScript. This enables users to trade the convenience of these languages for the security of not running any outside program in their machine.

Both browsers support the option of alerting users before accepting a cookie, and before sending a form in an insecure environment, and both let users choose if they want to continue this operation or not. They also enable the user to totally disable the cookie feature.

Both support the secure socket layer (SSL) implementation for communication with a secure server. When using Netscape Navigator 4.7 there is a small lock on the lower left corner that indicates if the communication is through a SSL(closed) or not (open). If the user clicks on this lock another window pops up with details of the page being shown, including the server that sent it and the certificates if there are any. When using Microsoft Internet Explorer 5.0 a lock appears on the right lower corner when the communication is through a SSL.

Both browsers support site certificates, which are digitally signed public keys used for digital signature verification. These are widely used by secure servers in conjunction with SSL. Microsoft calls its support of certificates Authenticode technology.

Explorer 5.0 divides sites into zones. It provides the user with four different zones that can be configured with the sites that the user wishes to have in them. These zones are Local Intranet Zone, Trusted Sites Zone, Internet Zone and Restricted Sites Zone. Explorer 5.0 allows the user to set four levels of security against active contents for each zone. The four levels are high, medium, low and custom. Although this at first seems like a good feature, the difference among the levels is very complex and the Explorer documentation fails to clearly explain what the difference is. For example, in the low security level the download of unsigned ActiveX controls is enabled, prompting the user about this operation, while in medium and high security levels only signed ActiveX controls can be downloaded. Another example is whether or not a user can download a file. In the low and medium security levels a user is able to download a file to his or her computer, while in the high security level this is not possible. The settings for Java are even more complex. It is possible to customize the access of Java applets to the file system, effectively allowing an applet to run with no sandbox protection.

Netscape Communicator 4.7 supports automatic updates. This feature enables automatic updates and the installation of browser and Plug-in applications from the network. The user is given the opportunity to look at the certificate for this software before the update or installation is completed. This is a very useful feature for the expert user, but it can be very dangerous for the novice.

Netscape Communicator 4.7 supports signed objects, which are Java applets, JavaScript scripts or Plug-in applications that were signed and that can request capabilities outside the sandbox. This enables these objects to read or write to/from the disk or to open network communications. In the case of Java applets, Netscape has developed classes, called capabilities classes, which have methods to request the desired capabilities. JavaScript also has a similar model. Microsoft Internet Explorer 5.0 also supports signed programs, which in its case are programs that were signed using Microsoft's certificate technology, called Authenticode.

2.1.5.7 Browser Attacks

There have been many security flaws in browsers since they were introduced. This section discuss some important flaws and attacks that were uncovered in the browsers, in Java, in JavaScript and in ActiveX for early browser versions. Since then many others flaws and attacks have been discovered at various levels of security risk. There is always a race between the discovery of a risk and the patching of the flaw that generates it. This dissertation, however, does not have the goal of covering all possible attacks. The attacks presented here show just how unsafe Internet transactions are and how difficult it is to trust the existing browsers.

2.1.5.7.1 Attacks exploiting Java flaws

Java implements a much more secure language than the previously existing programming languages, and it also has the bonus of being portable to any platform. However, many security holes have been found. These flaws may be exploited both when Java is used to develop applets and when it is used to develop application code.

Applets run on an implementation of the Java Virtual Machine, with the security implemented by the browser that is used to navigate the web. In this situation, the security holes can either be due to a weak implementation of the browser's security or to a flaw in the specification of the supposed behavior of Java.

Some of the first Java security holes were described by [DFW96]. In this paper they describe problems due to the browser implementations of Netscape Navigator 2.0 and HotJava 1.0 alpha 3. Since HotJava 1.0 alpha 3 was an initial implementation of a browser, without much care for security, the security problems that were described for that browser are not described here. Even today, with HotJava 3.0 in its release version, the browser's main goal is the demonstration of the new technology incorporated in Java applets. It was not initially intended to become a widely used browser, but this changed after Sun teamed up with Netscape to make a version of Netscape's browser that is completely written in Java [NET97a]. Sun used the HotJava technology to build a HotJava-Navigator browser with Netscape, HotJava 3.0, which is being marketed as a small footprint browser that complements Netscape Navigator [SUN99].

One class of attack that [DFW96] exposed was denial of service attacks. Since Java applets have access to all memory resources and CPU cycles in the user's computer, a malicious applet can consume one or both of these resources up to exhaustion. These programs can also be JavaScript programs or any content that when downloaded executes in the user's machine. The consumption of the resources slows down or even hangs the user's computer. Many examples of such attack can be found in [DIGI97]. Another denial of service attack that can be implemented using Java applets is to acquire locks on resources that are needed by the browser to work properly, which denies access to them. As an example of this attack, consider an applet that acquires a lock to the class java.net.InetAddress. All lookups for host names are blocked in Netscape 2.0, which results in no other new page being able to be loaded from any server. After being confronted with these possible threats, Sun answered that in their view these kinds of attacks are low-priority problems [DFW96].

Another class of attacks are those that exploit covert and overt channels. Many covert and overt channels exist in Netscape Navigator 2.0 and HotJava 1.0 [DFW96]. Even though the difference between covert and overt defined in [DFW96] is not the one usually used in the security community, some channels were identified in this paper. One such channel is that it is possible to connect to an SMTP mail daemon, if one exists in the network of the user's machine, and send e-mail to any machine in the Internet. The use of a tailored DNS server provides another channel. With this attack an applet requests a name resolution from the attacker's DNS server, and the server sends back a set of IP addresses. This exchange of information, requesting a name and returning IP addresses, could be used as a covert channel where host name and IP addresses would be interpreted as messages.

The covert channel that is the most difficult, if not impossible, to close has to do with the way the World Wide Web works. Since an applet can load any URL on the browser, the applet can also use this to send a message to some server. With this attack, the server interprets the request for a URL as a message from the malicious applet.

Some of these channels, such as the overt channel that uses SMTP, have been closed. The Java developers answered in [MUL45] that the analysis done in [DFW96] was based on the alpha 3 version of the JDK (Java Development Kit). In the JDK beta, applets could no longer connect to the SMTP port of the user's computer to attempt to send mail and also could not get the IP addresses of an arbitrary host.

One attack that was taken very seriously and resulted in long discussions among researchers interested in the security of browsers was known as the DNS attack. The DNS attack was formulated in [DFW96] and had independently been proposed by [GIB96] at an earlier date. It consists of setting up a DNS server that will lie about IP addresses when asked to do name resolution. An applet will ask to open a connection to a host name that is in the domain name from where the applet came. The attacker controlled DNS will resolve the host name to the IP address of any machine in the Internet, allowing the applet to connect to any machine in the Internet, which should not be allowed if the Java security specifications were followed. This attack generated a CERT advisory [CA96.05] to alert users about this possibility, which could be used to pass through a firewall. It was corrected in

Netscape 2.01 and JDK 1.0.1 by using IP addresses instead of host names to decide if a connection is allowed or not [ROS96].

Netscape Navigator 2.0 and JDK 1.01 both allowed an applet to instantiate the Classloader [DFW96]. Since the Classloader is responsible for loading any class required by an applet, a Classloader defined by an applet can resolve a class as one type in one instance and as a different type in another. This is the same as using explicit casts in C. With this, the pointer restriction imposed on Java becomes ineffective. This is possible because, although the constructor of a Classloader object checks to see if it is called from an applet, its reaction is to throw an exception that can be caught by the same applet. This leaves the applet with a partially initialized Classloader. Since the legal Classloader has already made all necessary initialization, this Classloader can be used just like any Classloader. This can then be exploited to execute any code, even native code, and to execute any command in the user's computer, with the user's privilege. This is also true for the Security Manager classes and FileInputStream classes, but no attack using these classes has been published. A CERT advisory [CA96.07] was issued to warn users of this problem. This problem was fixed in Netscape Navigator 2.02 and JDK 1.02 [SFAQ97].

In Netscape Navigator 2.0 and JDK1.0 an applet was able to invoke native code in a dynamic link library (DLL), which is a library that is loaded when an application requests a function that is implemented inside the library. This attack, which was first pointed out by [HOP96a], is also described in [DFW96]. By using this attack, an attacker could run any native code on the user's machine, without any of the Java restrictions. In order for this attack to succeed the user must download two files from the attacker in his/her machine: a Java loader and a native DLL. When an applet or application requests a package to be loaded, the Java runtime system loads it from a path of the file system derived from the package name, with every dot in the package name changed to a '/'. Although the Java language does not allow a package name to begin with a dot, the bytecode verifier allows a package name to start with a '/', resulting in this package being loaded from an absolute path in the user's machine. Code that uses this trick would have to be compiled with a compiler, since the standard Java compiler, javac, would not accept such a package name. Assuming the attacker had been able to download the files to the user's machine and knows the path of these files, he/she could load the package "loader" from an applet. This could invoke the native code DLL, since it is resident in the user's machine. In [DFW96] it is mentioned that the requirement that the user downloads the files necessary for the attack can be easily satisfied using the cache feature of Netscape. The attacker's applet would request the files from the server, which would be retrieved by Netscape and cached. If the applet can find out, or guess, the name under which the file is cached, it can accomplish the attack without any additional effort. [HOP96b] also discusses this possibility, with the addition that JavaScript could be used to define the path and filename known by the attacker that should be used by Netscape to cache these files. In [HOP96c], after some experimenting, he concludes that the use of JavaScript for defining path and filenames is only possible in Windows95 and Windows NT. This security hole was fixed in Netscape Navigator 2.02 and in the JDK 1.01 [SFAQ97].

An example of another attack to firewall protected hosts, which will only work for a very particular configuration of the network, is the attack described by [WIL96]. The author claims that he has found a security flaw in the implementation of Java sockets that allows applets to connect to any host, not just the host that the applet was loaded from. The attack works only in networks with a particular implementation and it requires that an HTTP proxy is being used to isolate the network from the Internet. In addition, Java-enabled web browsers need to be used on the network. The attack works in networks protected by firewalls, but they have to have particular domain names [REN96]. The target network and the attacker's domain name have to be identical, with the attacker's domain name being the officially registered domain name. This happens if the target network uses an internal domain name not registered with Internic and the attacker has control over the registered domain name. This problem was recognized by Javasoft and Netscape and was fixed in Netscape 2.02 and JDK 1.02.

Another firewall attack that requires a particular configuration allows an attacker outside the firewall to access computers inside it [MRR97]. This attack is based on the well known feature that in order to allow ftp transfers originating from the protected side of firewalls the firewall must allow a connection to be opened from outside in response to a PORT command issued from inside. When a user starts an ftp connection from the protected side of a firewall he/she issues a PORT command announcing at what port the transfer of data is expected. The server to which the connection is made receives this announcement. The firewall then sees an attempt to connect from the ftp server to the client machine at the announced port. If the usual behavior of the firewall, which doesn't allow any connection originating from outside of the firewall, were in place, then the ftp transfer would fail. However, some firewalls allow the outside server to connect to the client computer at the announced port after seeing a PORT command from the protected network to the outside. This was not considered to be a problem since only users on the protected side of the firewall, who had already been authenticated at login, could originate the PORT command. Unfortunately, with the possibility of Java applets starting socket connections in the user machine, this is no longer true. A Java applet can issue a PORT command for the host on the protected side of the firewall, announcing that it is going to listen at some known port, such as the telnet port. After this step, the ftp server can start a telnet session with this host without restriction from the firewall, since the firewall thinks that the connection is an answer to the PORT command.

A solution to the attack that uses the ftp PORT command is presented in [MRR97]. They suggest that sites that require strong authentication to outbound ftp connections would not suffer this attack, since the applet would not be able to authenticate itself in the user's behalf. Another solution would be for the firewall not to accept PORT commands that announce that a host is listening on a well-known port. In this case if an innocent host issues a PORT command that announces one of these ports, the ftp would fail, but the innocent and legitimate host could always start a new ftp transfer to a new port. A third possible solution discussed in [MRR97] is to allow only passive mode ftps. In this mode, instead of the client host starting the protocol and announcing at which port it will be listening, the server announces a port for the client to open. In this case all connections are originated from the client and the problem is solved. Although many ftp servers do not support the passive mode ftp protocol, a proxy could be used to implement it on the protected network. All client passive mode ftp requests would be made to the proxy, and the proxy would be in charge of communicating to the servers outside the firewall.

There is a variant of the attack proposed by [MRR97] that requires a much more particular configuration of the firewall and much more knowledge on the part of the attacker. For this attack an explicit proxy is used, and all requests from the client are channeled through the proxy.

2.1.5.7.2 Attacks exploiting browser flaws

One attack that exploits flaws in the implementation of the browser itself, as opposed to a Java or JavaScript flaw, was named the Danish Privacy bug after the nationality of the programmer who discovered it. The bug exploits a flaw in the implementation of Netscape Navigator 3.01 and Communicator 4.0 that enables a malicious site, which knows some file names and directory structure in the user's computer, to download these files [ORE97]. This enables the malicious site to acquire files from the user's computer without the knowledge of the user. The bug does not affect Microsoft Explorer and was fixed by Netscape in Navigator 3.02 and Communicator 4.01.

Another attack that does not require the use of Java or JavaScript is described in [MRR97]. It only requires the support of forms from the HTML language and http protocol. In this attack the attacker tricks the client into uploading a form to the ftp port in the attacker's machine, for example to http://evil.com:21/. Instead of containing information from the user, the form has a properly formatted PORT command announcing that the client is going to listen to a well-known port. The firewall will see this as an ftp connection since it is uploaded to port 21 of the server. The attack will work similarly to the PORT attack implemented using an applet, which was described above. Netscape does not allow http on low numbered ports, so the attack is not possible using Netscape [MRR97].

A spoofing attack, reported by [FBD97], relies on users thinking they are at a particular site when, in fact, they are at a copy of the site on the attacker's server. In this attack the attacker modifies a web server to work in a different way. A page in the attacker's server has all of the hyperlinks faked to address URL's in the attacker's server. If the user navigates through one of these hyperlinks the attacker's server receives the request. After receiving the request, the attacker's server downloads the page that the hyperlink should refer to and rewrites the page before sending it to the user. The rewrite changes all hyperlinks on the page to refer to the attacker's server. In this way the user receives the page that he/she wants, but everything is monitored by the man-in-the-middle, which is the attacker's server.

2.1.5.7.3 ActiveX Attacks

The use of ActiveX in Microsoft's browser has proven to be very dangerous. This is because of Microsoft's model of complete trust or no trust. In ActiveX, the analogs to Java applets are the ActiveX controls. If a site is trusted, it can download an ActiveX control to the user's computer, and this ActiveX control has access to all of the resources on the user's machine. Microsoft claims that ActiveX does not compete with Java, but with Plug-ins [MS96]. They also claim that while Plug-in applications have to be installed, the installation of ActiveX controls is transparent. This transparency is what makes ActiveX technology so dangerous from a security perspective.

Microsoft claims that the security of ActiveX controls comes from the digital signature of the developer of the control. This could be true if there was a tight control on who can get an Authenticode certificate. Although this control could improve the security of ActiveX, it would result in a process that would be lengthy.

This would restrict the number of users and companies that would apply for a certificate, making the model impractical.

Steffen Peter, a.k.a. Lutz Donnerhack from the Chaos Computer Club, demonstrated how to use an ActiveX control to make a monetary transfer from one bank account to another without the knowledge of the user [PET97]. His attack was based on the user downloading an ActiveX control that changed files used by a very popular financial software package called Quicken. His demonstration used a site that advertised "Becoming a millionaire in 5 minutes", which had an ActiveX control. Since the sandbox model does not bound ActiveX controls, this control was able to modify some files that would be used by Quicken the next time the user went online for financial operations.

2.1.5.7.4 Attacks exploiting JavaScript flaws

There have been many attacks that were implemented by exploiting JavaScript flaws. Some of these attacks rely on a flaw in the model that enables a form to be sent by e-mail, as was the case for the automatic submission of e-mail first used by [DIG97] and also used in [DDK97]. Other attacks rely on flaws in the implementation of the language, as was the case in the Bell Labs Privacy bug, the tracker bug, and the Singapore Privacy bug. These attacks are described in the following paragraphs.

Vinod Anupam, from Bell Labs [WL97], reported one of the first attacks that exploited JavaScript to compromise the privacy of a user. This attack exploited a flaw in JavaScript that enabled a malicious site to monitor which sites a user visits and to capture information from forms filled in by the user. The information is captured even when the user is interacting with a site that uses the secure socket layer (SSL). The attack uses two browser windows: one window is set to a size small enough not to be perceived, and this window monitors the user activity on the second browser window. This JavaScript flaw was present in Netscape Navigator 2.0, 3.0, Communicator 4.01 and Internet Explorer 3.02. It was fixed in Navigator 3.02 and Communicator 4.01a, and Microsoft released a patch to Explorer 3.02. Because of this finding, the Computer Emergency Response Team (CERT) issued an advisory [CA97.20], telling users concerned with privacy to disable JavaScript when browsing

Another JavaScript flaw, called the tracker bug exploited the possibility of JavaScript calling a function when leaving a page [BRU97]. This flaw uses a second window to set a tracking JavaScript function that is called when a page is unloaded (changed) in the first window. Using correct timing and a flaw in the JavaScript implementation, a malicious JavaScript script can use this function to monitor the activities of a user on the web in a way similar to that used for the Bell Labs Privacy bug. This flaw affects Netscape Navigator 3.02 and has been fixed on Netscape Navigator 3.03 [NET97f]. It is not present in Internet Explorer [MS97b].

The Singapore bug exploited an implementation flaw of JavaScript that occurs when a JavaScript function is called from a Java applet using LiveConnect

[LASH97]. The flaw enables an attacker to monitor the activities of a user in the same way as the previously described attacks. The flaw is due to LiveConnect not imposing the JavaScript access restrictions on Java applets that access JavaScript functions. This flaw does not affect Netscape Navigator 2.x or Internet Explorer, since LiveConnect is not implemented in these browsers. It affects Netscape Communicator 4.0 up to 4.01a. Netscape fixed the flaw in Netscape Communicator 4.02.

2.1.6 Secure Socket Layer and Public Key Infrastructure

Many security flaws have been found in the implementation of the new technologies used by browsers or in their specification. One main security concern has been spoofing attacks, where the user is made to believe that he/she is interacting with one site while in fact he/she is interacting with a malicious site [FBD97, DDK97]. A cryptographic protocol was designed to provide a higher degree of security to the user. This protocol, called secure socket layer (SSL) is a socket implementation that uses public key cryptography to authenticate a server site to a user. It can also be used to authenticate a user to a server site, although this is not widely used. In addition, SSL uses symmetric key cryptography to provide an encrypted channel between the user’s browser and the server site.

One difficulty of any protocol based in public key cryptography is to distribute public keys that can be trusted. Because of this, a Public Key Infrastructure (PKI) has been suggested. This infrastructure provides public keys cryptographically signed (certified) by Certification Authorities (CAs). The main idea behind this infrastructure is that it is easier to provide a small number of trusted public keys than a large one. A public key signed by a Certification Authority could then be used to certify that a user is looking at a page that carries the digital certificate of a named company, or is running a downloadable program using one of the technologies available on the Internet that has been signed by a company or person. This still does not protect users from attacks that exploit flaws on the technology that enable a malicious program to monitor a user's activity, but it does give a higher degree of security if the information exchange is encrypted.

One problem with CAs and PKIs is that the certificates prove that some person or company is the owner of a public key, they do not prove the good intentions of this person. This does not represent a significant threat when certifying a page; however, it is significant when the public key is used to sign downloadable programs that request special permissions to use resources in the user's computer.

As mentioned above, the SSL uses a session key to encrypt all messages passed from the client to the server and vice-versa. Currently, many Internet transactions are claimed to be secure because they use SSL to establish an encrypted link between the client and the server. Although the SSL protocol has the potential to be secure, it is generally misused. The result is that the transactions are not secure. Some of the problems are:

a) *Users don't check certificates*. The use of SSL does not guarantee that a user is interacting with the site that he/she wants to interact with. It only guarantees that the site implements SSL. A user needs to check the site certificate in order to verify the name under which the site is certified. This is usually neglected even by users that are aware of the existence of certificates, and it is unknown to the rest. Another problem is that some sites do not get a certificate using the same name by which they may be known to the user.

b) *Anybody can have a certificate*. Anybody can apply for a certificate and receive one.

c) *User credentials are stored in an unsafe place*. Most of the sites that use SSL do not require a user to have a credential to authenticate his/herself to the site. However, even when a site requires a credential and the browser supports the use of credentials, this alone does not guarantee that the user identified is the one interacting with the site. For example, credentials that are stored in an unsafe place, such as the user's computer, can be stolen. These credentials can then be used later by an attacker.

d) *The loss of a credential may not be noticed by the user*. Credentials that are stored in unsafe areas can be compromised without the user ever noticing. In contrast, if a token is used for safe transactions, the stealing or loss of a user's token is easily noticed by the user.

Commercial products that use tamper resistant devices to store certificates in order to prevent attacks against these certificates already exist. These products represent a move in the right direction and provide some protection against attacks to a user's credentials. They, however, use the tamper resistant device primarily for certificate storage, while still making most of the decisions in an unsafe area.

2.1.7 Summary

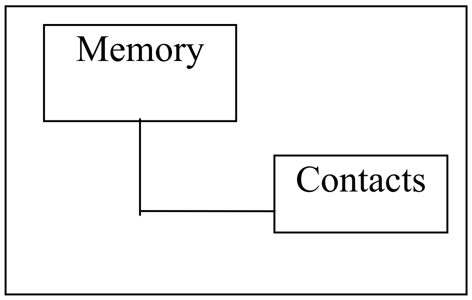
The Internet has grown at a phenomenal rate. One of the biggest contributor to this growth was the World Wide Web, which is accessed using browsers. Today, every person that uses the Internet knows how to use a browser. Because of this, browsers are not only used by expert computer users, but also by many users that have had little or no exposure to computers before the explosion in popularity of the World Wide Web. In addition, browsers use technologies like Java, JavaScript, ActiveX, and Plug-ins that were implemented with many security flaws. Although the implementations are patched as soon as a flaw becomes public, the "find and fix" approach is a band-aid solution.

2.2 Smart Card Technology

Although smart cards were patented about 20 years ago, they have only recently begun to be used in real applications. They have a strong presence in Europe, where they have been used with success for some time as pre-paid telephone cards. However, only after a big launch by VISA/Mastercard at the Olympic games are they becoming more known in the U.S.

Smart cards are cards the same size as magnetic strip credit cards that have an integrated circuit embedded in them, and they usually have contacts on one face. The shape, size, and voltages that are used are regulated by ISO 7816-x, which has many subtypes (from 1 to 7). ISO 7816-x also regulates some aspects of the operating system that is used in microprocessed smart cards. In the following subsections the available card technologies are discussed, beginning with cards with memory and no protection or processing and progressing to contactless cards.

2.2.1 Cards with memory

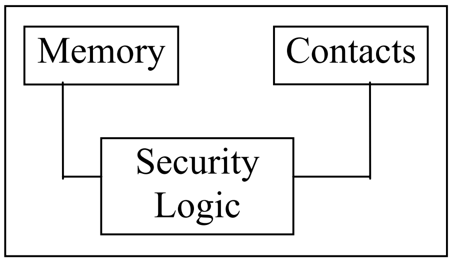


Memory cards are used mainly for the transport of information. They have an area of EEPROM (electrically erased programmable read only memory) that can vary in size from 256 bytes to 8 Kbytes depending on the card.

Any application can interface with this kind of card using a card reader and reading the memory. Because of this, these cards cannot be used for any secure application. Usually they are used to transport information that is used to initialize a system or by applications that only need to read information (often only once).

These cards were used when the price of cards with more functionality were expensive. They were used mostly as a proof of concept for the smart card, and they are being phased out due to a reduction in smart card prices and the limited functionality in these cards.

2.2.2 Cards with protected memory

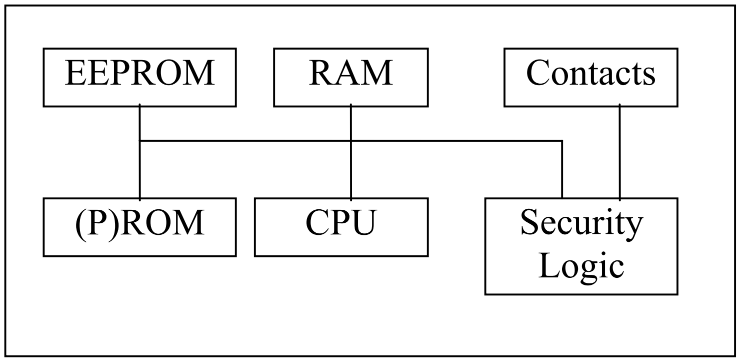


Cards with protected memory are used as pre-paid telephone cards in most of the world. They have between 256 bytes and 8 Kbytes of EEPROM memory. The difference between this kind of smart card and the former is that the memory in this case can not be written without first presenting a PIN code to the card, although it can still be read. The memory in these cards is divided into zones, and with the presentation of the pin code a zone can be individually blocked from write access forever.

They are used mainly for applications that need a medium level of security. Most applications that use these cards have a security module embedded in the card reader or card interface, where data processing takes place. The security modules are integrated circuit hardware with processing power, which are manufactured using the same layout and hardware protection that gives physical protection to the integrated circuits used in microprocessed smart cards. These security modules have software in their ROM that implements all the cryptographic functions and protocols to present the pin code to the protected memory cards and to log the transactions.

Although these cards have been used as pre-paid telephone cards, they have been proved to be unsafe, even for this application. The main reason for this is that the card has no processing power and all of the information flow, although encrypted, is information that is read from or written to a memory bank.

2.2.3 Microprocessed cards

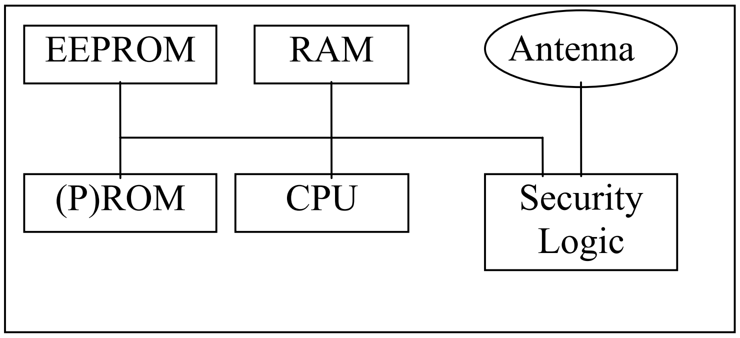


Microprocessed cards are the real “smart” cards. They have processing power (CPU) and an area of (P)ROM where the dedicated software is stored. They differ in the type of CPU and the type and size of memory. There are basically 2 different kinds of CPUs. One is from Motorola and has the same set of instruction as the 6805 line of microprocessors (8 bits). The other is compliant with the 8051 Intel set of instructions (8 bits). Only Motorola makes the former and many manufacturers, including Intel, Siemens and Philips, make the latter. The integrated circuit in these cards may also include a co-processor, which is used primarily for computing exponentials for RSA-like algorithms. The EEPROM memory, which is used for storing non volatile information, can range from 1 Kbyte to 16 Kbytes.

These cards are used for applications that need a high level of security, such as for electronic money. In most cases the application providers develop proprietary code that includes file manipulation and management, cryptographic protocols, and validation protocols, which are then given to the manufacturer of the integrated circuit. This code is burned in the ROM in a secure environment and sold back to the provider who puts it in the cards and initializes them.

Since these cards have processing power, they can be well protected against fraud. They use cryptographic protocols to communicate safely with the outside without compromising their private cryptographic keys, and they use random number generators to agree on a unique session key per transaction, eliminating the chances of replay attacks. They also have physical protection against corrosion attacks and electronic microscope analysis of the code.

2.2.4 Contactless Cards



Yet another category of cards that are emerging and that can similarly be divided into the above groups are the contactless cards [GEM97]. These cards do not have contacts and send and receive information using radio signals. They have the chip designed in the form of a square, as is the case in the cards with contacts, plus a circular antenna. The antenna and chip are embedded inside the plastic of the card. They have a very promising future, but there are still some technical problems to be solved.

2.3 Cryptography

Cryptography is defined as a collection of techniques that enables one to keep information secure [GAR97]. The idea of cryptography dates back to the Roman Empire. Roman and Greek generals used it to send messages to commanders in the field in a way that if intercepted would not disclose the content of the message.

The first techniques used for cryptography were substitution and transposition. Substitution is the technique of substituting each block of the message, where a block usually is considered to be one letter, by another block. The most famous substitution technique is called Caesar cipher and is attributed to the Roman Emperor Julius Caesar. His scheme for hiding the message, called encryption, replaces each letter by a letter that comes three letters after it. For example "d" is substituted for "a", "e" is substituted for "b", and so on. Thus, using the Caesar cipher "dissertation" would be encrypted to "glwwhuvdvlrq". The scheme for recovering the message, called decryption, is the reverse operation. Transposition is the technique of scrambling blocks, usually a sequence of letters, in the message. The goal of the technique is to make the scrambled message not understandable to anyone but the person who knows how to reverse the scrambling. For example, a block may be defined to be of size 6 and the scramble is to place the fifth letter of the original text in the first position of the scrambled text, the third letter in the second position, the sixth letter in the third position, the fourth letter in fourth position, the second letter in the fifth position, and the first letter in the sixth position. The text "dissertation" would have two blocks using this algorithm, since it has twelve letters. Each of the blocks would be encrypted using the rules above. The first block, "disser", would be encrypted as "esrsid" and the second, "tation", would be encrypted as "otniat". Thus, "dissertation" would be encrypted as "esrsidotniat".

A cryptographic algorithm usually employs the concept of keys in order to enable the encryption and decryption of a message. In a simple substitution cipher the key is the shift size (3 for Caesar cipher). In a transposition cipher it is the block size and letter positions ([5,3,6,4,2,1] in the example above). In modern cryptography a key is usually a set of bits that are used for encryption and/or decryption. The number of bits of the key, called the size of the key, is important to forecast the time needed by an attacker to try all possible values of keys in order to decrypt an encrypted message. The minimum key size to protect information is not constant and changes with time, as more computing power is available for a constant amount of money.

Modern cryptographic algorithms use mathematical properties to implement encryption and decryption transformations. The result of such transformations depends on the message and on the key. Encryption transformations are chosen such that it is very hard to apply an inverse transformation unless you have a correct key.

Modern cryptography can be divided into two basic types: symmetric key algorithms and asymmetric or public key algorithms. Symmetric key algorithms use the same key for encryption and decryption. Usually they are faster to encrypt and decrypt messages, but for each key all parties exchanging information based on this key need the same key and must keep it secret. Public key algorithms use a pair of keys, one for encryption and another for decryption. One of the keys of the pair is called a public key and is usually published somewhere so everybody knows it. The other is called a private key, and it must be kept secret.

In this research only well studied cryptographic algorithms are used. The reason for this is that because of the extensive scrutinization that those algorithms have been submitted to they are more probable to be safe and correct. The choice of algorithms will also be influenced by characteristics of smart card technologies, which enable some algorithms to run faster than others. These characteristics are due to the hardware design, which favors some mathematical operations over others. The following section briefly describes some of the algorithms that were analyzed, and the reasons for choosing the algorithms that were used are presented in section 3.1.

2.3.1 Description of the Algorithms

In the SAC approach the RSA algorithm is used for asymmetric encryption and decryption, and the triple DES algorithm is used for symmetric encryption and decryption. A number of other algorithms were considered, some of which are described in the next subsections.

2.3.2 DES

The Data Encryption Standard (DES) was officially published in 1977 [NBS77], but it is based on an algorithm developed by IBM in the early 70's called Lucifer. It has been widely used since then, and it is believed that there is no better method to break it other than a brute force attack (i.e., an attack that tries all possible keys). It is a symmetric block cipher that operates on 64 bit blocks. Its key length is 56 bits, but in many cases is expressed as a 64 bit key where every eighth bit is used for a parity check. The algorithm is a 16 round algorithm, where a round is the name given for the application of substitution and permutation based on the key and on the text resulting from the previous round.

As mentioned above, DES always operates on 64 bit blocks of plaintext. It divides the block into a left part (L), consisting of the 32 most significant bits and a right part consisting of the 32 least significant bits of the block. Starting with an initial permutation of bits, the algorithm applies a function f every round, using a different key k_i derived from the original key. The details of the function f and of the algorithm can be found in [NBS77].

2.3.3 RSA

RSA is named after the inventors, Ron Rivest, Adi Shamir and Leonard Adleman, and was first discussed in 1977 in a column of Scientific American[GARD77]. It is a very simple algorithm that uses private and public keys, and gets its security from the difficulty of factoring prime numbers. When a message must be encrypted, a public key, which is publicly known, is used, and the decryption can only be accomplished using the corresponding unique private key. The RSA algorithm is discussed in [RIV78].

2.3.4 Feige-Fiat-Shamir identification scheme

Uriel Feige, Amos Fiat and Adi Shamir modified a scheme originated by the last two for authentication and digital signatures so that it could be used as a zero knowledge proof of identity[FEI87]. The new scheme uses two large primes to generate a number n. The public key in this case is a string of k numbers and the private key is also a string of k numbers. After t transactions where there is a challenge and a response, the challenger is convinced that the party that is responding has knowledge of n and the private key. The authors of this scheme suggest using k = 5 and t = 4.

2.3.5 Diffie-Hellman key-exchange algorithm

The Diffie-Hellman key exchange algorithm[DIF76], like the other public/private key schemes mentioned here, is implemented using exponentiation and primes. The two sides that wish to exchange a key, initially agrees on a prime number n and a g, such that g is a primitive mod n. g and n do not need to be secret and can be exchanged at the beginning of the protocol over an insecure line. Side A chooses a large random number x and sends $X = g^x \bmod n$ to side B. Side B chooses a large random number y and sends $Y = g^y \bmod n$ to side A. Side A computes $k = Y^x \bmod n$ and side B computes $k' = X^y \bmod n$. It is easy to see that $k = k' = g^{xy} \bmod n$. So k is the key that was agreed upon by sides A and B.

2.3.6 Elliptic Curve Algorithms

The use of elliptic curve geometry over finite fields cannot be considered as an algorithm by itself. This field of number theory and algebraic geometry is used to implement public key algorithms.

Elliptic curve cryptography uses a field K that is usually the real numbers, the rational numbers, the complex numbers, or a finite field F_q of $q = p^r$ elements. A field that is particularly interesting for computation is the field F_q of $q = 2^r$ elements. The reason for this is that the computations over this field can easily be performed using bit operations like *and*, *or* and *xor*.

An elliptic curve over field K is defined as a single point at infinity together with the set of points (x, y) with $x, y \in K$ that satisfy:

$y^2 + y = x^3 + ax + b$, if K is a field of characteristic 2;

$y^2 = x^3 + ax^2 + bx + c$, where $x^3 + ax^2 + bx + c$ has no multiple roots and K has characteristic 3;

$y^2 = x^3 + ax + b$, where $x^3 + ax + b$ has no multiple roots and K has characteristic different from 2 and 3.

Elliptic curve geometry defines two basic operations: addition of two points and negation of a point. These operations are defined in such a way that the set of points over an elliptic curve form abelian groups. This is very important for public key cryptography, which uses finite abelian groups for its computation.

Many public key algorithms rely on the difficulty of the discrete logarithm problem. Given an elliptic curve *E* over a finite field and a point *B* in *E*, the discrete logarithm problem is defined as the problem of given another point $P \in E$, find an integer *x* such that $xB = P$, if such an x exists.

An example of an elliptic curve algorithm will help to clarify the concepts. The analog of the Diffie-Hellman algorithm, over an elliptic curve is as follows. Alice and Bob agree on a finite field, an elliptic curve *E* and a point *B* on this curve. Alice chooses a random integer *a* and sends to Bob *aB*. Bob also chooses a random integer *b* and sends to Alice *bB*. Alice, knowing *a*, can then compute *abB*. Bob, knowing *b*, can also compute *abB*. Anybody listening to the exchange is only able to know *aB* and *bB*, which is not sufficient to compute *abB* without solving the discrete logarithm problem.

The basic operation of a public key cryptosystem that uses elliptic curves is the multiplication of a point by a number. In order to perform this operation one needs to be able to do the operations of doubling a point *P* (*P* + *P*) and adding a point. An example would be the multiplication of a point *P* by 30. The result would be:

$$Q = 2*(1 + 2*(1 + 2*(1 + 2*P)))$$

The operations on the generic elliptic curve $y^3 + xy = x^3 + ax^2 + b$ are [SCH95]:

Addition of $P = (x_1, y_1)$ and $Q = (x_2, y_2)$.

- If P or Q is the point at infinity (O), then the result is the other point.
- If $x_1 = x_2$ and $y_1 \neq y_2$, then the result is O
- If $x_1 = x_2$ and $y_1 = y_2$, use the doubling rule
- If $x_1 \neq x_2$ then $(x_3, y_3) = P + Q$ is

  $\lambda = (y_2 + y_1)/(x_2 + x_1)$,

  $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$,

  $y_3 = \lambda(x_1 + x_3) + y_1 + x_3$

Doubling a point $P = (x_1, y_1)$

- If $x_1 = 0$ then $2P = O$
- If $x_1 \neq x_2$ then $(x_2, y_2) = 2P$ is:

  $\lambda = x_1 + y_1/x_1$,

  $x_2 = \lambda^2 + \lambda + a$,

  $y_2 = x_1^2 + (\lambda + 1) x_2$

Negating a point $P = (x_1, y_1)$

- $-P = (x_1, x_1 + y_1)$

Elliptic curve algorithms have characteristics that are very appealing to devices with limited storage space, such as smart cards. The reasons why some cryptographic algorithms were chosen over others for the SAC approach is discussed in Chapter 3.

2.4 The Container Model

Military systems usually handle sensitive information. Because of this, military systems classify all information as having a security level. The security level represents the amount of damage that would result from the public release of the information. An individual piece of information is called an object. Military systems also classify every user, process and device that handles information. These active entities are called subjects. This classification is called the clearance of the subject and represents the set of information that the subject is allowed to access. Several security models were developed to dictate the way a subject can access an object. They use a hierarchical model that orders classifications, e.g. unclassified, confidential, and secret. In addition, they use this ordering within compartments, e.g. crypto or nuclear, to give a finer granularity to the accesses.

A very popular security model used by military systems is called the Bell and LaPadula model [BEL73]. This model can be summarized by two properties. One, called the simple security property, does not allow a subject to read an object if the subject has lower clearance than the object's security level. The other, called the *-property (pronounced star property), does not allow a subject that has read access to an object to write to another object that has a security level lower than the object to which the subject has read access. Because of the very restrictive nature of the two properties, the Bell and LaPadula model includes the concept of trusted subjects, which are subjects that are allowed to perform operations that violate the *-property.

Military message systems have several particularities that make the Bell and LaPadula model unsuitable. [LAND84] describes the problems that four military message systems had when using the Bell and LaPadula model for their security policy. These systems are the military message experiment, the air force data services center multics, the kernelized secure operating system, and the guard message filter.

An important characteristic of military messages systems is that a message may have portions of it that are classified at different security levels. These portions are the fields of the message, e.g. the *to*, *from*, *subject*, and *text* fields, all of which have a classification; or it may be a subfield, e.g. paragraphs of the text field. The whole message is classified at the highest of these, possibly different, security levels. [LAND84] proposes a security model to handle the multilevel security of these messages based on a container model. Using this model, information can be objects and containers. Objects are single-level units of information and containers are multilevel information structures. Objects and containers have their security classification and can be referred to as entities. An entity may be accessed directly using a unique identifier (e.g. the file /mail/andre/di.txt), or indirectly using a set of entities names (e.g. the aerial photo in the container photos from Washington). Containers may require users to have a minimum clearance, which may be higher than some of the classifications of the entities contained in it, in order to access these entities through an indirect reference that uses the container name. An attribute called *container clearance required* specifies if a minimum clearance is required or not. Direct accesses are only based on the clearance of the subject and the security level of the entity.

Giving different security levels for fields of a container allows subjects to manipulate these containers without requiring it to have the highest clearance of all fields. This enables, for example, routers to route a message based on the *to* field. The router must only have clearance to read the *to* field. It does not need clearance to read the text of the message, which in this case is clearly not necessary. The container model also allows a user to browse messages by the *subject* fields that he/she has clearance for, even if this user is not cleared to read all, or part of, the *text* fields of every browsed message.

2.5 Alexandria Digital Library

The Alexandria Digital Library (ADL) project was one of six projects that were part of the Digital Library Initiative (DLI). Phase one of the Digital Library Initiative (DLI1) ended in 1998, followed by a phase two (DLI2), which will last for another five years. The digital library research that started with the ADL continues to be supported in DLI2 under the name of Alexandria Digital Earth Prototype (ADEPT). The ADL developed in DLI1 is currently being used at the University of California and will be part of the California Digital Library (CDL).

This dissertation refers to Alexandria Digital Library (ADL) without differentiating between the digital library designed under DLI1 and the one in development in DLI2. The ADL, which is an interdepartmental project based at the University of California at Santa Barbara, has two goals [FRE96]:

a) Explore and research the problems of a distributed digital library for geographically-referenced information;
b) Build a test bed system that can be used in the operations of a digital library.

In order to achieve these goals the researchers have built a test bed that is in constant evolution. The architecture of this system is a three-tier system that is described in [FRE98] and reproduced in Figure 1.

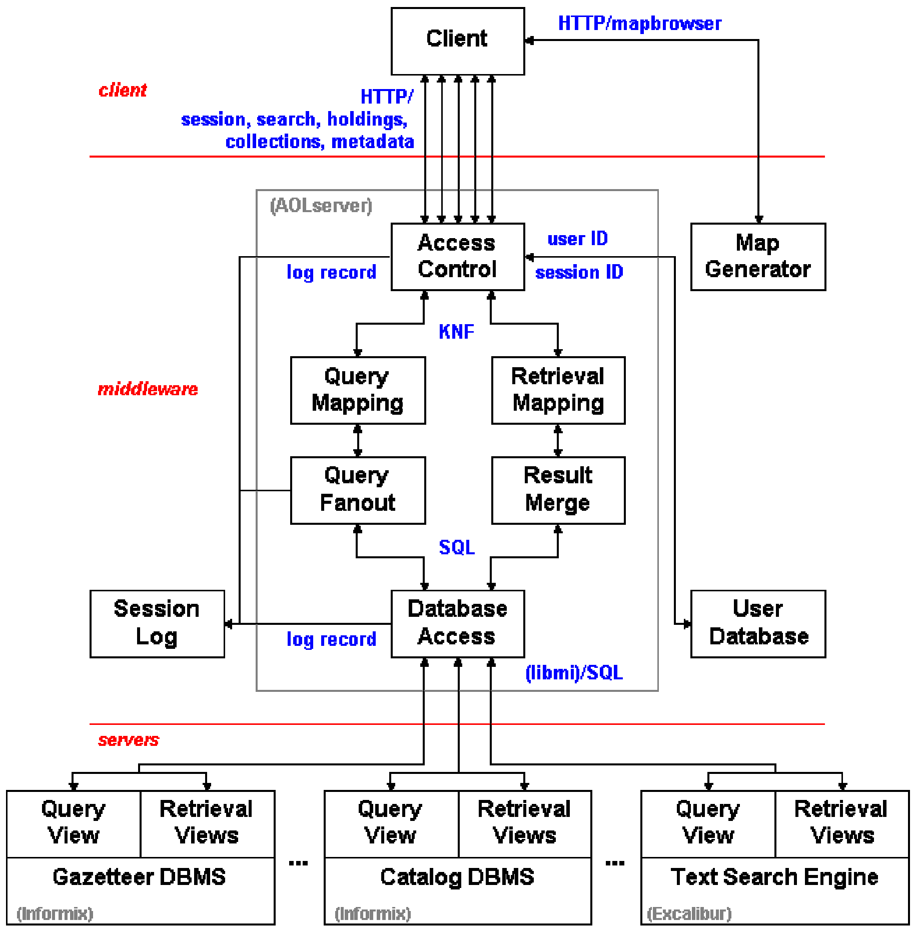


Figure 1: Current ADL architecture

The Alexandria Digital Library (ADL) uses the Internet and the HTTP protocol, which will be replaced by the secure socket layer version of this protocol (HTTPS), for accessing information. The client application is a proprietary program called Java Interface to Geospatial Information (JiGi). JiGi is a Java application that provides a graphical interface for interacting with the ADL. Although JiGi is a stand-alone application, it uses standard browsers for some operations.

In the current system, JiGi opens an HTTP connection to the middleware, and this connects to the ADL server, which is centralized and unique. The middleware interacts with the other parts of the system (basically authentication and database operations) through CGI scripts that access files of map data using C calls and database calls, which in turn use SQL for accessing database information. In the next implementation the access to the database, which will be distributed, will be implemented using CORBA and SQL.

The only security features currently in place are authentication through passwords and restrictions on the client hosts that can access the library. The middleware running a CGI script authenticates the user using a database that cross-references the user ID with the password. The restriction on clients able to access the library is accomplished by checking the host name of the client requesting service. If the host name belongs to a domain that is authorized to access the library, then the access is granted.

A big concern of the Alexandria Digital Library is how to be able to expand to include different types of users that have different access permissions and different data providers that have their own security policies. The users will be divided into groups, and each group will be able to receive different sets of data. In some cases a group will be able to check out information that will be degraded; in other cases the group will not be able to access any of the desired information. Some degradation that may be done to data is:

a) Reduced data resolution (for example in images);

b) Layers of data not available (in multi-layer data, e.g. GIS);

c) Summary for text data;

d) Clip for multimedia information.

The data providers will be a very important part of the Alexandria library. Through its system, ADL will also be able to provide data that third parties are interested in making available or in selling. These third parties will have their own clients, which will also have their access to the data defined with respect to the group to which they belong. The access will be defined by the third party that is interested in joining the digital library.

For more information on the Alexandria Digital Library the reader should see [FRE96]. For more information on the Alexandria Digital Earth Prototype the reader should see [ADE99].

CHAPTER 3

Safe Areas of Computation

The Safe Areas of Computation approach is a generic approach that can use any tamper resistant devices. On the server side the implementation can use many different devices, such as the IBM crypto controller [SMI99]. This is because the server has fewer restrictions on mobility and costs than the clients. In contrast, for a client implementation, smart cards are the devices that more closely meet the requirement of a Safe Area of Computation.

The next subsection describes the reasons for selecting the cryptographic algorithms to be used by the approach. Section 3.2 describes the SAC model, and section 3.3 describes the test bed implementations used for protecting Internet transactions. Section 3.4 analyzes the performance of the implementation. The chapter ends with section 3.5 describing extensions that can increase the smart card security perimeter and with section 3.6 describing possible schemes for charging fees for the use of a system based on the SAC approach.

3.1 Choice of Cryptographic Algorithms

The Safe Areas of Computation approach uses cryptography to provide strong authentication and secrecy of data. There are several algorithms that could be used in the SAC approach to accomplish this goal. This section explains the reasons for the choices made and describes some of the reasons for not choosing other algorithms.

The use of smart cards as one of the means to implement the client Safe Area of Computation put restrictions on which cryptographic algorithms are better suited to be used. The main restrictions placed on the algorithms are memory size and processor speed. However, as will be explained, these restrictions do not force the use of weak encryption schemes or keys.

Smart cards have a ROM area that is programmed during the manufacturing of the integrated circuit (IC) and an EEPROM area that can be programmed at any time (restricted by a finite number of erases). Usually the ROM area contains the operating system, which provides as much functionality as possible. In addition to standard functions, the operating system may also implement some cryptographic algorithms. Clearly an application that uses one of these cryptographic algorithms only needs to use three bytes in order to make a function call to the operating system. Thus, the application size will be much less than what is needed for an application that has to implement its own cryptographic algorithms. The operating system used for the test bed implementation is CardOS M 3.0 from Siemens. It currently has both DES and triple DES implemented, and it will have RSA implemented in a future version.

The processors used for smart cards run at a very low speed, as compared with traditional personal computer processors. The processor used for the test bed implementation was the top of the line Siemens SLE66CX160S running at 5 MHz. One advantage of this processor is that it has an integrated coprocessor that is optimized to perform modular arithmetic on large numbers.

The algorithms chosen to implement the SAC cryptographic protocols take these restrictions into consideration. The SAC implementation uses triple DES with three different keys for symmetric encryption and decryption and uses RSA with a 1024 bit key for asymmetric encryption and decryption.

3.1.1 Strength of the chosen Algorithms

Recent experiments have demonstrated that single DES is easy to break given the power of currently available processors [EFF98], and a new symmetric algorithm called Advanced Encryption Standard (AES) is being developed as a substitute for the DES [NIS99a]. While there is no decision about which algorithm will be chosen as the AES, the triple DES algorithm is still believed to be appropriate for encryption and decryption of sensitive data [NIS99b].

RSA has its strength in the difficulty of factoring large numbers. There have been some new ideas about using identification of smooth numbers to break RSA [SHA99]. Although this theoretical attack would reduce the complexity of the key, a 1024 bit key would still be appropriate.

3.1.2 Other Algorithms Considered

Many other algorithms were considered for use in the SAC approach. Some of them were immediately discarded because it was a prerequisite not to use any algorithm that requires licensing, either because they have a valid patent or because they are proprietary. Others were discarded in favor of algorithms that have been more rigorously scrutinized. Some of the algorithms that were considered for the protocol are discussed in the next subsections. They are Diffie-Hellman, Feige-Fiat-Shamir and algorithms based on elliptic curve mathematics. The following subsections describe the results of testing these algorithms and the reasons for not using them.

3.1.2.1 Diffie-Hellman

The Diffie-Helman algorithm was originally considered for the session key exchange. It is a well-studied algorithm that was patented, although the patent was valid only until April 29, 1997 [SCHN96]. One problem with this algorithm is that it is susceptible to man-in-the-middle attacks [SCHN96]. Because of this its messages must be protected using an additional algorithm. The original SAC prototype used RSA encryption to encrypt messages used by the Diffie-Helman algorithm. The two main problems with this approach are:

a) The protection against man-in-the-middle attacks used in this approach is RSA encryption and decryption used as a digital signature. If RSA can be broken, then it is possible to perform a man-in-the-middle attack. Thus, no protection is provided if RSA can be broken.

b) RSA has its strength in the difficulty of factoring a large number, and Diffie-Hellman has its strength in the difficulty of finding discrete logarithms. These two problems are related and although there is no proof that factoring large numbers can lead to finding discrete logarithms, the opposite has been proved [SCHN96]. So there is reason for concern that if RSA can be broken by a method for factoring big numbers, the same method could possibly be applied to find discrete logarithms. For example, the Number Field Siege (NFS) is a method that can be applied in both cases.

Of course there is the possibility that if RSA can be broken, the method used may not also be applicable for breaking the Diffie-Hellman algorithm. However, even if this is true, the continued use of the Diffie-Hellman algorithm would only be secure if the method used to break the RSA algorithm would not be efficient enough to allow a man-in-the-middle attack, or if an attacker would not be able to get positioned in a place that enables such an attack. It was decided that this is not enough to justify the use of the Diffie-Helman algorithm. Therefore, the current SAC protocol does not use the Diffie-Hellman algorithm.

3.1.2.2 Feige-Fiat-Shamir

The Feige-Fiat-Shamir algorithm was used in early versions of the SAC approach to mutually authenticate the client and the server. However, it was abandoned because:

a) The algorithm was used at the beginning of the data exchange between the client and the server, but it did not directly result in a session key. The

RSA algorithm was used after authentication to agree on a session key. Due to the possibly open nature of the connection between the client and the server (e.g. Internet) it is reasonable to assume that if this algorithm is used this connection may be tampered with just after authentication and before the beginning of the RSA algorithm. The result of this would be that the authenticity of the parties exchanging data could not be guaranteed during the session only based on the result of Feige-Fiat-Shamir algorithm.

b) This algorithm requires many interactions between the client and the server. This results in a close correlation between the client-server connection speed and the time required to execute the algorithm. This may be unacceptable for slow speed connections.

It is possible to use Feige-Fiat-Shamir with only one interaction between client and server and as a signature scheme. This is faster than RSA [SCHN96] and could be used for the SAC approach. However it would require an additional encryption algorithm to keep the session key secret. Thus, it was decided that this algorithm did not provide enough benefits to justify its use.

3.1.2.3 Elliptic Curve Algorithms

Elliptic curve geometry was also considered for use in the SAC approach, since algorithms based on elliptic curve geometry have many desirable features. The remainder of this section describes the preliminary results of its use.

The discrete log problem that makes it hard to break algorithms like Diffie-Hellman, is likely to be more intractable on elliptic curves than on finite fields [KOB87]. Based on this assumption one can conclude that for a fixed integer *r* it is more difficult, and takes more time, to compute the discrete logarithm over a finite field F_{2^r} on elliptic curves than over the same field on natural numbers. As a result, it is possible to use a smaller *r* for cryptographic algorithms that use elliptic curves. This is especially interesting for protocols and systems that use hardware devices with limited storage capacity.

A test implementation of the basic mathematics of elliptic curve geometry was constructed based on the work of [SCH95]. This implementation used the Siemens SLE66CX160S chip. This chip has an advanced crypto-engine (ACE), which is optimized to perform modular multiplications and reductions. The ACE has three registers of 1120 bits and a memory area of 2240 bits, which can be software configured to work either as two 1120 bit registers or four 560 bit registers.

The field chosen was the Galois field F_q, with $q = 2^{155}$. In this binary field, addition and subtraction are not computing intensive, since these operations are represented by the boolean "xor" operation. Multiplications use shifts and additions, which require that the processor have the ability of shifting large numbers. Another operation that is intensely used for inversions is the "and" operation.

Two implementations were tested: one using the advanced crypto engine to perform the necessary operations for large numbers and another without its use. A number of performance measurements were made on some of the basic routines, and it was clear that the advanced crypto engine (ACE) speeds up the calculations. The reason for this is that the ACE performs operations with large numbers in less processing cycles, on average, than the eight bit microprocessor. The ACE, however, has some limitations that result in unexpected performance hits.

One of the limitations of the ACE is that it is not able to perform “and” operations. The ACE, however, has the ability to expose the most significant bit of one of its registers to the eight bit microprocessor. This can be used to perform an “and” operation with the help of the eight bit microprocessor. An implementation was designed to test this performance. Unfortunately, it was found that this implementation had worse performance than performing the “and” of two 155 bits numbers, 8 bits at a time, without using the ACE.

The ACE has one register that can perform variable length shifting of bits. However, the ACE has another limitation which is the difficulty of performing variable length shifting of bits by two registers. This was overcome by using pre-shifted copies of numbers in different registers.

The ACE also has very limited processing power. The eight bit processor sends one or two byte commands to the ACE for processing, and this results in the eight bit processor having to drive all processing that has not been optimized by the ACE designers, such as modular multiplication. Another disadvantage of the ACE is that it does not have the ability to perform software loop operations. Many applications, including the SAC application, that require the processing of large numbers (numbers with much more than eight bits) could use the ACE in a more efficient way if the ACE was able to store a few instructions and repeat them in a loop without the intervention of the eight bit processor. This is already done by the modular multiplication instruction, but it cannot be programmed in software.

Because of the limitations mentioned above, the results of these benchmarks were very disappointing. A multiplication of a point by an integer took approximately 1.75 minutes, which is not acceptable. Therefore, the use of elliptic curves was postponed until a future time when smart cards have better processing power.

The implementation tested used a binary field for the elliptic curve, because this is the method used in generic computers. This does not take advantage of the fast modular multiplication characteristic of the ACE. Possible future work that could result in improved performance would be to use integer fields instead of binary fields. Although this would result in fast multiplications and additions, a penalty would still have to be paid for inversions. It is not known at this time if the inversions would be representative enough to discount the gains of the multiplication and addition speed up.

3.2 The SAC Model

In a client-server configuration a client SAC interacts with a server SAC through an insecure medium. In this configuration, the user is not protected when interacting with either a client SAC or a server SAC alone. For example, in order to safely browse the web a user must interact with a server site through a client, both of which have safe areas of computation. If either the client or server site does not have the SAC the interaction will not be protected.

The SAC, however, does not need to enforce the security policy of the computing system as a whole. The purpose of the SAC is to enforce a security policy tailored to the interactions of specific applications that need to be secure. The use of generic trusted areas that implement security critical functions to protect particular applications makes this approach different from approaches that use specific hardware for integrity checks of the computing system [YEE94].

The SAC approach uses the concept of a security perimeter, which is defined as the imaginary boundary of a trusted computing base (TCB) within which all security-related functions are executed [RUS91]. The trusted devices that compose a SAC define the SAC's security perimeter. The SAC can be as simple as a standalone smart card or secure coprocessor or it can be a collection of trusted processors, keyboards, displays, interfaces, etc.. The idea is to increase the number of safe operations that can be performed by adding trusted devices. As each new trusted device is added the security perimeter is extended.

3.2.1 Metadata

In the SAC approach data is organized in multi-level security containers. The container model used is a variation of the container model described in

[LAND84], which was proposed for military multi-level security documents, where each container is an abstraction for a set of data that has some attribute in common.

The SAC approach guarantees that only data to which a user has clearance is released outside the SAC. However, it is a function of the application software to present these containers and a description of the data inside the containers to the user for browsing and possible selection, after the SAC has approved access to them. To achieve this the SAC approach represents information using metadata, which has three fields: description, security attributes, and pointer. The metadata can be associated with both containers and data. For containers the pointer field in the metadata information is a pointer to the container header, and for data this field is a pointer to the actual data. The security attributes field represents the clearance a user must have in order to receive the data or header pointed to. A user who does not have the necessary clearance will not even know about the existence of the data or header to which the pointer points. The description field is used by the software that the SAC is protecting. It contains all the necessary information for the software to interpret the metadata and to present it to the user for browsing and selection (e.g., MIME type information).

A header is a set of zero or more metadata entries. The possibility of a header having zero entries enables a pointer to a container header to have any security label without releasing additional covert information about the existence of classified data. In particular, this enables an unclassified pointer to a classified container header. This fact is used by the SAC approach to enable users to start their sessions with unclassified pointers to (possibly empty) container headers. An analysis of these pointers does not reveal the existence or the number of containers with classified data, since the pointers can, and some will, point to empty container headers.

Consider a user making a query to request images of Santa Barbara, California. An example return value from the query could be the information shown in Table 2. In practice there would be many entries, each with metadata that satisfies the query parameters. Since the security label of the example container "Satellite images of Santa Barbara from July 14th, 1998" is unclassified, any user would be able to request the container header. If a user requests the container header, the information in Table 3 will be sent in a secure way to the client SAC. This information represents the metadata that can be used to request further data or containers. By using containers inside other containers, any level of indirection, with possibly different clearance requirements, can be supported using this approach.

The user is shown representations of only the metadata from Table 3 to which he/she has clearance. For instance, if the user does not have secret clearance, the "Satellite Image 4 (infra-red)" metadata will not be shown. That is, the decrypted metadata about "Satellite Image 4 (infra-red)" will never leave the client

SAC. Next, the user may choose and request further metadata of interest, based on the information shown.

DESCRIPTION	SECURITY LABEL	POINTER
Container: “Satellite images of Santa Barbara from July 14th, 1998”	Unclassified	ContainerServer::DB0::satellite::SB071498
Container: “Aerial Photos of Santa Barbara from July 14th, 1998”	Restricted to UC students and faculty	ContainerServer::DB1::aerphoto::SB071498
⋮	⋮	⋮

Table 2: Pointer to a container with satellite images

DESCRIPTION	**SECURITY LABEL**	**POINTER**
Visible Image: “Satellite Image 1”	Unclassified	Mainserver::DB0::satellite::SBvis071498
Visible Container: “Container of classified Satellite Images”	Confidential	ConfidentialServer::DB0::satellite::SBvis071498
Infra-Red Image: “Satellite Image 3” (infra-red)	Unclassified	Mainserver::DB1::satellite::SBir071498
“Satellite Image 4” (infra-red)	Secret	SecretServer::DB0::satellite::SBir071498

Table 3: Header of the container in Table 2

3.2.2 Access Control Lists

The SAC approach uses three types of access control lists (ACLs) to implement the various security policies based on the security required to access the data and the clearance of the user. These ACLs are stored in the client SAC, and this is where the checks are made. The access control lists are shown in Table 4 with the size of each of the record types. Although the sizes in the table, and in the test bed implementation, are in bytes, they can be changed to accommodate larger fields without changing the meaning of each ACL.

ACL	Size (fields)
Simple	1
Hierarchical	2
Complex	4

Table 4: Access control lists and record sizes

The simple ACL table contains one-byte records. When an access to a header or data item requires a simple clearance, the client system presents the required clearance to the client SAC with the request for data. The client SAC checks its simple ACL for the necessary clearance (a one-byte number). The data or header will only be transmitted or decrypted if the user has the required clearance.

The simple ACL has the ability to easily give a yes or no answer. However, in many cases, it is desirable to use hierarchical rather than absolute security clearances. This is the case, for example, when hierarchical clearances, such as classified, confidential, secret and top-secret, are used. For instance, a top-secret clearance could access any document. A secret clearance could access secret, confidential and classified documents, and so on. The hierarchical ACL is used to implement this policy. Each record has a field representing a low boundary and a second representing a high boundary. The security attribute of a header or data item that requires a hierarchical clearance contains the required clearance (a number), and the client SAC checks each record of its hierarchical ACL against the clearance. Access to the data or header is granted if the necessary clearance lies between the low and high boundary of one of the associated ACL records. For example, a hierarchical ACL could have the entries shown in Table 5, where each entry has the value that would actually be in the ACL, and the meaning of that value is given in parenthesis. In this example, data with a security attribute of 20 (<navy, secret>) can be accessed, while data with a security attribute of 87 (<nuclear, top-secret>) can not be accessed by this particular user.

Low	High
2 (<army, classified>)	4 (<army, secret>)
18 (<navy, classified>)	21 (<navy, top-secret>)
84 (<nuclear, classified>)	86 (<nuclear, secret>)

Table 5: Example hierarchical ACL

The two ACLs described above are used to represent static permissions. It is possible for the server to request changes to the ACL, but this is assumed to occur infrequently. For permissions that change frequently and for more complex security requirements complex ACLs are used. An example of this type of permission is allowing a user to access documents only a certain number of times per day. Each record of a complex ACL has four fields that represent:

- Security Label. This is a yes or no field that represents the clearance of the user. The first check the client SAC makes on the complex ACL when it has to decide about a complex access (access that requires a check in the complex ACL), is to search for a record that has the required security label. The access is denied if there is no such record.

- Reference. This is a field that is used as reference for changes. It is this field that indicates what hour, day, year, concert, etc… is being considered to make a decision.
- Tokens Remaining. This is the field that is used to store the data that is going to vary frequently (often a counter). The base for a decision of granting or denying access to data will be in this field in most cases.
- Reset Value. This field is a value that is transferred to the "tokens remaining" field when some particular actions occur, for example the reference value is updated.

An example of a complex ACL is shown in Table 6. The reference field in Table 6 represents the day of the year of the last operation on this field. Thus, the first entry may represent that the user can perform transfers, that he/she has last performed a transfer on September 9th, that he/she still has 10 tokens (meaning for example $100.00) that can be transferred on September 9th, and that if the current day is later than September 9th, then the tokens remaining field must be reset to 30 (meaning for example $300.00). This is an example of a complex ACL that enforces the policy that a user can transfer up to $300.00 per day.

Security Label	Reference	Tokens Remaining	Reset Value
2 (transfer)	252	10	30
8 (buy stock)	12	20	30

Table 6: Example complex ACL

The security attribute of a header or data item includes information that describes which ACL the client SAC has to check to make a decision. This information is a one-byte (8 bits) value. If the 5 most significant bits are zeroes than the check is performed against the simple ACL or the hierarchical ACL as shown in Table 7, which also shows a value for empty containers. The possibility of a container being empty is discussed in section 3.2.1 and is used to prevent covert channels. The client SAC does not try to interpret data for empty containers, knowing that this is a disguise to prevent a user from knowing that the container is empty. In addition to specifying which ACL to check, this value describes an operation to be performed in complex ACLs, as shown in Table 8. In some cases when a complex ACL is required, another value called the comparison value is also sent to the client SAC. The comparison value is used for making a decision, such as what the current date or time is.

Operation	**B_7**	**B_6**	**B_5**	**B_4**	**B_3**	**B_2**	**B_1**	**B_0**
Check simple ACL	0	0	0	0	0	0	0	0
Check hierarchical ACL	0	0	0	0	0	0	0	1
Empty container/data	0	0	0	0	0	0	1	0

Table 7: Access control list operations for non complex ACLs

Operation	B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
Add the value $B_2B_1B_0$ to the "tokens remaining" field	1	0	X	X	X	X	X	X
Subtract the value $B_2B_1B_0$ from the "tokens remaining" field	0	1	X	X	X	X	X	X
Check if comparison value is the same as "reference" field	X	X	0	0	1	X	X	X
Check if the "tokens remaining" field is positive.	X	X	0	1	0	X	X	X
Check if comparison value is the same as "reference" field. Copy the "reset value" field into the "tokens remaining" field if they are not the same.	X	X	0	1	1	X	X	X

Table 8: Access control list operations for complex ACL

Operation	**B_7**	**B_6**	**B_5**	**B_4**	**B_3**	**B_2**	**B_1**	**B_0**
Check if comparison value is lower than “tokens remaining” field.	X	X	1	0	0	X	X	X
Check if comparison value is greater than “tokens remaining” field.	X	X	1	0	1	X	X	X
Check if comparison value is lower or equal to “tokens remaining” field. If it is, subtract it from “tokens remaining” field and update the field for this value	X	X	1	1	0	X	X	X
Reserved for future use.	X	X	1	1	1	X	X	X
Reserved for future use.	1	1	X	X	X	X	X	X

Table 8 (cont): Access control list operations for complex ACL

Some of the operations performed on complex ACLs that are shown in Table 8 will change the “reference” field and/or the “tokens remaining” fields. It would not be safe to trust requests from the client platform to decide whether to commit these changes, since a malicious client could exploit the possibility to change the value of one or both of these fields. On the other hand, the SAC approach may need to use the updated value to decide whether or not to contact the server, and whether or not to deny the operation on the client side. To address this problem, the client SAC uses a temporary variable to hold the updated value and bases its decision to contact the server on the value of this temporary variable. If the server SAC approves the operation the temporary variable is saved as a field of the complex ACL record.

An additional threat when using a complex ACL is present when an addition operation is requested. As discussed above, the addition is committed only as a result of a server confirmation. This could be used for replay attacks during a session, since the session key would be fixed. In order to prevent this, the server sends an additional 64 random bytes when confirming an addition operation. The "xor" operation is applied to these 64 bytes and the session key to produce a new session key, which is used from this point on. The same scheme could be used to prevent replay attacks against other operations that change fields in a complex ACL entry. Currently, this is not the case, since in the current system replays of the other operations do not represent any advantage for a user.

Table 8 also specifies two operations in complex ACLs that may be combined with others. These operations are the addition and subtraction of $B_2B_1B_0$. If they are combined with any other operation, they are always executed after the other operation, and only if the other operation has been successful. For example, the value 10010001_b will add one token to the tokens remaining field after checking that this field is positive. Nothing happens if the tokens remaining field is non-positive. Both the addition and subtraction operations cannot be specified for the same operation; therefore, the 64 possible operations that start with the bits 11 are reserved for future use as shown in Table 8.

A number of example policies using the complex ACL are presented in section 3.2.4.

3.2.3 The Client-Server SAC Architecture

When used in a client-server configuration, the SAC architecture is as shown in Figure 2. The following subsections describe both the client SAC and the server SAC in more detail.

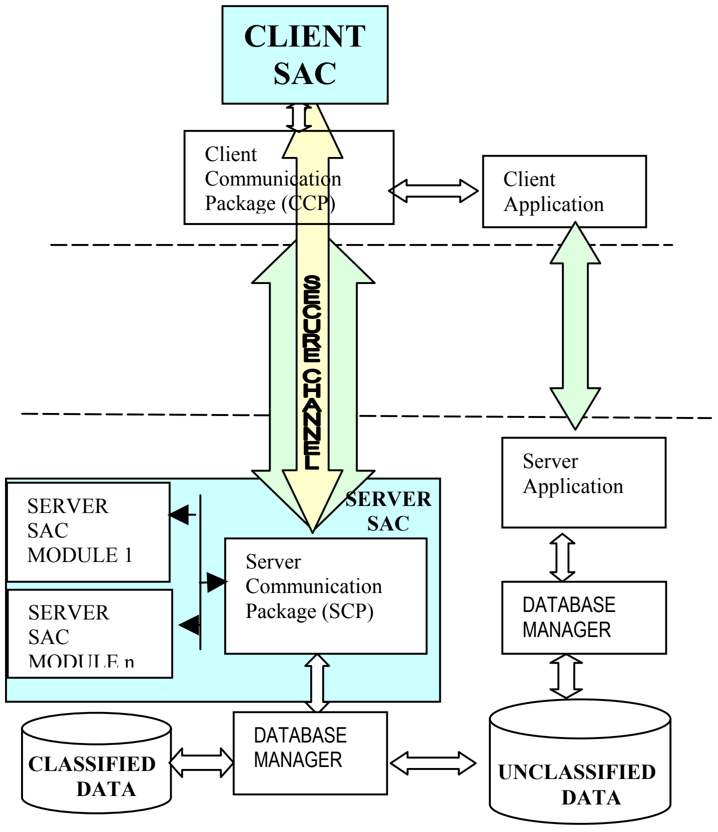
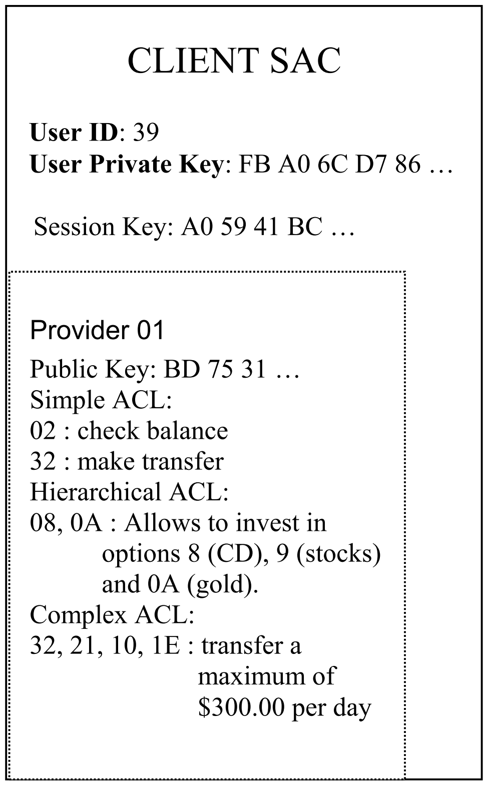


Figure 2: SAC in client-server configuration

3.2.3.1 The Client SAC

The client SAC and Client Communication Package (CCP) together are used to provide security without interpreting or presenting the data to users for browsing and selection. The client SAC has all of the cryptographic keys and the access control lists necessary for safe transactions with the server, as shown in Figure 3. After the client SAC receives encrypted metadata from the server SAC, it decrypts it and uses the security label fields to check if a specific user has the appropriate clearances to receive the metadata. If the user is allowed to access the data, then the client SAC hands the decrypted metadata to the particular application via the CCP. The application then presents this metadata to the user to browse and query. The application software primarily uses the description field of the metadata to accomplish this, but in some cases it may use additional fields.

The CCP is not inside the security perimeter and is not considered a safe area. Because of this it does not deal with clear text data until the data has been authorized for access. The application software interacts directly with the CCP using an API that has generic functions. It does not interact with the client SAC. The API provided deal with user and server authentication, requests for headers, and requests for data.



CLIENT SAC

User ID: 39
User Private Key: FB A0 6C D7 86 …

Session Key: A0 59 41 BC …

Provider 01
Public Key: BD 75 31 …
Simple ACL:
02 : check balance
32 : make transfer
Hierarchical ACL:
08, 0A : Allows to invest in options 8 (CD), 9 (stocks) and 0A (gold).
Complex ACL:
32, 21, 10, 1E : transfer a maximum of $300.00 per day

Figure 3: Client SAC

The initial queries of the server application do not involve the SACs; that is, the initial query of the server is made directly from the client application to the server application over an unprotected communication channel. The metadata returned by this query is a set of unclassified descriptions of the data or containers, in addition to their pointers. Although the metadata returned in the unclassified query is unclassified, it can point to a classified, potentially empty, container or data. The SAC components are required in order to request classified data or containers. The advantage of this approach is that if a user wishes to browse only unclassified data, then the SAC components will not be involved, and some may not even be present. The disadvantage is that a user may be returned a pointer to a container that has no data. This, however, is not believed to be critical since the user will immediately find that he/she is following a link that results in no data after the container header is requested. The operations for interactions with classified data or containers and the API functions that support these interactions are described in the next subsections.

3.2.3.1.1 User and Server Authentication

A user needs to start a session with a secure server to request classified data and containers. In order to start a session with a secure server the user must be authenticated to the client SAC and the client SAC and server SAC must mutually authenticate each other. If this is not done or if the authentication fails, then any additional requests from the application software to the CCP will be denied.

The application software interacts with the user to request a pin code. After this pin code has been entered, the application software calls the API function:

boolean

AskSACAuthentication(unsigned char pin[PIN_SIZE], unsigned char ServiceP[SERVP_ SIZE])

This function takes as input a pin code of size PIN_SIZE, represented as an array of unsigned characters. The SAC approach (and the test bed implementation) does not make any assumption about the size of the pin. However, currently the size is four.

The choice of a 4 digit pin code is used because the embedded functionality in some smart cards uses a 4 digit pin code to authenticate a user to the card. The Siemens SLE66CX160S used in the test bed does not have this functionality. In this case, the pin code is stored in a protected file that can store a pin code of any size. The card does not allow any transaction with it to occur without the correct pin code. Although four is a small number of digits, which could suggest a brute force attack, the card only allows three consecutive wrong pin codes to be presented. If a fourth wrong pin code is presented, the card will lock itself and will not work anymore. In order to unlock a card and return it to service, a special sequence of commands must be invoked on the locked card along with the use of a secret master key.

The second input parameter is also an array of unsigned char. This parameter, the service provider ID, is used to specify which service provider, from the many that may be stored in the card, is to be used for authentication. Each service provider has its own set of keys and access control lists, and these lists will be interpreted in the context of the authenticated service provider. The test bed implementation uses an ISO-7816 compliant subdirectory for each service provider. The service provider ID, which is the same as the subdirectory ID is specified as a two byte value. Because of this, although the SAC approach does not impose any restriction, the value of this parameter, which is given by *SERVP_ SIZE*, is currently fixed at two.

After the user has been authenticated to the client SAC the client SAC encrypts a message containing its unique ID and a random number using the RSA algorithm [RIV78] and the public key of the server. Upon receiving this message, the server looks up the public key associated with the specified ID and encrypts an answer message containing another random number generated by the server using this key. The Boolean "xor" of these random numbers will result in a 168 bit triple DES session key. Triple DES using the session key will then protect all information exchanged by the SAC's against eavesdropping and changes. This provides a virtual secure channel between the client SAC and the server SAC.

The encryption using the public keys of the server and client SACs prevents any eavesdropper from getting access to any part of the session key. The composition of the session key using two random numbers prevents any replay attacks.

3.2.3.1.2 Client SAC Updates

The Server SAC may request an update of the client SAC, immediately after mutual authentication. The request is encrypted using the session key and a session is not considered valid until the server has received a message from the Client SAC confirming the update. This update may result in the addition of service providers or a modification of the ACLs corresponding to the service provider involved in the authentication. In addition to these updates, special service providers can also invalidate a client SAC.

The server SAC checks a database for the necessary updates of a user ACLs before sending an authentication answer message to this user. This database is a set of files accessible to the server SAC, with each file being named as the user's ID and an extension "upd". Each update is encoded in a line of the update file as an operation code and a variable number of parameters. The server SAC encodes the number of updates that are going to be requested inside the authentication answer message. Following the authentication answer message the server SAC sends all the necessary updates as a stream of update codes and parameters, encrypted with the session key.

The SAC client implements the updating operations in order to be efficient and to accommodate many policies. These operations, and the operation code used for each are shown in Table 9, Table 10, Table 11, and Table 12. All parameters mentioned in the tables are currently one byte long unless explicitly specified otherwise.

Op. Code	**Description**	**Function**
FF	Delete ACLs	This operation has 1 parameter. The parameter specifies which ACL is to have all entries removed. The possible values, which refer to an ACL, are: 00: delete simple ACL. 01: delete hierarchical ACL. 02: delete complex ACL. A special parameter (FF) invalidates the service provider and has to be reset at a place specified by the service provider. However, a service provider can only invalidate the services it provides, not the whole client SAC. Special service providers can invalidate a client SAC.
FE	Add Service Provider	This operation can only be performed by special services providers. The operation has 2 parameters. The first parameter is a two byte parameter and specifies the service provider ID. Nothing will happen if this ID is already in the client SAC. The second parameter is a 128 byte parameter and represents the public key of the service provider. All operations after this operation is successful will be performed on the ACLs of this newly created service provider. They will be disregarded, unless it is another add service provider operation, if the operation is not successful.

Table 9: ACL update operation that is applied to all ACL.

Op. Code	Description	Function
00	Add an entry	This operation has 1 parameter, which is the entry to be added. The client SAC looks for the existence of the specified entry and adds this entry to the simple ACL if it is not present. Nothing happens if it is already present.
01	Delete an entry	This operation has 1 parameter, which is the entry to be deleted. The client SAC looks for the specified entry and deletes this entry from the simple ACL if it is present. Nothing happens if it is not present.
02	Delete a range of entries	This operation has 2 parameters. The first parameter represents the lower boundary of entries, and the second parameter represents the upper boundary. The client SAC deletes all entries from the simple ACL that lie between the lower and upper boundary, including the boundaries.
03	Add a range of entries	This operation has 2 parameters. The first parameter represents the lower boundary of entries, and the second parameter represents the upper boundary. The client SAC adds entries that lie between the boundaries, including the boundaries, if they are not present yet.

Table 10: ACL update operations that is applied to simple ACL

Op. Code	Description	Function
04	Add a range	This operation has two parameters: a lower and an upper boundary. The client SAC adds this range to the hierarchical ACL as an entry. In addition to adding the range, the client SAC merges overlapping ranges. For example, if the hierarchical ACL already has the range 1 to 10 and a request to add the range 5 to 15 is made, then the result will be one entry with the range 1 to 15.
05	Delete a range	This operation has two parameters: a lower and an upper boundary. The client SAC deletes this range in the hierarchical ACL. For example, if the hierarchical ACL already has the range 1 to 20 and a request to delete the range 5 to 15 is made, then the result will be one entry with the range 1 to 4 and another entry with the range 16 to 20.

Table 11: ACL update operations that is applied to hierarchical ACL

Op. Code	Description	Function
06	Add a label	This operation has 4 parameters. The first parameter is the label to be added. The client SAC looks for the specified label in the security label field of every complex ACL entry. The client SAC adds this entry (the 4 parameters) to the complex ACL if no other entry with the same label is found. Nothing happens if one is found.
07	Delete a label	This operation has 1 parameter, which is the label to be deleted. The client SAC looks for the specified label in the security label field of every complex ACL entry. The client SAC deletes any entry from the complex ACL that has the specified security label. Nothing happens if no entry is found
08	Delete a range of labels	This operation has 2 parameters. The first parameter represents the lower boundary of labels, while the second parameter represents the upper boundary. The client SAC deletes all entries from the complex ACL that have security label fields between the lower and upper boundary, including the boundary values.

Table 12: ACL update operations that is applied to complex ACL

Op. Code	Description	Function
09	Entry expiration by references	This operation has 3 parameters. The first parameter represents the lower boundary of labels, while the second parameter represents the upper boundary. The third parameter represents a maximum reference. The client SAC deletes all entries from the complex ACL that have security label fields between the lower and upper boundary, including the boundaries, and that have a reference field with a value lower than the maximum reference parameter.
0A	Token expiration by reference	This operation has 3 parameters. The first parameter represents the lower boundary of labels, while the second parameter represents the upper boundary. The third parameter represents a maximum reference. The client SAC zeroes the tokens remaining field of all entries from the complex ACL that have security label fields between the lower and upper boundary, including the boundaries, and that have a reference field with a value lower than the maximum reference parameter.

Table 12(cont): ACL update operations that is applied to complex ACL

Op. Code	**Description**	**Function**
0B	Token reset by reference	This operation has 3 parameters. The first parameter represents the lower boundary of labels, while the second parameter represents the upper boundary. The third parameter represents a maximum reference. The client SAC resets the tokens remaining field with the value of the reset value field, for all entries from the complex ACL that have security label fields between the lower and upper boundary, including the boundaries, and that have a reference field with a value lower than the maximum reference parameter.

Table 12(cont): ACL update operations that is applied to complex ACL

3.2.3.1.3 Request for Headers

From the client SAC perspective, the starting point of any transaction request is an unclassified metadata. This metadata is the result of a query run without the use of any SAC component. If the metadata pointer points to a container, then a header describing this container can be requested. The SAC components are required if the container is classified, as described in section 3.2.3.1. In the case of classified containers, the headers returned by the server are all encrypted and are assumed to be small, so they can be efficiently decrypted inside the security perimeter of the client SAC.

In order to request a header, the application software calls the API function:

int

*RequestHeader(DataPtr req, DataPtr *&ans)*

This function uses the struct DataPtr, which is defined as:

```
typedef struct _DataPtr {
        unsigned char secType;
        unsigned char secLabel;
        unsigned char value;            //only meaningful for complex types
        boolean ReportBack;
        CString         pointer;
        CString         description;
}DataPtr;
```

This structure uses three security attributes to specify the header. The first attribute is the security type of the header and is the one that was specified in Table 5, which defines what access control list must be checked. The second parameter is the security label of the header, which the user must have clearance to. The third parameter is only meaningful for requests to the complex access control list and is the value defined as the comparison value in Table 5.

The *ReportBack* attribute is used to force the client SAC to communicate with the server SAC even if the user is not allowed to access the header. The client SAC sends the server SAC the security attributes and the pointer if the *ReportBack* attribute is true and the user is not allowed to access the requested data. In addition, the client SAC sends the entry with the same security label as the requested header, if the request uses a complex ACL and there is an entry with the same security label as the requested header. Thus, requests that use the complex ACL and have the *ReportBack* attribute set are reported both when the request is granted and when it is denied. A server SAC sets this attribute if it wants to give a message to a user explaining the reason the request was denied, if it wants to report back a value from the complex ACL entry (e.g., the number of tokens remaining), or if it wants to take an action when a request cannot be fulfilled (e.g., logging the request).

When requesting a header, the security attribute field of DataPtr is filled in by untrusted software (including the CCP) that resides outside the client SAC. For this reason, the SAC approach does not rely solely on this information to grant or deny access to data. To be more specific, the SAC approach guarantees that even if a user provides a wrong or malicious security label to the client SAC, the server will not be misled into sending data that the user is not allowed to access. To assure this the server SAC performs a backup check comparing the claimed security attributes to the security attributes that are linked to the header stored in the server database.

When the API function returns, the application software will receive an array of DataPtr in the ans argument. The number of elements in this array is specified by the integer returned by the function. The elements returned to the user are only those elements of the container to which this user has access, as described earlier.

3.2.3.1.4 Request for Data

A request for data can be made if the user has a DataPtr structure that points to data. The request usually happens after a header is received and the available data is shown to the user. The user chooses the data of interest and the application software will make a request on behalf of the user. It is the function of the application software to format the data to be shown to the user and to respond to his/her selections.

When the application software wants data, it calls the API function

boolean

RequestData(DataPtr req, char filename[8])

The requested data is specified in the argument req, in the same way that requests for headers are specified. The function returns true if the data has been received by the client computer, and the filename argument contains a pointer to the file that has the requested data in clear text. Currently it is the responsibility of the application software to sanitize the file after using it.

3.2.3.2 The Server SAC

The individual Server SAC modules and Server Communication Package (SCP) are sensitive software that must be protected. The protection of these modules is centralized in the server that uses the SAC approach for security. The administrator of the server must ensure the protection of the server, which is usually easier than ensuring the protection of the distributed clients. In addition, it is expected that the system administrator of the server is more security aware than the individual users. For these reasons, the tamper resistant area where these modules run may be implemented using existing hardware on the server, such as implementing it in kernel space.

The Server SAC has a database that contains client SAC public keys to identify and interact with each user. Every time a user initiates an authentication with the server the SCP spawns a new server SAC module with properly initiated parameters, including the user ID, the user's public key, and the server private key. The SCP also checks if the specific user needs to have any of his/her accesses updated, such as for adding a new service provider. To accomplish this, the SCP uses a database of access control list (ACL) modifications. The modifications, if there are any, are requested by the server SAC through the secure channel, immediately after the channel is established.

The Safe Area of Computation in the server is composed of the dynamic set of server SAC modules, the databases that store keys and ACL modifications, and the Server Communication Package. This area, which must be well protected, is shown in Figure 4.

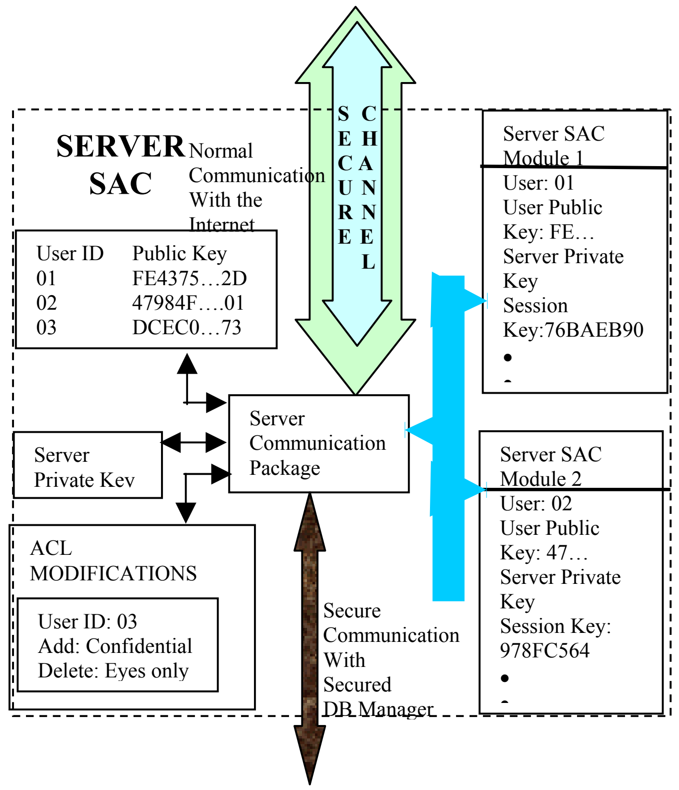


Figure 4: Server SAC Components

The following subsections explain the communication of the server with the Internet and with the secured database and discuss the security of the server.

3.2.3.2.1 Server SAC Communication with the Internet

Communication with the Internet is handled by the Server Communication Package (SCP). Every time a client SAC issues a request to the server through the Client Communication Package, the SCP sends the request to a specific server SAC module. After this server SAC module processes the request it sends an answer to the SCP, which sends it back through the secure connection used by the client to issue the request.

Every communication to and from the Internet is encrypted when the secure channel is used. Every pair of server SAC module/client SAC has specific cryptographic keys to encrypt and decrypt their transactions. This means that although only one secure channel is shown in Figure 4, there will be one distinct secure channel for each client. Thus, any adversary in the Internet cannot maliciously eavesdrop on a transaction that is conducted through the secure channel. In addition, since the keys are unique, malicious impersonation is not possible.

3.2.3.2.2 Server SAC Communication with the Secured Database Manager

The Server Communication Package also manages communication with the secured database manager, which handles all accesses to classified data. Since the database manager sends sensitive unencrypted data in answer to requests, the line of communication between the SCP and the secured database manager must be secure.

Each request that is sent to the secured database manager contains the data pointer sent from the client SAC and decrypted by the server SAC. Neither the server SAC nor the SCP interprets the pointer. As a result, the SAC approach can accommodate any database format without the need for any change, assuming the pointers are consistent with the secured database manager format. To accomplish this the database manager must export a function call, so that the SCP can send the pointer and receive back the security label of this data along with the data requested.

3.2.3.2.3 Security of the Server

A proper implementation of the server is an implementation where the modules shown in Figure 4 are kept in a safe area. In addition, this implementation trusts the data received from the secure communication with the secured database. This is a blind trust and nothing is done to guarantee the integrity of the data.

There is a possibility for denial of service attacks due to the fact that in every transaction with a client the server first needs to identify which user is requesting the transaction. This verification is accomplished by appending the user ID encrypted with the server's public key to the transaction. A malicious attacker cannot benefit from this since he/she will not be able to complete an authentication nor be able to get the session key used by a legitimate user interacting with the server. The attacker can, however, flood the server with transactions from possibly existing user IDs. This will require the server to process all these requests and may force legitimate users to restart sessions because of time-outs.

3.2.4 Example Policies

The SAC approach is very flexible and can enforce many reasonable policies. These policies may be used to restrict access to data based on an absolute or relative security label, or in parameters like number of times that data from a certain group can be accessed, dates and/or times they can be accessed. All of the different possibilities will not be discussed in this dissertation, since they are infinite, but in this section some basic policies that may be combined to express other, possibly more complex, policies will be discussed. The next subsections describe the basic policies.

Some of the policy descriptions refer to changing or comparing values from outside the client SAC to values in the client SAC access control list. This does not represent a security threat because the value outside the client SAC that is compared or used to change a value either comes from the server SAC through a secure channel or it will be validated by the server SAC before being sent.

3.2.4.1 Simple Access Policy

The simple access policy is responsible for enforcing that a user can only access data that he/she is cleared for. The access clearance may be based on an absolute clearance, where the user either has the clearance or not, or on a relative clearance, where clearances are hierarchical.

The absolute clearance is easily expressed using the simple access control list. The server site assigns a value for each possible clearance and a client SAC either has the clearance or not. The client SAC checks the simple ACL for the required clearance when requesting data or containers. For example, a digital library could assign the value 12 for data that can be accessed only by administrative personal. The access to this data would use the “check simple ACL” operation from Table 7 in conjunction with the value 12. Thus the attributes of this data would be 00 12.

The relative clearance is easily expressed using the hierarchical access control list. The server site divides the possible clearances in groups where an ordering has the required meaning. In addition, the server assigns a hierarchical value for each possible clearance. A user has entries in the hierarchical ACL that express his/her clearance for each group. The client SAC checks the hierarchical ACL for the required clearance when requesting data or container. For example, assume that the clearances are classified, secret and top secret, and the groups are navy and army. The server site can assign to <navy, classified> the value 1, <navy, secret> the value 2, <navy, top secret> the value 3, <army, classified> the value 4, <army, secret> the value 5, and <army, top secret> the value 6. A user cleared at the level <navy, secret> and <army, top secret> would have two hierarchical ACL entries: [1, 2] and [4, 6] (where the values are expressed using the two fields that would be on the hierarchical ACL fields). A user cleared only at the level <navy, classified> would have only one entry: [1, 1]. The operation "check hierarchical ACL", from Table 7, would be invoked to access the classified data and container. Thus, the attribute to access <army, secret> data would be 01 06, where 01 is the operation and 06 is the label.

3.2.4.2 Simple Counter Policy

The simple counter policy is responsible for enforcing the restrictions that a user may access data that he/she is cleared for only a certain number of times. This policy may also require the server to be able to add more usage privileges to a user.

This policy can be implemented using the complex access control list in the client SAC. The security label field of the complex ACL represents whether the user is cleared to access the data. The tokens remaining field represents the number of times the user is still allowed to access the data. One token is subtracted from the tokens remaining field every time data is accessed. This is accomplished through the "subtract from tokens remaining" operation defined in Table 8. Thus, in order to check if the tokens remaining field is greater than zero and if so subtract one token, the subtraction operation would be combined with the "check if tokens remaining field is positive", defined in Table 8, resulting in an operation code of 01010001_b. The value to be subtracted would be one in this case. The server site may also add tokens, effectively granting more usage privilege to a user, using the operation "add value to the tokens remaining field" of the complex ACL defined in Table 8. For example, the site would specify the operation 10000010_b in order to add two tokens.

3.2.4.3 Pay Per Use Policy

The pay per use policy enforces that a user has to pay for accessing data. For the SAC approach, the payment is always in tokens. It is the responsibility of the service provider to give a meaning to a token.

This policy can be implemented using the complex ACL. The implementation is similar to the one described for the simple counter policy. The difference is that the server requests the cost of the data, expressed in number of tokens, to be subtracted from the tokens remaining field instead of one token. For example, the server can specify a cost of three tokens with the “check and subtract tokens” operation (00110000_b), as defined in Table 8, with a comparison value of 3. This operation subtracts three tokens from the tokens remaining field and accesses the data only if the tokens remaining field is greater than three. The server may add tokens to the complex ACL entry of a user using the “add value to the tokens remaining field” operation defined in Table 8. For example, the site would specify the operation 10000011_b in order to add three tokens.

3.2.4.4 Temporal Counter Policy

The temporal counter policy is responsible for enforcing the restrictions that a user may access data that he/she is cleared for only a certain number of times in a certain period of time. For example, a user may be restricted to access a data item only 10 times per day.

This policy can be implemented using the complex ACL of the client SAC. The tokens remaining field of a complex ACL entry represents the number of times a user may still access data. The reference field represents the time of last access. The server uses its date value to extract a reference time. The server uses two operations, as defined in Table 8, to enforce the policy. The server resets the tokens remaining field if the reference field in the client SAC is not the same as the reference time generated by the server using the “check and reset” operation defined in Table 8 as 00011000_b. Then, for each access the server needs to check if the tokens remaining field is greater than zero. If it is, one token is subtracted from the tokens remaining field and the access will be granted. These operations of checking and subtracting can be combined in one operation with code 01010001_b.

3.2.4.5 Simple Temporal Policy

The simple temporal policy is responsible for enforcing that a user may access data only for a certain period of time. This can be used, for example, for subscription services where a user has to pay monthly or yearly to access the service. This can also be used to allow a user to try a service for a certain time before having to pay for it.

This policy can be implemented using the complex ACL. The complex ACL will have the reference field of a specific complex ACL entry as the day, month, year, or any other temporal parameter. The security attributes of the data being accessed specifies, using one of the "check if comparison values is the (same as or greater than or lower than) reference field" operations described in Table 8, that the client SAC must check the complex ACL entry specific to the service, comparing the reference field with a value, which can represent the current day, month, year, or temporal parameter. The access is only granted if the values satisfy the policy specification. For example, in order to allow access to data only in a specific month, the server uses the attribute (00001000_b) (00000111_b) to specify the operation (08) and that the current month is July (7). In order to allow access to data only before a specific month, the server uses the operation (00100000_b). As before, the server must specify the current month as a comparison value. In order to allow access to data only after a specific month, the server uses the operation (00101000_b).

3.2.4.6 Temporary Limited Use Policy

The temporary limited use policy enforces that a user may access a data item only a certain number of times. The difference between this policy and the simple counter policy is that this policy is used for temporary accesses and assumes that the server adds the permission to access data only on the first time a user requests that data. The simple counter policy assumes that the server adds the permission independently if the user will ever access the data and service to which that permission refers. The temporary limited use policy may be used in a University, for example, to enforce that a student can buy only a certain number of tickets for specific events.

The SAC approach uses the complex ACL in the client SAC to enforce this policy. The policy could be satisfied by adding an entry in the complex ACL with the respective use limit in the tokens remaining field the first time a user requests the service. In this case, data could be requested using, for example, the operation 01010001_b, which allows access only if the tokens remaining field is positive and if so, subtracts one token. This would enforce the policy that the data is accessed only a certain number of times. However, the policy is used for temporary data and this implementation does not use the complex ACL efficiently since all past permissions would be stored in the complex ACL until the server SAC explicitly requests their exclusion. In addition, the service provider would have to keep generating unique access id's, e.g. event id's in the ticket example, until the explicit deletion of id's in all possible client SACs. A better implementation uses the reference field to store the expiration date of an access clearance. This implementation uses the "entry expiration by references" operation described in Table 12, when a user connects to the server. The server uses its current date as a parameter to expire the entries. After that the server SAC requests a specific entry to be added to the client SAC complex ACL using the "add a label" operation described in Table 12. The reference field of this clearance represents its expiration data and the tokens remaining field represent the number of times the user can access the data. The "add a label" operation does not have any effect if that label is already present. This prevents the reset of non-expired access labels. The user can then request access to the data, which has the security attributes specifying the number of tokens that must be deducted.

As an example, assume that the current date can be represented by the number 21, the data attribute is 14, and the data can be accessed only one time. The server would first expire all entries on the complex ACL with label 14 and expiration date (stored in the reference field) lower than 21 using the operation 09 14 14 21, as defined in Table 12. After that the server would add an entry allowing a one time access, and with an expiration date (e.g. expiration date that can be represented by the number 61), using the operation 06 14 61 01 00, also defined in

Table 12. The data would then specify the “check and subtract tokens” operation (00110000_b), as defined in Table 8, with a comparison value of 1.

3.2.4.7 Good Before Policy

The good before policy enforces that a user is cleared to access some data a specific number of times, but only before a deadline. This can be used to allow a user to try a service up to a specific date, but not to try to access a huge amount of data, which might compromise the service. For example, a library may allow a user to try its service for a month but does not want the user to download all the data that the library has during this initial month.

This policy can be implemented using the complex ACL. The reference field of a complex ACL entry will represent the expiration time and the tokens remaining field will represent the number of times the user is allowed to access data. The server SAC requests the “token expiration by reference” operation described in Table 12 for all users that connect to this server SAC. It uses the server’s current date as the value to compare to the reference field from the client SAC complex ACL. The data has security parameters specifying that a number of tokens must be deducted. The data will be accessed up to the number of times specified on the tokens remaining field of a complex ACL entry. The tokens remaining field will be zeroed after the expiration date stored on the reference field.

As an example, assume that the current date can be represented by the number 21, the data attribute is 14. The server would first zero all tokens on the complex ACL with label 14 and expiration date (stored in the reference field) lower than 21 using the operation 0A 14 14 21, as defined in Table 12. The data would specify the "check and subtract tokens" operation (00110000_b), as defined in Table 8, with a comparison value that represents the number of tokens that must be deducted.

3.3 Use of SAC for Secure Internet Transactions

The goal of the SAC approach is to protect specific servers. Thus, it does not use the general approach of certificates, which was discussed in section 2.1.6. Figure 5 shows a high level view of the SAC approach to protecting Internet transactions. Each client has a client SAC, which holds its access control list and other security relevant information, and each server has a server SAC. There are two different types of server sites: central authority sites and service provider sites. Initially there is a single central authority site with which the client SAC can communicate. A central authority site is the only site that can add access to other sites to a client SAC. However, once a service provider site has been added to a client site, Internet transactions are performed between the service provider site and the user without intervention from a central authority. A service provider can also modify the portion of the client SAC access control list that refers to that particular service provider, and can remove all of the user's access to the site. It cannot, however, modify the user's access list for any other site.

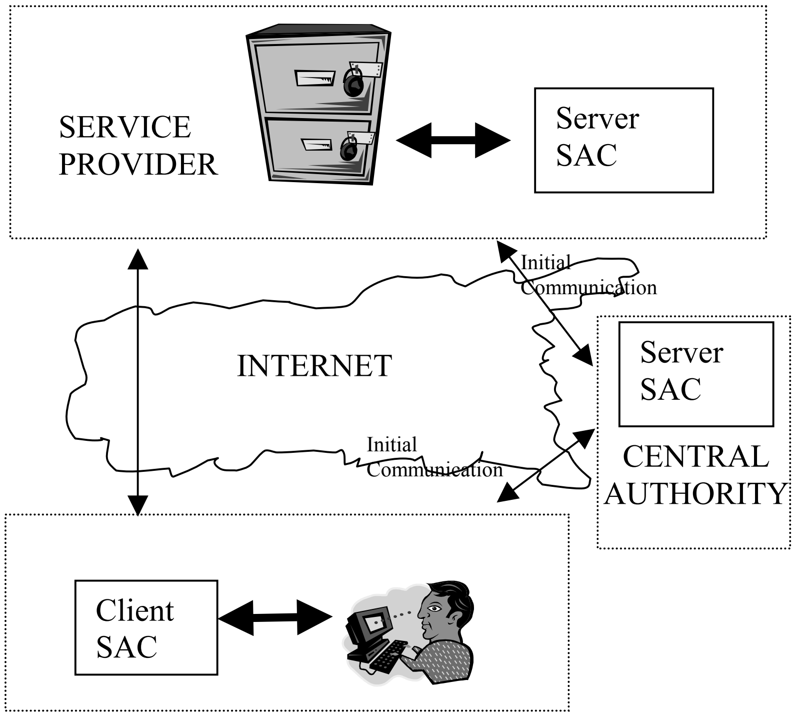


Figure 5: SAC approach for Internet Transactions

The use of the SAC approach for Internet transactions has many desirable features. The protocol used by the SAC approach provides strong authentication and prevents spoofing attacks against a user in a user-friendly manner. As discussed in section 3.2.3.1.1, a private identification number (PIN) is used to authenticate a user to the SAC. This number can be any easy to remember number. To protect the data in the SAC, in case the SAC is lost or stolen, the client SAC has a lock out procedure that locks itself after a specified number of wrong attempts. Because of this, any dictionary attack against the client SAC has a high probability of being unsuccessful. In addition, because the SAC is tamper resistant, the data cannot be copied for repeating the attack. After entering the PIN, a user does not need to perform any additional authentication. The PIN is the only secret information that a user has, and it is only useful in the presence of the correct client SAC. In addition, any spoofing attack will not be successful since the protocol does not reveal any secret in the clear and the user can at most be tricked into revealing his/her PIN. A critical element of the SAC model is that the SAC exchanges a secret key in the authentication step. Thus, an attacker who tries to perform a man-in-the-middle attack, during authentication, will be unsuccessful, because either the user exchanges the secret key with the attacker, which does not authenticate the user to the server, or with the server, which does not reveal the secret key to the attacker. The use of the server public key by the user and the user's public key by the server for encryption guarantees that only the end parties can decrypt and retrieve the secret key.

Another desirable feature of the SAC approach is that the use of access lists and security labels enables the personalization of a user's services without overloading the server. Personalized data that is security related is safely stored in the client SAC. This data is used by the SAC to determine if the user can access the data requested, and only accesses that are pre-authorized are transmitted to the server. Thus, any access that would not be authorized will not even be sent to the server, which reduces the load on the server. Although only pre-authorized requests are transmitted to the server, there is also a back up check at the server to prevent any malicious requests.

A user can only be impersonated if his/her PIN is compromised and his/her SAC is lost or stolen. If this is the case, the user can easily notice the loss and the particular SAC can be disabled the first time that it is used. Furthermore, in the event that the SAC is not disabled, it can only be used to the extent of the user's clearance. For example, in the case where a bank is the service provider, a user can choose not to have access that allows money transfers over the Internet, which the SAC will enforce. Then if this client SAC is compromised it will still only have limited access, such as the ability to check the user's balance.

The client SAC must be a device that is either integrated into a computing base or that is easy to carry and can interact with a computing base. As discussed in chapter 2, smart cards are plastic cards that resemble magnetic strip cards, which most users are already familiar with and that most users associate with secure operations. Smart cards are an appropriate choice for the client SAC requirements. Current production versions of smart cards can have a cryptographic co-processor, 16 Kbytes of EEPROM, 2 Kbytes of RAM and 32 Kbytes of ROM. This is sufficient to implement a SAC client, although it still places some restrictions on the number of service providers that can be accessed by one card. In addition, smart cards and their readers are dropping in price, their memory size is increasing, and computer operating systems already support smart card readers.

3.3.1 Implementation for Protecting Internet Transactions

The implementation for protecting Internet transactions uses the configuration shown in Figure 6. The data used in this implementation is generic MIME data. The intent was to explore the ability of a browser to directly handle many types of MIME data securely. The transactions may use the Netscape Navigator or the Microsoft Internet Explorer browser. The ActiveX controls technology is used to communicate with the Client Communication Package (CCP). This communication could be implemented using other technologies. In fact, in an earlier implementation this was done using Plug-In technology. The decision to change to ActiveX technology was made because the implementation primarily targets Windows platforms and the ActiveX technology provides a richer and more powerful environment than other technologies.

A client can visit a secure server without interacting with either the client or server SAC as long as he/she browses through unclassified data. However, as soon as the user requests classified data, the secure server starts an ActiveX control, which loads the CCP if it is not running yet. The client SAC is implemented using smart cards. The ActiveX control draws a text box window requesting the pin code from the user. After the user enters this information, it calls the authentication function.

If the user is authenticated, one of the frames in the main page allows the user to request a search from the secure server. The answer to this search is a set of unclassified pointers to collections.

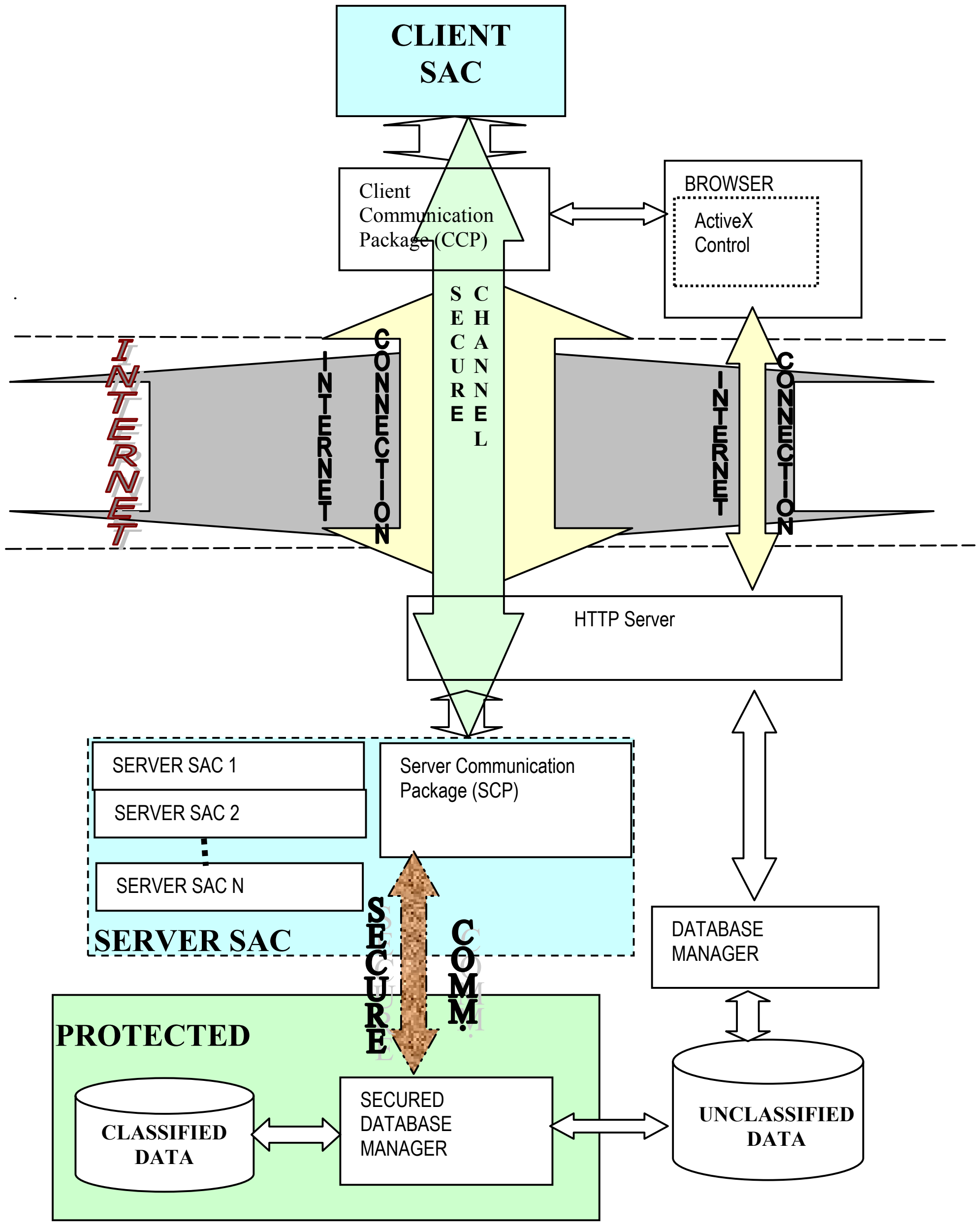


Figure 6: Safe Internet transactions using the SAC approach

The following subsections detail the components in Figure 6 and describe the steps of a simple transaction.

3.3.1.1 Client Side Components

In this section each of the components at a client site are described.

3.3.1.1.1 ActiveX Control

Several ActiveX controls were developed to provide a rich environment for requesting input, downloading data and visualizing data. The ActiveX control technology implements the active interaction between the browser and the Client Communication Package (CCP). The same design may be used with Plug-ins and Java applets with special capabilities.

By implementing this program as an ActiveX control, all the visual interfaces are provided to the user. The user interacts with the server using these visual interfaces, and the ActiveX controls interact with the CCP when classified data is requested.

3.3.1.1.2 Client Communication Package

While the ActiveX controls have a specific design developed to satisfy a particular application and server, the Client Communication Package (CCP) is designed to be generic and to support the functionality of the SAC approach. As discussed in section 3.2.3.1, the CCP does not require a safe area and never deals with sensitive plain text data that has not been first approved for release by the Client SAC.

Besides implementing all functionality that can run in the unprotected area, as discussed in section 3.2.3.1, the CCP implements the Internet communication line, which is used to exchange encrypted data between the client SAC and the server SAC and in which the virtual secure channel is established.

3.3.1.1.3 Client SAC

The Client SAC has limited memory and low processing power because it is housed in a smart card. The code that provides security, as well the cryptographic keys, are stored inside its security perimeter.

Because the client SAC has low processing power, it would take a long time to decrypt large files. Therefore, file decryption is performed by the CCP, outside of the client SAC. The server SAC encrypts each file with a unique one-time key and sends the encrypted result to the CCP using a standard Internet connection. The key is then sent through the secure channel to the client SAC. The client SAC can then hand the CCP this key to decrypt a sensitive data file. Thus, the client SAC always controls the decryption of sensitive data. This sensitive data is either used directly by the CCP, such as metadata in the clear, or is a key that can be used by the CCP to decrypt other sensitive data. The sensitive data or key will only leave the security perimeter if the access control list (ACL) inside the client SAC certifies that this user is cleared to access the data.

3.3.1.2 Server Side Components

In this section the SAC components at the server are described.

3.3.1.2.1 HTTP Server

The SAC approach does not make any assumption about the http server. That is, the generic nature of the approach does not require any particular http server. The http server transmits and receives encrypted sensitive data that is sent from the CCP to the SCP or vice-versa without compromising the security of the data. This is accomplished by adding a communication channel between the http server and the SCP using a technology that extends the http server, e.g. CGI and servlets. The current implementation uses a Microsoft Internet Server Application Programming Interface (ISAPI) extension to implement a channel between a Microsoft Internet Information Server (MIIS) and the SCP.

The client communication package could communicate with the server communication package directly using a specialized TCP/IP port. However, this communication could be filtered out by firewalls that do not allow communications through arbitrary TCP/IP ports. This is the main reason that the communication is performed with the aid of the http server. The firewall will allow the communication between the CCP and SCP if it allows connections to the http server for accessing the protected site.

3.3.1.2.2 Secured Database Manager

The secured database manager handles sensitive data and should be kept in a protected area satisfying only requests from trusted sources. No sensitive data will be leaked if the SCP runs in a safe area and its communication with the secured database manager is secure.

3.3.1.2.3 Server Communication Package

As discussed in section 3.2.3.2, the Server Communication Package (SCP), unlike the Client Communication Package, needs to run in a protected area in the server. One reason for this is that it interacts directly with the secured database manager to request sensitive data.

3.3.1.2.4 Server SAC Module

One Server SAC module is spawned for each user that is successfully authenticated to the server, as discussed in section 3.2.3.2. The Server SAC module has the necessary keys to interact with a particular client SAC.

The server SAC modules are kept inside the server SAC, while interacting with the corresponding client as shown in Figure 4. When the client SAC ends a session with the server, the corresponding server SAC module is deleted. However, a session between a client and a server SAC module often has transactions that cannot be anticipated. Therefore, it may be desirable to keep that specific server SAC module around. In the current implementation time-outs are used to solve this problem and to determine if a session has finished. If the time-outs are too short, this can cause some users to restart a session after a time-out.

3.3.1.2.5 Database Manager

In order to lower the workload when dealing with unclassified data, the server may run a second database manager that does not require a protected area. The system is responsible for assuring that this database manager can only access non-sensitive data. Using this approach the http server can request non-sensitive data directly from the unclassified database manager. This unclassified data can be used later to fetch classified data through the Safe Areas of Computation.

3.3.1.3 Example Session

The following sections describe in detail each step of a session. The necessary tests and functions that are executed in order to provide security are also discussed. A user does not interact with the client or server SAC to request unclassified data. In fact, a user will never interact with the SACs, and the client SAC may not even be present, if the user only requests unclassified data. The example session described in this section shows what happens when the user requests classified data. Thus, all data that this section refers to is classified, unless explicitly stated otherwise. In addition, a web page may have a mix of classified and unclassified data. In this case the classified data would be shown inside an ActiveX control, and would go through the steps described in this section, while the unclassified data may be shown in any way the server chooses.

3.3.1.3.1 Authentication

The ActiveX controls interact with the user, with the browser, and with the Client Communication Package providing the particular functionality needed by a specific application. The ActiveX controls and Client Communication Package are implemented as programs that run in the user's computer and that must be installed before any secure transaction can be accomplished. The Client SAC, which is a very sensitive set of modules, is implemented in a smart card. The user must therefore have a smart card reader connected to his/her computer.

When accessing the protected server, the user's computer displays a page that resides in this server. This page has an embedded ActiveX control, which starts the Client Communication Package program. Next, a text box window requests the user to enter a pin code in order to authenticate the user to the client SAC.

After the user enters a pin code and clicks the OK button the process of authentication starts. If the user enters the correct pin code, he/she will be authenticated to the client SAC. The client SAC will then authenticate itself to the server SAC, using an Internet connection opened by the CCP. The result of the successful authentication process is a session key that provides a secure channel over the standard Internet connection.

3.3.1.3.2 Request for Container

After the authentication and secure channel establishment takes place, the user or the ActiveX control can request classified containers by making queries. Either the user selects a container from a list shown by the ActiveX control or the

ActiveX control has a property that identifies a container to be requested. The user clicks a “Get Container” button to request the container or the ActiveX control does the request based on its property. In either case, the ActiveX control sends this request to the Client Communication Package (CCP), which forwards the request to the client SAC, specifying the pointer to the container header and its security label. If the user has the clearance necessary to request the container header, as specified by the security label of the container and the access control list, then the client SAC uses the secure channel to request the container header. Since the data used in the request is sent through the secure channel, neither the CCP nor the Internet can tamper with the request.

The request includes the security label of the container against which the user’s clearance was checked and a header with the user ID. The security label will be used to prevent software outside the client SAC from maliciously faking a request. The user ID is used in order to identify which spawned server SAC module has established a secure channel with the particular user.

3.3.1.3.3 Server Processing of Container Requests

The appropriate server SAC module, from the possibly many spawned by the Server Communication Package (SCP), receives a request for a container from the client SAC that is associated with it. The server SAC uses the SCP to request the container specified by the pointer sent by the client SAC. The SCP requests the container from the Secured Database Manager using this pointer.

The Secured Database Manager fetches the requested data (a container header) and returns it to the SCP, together with the security label associated with this data. The SCP returns the container with this security label to the appropriate server SAC module. The result is tested against the security label sent by the client SAC when requesting the specific container. If both security labels are not the same the operation fails and an error is sent to the client SAC. This is the second time that the security label of the requested data is checked and will catch malicious requests, where the security label has been changed before it is sent to the client SAC. In the case where the security labels agree, the server SAC will send the container and the security label received from the SCP back to the client SAC through the secure channel.

3.3.1.3.4 Results of Requesting a Container

The client SAC receives the container header through the secure channel and processes it. Before processing any of the data inside the header the client SAC uses the access control list and the security label of the header received from the server SAC, to check if the user has access to this header. This is the third time that the clearance is checked and should never fail. A failure at this point is unexpected and should be logged for further studies.

The client SAC checks the user clearance for each data or container embedded in the container, as represented in the container header. It sends the client communication package the information for only the data and containers to which the user has clearance. The CCP will send this information to the ActiveX control, which shows it to the user for browsing and selection.

3.3.1.3.5 Request for Data

After browsing the information about the metadata inside the container, the user can select one of interest by clicking the “Get Data” button. The client SAC makes the request in the same way that it requests container headers.

3.3.1.3.6 Server Processing of Data Requests

The process of directing a request for data to the correct server SAC module is the same as the process of directing a request for a container. The correct server SAC module again uses the Server Communication Package (SCP) to request the data from the secured database manager. When the server SAC receives the potentially large data back, it generates a one-time key to encrypt this data. The server SAC will only continue operations, as in the request for a container header, if the security label of the data reported by the secured database manager agrees with the security label reported by the client SAC when requesting the data.

The server SAC encrypts the data using triple DES and the generated one-time key, and sends it to the SCP, which sends it to the Client Communication Package (CCP) via the standard Internet connection between them (outside the secure channel). Additionally, the server SAC uses the secure channel to send the key used for the encryption and the security label of the data to the client SAC. The client SAC will release the key to the CCP only if the user is cleared to access this data. This is the third check of clearance, analogous to the procedure used when requesting container headers.

3.3.1.3.7 Results of Requesting Data

The Client Communication Package receives the encrypted data from the Server Communication Package and the key for the decryption of this data from the client SAC. It uses this key to decrypt the data and to write it to a local file, which is returned to the ActiveX control. The ActiveX control shows the data to the user as appropriate to the particular application.

The ActiveX control must interpret the data in a way appropriate for the application. The SAC approach only delivers the data to the ActiveX control without interpreting it. A specific application can use the description field to specify possible formats. The description field may include the MIME type of the data, which is easily interpreted by the browser.

3.3.1.3.8 Access Control List Updates

The steps above describe a simple session. Additionally, the current implementation supports dynamic modification of the client SAC access control list (ACL). For this the server SAC may send a request to the client SAC to update its ACL when the client SAC authenticates itself. The request is sent through the secure channel just after it is established. This request is specified as one of the operations described in Tables 6, 7, 8, and 9. It is not possible to tamper with this data since it uses the established secure channel, which uses a different session key each time. The updates support the policy to be enforced.

3.3.2 Test Beds

As a proof of concept, two test bed systems were built that use the SAC approach for Internet transactions. These test beds use the Siemens SLE66CX160S integrated circuit and the CardOS M3.0 operating system. The functionality of the client SAC is implemented by applications loaded on the EEPROM memory.

The Siemens SLE66CX160S integrated circuit has 16 Kbytes of EEPROM (CardOS uses 116 bytes of EEPROM and an additional small amount to keep a file system that will not be considered in this analyses). The applications that implement the client SAC functionality use 4,925 bytes. From the remaining 11,459 bytes, 10 bytes are used for the pin code and user's id (assuming a four byte pin code and four byte user id), and 339 bytes are used for the client SAC private key. Thus, there are 11,110 bytes that can be used to support service providers. Each service provider requires 141 bytes to store its public key plus the storage necessary for the ACLs. Therefore, the smart card used for the test bed can support 76 service providers that have security policies that require a maximum of four different simple labels (labels that can be accommodated in a simple ACL). The smart card can support 30 service providers that implement security policies requiring all three ACLs with each having 32 entries. The smart card can, therefore, support a sufficiently large number of service providers, even if they require complex security policies.

The current test bed implementations impose some particular requirements for the client and server computers. The client platform requirements are:

a) Windows 98 or NT system;

b) A smart card reader;

c) A standard browser that supports ActiveX controls; e.g., Microsoft Internet Explorer or Netscape Navigator;

d) A set of programs that may be downloaded as an ActiveX package, which consist of the necessary controls and the Client Communication Package.

The server platform requirements are:

a) The server platform must be tamper resistant. This is a very strong assumption and as such requires very specific servers;

b) Windows NT server;

c) Microsoft Internet Information Service (MIIS);

d) A particular ISAPI extension that communicates with the server SAC;

e) The server SAC application;

f) A secure connection to the database that stores classified data.

These requirements can be easily met by a Windows platform, which is the platform used for the implementation. In addition, the server could be easily migrated to a Unix platform.

The first test bed was built for the Alexandria Digital Library where it is used to download files in a secure way and only to authorized persons. This system is very simple and demonstrates only a few of the approach's characteristics. Because of this a second system was built to demonstrate how the SAC approach can be used to protect transactions for online banking. The following subsections describe these systems.

3.3.2.1 Alexandria Digital Library

The current requirement of the Alexandria Digital Library is to be able to provide access to a particular subset of its data to only a selected group of users. ADL does not require fine granularity control; the data is either restricted or not restricted.

In the current implementation the restricted data is composed of compressed files. Each of these files is a collection of data that must be transmitted in the compressed format to the user's computer. Programs in the user computer uncompress the downloaded data.

The SAC approach can meet these simple requirements very easily. Each privileged user has a client SAC implemented in a smart card, as shown in Figure 7. Each client SAC has the same set of permissions. It has no elements on either the hierarchical or the complex access control lists (ACL). In addition, the simple ACL has only one element that gives the user access to the one and only type of protected data.

Figure 7: Smart card used in the Alexandria Digital Library

Alexandria Digital Library uses the JiGi graphical interface to allow users to perform queries. These queries are not classified, and a user does not need a client SAC to perform them or to show unclassified data. However, a query may generate pointers to classified data. This also is not considered classified in the ADL security policy. Thus, a client SAC is not required if the user does not request the download of classified data. A browser window pops up if classified data is required. This browser window displays an HTML page generated by the middleware that has two embedded ActiveX controls. The details of these ActiveX Controls and the steps of a user interaction are discussed in the next subsections.

3.3.2.1.1 Authentication control

Every HTML page that uses the client SAC and can be bookmarked must have an embedded authentication control. The client SAC establishes a session key using the authentication process. Thus, a session key is not established without authentication through an authentication control, and without authentication the client and server SAC cannot communicate.

The authentication control has 3 properties, 2 events and 1 method. The properties are used to provide the control and the CCP static information (within that page). The events and method are used to enable communication among controls.

The authentication control properties are *First Pointer*, *Service Provider*, and *Web Server*. Their details are as follows:

a) *First Pointer*. This property is optional. It is used if an authentication control wants to use its window to show pointers inside a container after authentication. The property was designed to support a convenient flow for users that access systems exclusively through browsers. It can provide a login page that shows options after a correct authentication. The ADL test bed never uses this property since JiGi does all the initial steps. When used, this property is a string that starts with six digits and is followed by a pointer to a container. The six digits represent the security type, security label and comparison value of the container.

b) *Service Provider*. This is a required property and represents the service provider ID. The client SAC may support several service providers. It is this property, which is represented as a short integer that defines the service provider ID to which the user will be authenticated.

c) *Web Server*. This is also a required property. It represents the URL of the web server that intermediates the communication with the server SAC. Although one might think this could be used for man-in-the-middle attacks, this is not the case. The SAC approach is protected against man-in-the-middle attacks without making any assumption as to how the web servers are implemented, as discussed in section 3.2.3.1.1.

The control events are *ChangePage* and *Authenticated*. They work as follows:

a) *ChangePage*. This event is designed to enable the authentication control to substitute for the page being shown to the user. It takes as a parameter the URL of the page that the control wants to show. This page will replace the page currently being shown to the user.

b) *Authenticated*. This event is used to signal that the client SAC has authenticated the user and itself to the server to any other control embedded in the same HTML page. Thus, it is used to synchronize requests for data and containers after authentication. The authentication control always broadcasts an *Authenticated* event after a successful authentication.

The control method is called *IsAuthenticated(void)*. This method is called by other controls to verify that the user has been authenticated. The authentication control responds to an invocation of this method by broadcasting an *Authenticated* event to all controls embedded in the same page, if the user has been authenticated.

This enables a control to query the authentication control when it believes it missed the first *Authenticated* broadcast. This may, for example, occur due to initialization of the control.

The authentication control has a transparent background and uses a state machine to show messages to the user. The first task of the authentication control is to start the Client Communication Package (CCP) in the user's machine, if it has not yet been started. If the CCP has already been started, the control queries the CCP to check if the user has already been authenticated. When the user has not been authenticated, the control requests a pin code from the user, as shown in Figure 8. In addition, the control requests the user to insert a card if a card has not yet been inserted in the card reader. After a card has been inserted in the reader and the user enters the pin code and clicks the ok button, the control shows a changing message while waiting for the SAC to authenticate all parties (user, client SAC and server SAC) as shown in Figure 9. The control shows one of the boxes that says "authenticating" in Figure 9 each time, following the succession shown. This results in an animation effect, with the letters decreasing in size in a cyclic manner.

After the user has been authenticated the authentication control has different behaviors, depending on the presence of a valid *FirstPointer* property. The authentication control does not show anything, becoming transparent, if the *First Pointer* property is not present. This is what happens in the Alexandria Digital

Library test bed. An example of what happens if there is a valid *First Pointer* will be seen in the bank example.

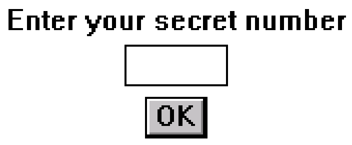


Figure 8: Interface used by the authentication control to request pin codes

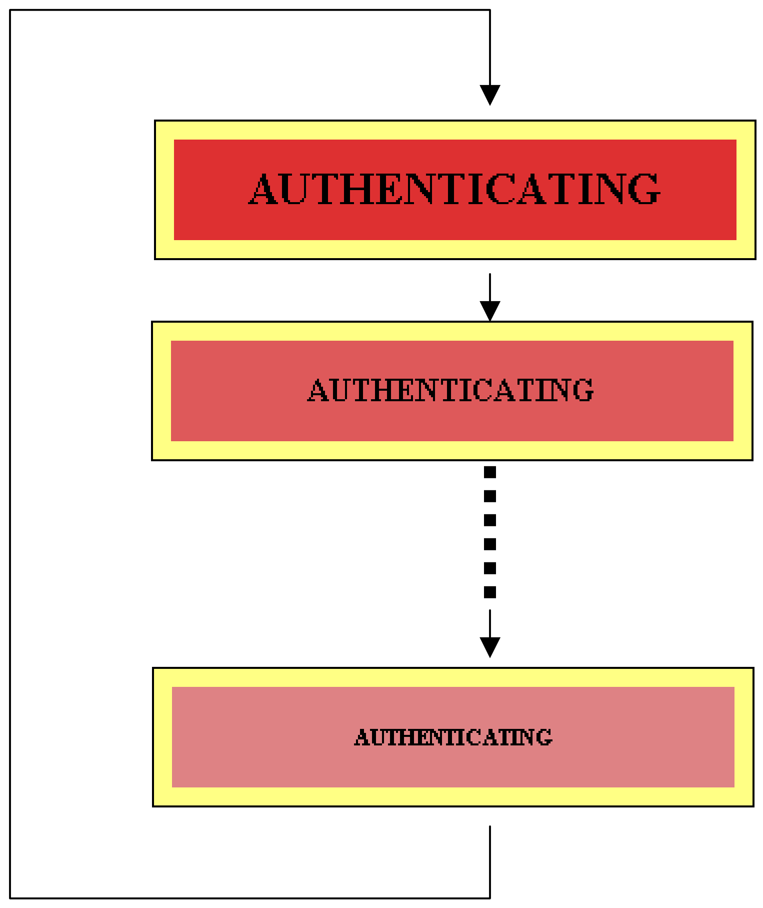


Figure 9: Interface to let a user know that authentication is being performed

3.3.2.1.2 Download Control

The download control is a transparent control that does not show any information in its client window. This control is used to determine the pointer and security attributes of the restricted data that is supposed to be downloaded. The control uses pop up windows to allow a user to define a file name under which the data will be stored on the local computer. The control shows the progress of the download of the requested data after the file name is defined. The control uses a standard Windows common file dialog box to allow the user to choose a name to save the downloaded file. In addition it uses the pop up window shown in Figure 10 to inform the user about the download.

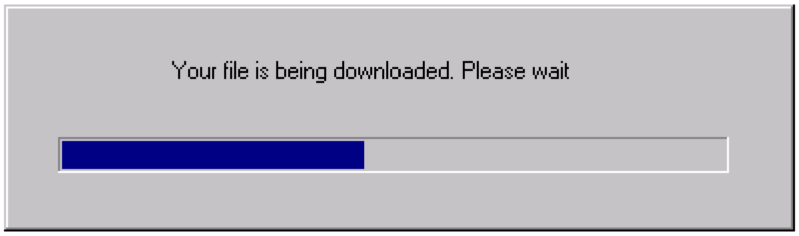

Your file is being downloaded. Please wait

Figure 10: Window that shows the download progress

The download control has one property, one event and one method. The property is called *First Pointer* and is similar to the property from the authentication control with the same name. The difference is that this property points to data on the server and must be present or there is no reason to include this control. It is this property that defines what data will be downloaded from the server.

The method of this control is called *OkAuthenticate(void)*. It is used to communicate to this control that the user has been authenticated. The request for the download of the data is performed only after this method is invoked.

The event is called *ReqAuthenticated* and is fired after the control has been initialized. This event invokes the *IsAuthenticated(void)* method of an authentication control to handle cases when its *OkAuthenticate(void)* method is called before proper initialization and the authentication acknowledgement is missed.

3.3.2.1.3 Steps of a User Transaction

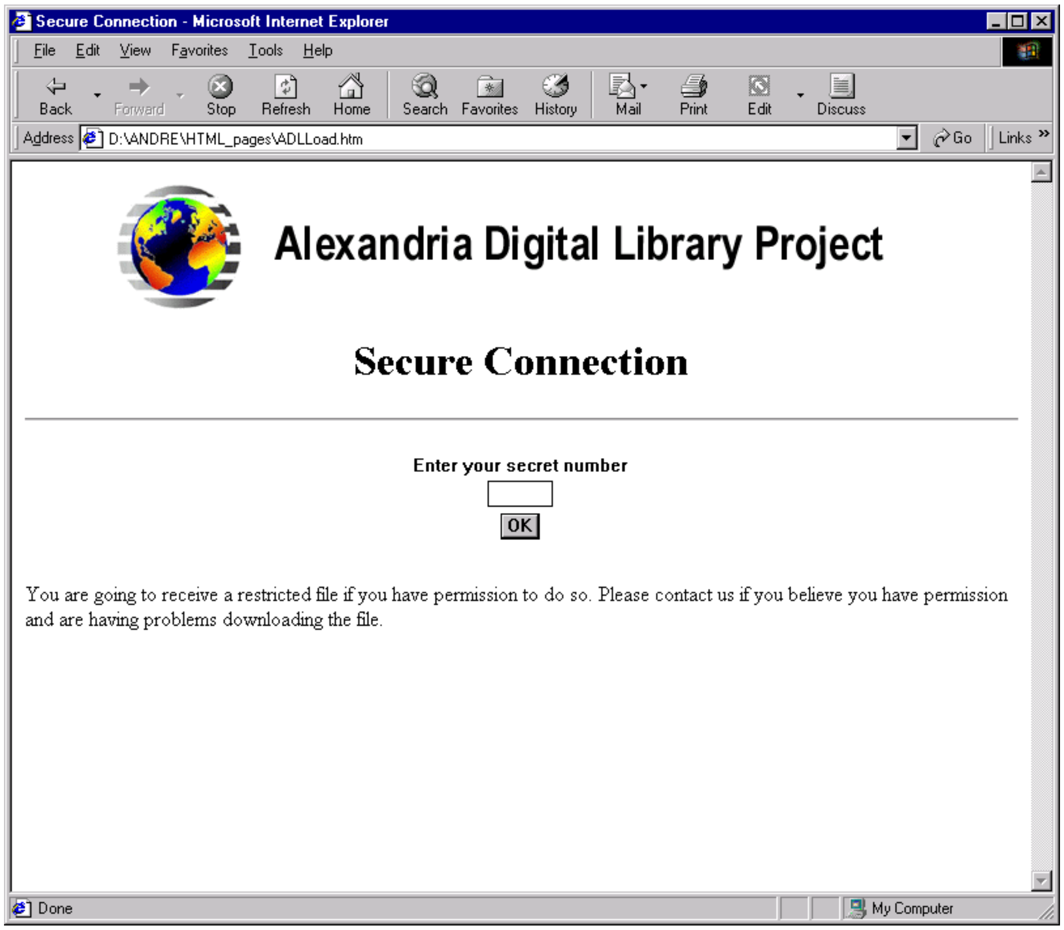
Almost all interaction of a user with the Alexandria Digital Library is done using the JiGi interface. In many cases the user will never request a protected data item and because of this does not even need a smart card reader. In some cases, however, the user will choose a protected data item from the JiGi Interface. If this is the case, a browser window will pop up and JiGi will display the HTML page shown below.

```
<html>
<head>
<title>Secure Connection</title>
</head>

<body>
<h1 align="center"><img border="0" src="alex_bar.gif" width="620"
height="100"></h1>
<h1 align="center">Secure Connection</h1>
<hr>
<p align="center">
<object classid="clsid:2D482096-20FF-11D3-A349-00105AA40330"
id="AuthenticationCtrl1" width="521" height="78">
  <param name="_Version" value="65536">
  <param name="_ExtentX" value="13785">
  <param name="_ExtentY" value="2064">
  <param name="_StockProps" value="0">
  <param name="FirstPointer" value="NAO">
  <param name="ServiceProvider" value="Service Provider ID">
  <param name="WebServer" value="Server Name">
</object>
</p>
<p align="left">You are going to receive a restricted file if you have permission to
do so. Please contact us if you believe you have permission and are having
problems downloading the file.</p>
<p>&nbsp;</p>
<p>
<object classid="clsid:00DC3B88-6D35-11D3-B1AF-00AA006FD6CB"
id="DownloadCtrl1" width="100" height="50">
  <param name="_Version" value="65536">
  <param name="_ExtentX" value="2646">
  <param name="_ExtentY" value="1323">
  <param name="_StockProps" value="0">
  <param name="FirstPointer" value="Security Attributes + Pointer">
</object>

   <SCRIPT LANGUAGE="JavaScript" FOR="AuthenticationCtrl1"
EVENT="Authenticated()">
<!--
DownloadCtrl1.OkAuthenticated()

-->
   </SCRIPT>

</p>
<p>&nbsp;</p>
<p align="center">&nbsp;</p>
</body>
</html>
```

The places marked in bold are the places that are filled by the JiGi interface. They will specify the service provider ID for the Alexandria Digital Library, the web server that is servicing the SAC and the pointer to the data along with its security attribute.

The user needs to authenticate his/herself to the client SAC the first time he/she requests protected data. The browser will show the page in Figure 11 requesting authentication. A standard Windows common file dialog box like the one shown in Figure 12 pops up as soon as the user is authenticated. This allows the user to choose the name under which the downloaded file will be saved. A progress window, like the one shown in Figure 10 pops up and keeps the user updated on the progress of the download process after the user chooses a file name.

After the user has been authenticated to the client SAC the first time, no further authentication will be requested. In this case, the area in Figure 11 requesting the pin code will be transparent and the Windows common file dialog box will pop up as soon as the page is loaded, requesting the user to give the name under which the file will be saved. As before, a progress window will keep the user updated on the status of the download.


Secure Connection - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address D:\ANDRE\HTML_pages\ADLLoad.htm
Go Links
Done
My Computer

Alexandria Digital Library Project

Secure Connection

Enter your secret number

OK

You are going to receive a restricted file if you have permission to do so. Please contact us if you believe you have permission and are having problems downloading the file.

Figure 11: Download page requesting user authentication

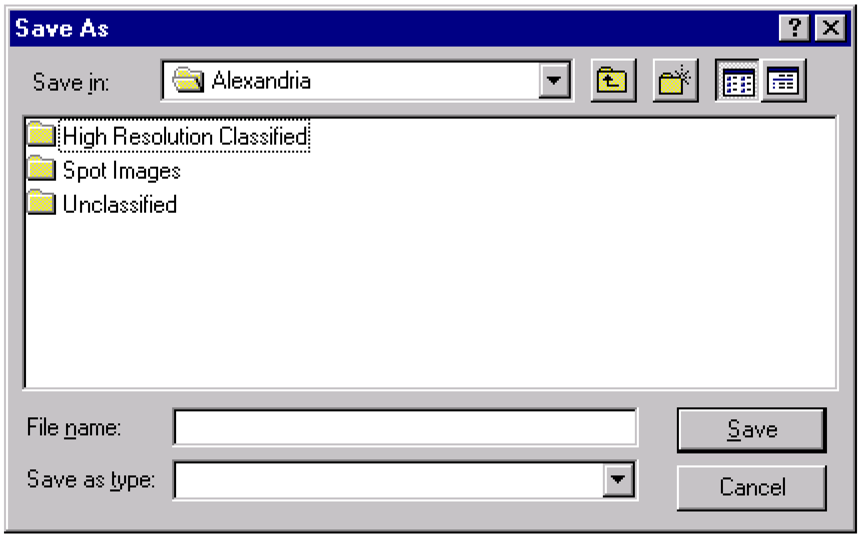


Figure 12: Windows common file dialog box for saving data files

3.3.2.1.4 Future Expansions

The current requirements of the Alexandria Digital Library (ADL) are very simple and easily fulfilled using the SAC approach as described above. ADL has, however, some additional requirements that are being considered. This subsection discusses these requirements and how the SAC approach fulfills them.

A major change that the ADL Internet interface is undergoing is the change from using JiGI for all interactions to using standard web browsers for interactions. This does not affect any of the designs described in the previous sections. The reason for this is that the design of the test bed always used only pure web browsers. The test bed never used any functionality of JiGi.

An additional requirement that is being considered is the implementation of a financial model. The ADL is considering charging for some of the data it offers. The data would only be charged to users outside of the University of California (UC). In order to enforce this policy, the same data would be accessible through two pointers. One pointer, accessible to the UC community, would require an entry in the simple ACL that only smart cards from the UC community have. The access would follow the simple access policy described in section 3.2.4.1. The other pointer, accessible to everyone else, would require an entry in the complex ACL. The access would follow the pay per use policy described in section 3.2.4.3.

ADL is also concerned about the possibility of having to terminate offering some data to non-members of the UC community after providing it for some time. This may occur due to legal constraints not considered initially or simply the desire to no longer offer the service. This is easily accommodated using the *delete a label* operation in Table 12, which deletes an access from the complex ACL.

Finally, the ADL wants to be able to report to non UC users how many tokens are remaining in their smart cards. This is easily accomplished by having the server request this value using the ReportBack field of the security attribute, as described in section 3.2.3.1.3.

3.3.2.2 Generic Online Banking

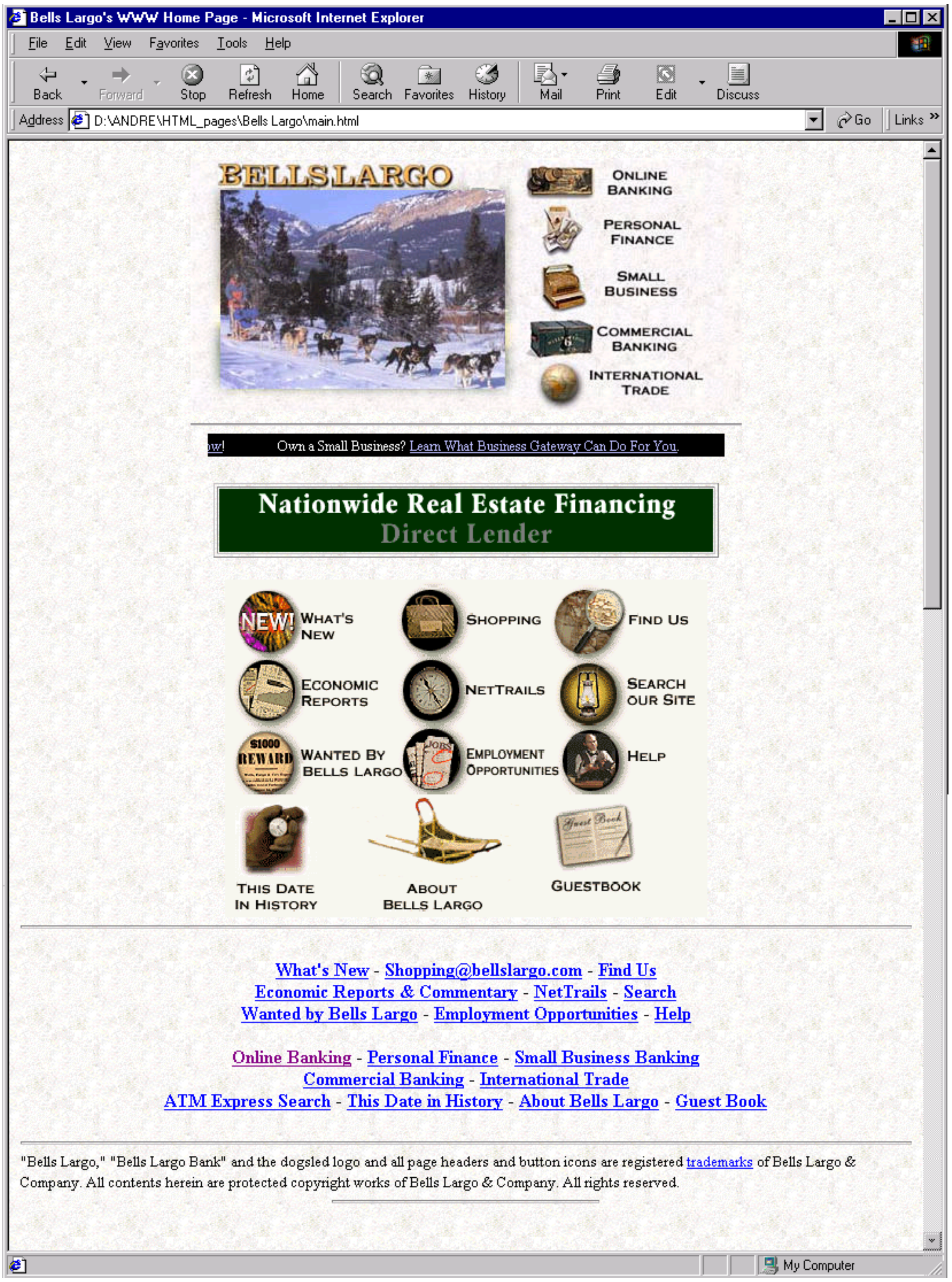
A generic online banking test bed was implemented to better illustrate some characteristics of the SAC approach that were not demonstrated in the ADL test bed.

This test bed implements a fictitious bank, called Bells Largo, which would provide its clients with the ability of checking their account balance and performing transfers. The bank could allow some users to only check balances, others to only perform transfers and still others to perform both operations. In addition the user could place some personalized restrictions on the transfers, such as limiting the amount per day.

Bells Largo distributes client SACs (smart cards) to all of its client. The clients will only be able to access the online banking operations using properly equipped computers. This means that these computers must have card readers and the necessary programs installed.

Each SAC client is personalized to the needs of each client of the bank, allowing the client to check his/her balance and/or perform transfers. In addition, for each client, it imposes a restriction on the value that may be transferred per day.

The main web page of Bells Largo is shown in Figure 13. The next subsections describe the ActiveX controls used for allowing users to interact with the bank in a safe and controlled way and the steps of a user interaction.



ow! Own a Small Business? Learn What Business Gateway Can Do For You.

Nationwide Real Estate Financing
Direct Lender

What's New - Shopping@bellslargo.com - Find Us
Economic Reports & Commentary - NetTrails - Search
Wanted by Bells Largo - Employment Opportunities - Help

Online Banking - Personal Finance - Small Business Banking
Commercial Banking - International Trade
ATM Express Search - This Date in History - About Bells Largo - Guest Book

"Bells Largo," "Bells Largo Bank" and the dogsled logo and all page headers and button icons are registered trademarks of Bells Largo & Company. All contents herein are protected copyright works of Bells Largo & Company. All rights reserved.

Figure 13: Bells Largo main page

3.3.2.2.1 Authentication control

The authentication control is the same as described in item 3.2.1.1. This control can be used in many different applications without change and is an important control that does most of the necessary initializations on the client side of the interaction.

In contrast to the Alexandria Digital Library test bed, in the online banking test bed the *First Pointer* property points to a valid container. There is a request to the CCP to request the container, after the user has successfully authenticated to the client SAC and the client and server SAC have mutually authenticated each other. After the container is properly requested from the server SAC, it is received by the Client SAC. The client SAC then hands the user the headers for which the user is cleared in clear text representation. The authentication control represents these headers as buttons for the user's selection, as shown in Figure 14. The control does not make any assumption about the number of headers that must be shown as buttons. It just centers and equally spaces them. The page designer must make sure that a large enough area will be reserved for the control to show all possible headers as buttons; otherwise, some of them may not be displayed, even though the user may want access to the data they represent. The control fires the *ChangePage* event after selection of one of these buttons. This continues the interactions through a new dedicated page. All information, such as the text to be shown in the buttons and the page to be changed to, are encoded in the description field of the headers.

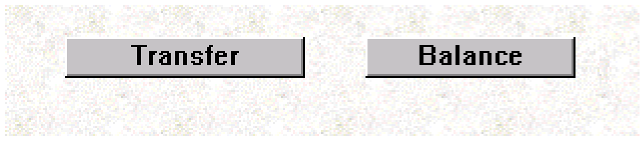
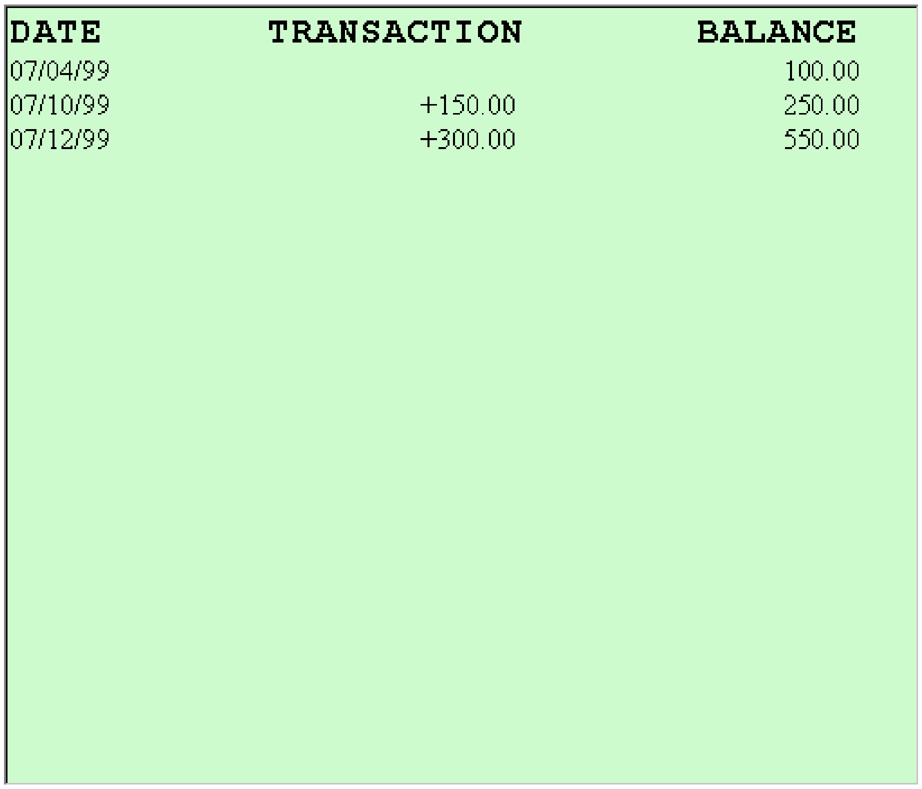


Figure 14: Control showing a container with pointers to two authorized operations

3.3.2.2.2 Rich Text Control

This control is used to show a text message to the user. The message is formatted in rich text format (RTF), which enables it to be displayed using the many features this format is able to specify, e.g. font, size, and appearance. This control is used to display restricted messages to a user. For example, the control that is used to show a user's balance is shown in Figure 15.

This control, like the download control described in section 3.3.2.1.2, has one property, one method and one event. The property, method and event have the same functionality and behavior as the ones described for the download control. However, unlike the download control, this control is visible in order to show the text to the user.



DATE	TRANSACTION	BALANCE
07/04/99		100.00
07/10/99	+150.00	250.00
07/12/99	+300.00	550.00

Figure 15: Rich text control showing a formatted balance

3.3.2.2.3 IO Control

The rich text control is a very simple and efficient way to show sensitive text to a user. Sometimes, however, input from the user is needed. If this input is not sensitive, it can be done outside the control of the SACs. On the other hand, the SACs must be invoked when the input is sensitive. Because of this the IO control was developed.

The IO control is used to show sensitive text and receive sensitive input from the user. Its functionality is not a super set of the rich text control, since it cannot show text formatted as rich text. This may change in the future and the IO control may become the only control used for the output of sensitive text and/or input of sensitive data.

The IO control can show sensitive text, which is located on the top of the control. The sensitive text is followed by a certain number of input fields, windows text boxes, that are used to input sensitive data. Each one of these text boxes may be preceded by text with a description of what input should be entered in that box. Two buttons are drawn in the bottom of the control so the user can confirm that all inputs have been entered or cancel the operation. The IO control used for specifying the account to transfer to and the amount in the online banking system is shown in Figure 16.

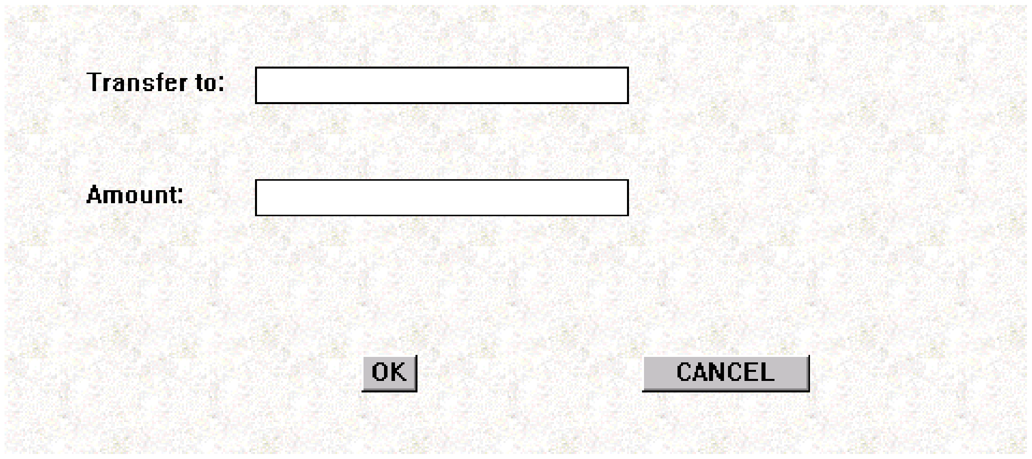


Figure 16: IO control with two input fields used for online transfers

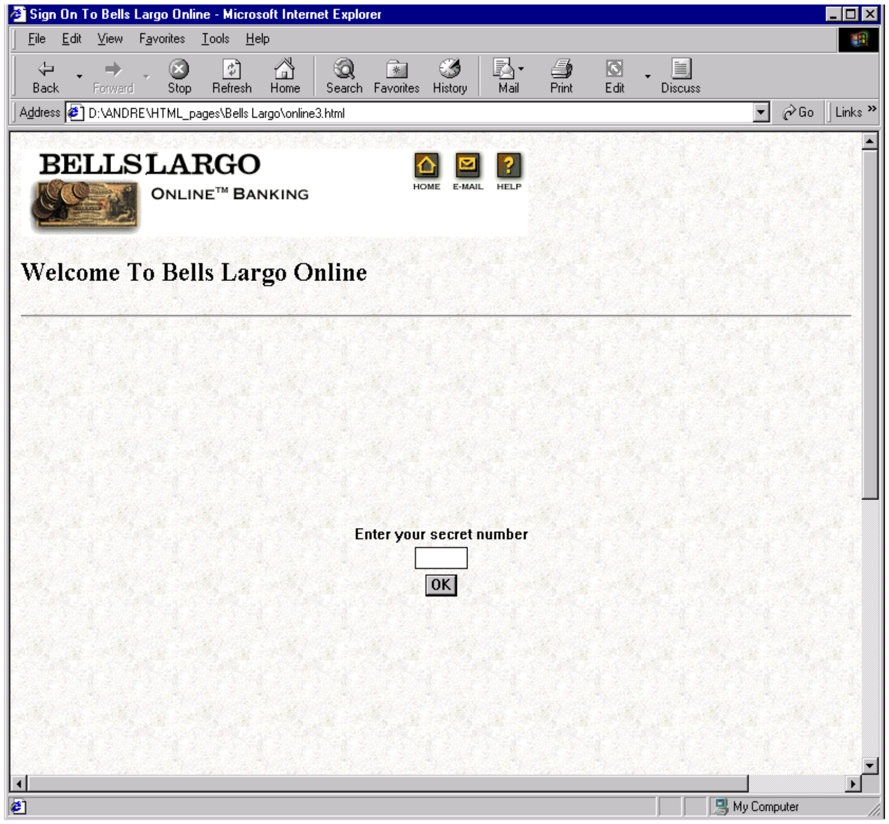
Some or all of the inputs may be numbers that must be cleared by the complex access control lists (ACL) in the client SAC. Because each access control list entry has limited space, it is not able to clear large values. Therefore, the control may use a scale factor for the entered value. This scaling operation is used for translating a value to a token; e.g., every $20.00 is one token. The scaling value is embedded in the security attributes associated with the control and may truncate the value requested by the user. The SAC server will scale the value back up, before sending it for processing. This scheme is used in the test bed for demonstration purposes only. A real application would need to change the scale factor to give more flexibility to the meaning of the tokens, and in some cases the complex ACL may have to be expanded to have more than one byte in some of its fields.

The button in the bottom of the control is used to signal the control that all input has been entered and to keep the current state of the control. This is required in cases where the control changes its state as inputs are entered and sent to the server. An example of such an interaction will be shown for the transfer operation in the Bells Largo test bed.

The IO control has one property, one event and one method. The method and event are used in the same way as those used in the download control described in 3.3.2.1.2. The property is called *Pointer* and it is initially used for the same purpose as the *FirstPointer* property of the RTF and Download controls. The difference here is that this property changes each time a user submits input and it keeps track of the characteristics (security attributes and pointer) of the data being shown in the control.

3.3.2.2.4 User Interaction

The main page of the Bells Largo Bank is shown in Figure 13. A user will start interacting with the SAC when he/she chooses to perform an online banking operation. The web page shown in Figure 17 is displayed to the user at the first interaction with the online banking that requires restricted access.


Sign On To Bells Largo Online - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address D:\ANDRE\HTML_pages\Bells Largo\online3.html
Go Links
BELLSLARGO
ONLINE™ BANKING
HOME E-MAIL HELP
OK
My Computer

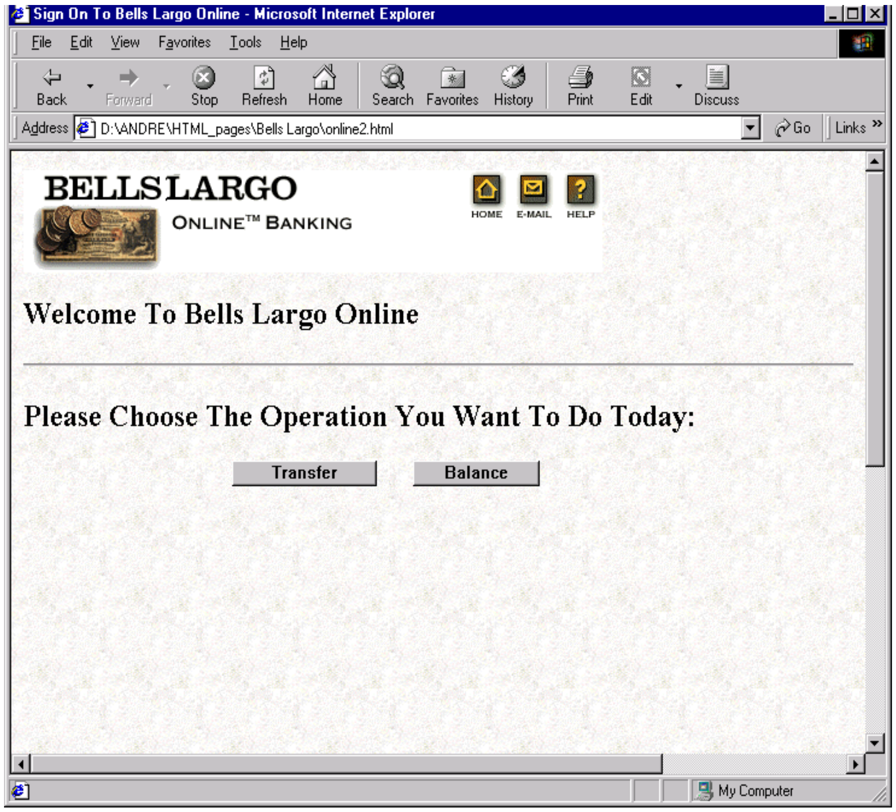
Welcome To Bells Largo Online

Enter your secret number

Figure 17: User authentication for online banking

The user has to enter his/her pin code in order to authenticate himself/herself to the client SAC. The authentication control shows a message requesting the user to insert the smart card (client SAC) in case he/she has not yet done so.

The fictitious bank provides two services: balance and transfer. The authentication control has a pointer to the unclassified container that has these two classified data pointers. The client SAC and server SAC mutually authenticate themselves and create a secure channel between them, after the user enters a correct pin code. After that, the authentication control requests the unclassified container, using the *FirstPointer* property that points to the container. The client SAC receives this request and validates it. If the validation is successful it requests the container from the server SAC. The server SAC fetches the container and sends it to the client SAC, together with the security attributes of the data pointers. The two data pointers have different classifications and the client SAC will only send the pointer(s) for which the user is cleared to the authentication control through the client communication package. Figure 18 shows the case where the user is cleared to perform both operations that this bank offers, and Figure 19 shows the case where the user is cleared only to check his/her balance.

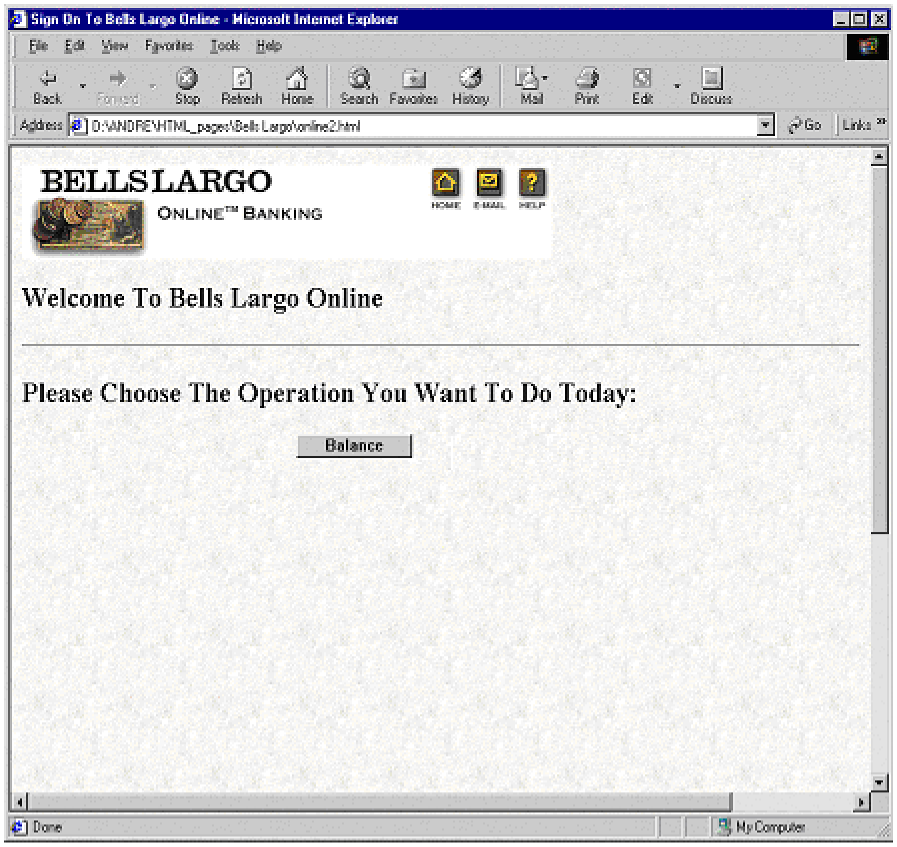


Sign On To Bells Largo Online - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Print Edit Discuss
Address D:\ANDRE\HTML_pages\Bells Largo\online2.html
Go Links
BELLS LARGO
ONLINE™ BANKING
HOME E-MAIL HELP
Transfer
Balance
My Computer

Welcome To Bells Largo Online

Please Choose The Operation You Want To Do Today:

Figure 18: Web page shown to users cleared to perform balance checks and transfers


Sign On To Bells Largo Online - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address D:\ANDRE\HTML_pages\Bells Largo\online2.html
Go Links
BELLSLARGO
ONLINE™ BANKING
HOME E-MAIL HELP
Balance
Done
My Computer

Welcome To Bells Largo Online

Please Choose The Operation You Want To Do Today:

Figure 19: Web page shown to users cleared to perform only balance checks

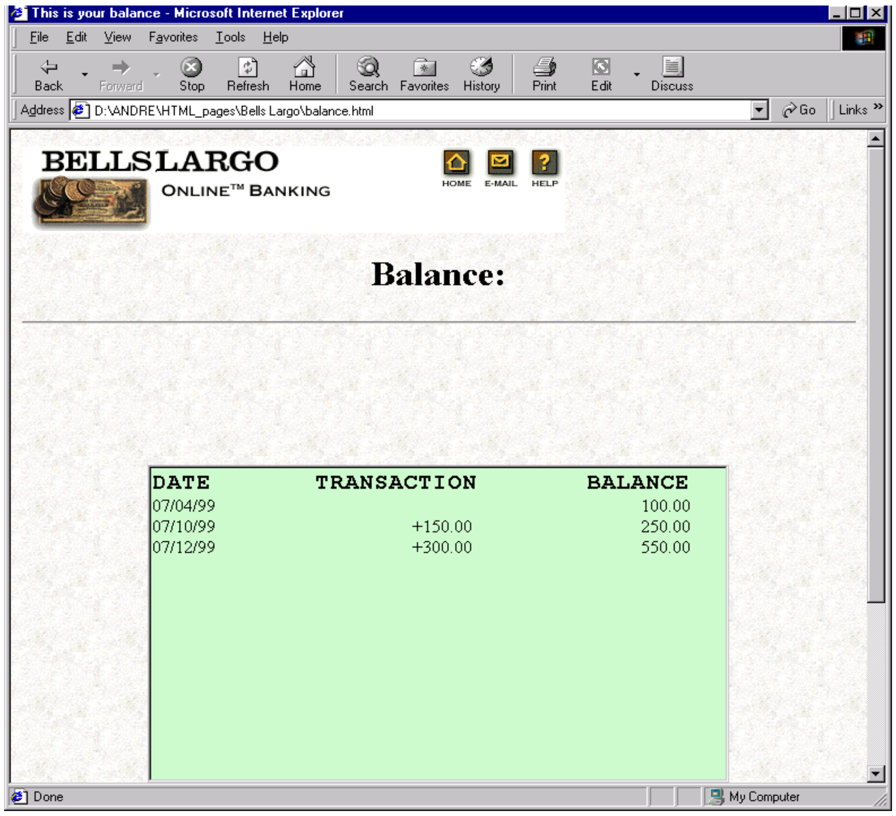
A user that has both clearances can choose to check his/her balance or to perform a transfer. The user clicks on the corresponding button shown in Figure 18 in order to accomplish this. The authentication control fires its *ChangePage* event as a result of the button click. This event is used, together with a Javascript script to change the page shown in the browser to the appropriate one. The balance check and transfer operations are discussed in detail in the next subsections.

3.3.2.2.4.1 Balance Check

The balance check is a very simple operation in which sensitive text is presented to a user. The rich text control described in section 3.3.2.2.2 can present this data as a nicely formatted text. It is this control, embedded in a web page, that is used for showing the balance. An example page that displays a fictitious balance is shown in Figure 20. A green background is used to emphasize the boundaries of the rich text control.

The page shown in Figure 20 assumes that a user has already been authenticated to the client SAC. This may not be the case, and the user may have requested this page through a direct link, e.g. a bookmark. For this reason, the page has an authentication control embedded in it. This authentication control is transparent, and that is why it is not shown in Figure 20. The authentication control would be drawn, as it is in Figure 17, if the user had not authenticated himself/herself. The rich text control would be transparent and waiting for a call to its *OkAuthenticated( )* method in this case.

The rich text control has a property called *FirstPointer* to indicate the security attributes of the data to be shown in the control, and a pointer to locate it. A user that sees this page after going through the page shown in Figure 18 will have been cleared to have this data; otherwise, he/she would not have received a button that would take him/her to this page. However, the SAC does not keep track of previous steps and does not take this into consideration. It only checks the *FirstPointer* property to decide whether or not the user can receive the data. A user that does not have the necessary clearance will not receive the data, being blocked either in the client SAC or in the server SAC. The blocking on the server SAC occurs in situations where the user tries to cheat the SAC and changes the security attributes of the *FirstPointer* property locally.


This is your balance - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Print Edit Discuss
Address D:\ANDRE\HTML_pages\Bells Largo\balance.html
Go Links
BELLSLARGO
ONLINE™ BANKING
HOME E-MAIL HELP
Done
My Computer

Balance:

DATE	TRANSACTION	BALANCE
07/04/99		100.00
07/10/99	+150.00	250.00
07/12/99	+300.00	550.00

Figure 20: Web page that shows a balance

3.3.2.2.4.2 Transfer

The transfer operation is more complex and has many steps. The operation can also enforce many different policies. As an example, Bells Largo enforces the policy that: “some users may transfer up to X dollars per day”. It uses the temporal counter policy described in section 3.2.4.4 to enforce this policy. For this particular case, an entry in the complex ACL has the fields:

- Security Label: This field represents the clearance that the user has to have to perform a transfer.
- Reference: This field represents the day of the last transaction.
- Tokens remaining: This field represents the amount that a user can still transfer on the reference day stored in the reference field.
- Reset Value: This field has the maximum value that may be transferred in a day (the X value in the policy). It is used to reset the tokens remaining field when the reference day is changed.

The transfer operation is performed in a number of steps, each one depending on the previous. Because of this these steps cannot be implemented as different web pages that can be randomly accessed. Therefore, the web page that allows a user to perform a transfer has an embedded IO control. It is this control that keeps track of what step a user is performing. The first step of the interaction is shown in Figure 21.

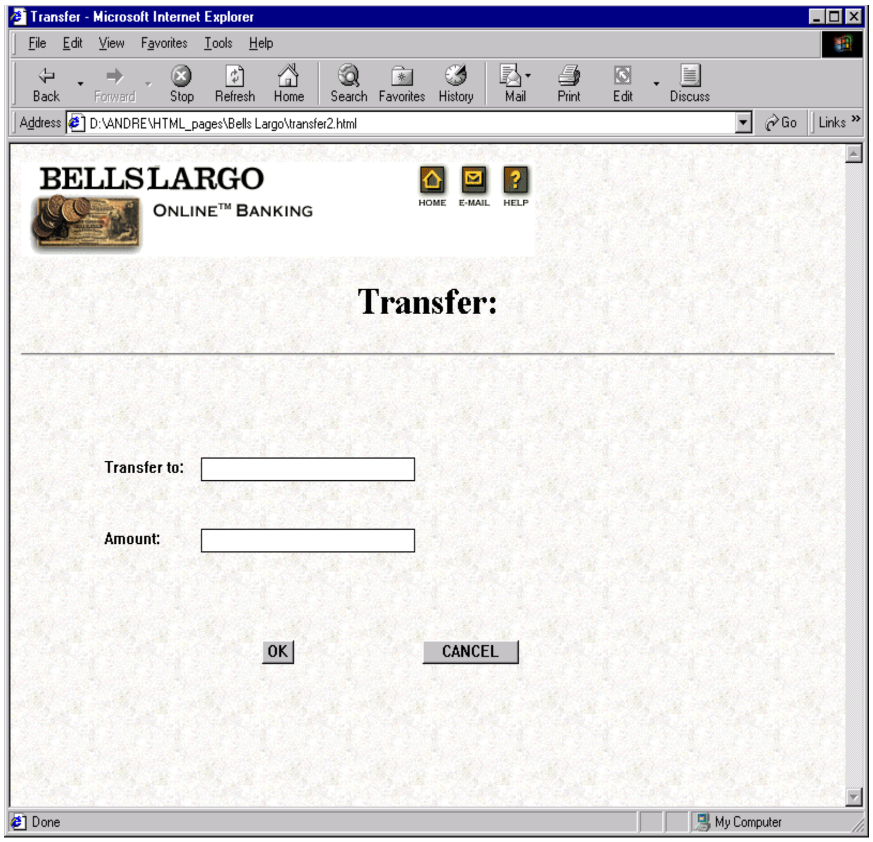

Transfer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss
Address D:\ANDRE\HTML_pages\Bells Largo\transfer2.html
Go Links
BELLSLARGO
ONLINE™ BANKING
HOME E-MAIL HELP
Transfer:
Transfer to:
Amount:
OK
CANCEL
Done
My Computer

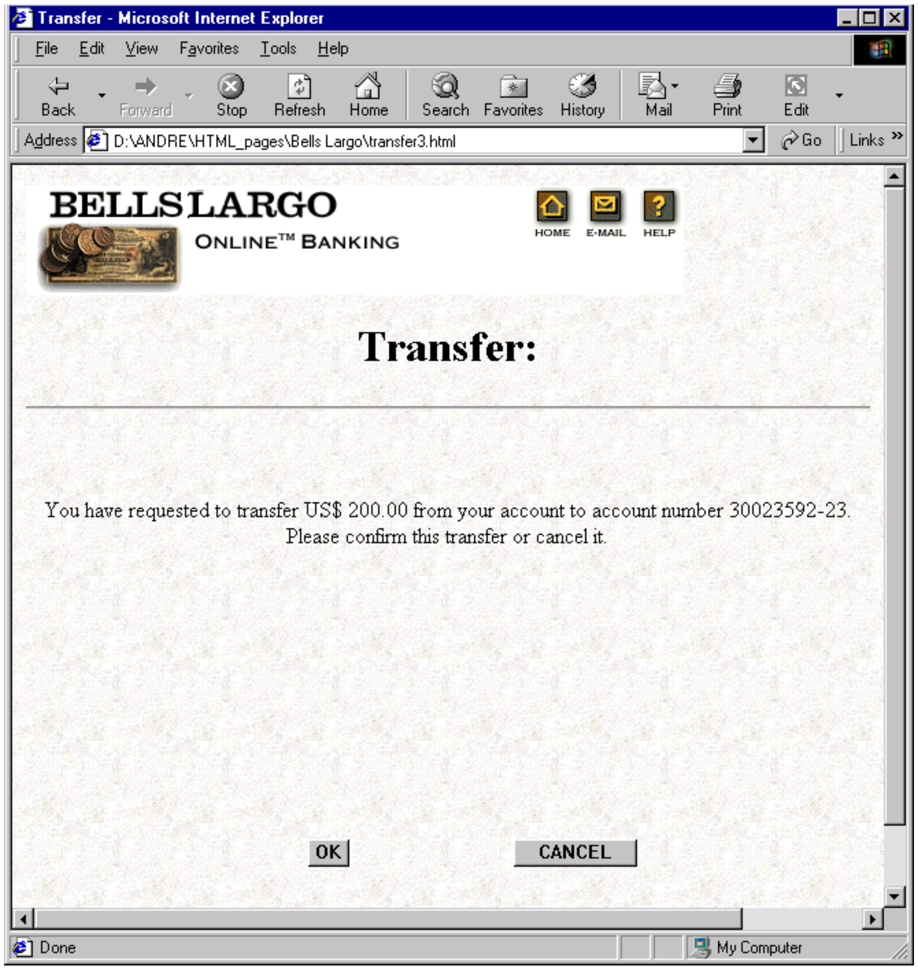
Figure 21: Web page showing the first step of a transfer operation

The web page shown in Figure 21 is only shown to the user that has the necessary clearance to perform a transfer. This is restricted data that the server returns to the client SAC; therefore, it has security attributes associated with it. Besides checking that the user is cleared to access this data, the security attributes have the function of performing an operation to test a value against the reference field of a complex ACL entry and to reset the tokens remaining field, if necessary, as detailed in Table 8. The value that the reference field is tested against is a value sent by the server SAC, which represents the current day. This will ensure that the tokens remaining field is always, and only, reset when the day of the last transfer is before the current day (as perceived by the server). If the current day is after the day of the last transfer, then the value sent by the server will be placed in the reference field of the complex ACL entry, and the value in the reset value field will be copied to the tokens remaining field.

The value entered will be part of the security checks to decide if the user can perform the operation. The account to which the transfer will occur will be part of the data request.

If the user clicks the “Cancel” button, the IO control sends a message to the client SAC requesting it to report this to the server SAC. If the user clicks the “OK” button the IO control composes the security attribute using the value of the transfer and the data pointer using the account number to which the transfer is to be performed. The IO control fills a *DataPtr* structure, described in section 3.2.3.1, with these security attributes and with the pointer from the *FirstPointer* IO control property. This *DataPtr* structure is used to call the function *RequestData(DataPtr req, char filename[8]).* The client communication package sends the request to the client SAC, which checks if the operation is allowed. The client SAC uses the complex ACL for this check and will only send the request to the server SAC if the user is allowed to perform the transfer for the indicated amount. To perform the check the client SAC subtracts the amount requested from the tokens remaining and uses this result to make its decision. It does not commit the result to the tokens remaining field at this time.

When the server SAC receives the request for the transfer it checks if the security attributes are valid, and requests the transfer. This still does not guarantee that the transfer will take place. The server SAC needs to receive a confirmation from the server database that the operation is allowed, e.g. the user has the necessary amount in his/her account. After the server SAC receives an answer that the transaction can take place, it sends a notice saying what was requested to the client SAC. The security attributes of this confirmation request that the transferred value be subtracted from the tokens remaining field of the client SAC complex ACL and committed. Thus, at this time the tokens remaining field stores a new value, which is the old value minus the amount of tokens that represent the transferred value. The client SAC then hands the key necessary to decrypt the data sent by the server to the client communication package. The client communication package does not interpret the data; it only decrypts it and sends it as the answer for the *RequestData* function as described in section 3.2.3.1.3. This data specifies the next step using a pointer that will be used in the next request from the client to the server. This pointer will be stored in the *FirstPointer* IO control property. The data also specifies that the IO control should not draw any input field. This confirmation presents two buttons to the user: one that confirms the transfer and another that cancels it. A sample page that shows this request for confirmation is shown in Figure 22.

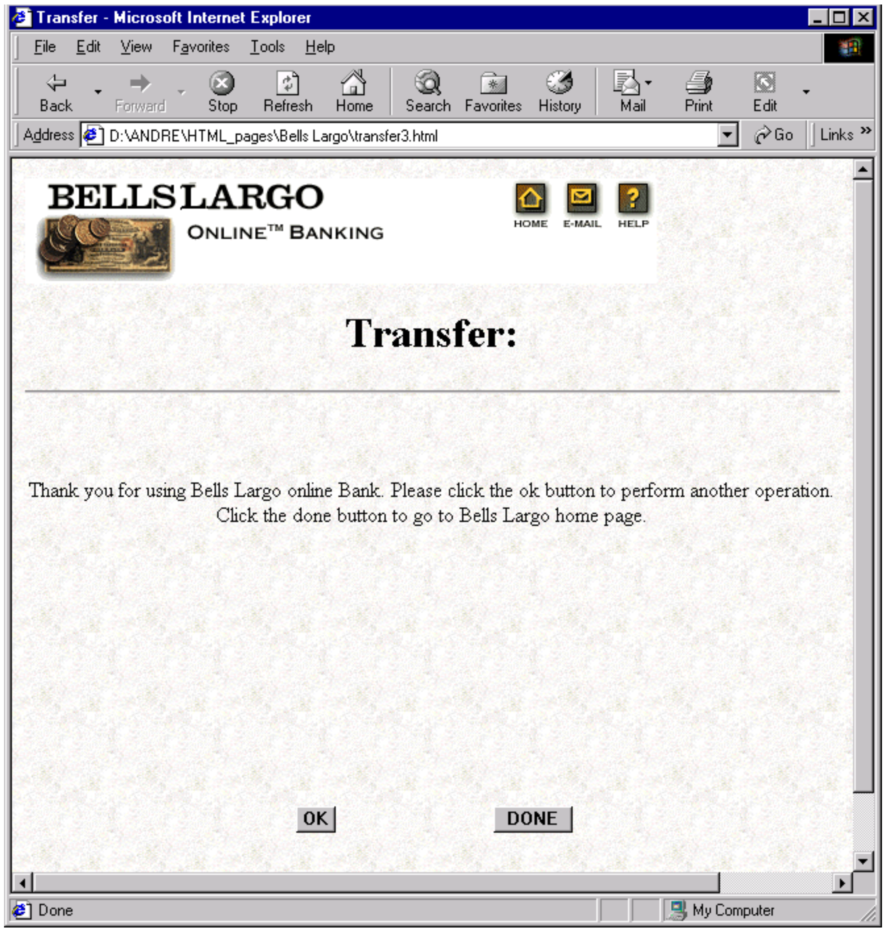

Transfer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address D:\ANDRE\HTML_pages\Bells Largo\transfer3.html
Go Links
BELLS LARGO
ONLINE™ BANKING
HOME E-MAIL HELP
Transfer:
OK
CANCEL
Done
My Computer

You have requested to transfer US$ 200.00 from your account to account number 30023592-23. Please confirm this transfer or cancel it.

Figure 22: Confirmation of a transfer operation

The ActiveX control shows the confirmation message to the user and waits for an interaction. If the user clicks on the “OK” button, the IO control sends the confirmation to the client communication package (CCP) using the *RequestData* function, which sends this to the client SAC. The client SAC sends this confirmation to the server SAC using the secure channel. The server will only validate the transfer request when it receives this confirmation. The server sends data that specifies a message of thanks for using the service. This data will be decrypted by the CCP using a key handed by the client SAC and will pass this data to the IO control. The IO control shows this message as shown in Figure 23.

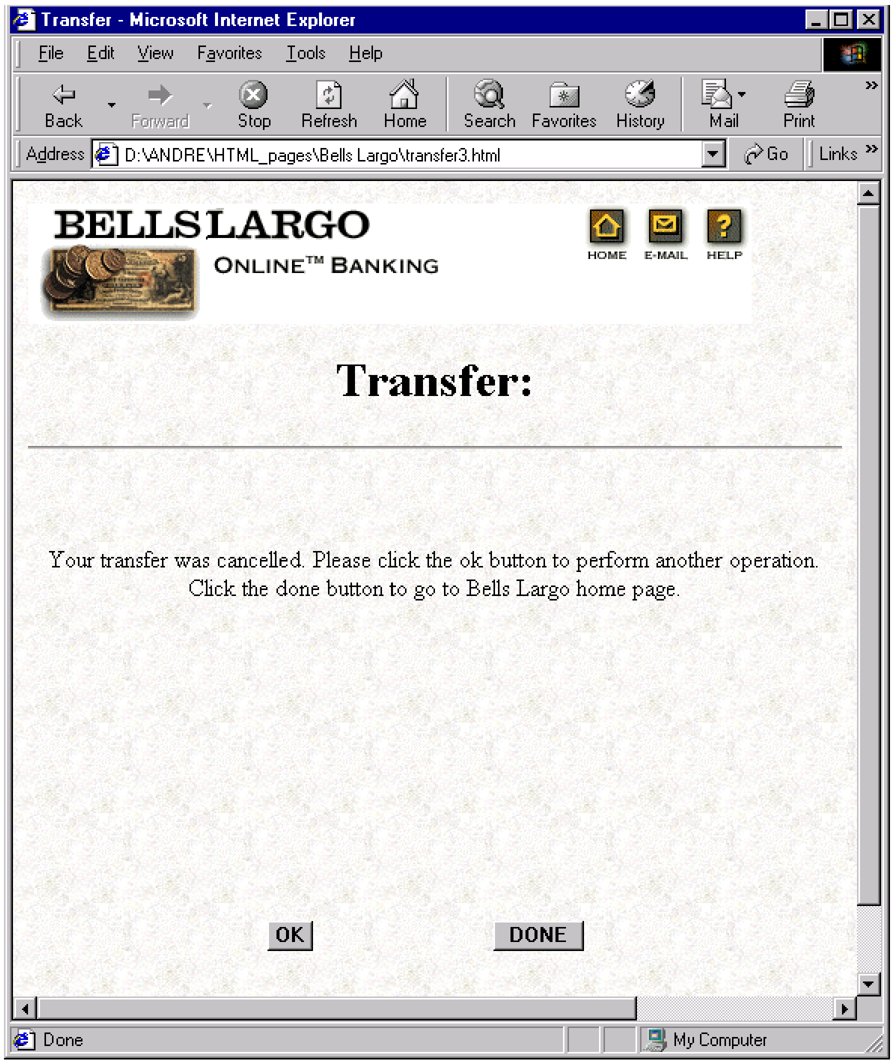
If the user clicks the "Cancel" button in Figure 22, the IO control sends the cancellation to the CCP, which sends it to the client SAC. The client SAC sends the cancel request to the server SAC. The server SAC, after performing the cancellation, sends a cancel confirmation to the client SAC. The IO control shows a confirmation of the cancel as shown in Figure 24. This cancel confirmation has security attributes that request that the transfer value must be added back into the tokens remaining field of the client SAC's complex ACL. This will guarantee that canceled transfers do not penalize the user.


Transfer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit
Address D:\ANDRE\HTML_pages\Bells Largo\transfer3.html
Go Links
BELLS LARGO
ONLINE™ BANKING
HOME E-MAIL HELP
OK
DONE
Done
My Computer

Transfer:

Thank you for using Bells Largo online Bank. Please click the ok button to perform another operation. Click the done button to go to Bells Largo home page.

Figure 23: Confirmation of the transfer operation


Transfer - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print
Address D:\ANDRE\HTML_pages\Bells Largo\transfer3.html
Go Links
BELLSLARGO
ONLINE™ BANKING
HOME E-MAIL HELP
Transfer:
OK
DONE
Done
My Computer

Your transfer was cancelled. Please click the ok button to perform another operation.
Click the done button to go to Bells Largo home page.

Figure 24: Confirmation of the cancellation of a transfer operation

3.4 Benchmarks

The SAC approach assumes that the client SAC is implemented using a hardware device that has limited computing power. Thus, the client SAC has to perform the minimum processing that is needed to assure the required security; otherwise, the application will take an unacceptable performance hit. The SAC design and the test bed implementation have taken enormous precautions to not compromise the performance of the applications. This is accomplished by minimizing the use of public key cryptography to only what is necessary to assure security and not performing lengthy encryption and decryption in the client SAC. This section will describe the times required for the operations that use the SAC. The times are an average of five measurements.

The benchmarks are based on measurements taken with the configuration using personal computers running Windows NT server 4.0, service pack 6, as both client and server. In order to eliminate network delays, the client communicates with the server through a Windows NT named pipe. The personal computer has an Intel Celeron processor running at 333 Mhz, with 128 MB of memory. The smart card has a Siemens SLE66CX160S processor running at 5 Mhz. The Windows NT was not optimized before taking the measurements. This was intentional so that the performance measurements would reflect average use and not the ideal.

Description	Time (ms.)
Time necessary to authenticate the user to the client SAC	290
Time necessary for the server SAC to authenticate the client SAC	561
Time necessary for the client SAC to authenticate the server SAC.	621
Total time for authentication	1462

Table 13: Performance of Authentication Protocol

Description	Time (ms.)
Time necessary to request data from the client SAC.	241
Time necessary to request data from the server SAC.	540
Time necessary for the client SAC to decrypt the data key.	321
Time necessary for the client host to encrypt/decrypt 100 Kbytes of data.	186
Total time necessary to request data.	1102 + 186*size (in 100 Kbytes)

Table 14: Performance of Data Requests

Tables 13 and 14 shows the times required to perform the basic operations that the Safe Areas of Computation approach provides. Table 13 represents the overhead necessary to perform the initial authentication and establishment of the secure channel. This is the most costly operation performed in the client SAC, since it uses a public key cryptographic algorithm. The client SAC measurements reflect both the time required for the cryptographic protocol inside the smart card and the time required for the communication with the personal computer using the serial port. The server SAC measurements reflect the time required for the cryptographic protocol and the time required to access the correct server SAC that corresponds to the client SAC that is communicating with the server SAC. This time may increase with an increased number of client SACs concurrently communicating with the server SAC. This should not represent a problem, however, since the server SAC's processing power can easily be upgraded to accommodate the number of clients that may be serviced concurrently.

Table 14 represents the time required to access data. Small amounts of data like the update requests from the server and for containers will not require the time for the client host to decrypt any data, since all data will be decrypted and/or interpreted inside the client SAC. For instance, the decryption of a header would take the fixed amount of 1102 ms., while the decryption of a 10 MB file would take (1102 + 186*100) ms. or 19602 ms. In addition, the time required for the client host decryption will change with the processing power of the client host.

The authentication of the user to the client SAC results in a message encrypted using the RSA algorithm with the server SAC public key, assuming the user enters the correct pin code. The server SAC decrypts the message sent from the client SAC using its private key and encrypts an answer using the RSA algorithm and the client SAC public key, during the authentication of the client SAC. The client SAC decrypts the answer from the server SAC using its private key during the authentication of the server SAC.

If the overhead of the serial communication between the client host and the smart card is not considered, Table 13 shows that the server SAC performs a decryption and an encryption in 561 ms., while the client SAC performs an encryption in 290 ms. and a decryption in 621 ms., for a total time of 911 ms. This shows that although the clock speed of the server SAC is much higher than the clock speed of the client SAC, their performance only differs by a factor of less than 2. The main reason for this performance difference is that the client SAC has a coprocessor dedicated to performing and speeding up modular exponentiations, which are used in the RSA algorithm.

Table 13 also indicates that the client SAC decryption based on the RSA algorithm is less than 3 times slower than the encryption based on the same algorithm. [SCHN96] mentions an implementation of the RSA algorithm on a SPARC II where the decryption is more than 10 times slower than the encryption. The implementation used for the measurements shown in Table 13 uses the Chinese Remainder theorem in addition to the dedicated coprocessor. This decreases the difference between the times required for encryption and decryption. In addition, the overhead of the serial communication between the host computer and the smart card represents a reasonable portion of the time. This communication overhead was not separated out, since it would require specialized hardware to measure it. The relative difference between the decryption and encryption time would certainly increase if this constant overhead were subtracted.

The results shown in Table 14 confirm the assumption that the overhead of communication from the host computer to the smart card must be a reasonable portion of the total time, since the triple DES encryption/decryption is much faster than RSA encryption/decryption. Besides this communication overhead, the lack of a hardware device for DES speed up also contributes to the measured time of 321 ms.. The time of 540 ms. necessary to request data from the server SAC is not the time for the decryption of the request. This time is due to overhead like the communication with the secured database to request the data size. The time necessary for the server to have a clear text request was measured and is less than 1 ms.

From these measurements it is clear that the limited interactions with the SAC will not represent a significant hit on an application's performance. The measurements do not consider network delays. These delays are independent of the SACs and can be much higher than the overhead that the SACs impose.

Besides the processing, the client SAC introduces the overhead of the IO between the host computer and the smart card. The benchmark used a smart card reader from Towitoko Electronics connected to the serial port of the host computer. It is clear from the measurements that because the SAC approach keeps data transactions between the host computer and the client SAC at a minimum this was not a factor for degrading the performance.

The server SAC is not assumed to have limited processing power, as is the case with the client SAC. This is because the server SACs will be limited in number and one can justify the use of more expensive hardware, e.g., using an IBM secure coprocessor. Because of this the server SAC performance is not as critical as the client SAC. However, the measurements show that a server SAC with the processing power of a personal computer will satisfy the requirements of the approach.

The time necessary to mutually authenticate server and client SACs is less than 1.5 seconds plus network time. The network time is small since the client and server only exchange 128 bytes; unless there is some unrelated network problem, which would affect all Internet operation. This is very small and the user should easily accept this one-time overhead. The time to decrypt a header is 1.1 seconds and should also easily be accepted by the user. The time of 186*"size of the file" (in 100 Kbytes), necessary to receive a file may appear to be unacceptable. For a 10 MB files this results in more than 18 seconds. This is not the case, however, since the file is decrypted as it arrives at the client host, and the time necessary to receive 10 MB over the Internet is much higher than 18 seconds.

3.5 Extensions

The Safe Areas of Computation approach increases the trust a user can place in an application. A SAC protects the operations that are performed inside its security perimeter. The server SAC may have a very large security perimeter, since only one is used per service provider. However, every user of the services that the service provider is offering must use a client SAC. As a consequence, the client SAC must be financially feasible and as unintrusive to a user's computing system as possible. This limits the scope of the client SAC security perimeter.

The smart card used in the test beds protects the transactions a user performs. However, there is a threat that a malicious application can use visual output to the user and/or malicious input to the card to perform a transaction that is not intended by the user. This is very hard to prevent, since normal smart cards do not have input and output (IO) devices directly connected to them. In particular, this can represent a threat for users that want to access a protected service provider from a computing system that might have a compromised operating system or application, such as from an Internet Cafe. In this case, the SAC approach would still protect the user against transactions that he/she did not sign up for, but this may not be enough.

The limitation of not being able to perform input and output to the card from inside the security perimeter was recognized in the early stages of smart card development. Because of this, VISA designed a smart card that included a small keypad and display, called super smart card, in 1985 [AS99a]. VISA started testing prototypes of super smart cards, named Ulticard, in May 1986 [AS99b]. The prototype Ulticard is shown in Figure 25. Unfortunately, these cards were not readily accepted due to the small size of the keyboard and display.

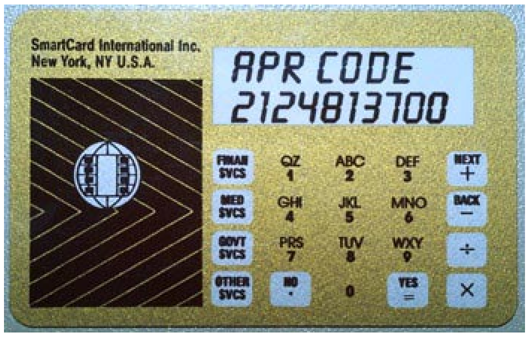


Figure 25: Super smart card designed by VISA

A possible solution to the problem of the lack of trusted input and output is to use an additional pin code (or to require the user to answer a question that only the legitimate user is likely to know, such as maiden name) to authorize some operations. The operations protected by an additional pin are those that are performed only a few times, or never, in a computer that has a likely chance of being tampered with. This could occur, for example, if a user needs to perform an online banking transfer using a computer in an untrusted location. A Trojan horse in a computer used by the user but where the user never performs the operations that require the additional pin code will never learn the additional pin code and, therefore, will not be able to fake any of these operations. The biggest advantage of this solution is that it can be implemented immediately, because it does not need additional tamper resistant hardware. This, however, will not solve the threat of modifications to the parameters used for legal operations after the user presents the additional pin code. For example, a Trojan program could change the account number to which a cash transfer is to be performed in an online banking operation.

Another solution is to have a smart card with limited input and output capability, without using a full feature keyboard and display like the VISA Ulticard had. [GOB96] studied the problems associated with the requirements for trusted input and output for smart cards that were used for electronic cash. In particular, they showed that trusted input with one bit of trusted output can provide general trusted output, trusted output with one bit of trusted input can provide general trusted input, and general trusted input is equivalent to general trusted output (i.e., if a smart card can provide either trusted input or trusted output, then it can provide both). Because smart cards used in the SAC approach are vulnerable to the same threats as smart cards used for electronic cash, a possible solution is to use a modified smart card and take advantage of the property that a trusted output with one bit of trusted input provides general trusted input and output. The proposed smart card would have a small display and a red light that would blink to get the user's attention. The function of the red light is to assure that messages in the display do not go unnoticed by the user. When using this approach, the service provider requests the smart card's red light to blink before committing operations that are considered to be highly sensitive. In addition, the smart card's small display would present messages to the user (i.e., the red light and the display have the function of trusted output). The user, seeing the blinking light, could take the card out of the reader if he/she did not approve the operation. Thus, the user removing the card from the reader or not has the function of a one bit trusted input. Although this solution does not require much effort on the user's part, it is not very user-friendly. It would also present problems to the vision impaired. This solution provides a trusted interface to the user without the problems that the small keyboard of super smart cards had, but a real solution requires further cost and hardware integration analyses.

As an alternative, implementations of the SAC approach could use devices that integrate both a computer keyboard and a card reader, which are already available commercially [HP97, SWE96]. These devices do not incorporate a display, and a user could still suffer an attack that tries to convince him/her to perform some non-desirable input through false visual output. The smart card could be modified, however, to add a one bit trusted output to the trusted input and provide general trusted input and output, as shown in [GOB96]. For instance, the trusted output could be provided by an led integrated into the smart card. A major deficiency of this approach is the lack of assurance as to what degree of resistance to tampering the keyboard and reader offer.

A completely different solution is to use hardware devices other than smart cards. [BAL99] claim that a hand-held computer can be used as a smart card. If hand-held computers were tamper resistant this would allow trusted input and output inside the security perimeter. However, hand-held computers were designed to be generic and to run a variety of applications, like personal computers are. Because of this, they cannot be considered tamper resistant, and they suffer the same vulnerabilities as personal computers. This, however, is a step in the right direction, since due to their form factor hand-held computers are likely to be in the user's possession most of the time. Therefore, there is less chance that a malicious user could access them or insert malicious programs into them than there is with a personal computer.

The tests that VISA performed with the super smart card demonstrated the problem that having input and output inside the security perimeter is not only one of extending the security perimeter, but also of extending the perimeter in a user-friendly manner. The discussion above proposed some devices that could provide more user-friendliness by limiting the input and/or output that would have to be incorporated into the smart card, without sacrificing security. Possible solutions may use some, or all, of the methods described above to provide different ways for a user to interact with a safe area of computation. In the end, however, the choice will depend on the user's background, his/her access clearances, his/her acceptance of additional costs, and his/her willingness to perform additional operations in return for the additional security.

3.6 Fee Models

The SAC approach allows special server SACs, called Central Authorities, to add service providers to client SACs. After the initial set up, the newly added service providers can communicate directly with the client SACs. For some applications the Central Authority may want the service providers to pay a fee based on the amount of use. There are many different payment models to consider. This section discusses a few of the possible ways of charging a fee and how they can be implemented in the client SACs.

The easiest way of charging a fee is to require that a new service provider pay a one time fee for being added to a client SAC. This fee can be charged for each client SAC that the service provider wants to add or it can be a one time fee that is paid for adding as many client SACs as the service provider wants.

Another way of charging a fee is to require that any ACL update operation and/or any addition to the complex ACL needs to be digitally signed by a Central Authority. The client SAC then verifies the signature using the public key of each Central Authority, and the client SAC only completes the update after verifying the correctness of the signature. This is very effective for charging per use when the service provider uses a policy based on counters to charge a user per access for its data. However, this scheme may not be reasonable if the service provider uses temporal policies, such as the temporal counter policy discussed in section 3.2.4.4.

A third way to charge a fee is to have a table for each Central Authority. Each entry in this table would contain a service provider id and a counter. The counter for a specific entry would be decremented every time a client SAC accesses a specific service provider. After this counter is decremented to zero the service provider could no longer be accessed by the client, until a Central Authority updates the counter to a positive value. This scheme charges the service provider a fee per use of the service provider's service. This scheme can be further refined to have different ways of decrementing the counter. A specific charging scheme could be specified in a field of the table entry. For example, one charging scheme would be to charge one token per access that checks the simple or hierarchical access control list and results in a success, but to only charge a token for accesses to the complex access control list if a field of an entry is modified. This table scheme is based on the same principles as the general counter policies in the SAC approach. The table has the access rights of the service provider, controlled by the Central Authority. Therefore, for charging purposes, the Central Authority has the function of a server in the SAC approach, while each specific service provider has the function of a client SAC. This table-based scheme can be regarded as more fair to the Central Authority than the previous scheme, but some service providers may be more comfortable with the former. For example, if the service provider charges for downloads, and the user must buy tokens for these downloads, then the service provider may prefer to pay a fee only for tokens that it sells. That is, the service provider may not want to pay a fee for accesses that are restricted but that are provided free of charge.

A policy that is unlikely to work is to charge a fee based on a reliable time variable, such as a clock. This is difficult to enforce because the smart card does not have a clock inside its security perimeter. Therefore, it has to request the current time from either the client computer or from the service provider. Thus, if both the client and the service provider are in collusion against the Central Authority, then there is no reliable source for a time reference in a transaction. For example, a model could specify that an ACL entry for a service provider should have the tokens remaining field zeroed after a certain date. If it is advantageous to both the service provider and the user to keep the service available, then they could collude to set their clocks for a time before the expiration date. If this is the case, the field would never be zeroed (at least not until the user interacts with the Central Authority or with a service provider that has its clock set correctly). Because of this, until clocks are incorporated into the security perimeter, no reliable models can be based on a clock.

The SAC approach described in this dissertation does not incorporate any of these schemes. However, they can be easily incorporated into the SAC approach and, in particular, into the test beds, which already have the required structures. The exception is the service provider table approach described above, which would require additional structures (one file per Central Authority) to be implemented in the smart card. The additional structures can be easily added to future test beds that use fee models requiring a service provider table. A more difficult task is to have enough information to be able to decide which model best suits most of the possible service providers. An initial estimate is that a mix of the first three schemes described above, which do not use time variables, with a few different options for each scheme, will satisfy the requirements of most service providers. The different options and schemes can be coded into the service provider table and the client SAC can perform the necessary checks and updates depending on the scheme that the particular service provider is using.

CHAPTER 4

Conclusions

The Safe Areas of Computation approach, which is defined in this dissertation, is very flexible, and it can accommodate many different security requirements. This dissertation presented the SAC model, provided detailed information on the approach and its components, described the API used to communicate with the SAC, and demonstrated how it could be implemented to secure real-life applications.

This dissertation also showed that the technologies currently used for interactions between users and sites on the World Wide Web have many security weaknesses. Because of these weaknesses users are exposed to malicious acts. In some cases, such as stealing a credit card number, the liability of a user is limited. However, in other cases, like money transfers in online banking systems and stock dealings in online stock brokering systems, the liability of the user is not always well defined. Therefore, the protection of Internet transactions is a reasonable application to implement as a proof of concept for the Safe Areas of Computation approach.

The SAC approach uses three types of access control lists implemented on the client SAC. Because these access control lists are generic and provide flexible operations, the approach can be used to implement many different security policies. This dissertation presented the operations that use the access control lists to implement different policies and described how a number of basic security policies could be enforced. These basic security policies can be composed into more complex policies to implement most of the realistic policies currently in use.

Employing a user- or device-specific access control list enables the personalization of the accesses and restrictions a user will have (e.g., only allowing the user to view his/her bank account balance and not allowing withdrawals or transfers). This personalization has the advantage of providing access to only the services that the user signs up for, which also limits the user's liability, since the impersonation of the user is limited to only these services.

Another advantage of storing access control lists on the client SAC is that when the server receives a request for a service, the client SAC has already authorized this request at the client host. The result is that the server does not need to again verify that the specific client has the necessary access rights for the requested service. The server still needs to verify that the security labels for the service have not been changed at the client untrusted platform. However, changing security labels represents a malicious act that is not likely to happen frequently, so the client SAC filters requests, which reduces the server load.

The implementation of the protected browser demonstrated that much of the burden of providing a user-friendly interface is placed on the application specific non-secure software. It is not a function of the SAC components. In addition, the SAC approach imposes minimal additional requirements that could make the application software less user-friendly. This helps to satisfy the goal of user-friendliness.

The SAC approach uses trusted devices to improve the security of interactions between a client and a server application. This dissertation showed how a specific tamper resistant device, a smart card, could be used to implement the approach. Two test beds using smart cards for the client SACs were implemented as a proof of concept.

The test beds, which were used for safe Internet transactions, showed how the SAC approach can be easily implemented using ActiveX control technology. An advantage of the SAC approach is that standard applications only need to be enhanced to use the generic functions provided by the SAC Client Communication Package to get secure access to sensitive data. A service provider that needs the security of the SAC approach will likely already have a security policy that associates security labels with its data. The service provider then only needs to design ActiveX controls that are in agreement with this policy and to provide a user interface. In order to make it even easier to implement new services using the SAC approach, this dissertation described some ActiveX controls that can be used to generically support new services. The future work chapter also proposes the development of tools for easing the implementation of Web pages using generic ActiveX controls. These tools would include a graphical user interface that provides an easy way for an administrator to interact with the server SAC. These interactions include adding new users, setting the number of times a user can access a specific data item, invalidating client SACs, and changing a user's access.

The test beds that were implemented demonstrated that the client side application need not require interactions from the user, other than what is necessary to manipulate data. In fact, the only time a user is aware of the SAC is when inserting the smart card in the smart card reader, when entering the pin code, and when trying to perform non-authorized operations. All the other interactions are carried out by the ActiveX controls and are transparent to the user. This results in an application that is user-friendly and allows the client and server SAC to enforce a security policy without requiring complex password schemes.

The test beds also demonstrated that the approach can be implemented using devices that are financially accessible. Smart card readers have a cost of less than \$6.00 if ordered in a quantity greater than 1,000 readers [ZON00]. The smart cards used for the Alexandria Digital Library test bed had a final cost, including the operating system and printing, of \$10.00 per card, ordered in a quantity of 100 cards. These prices demonstrated the commercial feasibility of using smart cards to implement the SAC approach.

This dissertation also discusses the problems that a client SAC can have when input and output is outside of the security perimeter, and it discussed the problems of user-friendliness that occur when the input and output are included in the security perimeter. Finding an approach that uses limited input and/or output and has a minimal impact on the user-friendliness of an application remains an open problem.

CHAPTER 5

Future Work

This dissertation laid the ground work for a very powerful approach. However, as with any research, there are still many areas that need further study to improve the SAC approach. This section discusses some of these areas as pointers to future work.

To provide different services, ActiveX controls must be incorporated in web pages. Almost all the time necessary for the design of new services that use the SAC approach is spent incorporating the correct ActiveX controls to enforce a particular security policy. The test bed implementations showed that it is highly probable that a set of basic ActiveX controls can be built to allow service providers to use the SAC approach for secure transactions through the Internet. An area for future research is identifying these controls, studying the feasibility of building them, and determining how well they fulfill different requirements. In addition, the design and development of a toolset that uses these ActiveX controls to aid in the construction of web pages for services that use the SAC approach must be further researched.

The fee models described in section 3.6 are the result of preliminary work that takes into consideration the possible requirement of a Central Authority receiving a fee for adding other service providers to a system that it controls. These models must be field tested and optimized to satisfy real commercial requirements. The identification of these requirements, their implementation, testing, and tuning is another area for further work.

Some applications may require that a client be able to interact with different servers to request the same service. This may be the case, for example, when a service is offered worldwide and the service provider does not want to require a client to always communicate with a centralized server. Thus, another area for further investigation is the possibility of being able to have different servers offer the same service without the need for them to be separate service providers.

In section 3.5 some schemes for incorporating trusted input and output into the SAC security perimeter were discussed. A further investigation of these schemes with particular interest in how user-friendly they can be made is needed.

The grouping of users and/or data providers is one way to lower the memory requirements for a client SAC. This means, however, that private keys would be shared within a group, which represents a security weakness. The grouping used and the feasibility of grouping is an area for further investigation.

Currently, traffic analysis prevention is not considered to be a requirement for the SAC approach; however, it may be required in future implementations. Thus, the interactions between the user's computer and the secure server for the first search will need to be protected. This could be accomplished using the protection of

SSL. However, even the use of SSL will not prevent a man-in-the-middle attack whose goal is traffic analysis [DDK97]. Because of this, functions for protecting queries will need to be included in the SAC approach if traffic analysis protection is desirable.

References

[ADE99] Alexandria Digital Earth Prototype, “The Alexandria Digital Library Project. Past, Present and Future,” ADL web site, 1999, http://www.alexandria.ucsb.edu/adept/overview.pdf.

[AIM97] Aimtech Corp, “Jamba The Professional Design Tool for Creating Dynamic Java Applets,” Jamba web site, 1997, http://www.jamba.com:80.

[AS99a] Analyses & Synthèses, “The Birth of Smart Cards: 1985,” The Smart Card Cybershow, 1999, http://www.cardshow.com/museum/ex80/y85.html.

[AS99b] Analyses & Synthèses, “The Birth of Smart Cards: 1986,” The Smart Card Cybershow, 1999, http://www.cardshow.com/museum/ex80/y86.html.

[APP97] Apple Computer Inc., “History of the Web,” 1997, http://www.next.com/WebObjects/History.html.

[BAL99] D. Balfanz, and E. W. Felten, “Hand-Held Computers Can Be Better Smart Cards,” *Proceedings of the Eighth USENIX Security Symposium*, pp. 15-23, 1999.

[BEL73] D. Bell, and L. LaPadula, “Secure Computer Systems: Mathematical Foundations and Model,” MITRE Report MTR 2547, vol. 2, 1973.

[BRU97] D. Brumleve, “Tracker,” 1997, http://www.aleph2.com/tracker/tracker.cgi.

[BUSH45] V. Bush, “As We May Think,” *The Atlantic Monthly*, 1945. Reproduced in http://www.isg.sfu.ca/~duchier/misc/vbush/.

[CA96.05] CERT(sm), “Advisory CA-96.05 Java Implementations Can Allow Connections to an Arbitrary Host,” 1996, ftp://info.cert.org/pub/ cert_advisories/CA-96.05.java_applet_security_mgr.

[CA96.07] CERT(sm), “Advisory CA-96.07 Weaknesses in Java Bytecode Verifier,” 1996, ftp://info.cert.org/pub/cert_advisories/CA-96.07.java_bytecode _verifier.

[CA97.20] CERT(sm), “Advisory CA-97.20 JavaScript Vulnerability,” 1997, ftp://info.cert.org/pub/cert_advisories/CA-97.20.JavaScript.

[CARD94] Cardintell, Promotional Communications, 1994.

[DDK97] F. De Paoli, A.L. dos Santos, and R. A. Kemmerer, “Web Browsers and Security”, *Mobile Agents and Security*, LNCS vol. 1419, pp. 235-256, Springer-Verlag, 1998.

[DFW96] D. Dean, E. W. Felten, and D. S. Wallach, “Java Security: From HotJava to Netscape and Beyond”, *Proceedings of 1996 IEEE Symposium on Security and Privacy,* pp. 190-200,1996, http://www.cs.princeton.edu/ sip/pub/secure96.html.

[DIF76] W. Diffie and M. E. Hellman, “New Directions in Cryptography,” *IEEE Transactions on Information Theory*, vol. IT-22, no. 6, pp. 644-654, 1976.

[DIGI97] Digicrime site. http://www.digicrime.com.

[EFF98] Electronic Frontier Foundation, *Cracking DES: Secrets of Encryption Research, Wiretap Politics & Chip Design*, O'Reilly & Associates, 1998.

[FBD97] E. W. Felten, D. Balfanz, D. Dean, and D. S. Wallach, “Web Spoofing: an Internet Con Game,” *Proceedings of the Twentieth National Information Systems Security Conference*, pp. 95-103, 1997, http://www.cs.princeton.edu/sip/pub/spoofing.html.

[FEI87] U. Feige, A. Fiat, and A. Shamir, “Zero knowledge proofs of identity,” *Proceedings of the Nineteenth Annual ACM Symposium on the Theory of Computing*, pp. 210-217, 1987.

[FRE96] J. Frew, et. al. , "The Alexandria Digital Library Testbed," *D-Lib Magazine* [Online], July/August 1996, http://www.dlib.org/.

[FRE98] J. Frew, et. al., “The Alexandria Digital Library Architecture,” *Proceedings of the Second European Conference on Research and Advance Technology for Digital Libraries*, pp. 61-73, 1998, http://www.alexandria.ucsb.edu/~frew/alex-imp/architecture_paper/1998-05-15_2EuroDL/adl-arch.html.

[GAR97] S. Garfinkel and G. Spafford, *Web Security & Commerce*, O’Reilly & Associates, June 1997.

[GARD77] M. Gardner, “A New Kind of Cipher that Would Take Millions of Years to Break,” *Scientific American*, vol. 237, no. 8, pp. 120-124, 1977.

[GEM97] Gemplus, “The Future,” http://www.gemplus.com/welcome/future.htm, 1997.

[GEM97a] Gemplus, "GPK2000, Evaluation & Development Kit, Kit Presentation," Promotional Communication, 1997.

[GIB96] S. P. Gibbons, "Java Security," 1996, http://www.aztech.net/~steve/java/.

[GOB96] H. Gobioff, S. Smith, J. D. Tygar, and B. Yee, "Smart Cards in Hostile Environments," *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pp. 23-28, 1996.

[GON97] L. Gong, M. Mueller, H. Prafullchandra, and R. Schemers. "Going Beyond the Sandbox: An Overview of the New Security Architecture in the Java Development Kit 1.2," *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pp. 103-112, 1997, http://java.sun.com/security/usenix-jdk12security.ps.

[HOP96a] D. Hopwood, "Java Security Bug (Applets Can Load Native Methods)," Risks Forum, vol. 17, no. 83, 1996, ftp://ftp.sri.com/risks/17/risks-17.83.

[HOP96b] D. Hopwood, "More on Java Applet Loading," Risks Forum, vol. 17, no. 86, 1996, ftp://ftp.sri.com/risks/17/risks-17.86.

[HOP96c] D. Hopwood, "Re: Java Security Bug and the Netscape Cache," Risks Forum, vol. 17, no. 87, 1996, ftp://ftp.sri.com/risks/17/risks-17.87.

[HP97] Hewlett-Packard, “HP Develops First PC/SC-Compliant Smart-Card Keyboard For Worldwide Trials,” HP Press Releases, 1997, http://www.hp.com/pressrel/jun97/16jun97e.html.

[INVA97] Internet Valley Inc., “History of Internet and WWW – Part 8 Statistics,” 1997, http://www.internetvalley.com/intvalstat.html.

[KOB87] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, 1987.

[LAND84] C. E. Landwehr, C. L. Heitmeyer, and J. McLean, “A Security Model for Military Message Systems,” *ACM Transactions on Computer Systems*, vol. 9, no. 3, pp. 198-222, 1984.

[LASH97] A. Lash, “Netscape Fixes Latest Bug,” CNET online, 1997 http://www.news.com/News/Item/0,4,12840,00.html.

[LEV96] J. Levitt, “Internetview: An Application Infrastructure,” *Information Week* [Online], 1996, http://techweb.cmp.com/iw/582/82iojl.htm.

[LY96] T. Lindholm and F. Yellin, “The Java^{TM} Virtual Machine Specification,” 1996, http://www.javasoft.com/docs/books/vmspec/html/VMSpecTOC.doc.html.

[MRR97] D. M. Martin Jr, S. Rajagopalan, and A. D. Rubin, “Blocking Java Applets at the Firewall,” *Proceedings of the 1997 Symposium on Network and Distributed System Security*, pp. 16-26, 1997, http://cs-www.bu.edu/techreports/report96-026.

[MS97a] Microsoft Corporation, “Microsoft Security Advisor, Smart Card,” Microsoft web site, 1997, http://www.microsoft.com/smartcard.

[MS97b] Microsoft Corporation, “Frequently Asked Questions ”, Microsoft web site, 1997, http://www.microsoft.com/ie/security/bell.htm.

[MS96] Microsoft Corporation, “ActiveX FAQ,” Microsoft web site, 1996 http://www.microsoft.com/activex/actx-gen/faq.asp.

[MUL45] M. Muller, “Regarding Java Security,” Risks Forum, vol. 17, no. 45, 1995, ftp://ftp.sri.com/risks/17/risks-17.45.

[NAC96] D. Naccache, and D. M’Raihi, “Cryptographic Smart Cards,” *IEEE Micro*, vol. 16, no. 3, pp. 14-24, 1996.

[NBS77] NBS FIPS PUB 46, “Data Encryption Standard,” National Bureau of Standards, U.S. Department of Commerce, 1977.

[NIS99a] National Institute of Standards and Technology, “Advanced Encryption Standard (AES) Development Effort,” 1999, http://csrc.nist.gov/encryption/aes/aes_home.htm.

[NIS99b] NIST FIBS PUB 46-3, “Announcing Draft Federal Information Processing Standard (FIPS) 46-3, Data Encryption Standard (DES), and Request for Comments,” National Institute of Standards and Technology, 1999, http://csrc.nist.gov/cryptval/des/fr990115.htm.

[NET97] Netscape Communications Inc., “JavaScript Authoring Guide”, 1997, http://home.netscape.com/eng/mozilla/2.01/handbook/javascript/index.html.

[NET97a] Netscape Communications Inc, “Netscape to Create '100% Pure Java' Version of Navigator by Early 1998,” Netscape web site, 1997, http://home.netscape.com/newsref/pr/newsrelease471.html.

[NET97b] Netscape Communications Inc, “Using Data Tainting for Security,” Netscape web site, 1997, http://home.netscape.com/eng/mozilla/3.0/handbook/javascript/index.html.

[NET97c] Netscape Communications Inc, “Signed Scripts,” Netscape web site, 1997, http://developer.netscape.com/library/documentation/communicator/jsguide/js1_2.htm.

[NET97d] Netscape Communications Inc., “LiveConnect,” Netscape web site, 1997, http://home.netscape.com/comprod/products/navigator/version_3.0/building_blocks/ liveconnect/index.html.

[NET97e] Netscape Communications Inc., “Netscape Communicator Feature Comparison Summary,” Netscape web site, 1997, http://home.netscape.com/comprod/products/communicator/comparison/summary.html?cp=flh24se.

[NET97f] Netscape Communications Inc., “The Tracker Bug”, Netscape web site, 1997, http://home.netscape.com/assist/security/resources/tracker_bug.html.

[ORE97] C. Orellana, “The Bug,” Cabocomm web site, 1997, http://www.cabocomm.dk/bug.html.

[PET97] S. Peter, "ActiveX: Cornucopia for Pickpockets", *IX online*, 1997, http://www.heise.de/ix/artikel/E/1997/03/090/artikel.shtml.

[PCGS97] PBS Internet Publishing Group, Cochran Interactive, Georgia Public Television and Sun Microsystems, "The History of the Internet," 1997, http://www.pbs.org/internet/history.

[REN96] B. Renaud, "Security Problem with JDK1.0.1", comp.lang.java newsgroups, 1996.

[RIV78] R. L. Rivest, A. Shamir, and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems", *Communications of the ACM*, vol. 21, no.2, pp. 120-126, 1978.

[ROS96] J. Roskind, "Navigator 2.02 Security-Related FAQ," Netscape web site, 1996, http://home.netscape.com/newsref/std/java_security_faq.html.

[RUS91] D. Russel and G. T. Gangemi Sr., *Computer Security Basics*, O'Reilly & Associates Inc., 1991.

[SFAQ97] Javasoft, "Javasoft. Frequently Asked Questions - Applet Security," Javasoft web site, 1997, http:// www.javasoft.com:81/sfaq/index.html.

[SCHN96] B. Schneier, *Applied Cryptography*, John Wiley & Sons Inc, 1996.

[SCH95] R. Schroeppel et ali, "Fast Key Exchange with Elliptic Curve Systems." *Proceedings of the Fifteenth Annual International Cryptology Conference*, pp. 43-56, LNCS, Springer-Verlag, 1995.

[SHA99] A. Shamir, “Factoring Large Numbers with the TWINKLE Device”, Eurocrypt99 rump Session, 1999, http://jya.com/twinkle.eps.

[SIE97] Siemens Components Inc., “Security Controllers IC’s,” Siemens web site, 1997, http://www.sci.siemens.com/htdocs/catalog/ICs/smartcard/scics.html.

[SMI99] S. W. Smith and S. Weingart, “Building a High-Performance, Programmable Secure Coprocessor,” Draft paper, 1999.

[SWE96] A. Sweeney, “Smart Card Security Keyboard Secures OEM Support,” *ZDNet online*, 1996, http://www1.zdnet.com.uk/news/ns-693.html.

[SUN99] Sun Microsystems, “HotJava Browser 3.0,” Sun web site, 1999, http://java.sun.com/products/hotjava/3.0.

[TRE99] W. Treese, “The Internet Index,” no. 24, 1999, http://new-website.openmarket.com/intindex/99-05.htm.

[WS96] D. Wagner, and B. Schneier, “Analysis of the SSL 3.0 protocol,” *Proceedings of the Second USENIX Workshop on Electronic Commerce*, pp. 29-40, 1996.

[WIL96] E. R. Williams, “Java Applet Security: Socket,” 1996, http://www.sky.net/~williams/java/javasec.html.

[WL97] N. Wingfield and A. Lash, “JavaScript Hole Exposes Form Data”, *CNET online*, 1997, http://www.news.com/News/Item/0,4,12282,00.html.

[YEE94] B. S. Yee. “Using Secure Coprocessors,” PhD dissertation, Carnegie Mellon University, 1994.

[ZON00] Zone Development, “Product Pricing,” Zone Development web site, 2000, http://www.zonedevelopment.com/pricing.html.