NEAR EAST UNIVERSITY

INSTITUTE OF APPLIED AND SOCIAL SCIENCES

SECURE SOCKETS LAYERS AND DATA ENCRYPTION USING JAVA APPLICATION

Nehad Bader

Master Thesis

Department Of Computer Engineering

Nicosia 2002

Nehad Bader: Secure Sockets Layers and Data Encryption using Java Application

Approval of Director of the Institute of Applied and Social Sciences

Assoc. Prof. Dr. Dogan Ibrahim
Director

We certify that this thesis is satisfactory for the award of the degree Of Master of Science in Computer Engineering

Examining Committee in Charge:

Prof. Dr. Fakhraddin Mamedov, Chairman of Committee, Head of the Electrical and electronic Engineering Department, NEU

Assist. Prof. Dr. Dogan Haktanir, Computer Engineering Department, NEU

Assoc. Prof. Dr. Rahib Abiyev, Computer Engineering Department, NEU

ACKNOWLEDGEMENTS

I would like to thank my supervisor Assoc.Prof.Dr Dogan Ibrahim for his valuable advice given throughout the duration of this project.

I am also thankful to Assoc.Prof.Dr Adnan Khashman and Assoc.Prof.Dr Senol Bektas for their support during my studies at the Near East University.

ABSTRACT

This thesis specifies the Secure Sockets Layer protocol, a security protocol that provides communications privacy over the Internet. The protocol allows Client/Server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

To fully appreciate how web services will drive a security growth, a new development platforms such as Scalability and Control of Security Capabilities, Authentication and Authorization, Confidentiality, integrity and Security Mechanisms for Web Services should be developed.

In this thesis the author has developed a JAVA based program to allow a user to create RSA keys and encrypt and decrypt text or numbers.

TABLE OF CONTENTS

INTRODUCTION

Internet is an open system. The identity of people, companies and computers communicating on the Internet is not easy to determine and validate. Furthermore, the very communication path is inherently insecure all communications are potentially Open for an eavesdropper to read and modify as they pass between the communicating endpoints.

Many business executives see Web services fueling the next major wave of e-business growth and process efficiencies. IT groups expect Web services to significantly reduce application integration costs, and technology providers envision robust demand for a new generation of Web services-enabled offerings. With interest and motivation at such a high level, standards are evolving rapidly, vendor offerings are coming to market, and early adopters are wading in with the first implementations.

Despite all this forward motion, concerns about security are a major barrier to adoption. After all, many enterprises have not fully mastered security measures in their existing e-business environment. Now, Web services technology makes it even easier to expose critical data and business processes to the outside world, potentially increasing security risks.

Fortunately, standards bodies, vendors and key industries to address security requirements in the new world of Web services are doing much work. Rather than re-inventing the wheel, these efforts leverage the ubiquitous Web infrastructure and proven security technologies already deployed in e-business, such as authentication systems, Web access management and public key infrastructure (PKI). In fact, while Web services create a new set of security challenges, they also provide a new paradigm for delivering pervasive security services in a consistent and cost-effective manner across the enterprise. In my thesis I have described internet security and I discussed the problems that we have from Authentication and Authorization between clients and servers, before couple of

years we used to have such problems in Confidentiality, integrity and Security Mechanisms for Web Services.

There is strong momentum to bring Web services technology into the mainstream of network computing. Thus many companies tried to support Authentication and Authorization by some thing called Cryptographic Algorithms and digital signatures.

This thesis discusses the solution of the network security problem by using a technique known as the "*Secure Sockets Layers and Transport Layer Security*" (SSL/TLS), with new development platforms such as, Scalability, and Control of Security Capabilities, Authentication and Authorization, Confidentiality, integrity and Security Mechanisms for Web Services.

The thesis includes four chapters followed by a Java program developed by me to allow the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.

Chapter 1 is an introduction to the basic principles of network security and various security related issues are described here in detail.

Chapter 2 discusses the Secure Sockets Layers and Transport Layer security technique in detail. Various handshake and authentication protocols are described here.

Chapter 3 is about the Rivest, Adi Shanir, and Leonard (RSA) data encryption technique which is again widely used in network security systems.

Chapter 4 discusses the development of a Java applet that allows a user to create RSA keys and encrypt and decrypt text or numbers.

CHAPTER ONE

WEB SERVICES SECURITY

1.1. Overview

To fully appreciate how Web services will drive security growth, it is helpful to understand the fundamental mechanisms of Web services technology. As Forrester Research has put it, "Web services aren't a revolution rather, they're just as easier way to do distributed computing." Yet, therein lies their power by dramatically simplifying the way systems communicate, Web services make it far easier to exchange information and thus conduct business across the Internet.

An IBM tutorial defines Web services as "self-contained, self-describing modular applications that can be published, located and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes."

What all this means, in practical terms, is that Web services can announce themselves across the Internet and expose their functionality to other applications ("I'm the Village Bank Online Loan Application Service"). In turn, they can search for and invoke other Web services, ("I want to check an applicant's credit rating with any one of my three credit bureau partners"), and they can interact with those services, exchanging requests for information ("Tell me John Smith's credit rating") and receiving responses ("A-plus").

1.2 E-Business Today

During the last several years, enterprises have made great progress in using the Web to achieve their strategic aims Streamlining processes, driving down costs, and delivering innovative offerings to the marketplace. Despite this evident success, the most visionary aspects of e-business remain elusive. (Remember how industry exchanges were going to revolutionize the way trading partners do business with one another?)

The barriers to e-business growth could be summed up in two words: integration and security. [Ref 5]

• In moving their business processes to the Web, enterprises have learned a hard lesson. With current technologies, the task of integrating disparate resources whether to achieve efficiencies or create new services is more difficult and costly than was originally anticipated. As a result, the vast majority of e-business interactions are still fairly straightforward, involving individual users interacting with discrete applications, via Web browsers. Where application-to-application data exchange is essential, for example, in EDI or supply chain environments those capabilities can only be created through custom integration efforts, which cannot be easily leveraged for multiple purposes and therefore are more difficult to cost-justify.

• Persistent and well-founded concerns about security continue to undermine trust in e-business, among consumers and enterprises alike. Again, due to the limits of current technology, it is nearly impossible for an organization to consistently implement best security practices and enforce security policies across the extended enterprise. The result: security vulnerabilities that are readily exploited by intruders and insiders, further eroding trust in e-business.

Given these factors, business and IT executives have come to realize that it may be years before enterprises achieve the most ambitions of e-business: routinely conducting complex transactions in a secure and trusted environment, with dramatically streamlined business processes and the bare minimum of human intervention.

They also realize that Web services and Web services security will play a key role in transforming this vision into reality.

1.3 Web Services Capabilities

Like existing Web capabilities, consumer-oriented Web services will be browser-driven, as in the loan application example given previously. However, over time, the bulk of Web services traffic will be generated by enterprises and will consist of pure machine-to-

machine interactions that carry out routine business processes with no human intervention.

Take, for example, a Web services-enabled manufacturing environment, where the production management system has increased the quota on an automated production line. This action triggers a query to the inventory management system to determine whether inventory levels are sufficient for the planned production run. Based on the inputs provided by the requesting application, the inventory management system calculates the types and quantities of materials that need to be reordered and notifies the production management system accordingly. In turn, the system initiates one or more purchase requests to the appropriate suppliers' order entry systems all without any human involvement.

1.4 The Web Services Value Proposition

With new development platforms - Web services mechanisms, such as XML formatting and SOAP messaging, are an organic part of the application development environment. This makes it faster, easier and more cost-effective to integrate heterogeneous resources, both within enterprises and between trading partners. In turn, by reducing integration costs, enterprises can undertake a wider range of integration efforts, including initiatives that could not previously be cost-justified.

A Web services application can be designed to aggregate information and services from multiple back-end systems. This makes it possible to perform routine tasks more efficiently, for example, accessing data from multiple systems in order to calculate a business unit's profit and loss.

Because Web services are self-contained and modular, they can be reused for multiple purposes. For example, an enterprise's shipping status service can be invoked by the sales order system and the customer care system.

Support for Federated Identity Management:

Web services play a key role in solutions for federated identity management. A federated approach to identity management enables enterprises to enhance the user experience by providing single sign-on (SSO) and seamless navigation across multiple domains. The business benefits of Web services are well understood. These include faster time-to-market for new applications and services, reduced business process costs and reduced partnering costs. Gartner Research predicts that the delivery of software through a subscription-based hosting model will eventually eliminate the need for most software to be licensed and sold in physical packaging.

In turn, this will greatly reduce IT costs related to software distribution and maintenance. In addition to these gains, enterprises will implement new business models and innovative offerings that attract new customers, enhance revenue and provide a competitive advantage.

1.5 Web Services and Network Computing

There is a strong momentum on several fronts to bring Web services into the mainstream of network computing. There are many proposals that have been put forth and are in Various stages of review by two key standards bodies: the World Wide Web Consortium (W3C) and the Organization for the Advancement of Structured Information Standards (OASIS).

Many of these proposals address the requirements of specific industries. In fact, the outpouring has been so great that W3C is looking at mechanisms for determining which of these proposals are truly foundational and which are more suitably developed by communities of interest. Web services have also inspired a high level of cooperation among technology vendors. For example, Microsoft and IBM have developed a proposal for a Global XML Web Services Architecture (GXA).

Support for Web services is further reflected in vendors' product strategies. Virtually all major hardware and software vendors have committed to introducing Web services Technology into their offerings.

While enterprises are carefully weighing all their technology investments, the pace of Web services implementations is beginning to gain speed. As Forrester Research notes, "early adopters", such as Ford are using Web services today to solve problems that traditional EAI and B2Bi products can't tackle cost-effectively: lower-volume integration between departmental apps and automated exchanges with small suppliers.

1.6 Familiar Challenges with Some New Twists

Despite the rapid progress being made in developing Web services technology, security remains a pressing concern. Security is the number one Inhibitor of Web services adoption. Perhaps, this should come as no surprise, given that many enterprises have only made modest progress in securing their existing e-business environments.

In many respects, the security challenges posed by Web services are similar to those presented by the first generation of Web technology. At the highest level, the objective is the same: to create a convenient and trusted e-business environment, where enterprises and individuals can conduct communications, transactions and other business processes.

The underlying challenges are also the same:

- How do you scale security to protect a widening set of resources against increasing points of vulnerability.

- How do you manage user identities and verify who is on the other end of a network connection.

- How do you exert fine-grained control over user access to sensitive resources.

- How do you ensure the confidentiality and integrity of transactions and communications.

In addressing these familiar challenges of distributed computing, there are major new twists, related to the ever-increasing scale and complexity of e-business.

1.7 Scalability and Control of Security Capabilities

Today, most large enterprises support dozens of major applications, each with its own unique security implementation. The familiar result: multiple schemes for authenticating users and managing access to Web applications. Inefficient and costly, these "islands of security"

Also increase risk by forcing organizations to apply security policies in a patchwork manner. In the future, as Web services gain wide acceptance adding hundreds or even thousands more discrete services to the enterprise IT environment this already unwieldy approach will be unsustainable.

In addition, the distributed nature of Web services makes it more difficult than ever to enforce security policies. As research points out, "With CORBA and DCE, IT Bosses could ensure security by exercising control: Only a small set of security-savvy developers could touch the code.

But with [new tools] novices are easily building and deploying Web services interfaces to critical data and unknowingly exposing their firms to security risks. Though it won't happen tomorrow, Web services will eventually make it possible to address these challenges by creating a unified security infrastructure that functions much like a utility, delivering security services to be consumed by other Web services across the enterprise and between trading partners. As a result, proven capabilities such as authentication, authorization and encryption services will be applied in new ways to protect a much wider set of resources than is possible today.

1.7.1 Authentication and Authorization

In this domain, there are several challenges that need to be addressed.

• In today's enterprise environments, the existence of multiple authentication schemes which typically result in multiple identities for one individual create major interoperability problems while also thwarting efforts to centrally manage user identities.

• Current authentication and authorization solutions focus largely on human-machine interactions. The growth of machine-to-machine interactions requires mechanisms for Web services to establish trust with each other, with support for multiple methods, including public key, SAML and perhaps Kerberos-based authentication and authorization. [Ref 1]

• The demand for federated identity and single sign-on (SSO) solutions creates an additional set of challenges.

1.7.2 Confidentiality and Integrity

Current technologies, most notably cryptography and PKI, offer well-developed mechanisms for protecting whole documents as they cross public and private networks. In addition to safeguarding confidentiality through encryption, these solutions enable recipients to authenticate the sender, via digital signatures, and verify text integrity to ensure a document has not been tampered with. [Ref 1]

Because Web services can easily aggregate information from multiple sources, they present more complex security challenges. For example, a transaction may require that digital signatures and encryption to be applied by multiple parties to different informational components within a service and at different points within a multi-tier system.

1.8 Security Mechanisms for Web Services

With the first generation of e-business technologies, many enterprises made the decision to "implement now and secure later." Wiser now, enterprises realize that addressing fundamental security challenges is one of the prerequisites for the long-term success of Web services and e-business.

For this reason, they are urging vendors to accelerate the development of security standards into the Web services framework and vendors are responding. As of mid-year 2002, approximately 30 key standards relating to security and reliability were in various stages of development, review and approval. In addition, major vendors have pledged support for open security standards and promised to build security into their mainstream products. Some of these core security standards are summarized below.

1.9 Reduce Risk by Providing Network Security

Enterprises that are considering a Web services implementation need to make security an integral part of their planning efforts. One way to reduce the risk and complexity of deploying this new technology is to start with an application that can be easily secured using existing capabilities. For example, point-to-point interactions between two companies can be authenticated and protected using VPN or SSL sessions.

From a security perspective, the next level of complexity involves three-way communication in a peered environment. An example would be a portal or business exchange, where data needs to be passed from one trading partner to another, through a third party. Because information crosses multiple security domains, point-to-point security is inadequate; beyond the first "hop," information will be exposed. At a minimum, an enterprise will want to implement WS-Security to ensure the privacy and integrity of communications as they cross multiple nodes.

With experience gained from these types of deployments, an enterprise will be prepared to address more sophisticated security challenges. For example, with a SAML-based security application, a company can readily transfer information from its diverse authentication systems to its Web access management system, facilitating centralized management of user access privileges. Similarly, a SAML-based system enables trading partners to federate user identity and authentication information for the purpose of providing single sign-on.

1.10 Work with a Trusted Provider

Another way to help ensure the success of a Web services implementation is to work with a trusted provider: one who combines a proven track record for implementing e-business security solutions with a strong knowledge of emerging Web services security technologies. RSA Security is uniquely qualified in this respect. RSA Security is the most trusted name in e-security. The company offers a comprehensive and well-proven set of solutions including authentication, Web access management and developer solutions for organizations that need to create trusted business processes in traditional e-business environments.

RSA Security is moving quickly to make these same capabilities available in the Web services environment. For example, RSA Company has been a significant contributor in the development of security standards for Web services, most notably, the SAML specification for exchanging authentication and authorization information. To speed the adoption of SAML, the company has made two of its patents available on a non-exclusive, royalty-free basis. Further, RSA Security has pledged its support for the WS-Security specification.

As these and other standards for Web services security evolve, RSA Security is fully committed to incorporating them into the company's offerings. In turn, enterprises can create an environment of authenticated, private and legally binding electronic transactions and communications in the new Web services world.

Further, through its involvement in the Liberty Alliance, RSA Security is helping to facilitate the practical application of Web services security capabilities in commercial environments.

The company provides the Alliance with expertise in the critical areas of user authentication, access rights management, data privacy and transaction integrity.

1.11 Summary

Many people feel the Internet is not secure because it is a public network. It is true that connecting an internal network to the Internet can expose the internal systems of a company to very considerable risks. The transmission of a document between two parties over the Internet may result in the document passing through half a dozen networks, each managed by different organizations.

In order to address security properly, it is necessary to build it into the integration solution from the beginning. Security issues must be considered seriously whenever information is sent or received from enterprise systems and when interfaces to the systems are built.

Security concerns can be divided into concerns about access control, and concerns about information and transaction security. Access control mechanisms, such as password protection, encrypted smart cards, biometrics, and firewalls, ensure that only valid users and applications get access to information resources such as user accounts, files and databases.

Information and transaction security schemes such as secret key encryption and public key encryption are used to ensure the privacy, integrity and confidentiality of business Transactions and messages.

CHAPTER TWO
SECURE SOCKETS LAYERS AND TRANSPORT LAYER SECURITY

2.1 Overview

Understanding SSL requires an understanding of cryptographic algorithms, message digest functions (also known as one-way or hash functions), and digital signatures.
SSL is a protocol using different cryptographic algorithms to implement security using authentication with certificates, session key exchange algorithms, encryption and integrity check. It is a common protocol, often used to ensure that the communication between WWW-server and WWW-client is secure.

SSL v3.0 is the de facto standard and the Internet Engineering Task Force, IETF, has a group working to develop SSL v3.0 further and the work formed the basis for the IETF's TLS standard. This working group is named Transport Layer Security (TLS). SSL v3.0 has been developed by Netscape with opinions from several industrial companies and developers.

2.2 Cryptographic Algorithms

Suppose Alice wants to send a private message to her bank to transfer some money. Alice would like the message to be private, since it will include information such as her bank account number and transfer amount. One solution is to use a cryptographic algorithm, a technique that would transform her message into an encrypted form, unreadable except by those it is intended for. Once in this form, the message may only be interpreted through the use of a secret key. Without the key the message is useless: good cryptographic algorithms make it so difficult for intruders to decode the original text that it isn't worth their effort.

There are two categories of cryptographic algorithms: conventional and public key.

2.2.1 Conventional Cryptography

Conventional cryptography, also known as symmetric cryptography, requires the sender and receiver to share a key: a secret piece of information that may be used to encrypt or decrypt a message. If this key is secret, then nobody other than the sender or receiver may read the message. If Alice and the bank know a secret key, then they may send each other private messages. The task of privately choosing a key before communicating, however, can be problematic.

2.2.2 Public Key Cryptography

Public key cryptography, also known as asymmetric cryptography, solves the key exchange problem by defining an algorithm which uses two keys, each of which may be used to encrypt a message. If one key is used to encrypt a message then the other must be user to decrypt it. This makes it possible to receive secure messages by simply publishing one key (the public key) and keeping the other secret (the private key). Anyone may encrypt a message using the public key, but only the owner of the private key will be able to read it. In this way, Alice may send private messages to the owner of a key-pair (the bank), by encrypting it using their public key. Only the bank will be able to decrypt it.

2.3 Certificates

Although Alice could have sent a private message to the bank, signed it and ensured the integrity of the message, she still needs to be sure that she is really communicating with the bank. This means that she needs to be sure that the public key she is using corresponds to the bank's private key. Similarly, the bank also needs to verify that the message signature really corresponds to Alice's signature.

If each party has a certificate which validates the other's identity, confirms the public key, and is signed by a trusted agency, then they both will be assured that they are communicating with whom they think they are. Such a trusted agency is called a Certificate Authority (CA), and certificates are used for authentication.

2.4 Public Key Infrastructure (PKI)

A PKI (public key infrastructure) enables users of a basically unsecured public network such as the Internet to securely and privately exchange data and money through the use of a public and a private cryptographic key pair that is obtained and shared through a trusted authority. The public key infrastructure provides for a digital certificate that can identify an individual or an organization and directory services that can store and, when necessary, revoke the certificates. Although the components of a PKI are generally understood, a number of different vendor approaches and services are emerging. Meanwhile, an Internet standard for PKI is being worked on.

The public key infrastructure assumes the use of public key cryptography, which is the most common method on the Internet for authenticating a message sender or encrypting a message. Traditional cryptography has usually involved the creation and sharing of a secret key for the encryption and decryption of messages. This secret or private key system has the significant flaw that if the key is discovered or intercepted by someone else, messages can easily be decrypted. For this reason, public key cryptography and the public key infrastructure is the preferred approach on the Internet. (The private key system is sometimes known as symmetric cryptography and the public key system as asymmetric cryptography).

A public key infrastructure consists of:

1. A certificate authority (CA) that issues and verifies digital certificate. A certificate includes the public key or information about the public key

2. A registration authority (RA) that acts as the verifier for the certificate authority before

3. Digital certificate is issued to a requestor

4. One or more directories where the certificates (with their public keys) are held

5. A certificate management system

2.4.1 Who Provides the Infrastructure

A number of products are offered that enable a company or group of companies to implement a PKI. The acceleration of e-commerce and business-to-business commerce over the Internet has increased the demand for PKI solutions. Related ideas are the virtual private network (VPN) and the IP Security (IPsec) standard. Among PKI leaders are:

1. RSA, which has developed the main algorithms used by PKI vendors

2. Verisign, which acts as a certificate authority and sells software that allows a company to create its own certificate authorities

3. GTE CyberTrust, which provides a PKI implementation methodology and consultation service that it plans to vend to other companies for a fixed Price

4. Xcert, whose Web Sentry product that checks the revocation status of certificates on a server, using the Online Certificate Status Protocol (OCSP)

5. Netscape, whose Directory Server product is said to support 50 million objects and process 5,000 queries a second; Secure E-Commerce, which allows a company or extranet manager to manage digital certificates; and Meta-Directory, which can connect all corporate directories into a single directory for security management.

2.5 Certificate Contents

A certificate associates a public key with the real identity of an individual, server, or other entity, known as the subject. As shown in the table below, information about the subject includes identifying information (the distinguished name), and the public key. It also includes the identification and signature of the Certificate Authority that issued the certificate, and the period of time during which the certificate is valid. It may have

additional information (or extensions) as well as administrative information for the Certificate Authority's use, such as a serial number.

Subject:	Distinguished Name, Public Key
Issuer:	Distinguished Name, Signature
Period of Validity:	Not Before Date, Not After Date
Administrative Information:	Version, Serial Number
Extended Information:	Basic Constraints, Netscape Flags, etc.

Table1: Certificate Authority

2.5.1 Distinguished Name

A distinguished name is used to provide an identity in a specific context - for instance, an individual might have a personal certificate as well as one for their identity as an employee. Distinguished names are defined by the X.509 standard, which defines the fields, field names, and abbreviations used to refer to the fields (see below).

DN Field	Abbreviation	Description	Example
Common Name	CN	Name being certified	CN=Joe Average
Organization or Company	O	Name is associated with this organization	O=University of Bern
Organizational Unit	OU	Name is associated with this organizational unit, such as a department	OU=Institute of Computer Science
City/Locality	L	Name is located in this city	L=Bern
State/Province	ST	Name is located in this state/province	ST=Bern
Country	C	Name is located in this country	C=CH

		(ISO code)	

Table2: Distinguished Name

A Certificate Authority may define a policy specifying which distinguished field names are optional, and which are required. It may also place requirements upon the field contents, as may users of certificates. As an example, a Netscape browser requires that the Common Name for a certificate representing a server has a name which matches a wildcard pattern for the domain name of that server, such as *.unibe.ch.

2.5.2 ASN.1 notation, BER

The binary format of a certificate is defined using the ASN.1 notation. This notation defines how to specify the contents, and encoding rules define how this information is translated into binary form. The binary encoding of the certificate is defined using Distinguished Encoding Rules (DER), which are based on the more general Basic Encoding Rules (BER). For those transmissions which cannot handle binary, the binary form may be translated into an ASCII form by using Base64 encoding (MIME

2.6 Certificate Authorities

By first verifying the information in a certificate request before granting the certificate, the Certificate Authority assures the identity of the private key owner of a key-pair. For instance, if Alice requests a personal certificate, the Certificate Authority must first make sure that Alice really is the person the certificate request claims.

2.6.1 Certificate Chains

A Certificate Authority may also issue a certificate for another Certificate Authority. When examining a certificate, Alice may need to examine the certificate of the issuer, for each parent Certificate Authority, until reaching one which she has confidence in. She may decide to trust only certificates with a limited chain of issuers, to reduce her risk of a "bad" certificate in the chain.

2.6.2 Creating a Root-Level CA

As noted earlier, each certificate requires an issuer to assert the validity of the identity of the certificate subject, up to the top-level Certificate Authority (CA). This presents a problem: Since this is who vouches for the certificate of the top-level authority, which has no issuer? In this unique case, the certificate is "self-signed", so the issuer of the certificate is the same as the subject. As a result, one must exercise extra care in trusting a self-signed certificate. The wide publication of a public key by the root authority reduces the risk in trusting this key - it would be obvious if someone else publicized a key claiming to be the authority. Browsers are preconfigured to trust well-known certificate authorities.

It is also possible to create your own Certificate Authority. Although risky in the Internet environment, it may be useful within an Intranet where the organization can easily verify the identities of individuals and servers.

2.6.3 Certificate Management

Establishing a Certificate Authority is a responsibility which requires a solid administrative, technical, and management framework. Certificate Authorities not only issue certificates, they also manage them - that is, they determine how long certificates are valid, they renew them, and they keep lists of certificates that have already been issued but are no longer valid (Certificate Revocation Lists, or CRLs). Say Alice is entitled to a certificate as an employee of a company. Say too, that the certificate needs to be revoked when Alice leaves the company. Since certificates are objects that get passed around, it is impossible to tell from the certificate alone that it has been revoked. When examining certificates for validity, therefore, it is necessary to contact the issuing Certificate Authority to check CRLs - this is not usually an automated part of the process.

2.7 SSL Protocol

SSL is placed on top of the TCP/IP layer and below the application layer; this layer is sometimes called secure transport layer. To set up a SSL connection a TCP/IP connection must be established first as SSL uses the primitives of TCP/IP. The SSL connection can

be seen as a secure channel within the TCP/IP connection in which all traffic between the application peers goes encrypted.

Application (e.g., HTTP)
Secure transport layer (SSL/TLS)
TCP
IP
Subnet

Table3: SSL Protocol

These capabilities address fundamental concerns about communication over the Internet and other TCP/IP networks:

- **SSL server authentication** allows a user to confirm a server's identity. SSL- enabled client software can use standard techniques of public-key cryptography to check that a server's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the client's list of trusted CAs. This confirmation might be important if the user, for example, is sending a credit card number over the network and wants to check the receiving server's identity.

- **SSL client authentication** allows a server to confirm a user's identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check that a client's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the server's list of trusted CAs. This confirmation might be important if the server, for example, is a bank sending confidential financial information to a customer and wants to check the recipient's identity.

- **An encrypted SSL connection** requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software,

thus providing a high degree of confidentiality. Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism for detecting tampering - that is, for automatically determining whether the data has been altered in transit. All the calls from the application layer to the TCP layer are replaced with calls to the SSL layer, and the SSL layer will take care of the communication with the TCP layer. SSL itself is actually divided again into two different layers as showed below.

Application layer		
Change Cipher Specification Protocol	Alert Protocol	Handshake protocol
SSL record layer		
TCP layer		
IP layer		

Table4: SSL is divided into two different layers

The SSL record protocol defines the format used to transmit data. The SSL handshake protocol involves using the SSL record protocol to exchange a series of messages between an SSL-enabled server and an SSL- enabled client when they first establish an SSL connection. This exchange of messages is designed to facilitate the following actions:

- Authenticate the server to the client.
- Allow the client and server to select the cryptographic algorithms, or ciphers, that they both support.
- Optionally authenticate the client to the server.
- Use public-key encryption techniques to generate shared secrets.

- Establish an encrypted SSL connection.

In the SSL Record layer the messages are divided/combined to fit within the payload of SSL records, a MAC is created and then the whole record layer message is encrypted with the secret key algorithm agreed upon. The substance of the SSL records will be: content type, protocol version number, length, data payload, message authentication code (MAC, provides for data integrity). The messages are the sent to the other peer through a TCP/IP connection.

SSL record format:

Record Type	Version MSB	Version LSB	Length MSB	Length LSB
Ciphertext (*length* bytes)				
MAC (0, 16 or 20 bytes)				

Table5; SSL record format

- Implicit sequence number, cipher-state

- Types: handshake, alert (close), data, rekey

2.8 Change Cipher Specification Protocol

The change cipher specification protocol exists to signal transitions in ciphering strategies. The protocol consists of a single message, which is encrypted and compressed under the current (not the pending) CiperSpec. The message consists of a single byte of value 1. The change cipher specification message is sent by both the client and server to notify the receiving party that subsequent records will be protected under the just-negotiated CipherSpec and keys.

2.9 Alert Protocol

One of the content types supported by the SSL Record layer is the alert type. Alert message convey the severity of the message and a description of the alert. Alert messages

with a level of "fatal" result in the immediate termination of the connection. In this case, other connection corresponding to the session may continue, but the session identifier must be invalidated, preventing the failed session from being used to establish new connections. Like other messages, alert messages are encrypted and compressed, as specified by the current connection state.

2.10 Ciphers Used with SSL

The SSL protocol supports the use of a variety of different cryptographic algorithms, or ciphers, for use in operations such as authenticating the server and client to each other, transmitting certificates, and establishing session keys. Clients and servers may support different cipher suites, or sets of ciphers, depending on factors such as the version of SSL they support, company policies regarding acceptable encryption strength, and government restrictions on export of SSL-enabled software. Among its other functions, the SSL handshake protocol determines how the server and client negotiate which cipher suites they will use to authenticate each other, to transmit certificates, and to establish session keys.

The cipher suite descriptions that follow refer to these algorithms:

1. **DES**. Data Encryption Standard, an encryption algorithm used by the U.S. Government.

2. **DSA**. Digital Signature Algorithm, part of the digital authentication standard used by the U.S. Government.

3. **KEA**. Key Exchange Algorithm, an algorithm used for key exchange by the U.S. Government.

4. **MD5**. Message Digest algorithm developed by Rivest.

5. **RC2 and RC4**. Rivest encryption ciphers developed for RSA Data Security.

6. **RSA**. A public-key algorithm for both encryption and authentication. Developed by Rivest, Shamir, and Adleman.

7. **RSA key exchange**. A key-exchange algorithm for SSL based on the RSA algorithm.

8. **SHA-1**. Secure Hash Algorithm, a hash function used by the U.S. Government.

9. **SKIPJACK**. A classified symmetric-key algorithm implemented in FORTEZZA-compliant hardware used by the U.S. Government. (For more information, see FORTEZZA Cipher Suites.)

10. **Triple-DES**. DES applied three times.

Key-exchange algorithms like KEA and RSA key exchange govern the way in which the server and client determine the symmetric keys they will both use during an SSL session. The most commonly used SSL cipher suites use RSA key exchange. The SSL 2.0 and SSL 3.0 protocols support overlapping sets of cipher suites. Administrators can enable or disable any of the supported cipher suites for both clients and servers. When a particular client and server exchange information during the SSL handshake, they identify the strongest enabled cipher suites they have in common and use those for the SSL session.

Decisions about which cipher suites a particular organization decides to enable depend on trade-offs among the sensitivity of the data involved, the speed of the cipher, and the applicability of export rules.

organizations want to disable the weaker ciphers to prevent SSL connections with weaker encryption. However, due to government restrictions on products that support anything stronger than 40-bit encryption, disabling support for all 40-bit ciphers effectively restricts access to network browsers that are available only in the United States (unless the server involved has a special Global Server ID that permits the international client to "step up" to stronger encryption).

To serve the largest possible range of users, it's administrators may wish to enable as broad a range of SSL cipher suites as possible. That way, when a domestic client or server is dealing with another domestic server or client, respectively, it will negotiate the use of the strongest ciphers available. And when an domestic client or server is dealing with an international server or client, it will negotiate the use of those ciphers that are permitted under U.S. export regulations.

2.11 Cipher Suites with RSA Key Exchange

Unless otherwise indicated, all ciphers listed in the table are supported by both SSL 2.0 and SSL 3.0. Cipher suites are listed from strongest to weakest; for more information about the way encryption strength is measured, see Key Length and Encryption Strength. Cipher suites supported by the SSL protocol that use the RSA key-exchange algorithm

Strongest cipher suite: This cipher suite is appropriate for banks and other institutions that handle highly sensitive data.

- Triple DES, which supports 168-bit encryption, with SHA-1 message authentication. Triple DES is the strongest cipher supported by SSL, but it is not as fast as RC4. Triple DES uses a key three times as long as the key for standard DES. Because the key size is so large, there are more possible keys than for any other cipher - approximately 3.7 * 1050.

Strong cipher suites: These cipher suites support encryption that is strong enough for most business or government needs.

- RC4 with 128-bit encryption and MD5 message authentication. Because the RC4 and RC2 ciphers have 128-bit encryption, they are the second strongest next to Triple DES (Data Encryption Standard), with 168-bit encryption. RC4 and RC2 128-bit encryption permits approximately 3.4 * 1038 possible keys, making them very difficult to crack. RC4 ciphers are the fastest of the supported ciphers.

- RC2 with 128-bit encryption and MD5 message authentication. Because the RC4 and RC2 ciphers have 128-bit encryption, they are the second strongest next to Triple DES

(Data Encryption Standard), with 168-bit encryption. RC4 and RC2 128-bit encryption permits approximately 3.4 * 1038 possible keys, making them very difficult to crack. RC2 ciphers are slower than RC4 ciphers.
This cipher suite is supported by SSL 2.0 but not by SSL 3.0.

• DES, which supports 56-bit encryption, with SHA-1 message authentication. DES is stronger than 40-bit encryption, but not as strong as 128-bit encryption. DES 56-bit encryption permits approximately 7.2 * 1016 possible keys. Æ Both SSL 2.0 and SSL 3.0 support this cipher suite, except that SSL 2.0 uses MD5 rather than SHA-1 for message authentication.

Exportable cipher suites: These cipher suites are not as strong as those listed above, but may be exported to most countries (note that France permits them for SSL but not for S/MIME). They provide the strongest encryption available for exportable products. Note that for RC4 and RC2 ciphers, the phrase "40-bit encryption" means the keys are still 128 bits long, but only 40 bits have cryptographic significance.

• RC4 with 40-bit encryption and MD5 message authentication. RC4 40-bit encryption permits approximately 1.1 * 1012 (a trillion) possible keys. RC4 ciphers are the fastest of the supported ciphers.

• RC2 with 40-bit encryption and MD5 message authentication. RC2 40-bit encryption permits approximately 1.1 * 1012 (a trillion) possible keys. RC2 ciphers are slower than the RC4 ciphers.

Weakest cipher suite: This cipher suite provides authentication and tamper detection but no encryption. Server administrators must be careful about enabling it, however, because data sent using this cipher suite is not encrypted and may be accessed by eavesdroppers.

- No encryption, MD5 message authentication only. This cipher suite uses MD5 message authentication to detect tampering. It is typically supported in case a client and server have none of the other ciphers in common.

2.12 FORTEZZA Cipher Suites

The list below lists additional cipher suites supported by Netscape products with FORTEZZA for SSL 3.0. It provides a hardware implementation of two classified ciphers developed by the federal government: FORTEZZA KEA and SKIPJACK. FORTEZZA ciphers for SSL use the Key Exchange Algorithm (KEA) instead of the RSA key-exchange algorithm mentioned in the preceding section, and use FORTEZZA cards and DSA for client authentication. Strong FORTEZZA cipher suites: Permitted for deployments within the United States only. These cipher suites support encryption that is strong enough for most business or government needs.

- RC4 with 128-bit encryption and SHA-1 message authentication. Like RC4 with 128-bit encryption and MD5 message authentication, this cipher is one of the second strongest ciphers after Triple DES. It permits approximately 3.4 * 10^38 possible keys, making it very difficult to crack. This cipher suite is supported by SSL 3.0 but not by SSL 2.0.

- RC4 with SKIPJACK 80-bit encryption and SHA-1 message authentication. The SKIPJACK cipher is a classified symmetric-key cryptographic algorithm implemented in FORTEZZA-compliant hardware. Some SKIPJACK implementations support key escrow using the Law Enforcement Access Field (LEAF). The most recent implementations do not. This cipher suite is supported by SSL 3.0 but not by SSL 2.0. - Weakest FORTEZZA cipher suite: This cipher suite provides authentication and tamper detection but no encryption. Server administrators must be careful about enabling it, however, because data sent using this cipher suite is not encrypted and may be accessed by eavesdroppers.

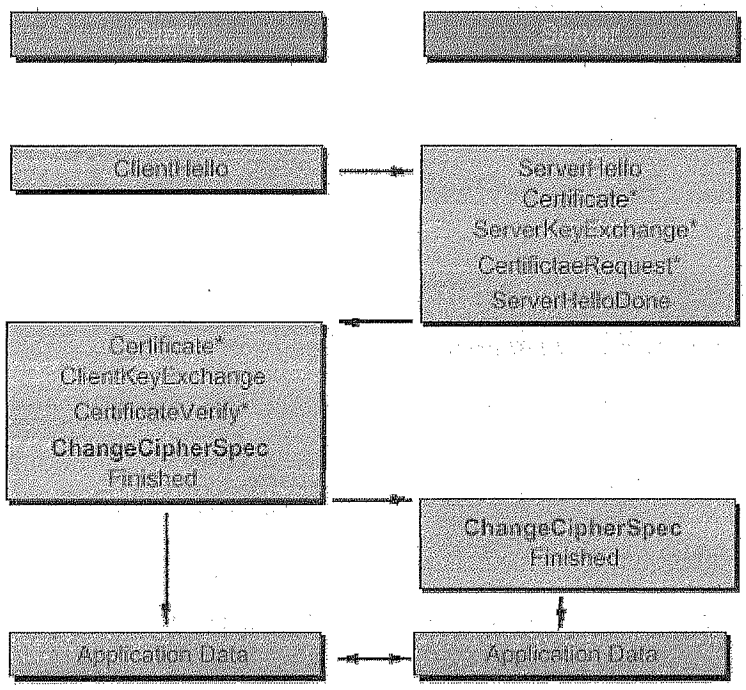
- No encryption, SHA-1 message authentication only. This cipher uses SHA-1 message authentication to detect tampering.

2.13 Handshake Protocol

The SSL protocol uses a combination of public-key and symmetric key encryption. Symmetric key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques. An SSL session always begins with an exchange of messages called the SSL handshake. The handshake allows the server to authenticate itself to the client using public-key techniques, then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows. Optionally, the handshake also allows the client to authenticate itself to the server. During the handshake there is a negotiation of what Cipher Suite to use during data transfer, establishment and sharing of the session key between the client and server and optionally authentication of the peers. The Cipher Suite consists of what methods to use for Key Exchange, Data Transfer and creation of Message Authentication Code, MAC.

1. The client sends the server the client's SSL version number, cipher settings, randomly generated data, and other information the server needs to communicate with the client using SSL.
2. The server sends the client the server's SSL version number, cipher settings, randomly generated data, and other information the client needs to communicate with the server over SSL. The server also sends its own certificate and, if the client is requesting a server resource that requires client authentication, requests the client's certificate.
3. The client uses some of the information sent by the server to authenticate the server (see Server Authentication for details). If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client goes on to Step 4.
4. Using all data generated in the handshake so far, the client (with the cooperation of the server, depending on the cipher being used) creates the premaster secret for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in Step 2), and sends the encrypted premaster secret to the server.

5. If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case the client sends both the signed data and the client's own certificate to the server along with the encrypted premaster secret.
6. If the server has requested client authentication, the server attempts to authenticate the client (see Client Authentication for details). If the client cannot be authenticated, the session is terminated. If the client can be successfully authenticated, the server uses its private key to decrypt the premaster secret, then performs a series of steps (which the client also performs, starting from the same premaster secret) to generate the master secret. After the master secret has been generated, the premaster secret will be immediately erased from memory.
7. Both the client and the server use the master secret to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity - that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection.
8. The client sends a message to the server informing it that future messages from the client will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.
9. The server sends a message to the client informing it that future messages from the server will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.
10. The SSL handshake is now complete, and the SSL session has begun. The client and the server use the session keys to encrypt and decrypt the data they send to each other and to validate its integrity.



(*)_Optional_messages

Fig1. Handshake Protocol [Ref 6]

It's important to note that both client and server authentication involve encrypting some piece of data with one key of a public-private key pair and decrypting it with the other key:

- In the case of server authentication, the client encrypts the premaster secret with the server's public key. Only the corresponding private key can correctly decrypt the secret, so the client has some assurance that the identity associated with the public key is in fact the server with which the client is connected. Otherwise, the server cannot decrypt the premaster secret and cannot generate the symmetric keys required for the session, and the session will be terminated.

- In the case of client authentication, the client encrypts some random data with the client's private key - that is, it creates a digital signature. The public key in the client's certificate can correctly validate the digital signature only if the corresponding private

key was used. Otherwise, the server cannot validate the digital signature and the session is terminated.

2.13.1 SSL Handshake Protocol in Deep [Ref 5]

Hello Request: The hello request message may be sent by the server at any time, but will be ignored by the client if the handshake protocol is already underway. It is a simple notification that the client should begin the negotiation process new by sending a client hello message when convenient.

Client Hello: When a client first connects to a server it is required to send the client hello as its first message. The client can also send a client hello response to a hello request or on its own initiative in order to renegotiate the security parameters in an existing connection. The structure of the client hello is as follows.

```
struct {
        ProtocolVersion client_version;
        Random random;
        SessionID session_id;
        CipherSuite cipher_suites<2..2^16-1>;
        CompressionMethod compression_methods<2..2^8-1>;
} ClientHello;
```

Server Hello: The server processes the client hello message and responds with either a handshake_failure alert or server hello message.

```
struct {
        ProtocolVersion server_version;
        Random random;
        SessionID session_id;
        CipherSuite cipher_suites<2..2^16-1>;
```

```
        CompressionMethod compression_methods<2..2^8-1>;
    } ServerHello;
```

Server Certificate: If the server is to be authenticated (which is generally the case), the server sends its certificate immediately following the server hello message. The certificate type must be appropriate for the selected cipher suite's key exchange algorithm, and is generally an X.509.v3 certificate (or a modified X.509 certificate in the case of FORTEZZA). The same message type will be used for the client's response to a certificate request message.

```
opaque ASN.1Cert<1..2^24-1>;

    struct {
        ASN.1Cert certificate_list<1..2^24-1>;
    } Certificate;
```

Server key exchange message: The server key exchange message is sent by the server if it has no certificate, has a certificate only used for signing, or FORTEZZA KEA key exchanges is used. This message is not used if the server certificate contains Difiie-Hellman parameters.

```
    struct {
        select (KeyExchangeAlgorithm) {
        case diffie_hellman:
        ServerDHParams params;
        Signature signed_params;
        case rsa:
        ServerRSAParams params;
        Signature signed_params;
        case fortezza_kea:
        ServerFortezzaParams params;
```

```
        };
    } ServerKeyExchange;
```

Certificate Request: A non-anonymous server can optionally request a certificate from the client, if appropriate for the selected cipher suite. If is fatal handshake failure alert for an anonymous server to request client identification.

```
struct {
        ClientCertificateType certificate_types<1..2^8-1>;
        DistinguishedName certificate_authorities<3..2^16-1>;
} CertificateRequest;
```

Server Hello done: The server hello done message is sent by the server to indicate the end of the server hello and associated messages. After sending this message the server will wait for a client response: struct { } ServerHelloDone;

Client Certificate: This is the first message the client can send after receiving a server hello done message. This message is only sent if the server requests a certificate. If no suitable certificate is available, the client should send a no_certificate alert instead. This alert is only a warning, however the server may respond with a fatal handshake failure alert if client authentication is required. The client certificate message is similar to the server certificate message above.

Client key exchange message: The choice of messages depends on which public key algorithm(s) has (have) been selected. See also Server key exchange message.

```
struct {
  select (KeyExchangeAlgorithm) {
    case rsa: EncryptedPreMasterSecret;
    case diffie_hellman: ClientDiffieHellmanPublic;
    case fortezza_kea: FortezzaKeys;
```

```
    } exchange_keys;
} ClientKeyExchange;
```

Certificate Verify: This message is used to provide explicit verification of a client certificate. This message is only sent following any client certificate that has signing capability.

```
struct {
    Signature signature;
} CertificateVerify;
```

Finished: A finished message is always sent immediately after a change cipher specs message to verify that the key exchange and authentication processes were successful. The finished message is the first protected with the just-negotiated algorithms, keys, and secrets. No acknowledgment of the finished message is required; parties may begin sending encrypted data immediately after sending the finished message. Recipients of finished messages must verify that the contents are correct.

```
struct {
    opaque md5_hash[16];
    opaque sha_hash[20];
} Finished;
```

2.14 Server Authentication

SSL always requires server authentication, or cryptographic validation by a client of the server's identity. As explained in Step 2 of The SSL Handshake, the server sends the client a certificate to authenticate itself. The client uses the certificate in Step 3 to authenticate the identity the certificate claims to represent. An SSL-enabled client goes through these steps to authenticate a server's identity:

1. **Is today's date within the validity period?** The client checks the server certificate's validity period. If the current date and time are outside of that range, the

authentication process won't go any further. If the current date and time are within the certificate's validity period, the client goes on to Step 2.

2. **Is the issuing CA a trusted CA?** Each SSL-enabled client maintains a list of trusted CA certificates. This list determines which server certificates the client will accept. If the distinguished name (DN) of the issuing CA matches the DN of a CA on the client's list of trusted CAs, the answer to this question is yes, and the client goes on to Step 3. If the issuing CA is not on the list, the server will not be authenticated unless the client can verify a certificate chain ending in a CA that is on the list.
3. **Does the issuing CA's public key validate the issuer's digital signature?** The client uses the public key from the CA's certificate (which it found in its list of trusted CAs in step 2) to validate the CA's digital signature on the server certificate being presented. If the information in the server certificate has changed since it was signed by the CA or if the CA certificate's public key doesn't correspond to the private key used by the CA to sign the server certificate, the client won't authenticate the server's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the client has determined that the server certificate is valid. It is the client's responsibility to take Step 4 before Step 5.
4. **Does the domain name in the server's certificate match the domain name of the server itself?** This step confirms that the server is actually located at the same network address specified by the domain name in the server certificate. Although step 4 is not technically part of the SSL protocol, it provides the only protection against a form of security attack known as a Man-in-the-Middle attack. Clients must perform this step and must refuse to authenticate the server or establish a connection if the domain names don't match. If the server's actual domain name matches the domain name in the server certificate, the client goes on to Step 5.
5. The server is authenticated. The client proceeds with the SSL handshake. If the client doesn't get to step 5 for any reason, the server identified by the certificate cannot be authenticated, and the user will be warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server requires client authentication, the server performs the steps described in Client Authentication.

6. After the steps described here, the server must successfully use its private key to decrypt the premaster secret the client sends in Step 4 of The SSL Handshake. Otherwise, the SSL session will be terminated. This provides additional assurance that the identity associated with the public key in the server's certificate is in fact the server with which the client is connected.

2.14.1 Man-in-the-Middle Attack

As suggested in Step 4 above, the client application must check the server domain name specified in the server certificate against the actual domain name of the server with which the client is attempting to communicate. This step is necessary to protect against a man-in the middle attack, which works as follows.

The "man in the middle" is a rogue program that intercepts all communication between the client and a server with which the client is attempting to communicate via SSL. The rogue program intercepts the legitimate keys that are passed back and forth during the SSL handshake, substitutes its own, and makes it appear to the client that it is the server, and to the server that it is the client.

The encrypted information exchanged at the beginning of the SSL handshake is actually encrypted with the rogue program's public key or private key, rather than the client's or server's real keys. The rogue program ends up establishing one set of session keys for use with the real server, and a different set of session keys for use with the client. This allows the rogue program not only to read all the data that flows between the client and the real server, but also to change the data without being detected. Therefore, it is extremely important for the client to check that the domain name in the server certificate corresponds to the domain name of the server with which a client is attempting to communicate - in addition to checking the validity of the certificate by performing the other steps described in Server Authentication.

2.15 Client Authentication

SSL-enabled servers can be configured to require client authentication, or cryptographic validation by the server of the client's identity. When a server configured this way requests client authentication (see Step 6 of The SSL Handshake), the client sends the server both a certificate and a separate piece of digitally signed data to authenticate itself. The server uses the digitally signed data to validate the public key in the certificate and to authenticate the identity the certificate claims to represent.

The SSL protocol requires the client to create a digital signature by creating a one- way hash from data generated randomly during the handshake and known only to the client and server. The hash of the data is then encrypted with the private key that corresponds to the public key in the certificate being presented to the server.

An SSL-enabled server goes through these steps to authenticate a user's identity:

1. **Does the user's public key validate the user's digital signature?** The server checks that the user's digital signature can be validated with the public key in the certificate. If so, the server has established that the public key asserted to belong to John Doe matches the private key used to create the signature and that the data has not been tampered with since it was signed. At this point, however, the binding between the public key and the DN specified in the certificate has not yet been established. The certificate might have been created by someone attempting to impersonate the user. To validate the binding between the public key and the DN, the server must also complete Step 3 and Step 4.
2. **Is today's date within the validity period?** The server checks the certificate's validity period. If the current date and time are outside of that range, the authentication process won't go any further. If the current date and time are within the certificate's validity period, the server goes on to Step 3.
3. **Is the issuing CA a trusted CA?** Each SSL-enabled server maintains a list of trusted CA certificates. This list determines which certificates the server will accept. If the DN of the issuing CA matches the DN of a CA on the server's list of trusted CAs, the answer to this question is yes, and the server goes on to Step 4. If the issuing CA is

not on the list, the client will not be authenticated unless the server can verify a certificate chain ending in a CA that is on the list (see CA Hierarchies for details). Administrators can control which certificates are trusted or not trusted within their organizations by controlling the lists of CA certificates maintained by clients and servers.

4. **Does the issuing CA's public key validate the issuer's digital signature?** The server uses the public key from the CA's certificate (which it found in its list of trusted CAs in Step 3) to validate the CA's digital signature on the certificate being presented. If the information in the certificate has changed since it was signed by the CA or if the public key in the CA certificate doesn't correspond to the private key used by the CA to sign the certificate, the server won't authenticate the user's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the SSL protocol allows the server to consider the client authenticated and proceed with the connection as described in Step 6. Netscape servers may optionally be configured to take Step 5 before Step 6.
5. **Is the user's certificate listed in the LDAP entry for the user?** This optional step provides one way for a system administrator to revoke a user's certificate even if it passes the tests in all the other steps. The Netscape Certificate Server can automatically remove a revoked certificate from the user's entry in the LDAP directory. All servers that are set up to perform this step will then refuse to authenticate that certificate or establish a connection. If the user's certificate in the directory is identical to the user's certificate presented in the SSL handshake, the server goes on to step 6.
6. **Is the authenticated client authorized to access the requested resources?** The server checks what resources the client is permitted to access according to the server's access control lists (ACLs) and establishes a connection with appropriate access. If the server doesn't get to step 6 for any reason, the user identified by the certificate cannot be authenticated, and the user is not allowed to access any server resources that require authentication.

2.16 Vulnerabilities

SSL v3.0 fixes vulnerabilities in v2.0: The SSLv2 protocol has two known flaws. The downgrade attack allows an active attacker (one who can change bits in the messages between client and server) to force an SSLv2 session to use weaker cipher suites than would ordinarily be negotiated. Since the SSLv2 handshake packets between the client and server are not integrity checked, there's no way for the server to tell if the cipher suite list has been modified. Once the attacker has forced the SSLv2 session into a weak cipher suite, the attacker can use brute-force techniques to crack the small key. The truncation attack allows an attacker to stop the SSLv2 session without the server or client knowing that the session was stopped by an attacker instead of by the other party. If the attacker knows something about the structure of the message and how it is sent in the SSLv2 packets, he can use the truncation attack to change the meaning of the message. For example, Alice the client is sending an order to buy stock to Bob the stock broker's secure server. Mark the attacker can change bits in the communications between Alice and Bob, and Mark wants Alice to lose money. He knows that Alice is sending a buy order, and he knows the format that the buy order will take (simple to figure out, he only has to look as the HTML for the form on Bob's web site). Alice sends "buy 10,000 shares of PBGX". Mark truncates her message to "buy 10,000 shares of PBG". Not being able to tell that Alice's message was truncated, Bob will execute the order to buy 10,000 shares of PBG. If Alice is unlucky, the price of PBG will be much greater than that of PBGX, and Alice will have to go on margin to cover it. At the least, she'll be out the commission and the opportunity cost of not buying PBGX when she wanted to do so.

2.17 Transport Layer Security (TLS)

Internet commerce requires secure communications. To order goods, a customer typically sends credit card details. To order life insurance, the customer might have to supply confidential personal data. Internet users would like to know that such information is safe from eavesdropping or alteration.

Many Web browsers protect transmissions using the protocol SSL (Secure Sockets Layer). The client and server machines exchange nonces and compute session keys from

them. The latest version of the protocol is called TLS (Transport Layer Security) it closely resembles SSL 3.0. Is TLS really secured? My proofs suggest that it is, but one should draw no conclusions without reading the rest of this paper, which describes how the protocol was modeled and what properties were proved. I have analyzed a much simplified form of TLS; I assume hashing and encryption to be secure.

2.17.1 Overview of TLS

TLS is the successor to the Secure Sockets Layer, and is nearly identical to SSL except that it implements

- an open, standards-based solution,
- more non-proprietary ciphers,
- better error reporting, and
- HMAC digests instead of MD5.

TLS and SSL are not interoperable. The TLS protocol does contain a mechanism that allows TLS implementation to back down to SSL. The most recent browser versions support TLS. The TLS Working Group, continues to work on the TLS protocol and related applications.

Internet browsers use security protocols to protect confidential messages. An inductive analysis of TLS, has been performed using the theorem prove Isabelle. Proofs are based on higher-order logic and make no assumptions concerning beliefs or finiteness. All the obvious security goals can be proved; session resumption appears to be secure even if old session keys have been compromised. The proofs suggest minor changes to simplify the analysis.

TLS, even at an abstract level, is much more complicated than most protocols that researchers have verified. Session keys are negotiated rather than distributed, and the protocol has many optional parts. Nevertheless, the resources needed to verify TLS are modest: six man-weeks of effort and three minutes of processor time.

A TLS handshake involves a client, such as a World Wide Web browser, and a Web server. Below, I refer to the client as A ('Alice') and the server as B ('Bob'), as is customary for authentication protocols, especially since C and S often have dedicated meanings in the literature.

At the start of a handshake, A contacts B, supplying a session identifier and nonce. In response, B sends another nonce and his public-key certificate (my model omits other possibilities). Then A generates a *pre-master-secret*, a 48-byte random string, and sends it to B encrypted with his public key. A optionally sends a signed message to authenticate herself. Now, both parties calculate the *master-secret* M from the nonces and the pre-master-secret, using a secure pseudo-random-number function (PRF). They calculate session keys from the nonces and master-secret.

Each session involves a pair of symmetric keys; A encrypts using one and B encrypts using the other. Before sending application data, both parties exchange finished messages to confirm all details of the handshake and to check that clear text parts of messages have not been altered.

A full handshake is not always necessary. At some later time, A can resume a session by quoting an old session identifier along with a fresh nonce. If B is willing to resume the designated session, then he replies with a fresh nonce. Both parties compute fresh session keys from these nonces and the stored master-secret, M. Both sides confirm this shorter run using finished messages.

TLS is highly complex. this version leaves out many details for the sake of simplicity:
1. Record formats, field widths, cryptographic algorithms, etc. are irrelevant in an abstract analysis.
2. Alert and failure messages are unnecessary because bad sessions can simply be abandoned.
3. The server key exchange message allows anonymous sessions among other things, but it is not an essential part of the protocol.

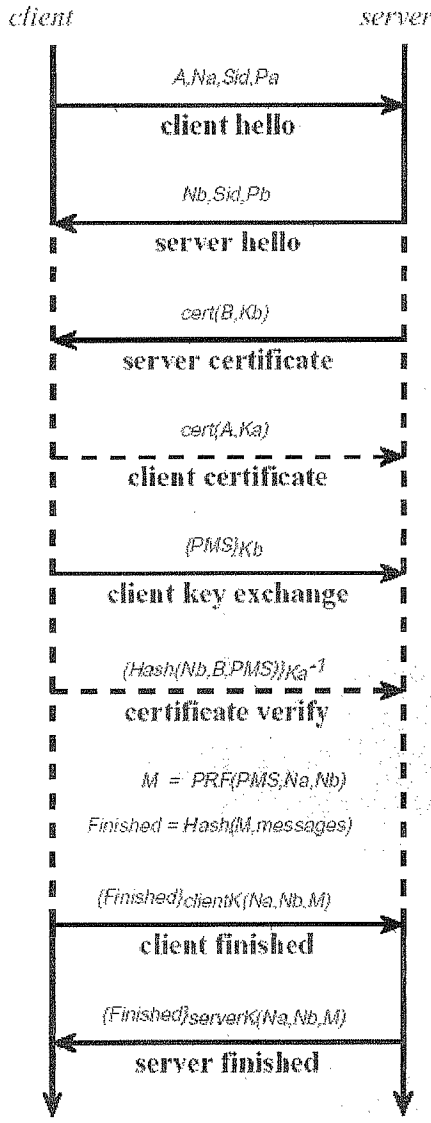


Fig3.1. The TLS Handshake Protocol as Modeled [Ref 6]

The inductive method has many advantages. Its semantic framework, based on the actions agents can perform, has few of the peculiarities of belief logics. Proofs impose no limits on the number of simultaneous or resumed sessions. Isabelle's automatic tools allow the proofs to be generated with a moderate effort, and they run fast. The full TLS proof script runs in 150 seconds on a 300Mhz Pentium.

2.18 Summary

To transfer information privately and securely across the Internet, the Secure Socket Layer (SSL) protocol was developed. This paper examines the SSL protocol and details how to improve the performance and scalability of Web sites by off-loading SSL processing and object delivery from Web servers.

CHAPTER THREE

RIVEST-ADI SHAMIR-LEONARD ADLEMAN (RSA) ENCRYPTION

3.1 Overview

Encryption is the act of encoding text so that others not privy to the decryption mechanism (the "key") cannot understand the content of the text. Encryption has long been the domain of spies and diplomats, but recently it has moved into the public eye with the concern of the protection of electronic transmissions and digitally stored data.

3.2 Public Key Cryptography

One of the biggest problems in cryptography is the distribution of keys. Suppose you live in the United States and want to pass information secretly to your friend in Europe. If you truly want to keep the information secret, you need to agree on some sort of key that you and he can use to encode/decode messages. But you don't want to keep using the same key, or you will make it easier and easier for others to crack your cipher. But it's also a pain to get keys to your friend. If you mail them, they might be stolen. If you send them cryptographically, and someone has broken your code, that person will also have the next key. If you have to go to Europe regularly to hand-deliver the next key, that is also expensive. If you hire some courier to deliver the new key, you have to trust the courier, et cetera.

3.2.1 Trap-Door Ciphers

But imagine the following situation. Suppose you have a special method of encoding and decoding that is "one way" in a sense. Imagine that the encoding is easy to do, but decoding is very difficult. Then anyone in the world can encode a message, but only one person can decode it. Such methods exist, and they are called "one way ciphers" or "trap door ciphers".

Here's how they work. For each cipher, there is a key for encoding and a different key for decoding. If you know the key for decoding, it is very easy to make the key for encoding, but it is almost impossible to do the opposite to start with the encoding key and work out the decoding key.

So to communicate with your friend in Europe, each of you has a trap door cipher. You make up a decoding key **Da** and generate the corresponding encoding key **Ea** your friend does exactly the same thing, but he makes up a decoding key **Db** and generates the corresponding encoding key **Eb** You tell him **Ea** (but not **Da**_) and he tells you **Eb** _ (but not **Db**) Then you can send him messages by encoding using **Eb**_ _ (which only he can decode) and vice-versa—he encodes messages to you using **Ea** (which only you can decode, since you're the only person with access to **Da**).

Now if you want to change to a new key, it is no big problem. Just make up new pairs and exchange the encoding keys. If the encoding keys are stolen, it's not a big deal.
The person who steals them can only encode messages—they can't decode them. In fact, he encoding keys (sometimes called "public keys") could just be published in a well-known location. It's like saying, "If you want to send me a private message, encode it using this key, and I will be the only person in the world who can read it." But be sure to keep the decoding key (the "private key") secret.

3.2.2 Certification

There is, of course, a problem with the scheme above. Since the public keys are really public, anyone can "forge" a message to you. So your enemy can pretend to be your friend and send you a message just like your friend can—they both have access to the public key. Your enemy's information can completely mislead you. So how can you be certain that a message that says it is from your friend is really from your friend? Here is one way to do it, assuming that you both have the public and private keys **Ea Eb Da Db** and as discussed in the previous section. Suppose I wish to send my friend a message that only he can read, but in such a way that he is certain that the message is from me. Here's how to do it. I will take my name, and pretend that it is an encoded message, and decode

it using Da I am the only person who can do this, since I am the only person who knows **Da** Then I include that text in the real message I wish to send, and I encode the whole mess Using **Eb** which only my friend knows how to decode. When he receives it, he will decode it using **Db** and he will have a message with an additional piece of what looks to him like junk characters. The junk characters are what I got by "decoding" my name. So he simply encodes the junk using my public key **Ea** and makes certain that it is my name. Since I am the only one who knows how to make text that will encode to my name, he knows the message is from me.

We can encode any text for certification, and in fact, you should probably change it with each message, but it's easy to do. Your message to your friend would look like this: "Attack at dawn. Here is my decoding of 'ABCDEFG': 'JDLEODK'." To assure privacy, for each message, change the "ABCDEFG" and the corresponding "JDLEODK".

3.3 RSA Encryption

In the previous section we described what is meant by a trap-door cipher, but how do you make one? One commonly used cipher of this form is called "RSA Encryption",
It is based on the following idea:

It is very simple to multiply numbers together, especially with computers. But it can be very difficult to factor numbers. For example, if I ask you to multiply together 34537 and 99991, it is a simple matter to punch those numbers into a calculator and 3453389167. But the reverse problem is much harder [Ref 2]. Suppose I give you the number 1459160519. I'll even tell you that I got it by plying together two integers. Can you tell me what they are? This is a very difficult problem. A computer can factor that number fairly quickly, but (although there are some tricks) it basically does it by trying most of the possible combinations. For any size number, the computer has to check something that is of the order of the size of the square-root of the number to be factored. In this case, that square-root is roughly 38000.

Now it doesn't take a computer long to try out 38000 possibilities, but what if the number to be factored is not ten digits, but rather 400 digits? The square-root of a number with 400 digits is a number with 200 digits. The lifetime of the universe is approximately 10/18 seconds – an 18 digit number. Assuming a computer could test one million factorizations per second, in the lifetime of the universe it could check 10/24 possibilities. But for a 400 digit product, there are 10/200 possibilities. This means the computer would have to run for 10/176 times the life of the universe to factor the large number.
It is, however, not too hard to check to see if a number is prime in other words to check to see that it cannot be factored. If it is not prime, it is difficult to factor, but if it is prime, it is not hard to show it is prime.

So RSA encryption works like this. I will find two huge prime numbers, p and q _ that have 100 or maybe 200 digits each. I will keep those two numbers secret (they are my private key), and I will multiply them together to make a number N=pq That number N is basically my public key. It is relatively easy for me to get N I just need to multiply my two numbers. But if you know N it is basically impossible for you to find p and q To get them, you need to factor N, which seems to be an incredibly difficult problem.

But exactly how is N used to encode a message, and how are p and q used to decode it? Below is presented a complete example, but I will use tiny prime numbers so it is easy to follow the arithmetic. In a real RSA encryption system, keep in mind that the prime numbers are huge.

In the following example, suppose that person A wants to make a public key, and that person B wants to use that key to send A a message. In this example, we will suppose that the message A sends to B is just a number. We assume that A and B have agreed on a method to encode text as numbers. Here are the steps:

1. Person A selects two prime numbers. We will use p=23 and q=41 for this example, but keep in mind that the real numbers person A should use should be *much* larger.

2. Person A multiplies p and q together to get pq= (23)(41)=943 _ is the "public key", which he tells to person B (and to the rest of the world, if he wishes).

3. Person A also chooses another number which must be relatively prime to (p-1)(q-1) In this case, (p-1)(q-1)=(22)(40)=880 so e=7 is fine. E is also part of the public key, so B also is told the value of e.

4. Now B knows enough to encode a message to A. Suppose, for this example, that the message is the number M=35

5. B calculates the $C=M^e \pmod N = 35^7 \pmod{943}$ value of $35^7 = 64339296875$ and $6433929875 \pmod{943} = 545$.
the number 545 is the encoding that B sends to A

7. Now A wants to decode 545. To do so, he needs to find a number d such that $ed = 1 \pmod{(p-1)(q-1)}$ or in this case, such that $7d = 1 \pmod{880}$.

A solution is d=503, since $7*503=3521=4(880)+1=1 \pmod{880}$.

8. To find the decoding, A must calculate $C^d \pmod N = 545^{503} \pmod{943}$ This looks like it will be a horrible calculation, and at first it seems like it is, but notice that 503=256+128+64+32+16+4+2+1 (this is just the binary expansion of 503). So this means that $545^{503} = 545^{256+128+64+32+16+4+2+1} = 545^{256}\ 545^{128} \ldots 545^{1}$.

But since we only care about the result (mod943), we can calculate all the partial results in that modulus, and by repeated squaring of 545, we can get all the exponents that are powers of 2. For example, $545^2 \pmod{943} = 545.545 = 297025 \pmod{943} = 923$

Then square again:

$545^4 \pmod{943} = 545^{2^2} \pmod{943} = 923.923 = 851929 \pmod{943} = 400$

and so on. We obtain the following table:

$545^1 \pmod{943} = 545$

$545^2 \pmod{943} = 923$

$545^4 \pmod{943} = 400$

$545^8 \pmod{943} = 633$

$545^{16} \pmod{943} = 857$

$545^{32} (\text{mod} 943) = 795$

$545^{64} (\text{mod} 943) = 215$

$545^{128} (\text{mod} 943) = 18$

$545^{256} (\text{mod} 943) = 324$

So the result we want is:

$545^{503} (\text{mod} 943) = 324.18.215.795.400.923.545 (\text{mod} 943) = 35.$

Using this tedious (but simple for a computer) calculation, A can decode B's message and obtain the original message N=35

3.3.1 Simple Explanation of RSA

Let p and q be distinct large primes and let n be their product. Assume that we also computed two integers, d (for decryption) and e (for encryption) such that d * e 1 (mod ø(n)) where $ø(n)$ is the number of positive integers smaller than n that have no factor except 1 in common with n The integers n and e are made public, while p, q, and d are kept secret. Let m be the message to be sent, where m is a positive integer less than and relatively prime to n. A plaintext message is easily converted to a number by using either the alphabet position of each letter (a=01, b=02, ..., z=26) or using the standard ASCII table. If necessary (so that $m<n$), the message can be broken into several blocks.

The encoder computes and sends the number

m' = m^e mod n

To decode, we simply compute

e^d mod n

Now, since both n and e are public, the question arises: can we compute from them d? The answer: it is possible, if n is factored into prime numbers. The security of RSA depends on the fact that it takes an impractical amount of time to factor large numbers.

NEAR EAST UNIVERSITY

INSTITUTE OF APPLIED AND SOCIAL SCIENCES

SECURE SOCKETS LAYERS AND DATA ENCRYPTION USING JAVA APPLICATION

Nehad Bader

Master Thesis

Department Of Computer Engineering

Nicosia 2002

Nehad Bader: Secure Sockets Layers and Data Encryption using Java Application

Approval of Director of the Institute of Applied and Social Sciences

Assoc. Prof. Dr. Dogan Ibrahim
Director

We certify that this thesis is satisfactory for the award of the degree Of Master of Science in Computer Engineering

Examining Committee in Charge:

Prof. Dr. Fakhraddin Mamedov, Chairman of Committee, Head of the Electrical and electronic Engineering Department, NEU

Assist. Prof. Dr. Dogan Haktanir, Computer Engineering Department, NEU

Assoc. Prof. Dr. Rahib Abiyev, Computer Engineering Department, NEU

ACKNOWLEDGEMENTS

I would like to thank my supervisor Assoc.Prof.Dr Dogan Ibrahim for his valuable advice given throughout the duration of this project.

I am also thankful to Assoc.Prof.Dr Adnan Khashman and Assoc.Prof.Dr Senol Bektas for their support during my studies at the Near East University.

ABSTRACT

This thesis specifies the Secure Sockets Layer protocol, a security protocol that provides communications privacy over the Internet. The protocol allows Client/Server applications to communicate in a way that is designed to prevent eavesdropping, tampering, or message forgery.

To fully appreciate how web services will drive a security growth, a new development platforms such as Scalability and Control of Security Capabilities, Authentication and Authorization, Confidentiality, integrity and Security Mechanisms for Web Services should be developed.

In this thesis the author has developed a JAVA based program to allow a user to create RSA keys and encrypt and decrypt text or numbers.

TABLE OF CONTENTS

INTRODUCTION

Internet is an open system. The identity of people, companies and computers communicating on the Internet is not easy to determine and validate. Furthermore, the very communication path is inherently insecure all communications are potentially Open for an eavesdropper to read and modify as they pass between the communicating endpoints.

Many business executives see Web services fueling the next major wave of e-business growth and process efficiencies. IT groups expect Web services to significantly reduce application integration costs, and technology providers envision robust demand for a new generation of Web services-enabled offerings. With interest and motivation at such a high level, standards are evolving rapidly, vendor offerings are coming to market, and early adopters are wading in with the first implementations.

Despite all this forward motion, concerns about security are a major barrier to adoption. After all, many enterprises have not fully mastered security measures in their existing e-business environment. Now, Web services technology makes it even easier to expose critical data and business processes to the outside world, potentially increasing security risks.

Fortunately, standards bodies, vendors and key industries to address security requirements in the new world of Web services are doing much work. Rather than re-inventing the wheel, these efforts leverage the ubiquitous Web infrastructure and proven security technologies already deployed in e-business, such as authentication systems, Web access management and public key infrastructure (PKI). In fact, while Web services create a new set of security challenges, they also provide a new paradigm for delivering pervasive security services in a consistent and cost-effective manner across the enterprise. In my thesis I have described internet security and I discussed the problems that we have from Authentication and Authorization between clients and servers, before couple of

years we used to have such problems in Confidentiality, integrity and Security Mechanisms for Web Services.

There is strong momentum to bring Web services technology into the mainstream of network computing. Thus many companies tried to support Authentication and Authorization by some thing called Cryptographic Algorithms and digital signatures.

This thesis discusses the solution of the network security problem by using a technique known as the "*Secure Sockets Layers and Transport Layer Security*" (SSL/TLS), with new development platforms such as, Scalability, and Control of Security Capabilities, Authentication and Authorization, Confidentiality, integrity and Security Mechanisms for Web Services.

The thesis includes four chapters followed by a Java program developed by me to allow the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data.

Chapter 1 is an introduction to the basic principles of network security and various security related issues are described here in detail.

Chapter 2 discusses the Secure Sockets Layers and Transport Layer security technique in detail. Various handshake and authentication protocols are described here.

Chapter 3 is about the Rivest, Adi Shanir, and Leonard (RSA) data encryption technique which is again widely used in network security systems.

Chapter 4 discusses the development of a Java applet that allows a user to create RSA keys and encrypt and decrypt text or numbers.

CHAPTER ONE

WEB SERVICES SECURITY

1.1. Overview

To fully appreciate how Web services will drive security growth, it is helpful to understand the fundamental mechanisms of Web services technology. As Forrester Research has put it, "Web services aren't a revolution rather, they're just as easier way to do distributed computing." Yet, therein lies their power by dramatically simplifying the way systems communicate, Web services make it far easier to exchange information and thus conduct business across the Internet.

An IBM tutorial defines Web services as "self-contained, self-describing modular applications that can be published, located and invoked across the Web. Web services perform functions, which can be anything from simple requests to complicated business processes."

What all this means, in practical terms, is that Web services can announce themselves across the Internet and expose their functionality to other applications ("I'm the Village Bank Online Loan Application Service"). In turn, they can search for and invoke other Web services, ("I want to check an applicant's credit rating with any one of my three credit bureau partners"), and they can interact with those services, exchanging requests for information ("Tell me John Smith's credit rating") and receiving responses ("A-plus").

1.2 E-Business Today

During the last several years, enterprises have made great progress in using the Web to achieve their strategic aims Streamlining processes, driving down costs, and delivering innovative offerings to the marketplace. Despite this evident success, the most visionary aspects of e-business remain elusive. (Remember how industry exchanges were going to revolutionize the way trading partners do business with one another?)

The barriers to e-business growth could be summed up in two words: integration and security. [Ref 5]

• In moving their business processes to the Web, enterprises have learned a hard lesson. With current technologies, the task of integrating disparate resources whether to achieve efficiencies or create new services is more difficult and costly than was originally anticipated. As a result, the vast majority of e-business interactions are still fairly straightforward, involving individual users interacting with discrete applications, via Web browsers. Where application-to-application data exchange is essential, for example, in EDI or supply chain environments those capabilities can only be created through custom integration efforts, which cannot be easily leveraged for multiple purposes and therefore are more difficult to cost-justify.

• Persistent and well-founded concerns about security continue to undermine trust in e-business, among consumers and enterprises alike. Again, due to the limits of current technology, it is nearly impossible for an organization to consistently implement best security practices and enforce security policies across the extended enterprise. The result: security vulnerabilities that are readily exploited by intruders and insiders, further eroding trust in e-business.

Given these factors, business and IT executives have come to realize that it may be years before enterprises achieve the most ambitions of e-business: routinely conducting complex transactions in a secure and trusted environment, with dramatically streamlined business processes and the bare minimum of human intervention.

They also realize that Web services and Web services security will play a key role in transforming this vision into reality.

1.3 Web Services Capabilities

Like existing Web capabilities, consumer-oriented Web services will be browser-driven, as in the loan application example given previously. However, over time, the bulk of Web services traffic will be generated by enterprises and will consist of pure machine-to-

machine interactions that carry out routine business processes with no human intervention.

Take, for example, a Web services-enabled manufacturing environment, where the production management system has increased the quota on an automated production line. This action triggers a query to the inventory management system to determine whether inventory levels are sufficient for the planned production run. Based on the inputs provided by the requesting application, the inventory management system calculates the types and quantities of materials that need to be reordered and notifies the production management system accordingly. In turn, the system initiates one or more purchase requests to the appropriate suppliers' order entry systems all without any human involvement.

1.4 The Web Services Value Proposition

With new development platforms - Web services mechanisms, such as XML formatting and SOAP messaging, are an organic part of the application development environment. This makes it faster, easier and more cost-effective to integrate heterogeneous resources, both within enterprises and between trading partners. In turn, by reducing integration costs, enterprises can undertake a wider range of integration efforts, including initiatives that could not previously be cost-justified.

A Web services application can be designed to aggregate information and services from multiple back-end systems. This makes it possible to perform routine tasks more efficiently, for example, accessing data from multiple systems in order to calculate a business unit's profit and loss.

Because Web services are self-contained and modular, they can be reused for multiple purposes. For example, an enterprise's shipping status service can be invoked by the sales order system and the customer care system.

Support for Federated Identity Management:

Web services play a key role in solutions for federated identity management. A federated approach to identity management enables enterprises to enhance the user experience by providing single sign-on (SSO) and seamless navigation across multiple domains. The business benefits of Web services are well understood. These include faster time-to-market for new applications and services, reduced business process costs and reduced partnering costs. Gartner Research predicts that the delivery of software through a subscription-based hosting model will eventually eliminate the need for most software to be licensed and sold in physical packaging.

In turn, this will greatly reduce IT costs related to software distribution and maintenance. In addition to these gains, enterprises will implement new business models and innovative offerings that attract new customers, enhance revenue and provide a competitive advantage.

1.5 Web Services and Network Computing

There is a strong momentum on several fronts to bring Web services into the mainstream of network computing. There are many proposals that have been put forth and are in Various stages of review by two key standards bodies: the World Wide Web Consortium (W3C) and the Organization for the Advancement of Structured Information Standards (OASIS).

Many of these proposals address the requirements of specific industries. In fact, the outpouring has been so great that W3C is looking at mechanisms for determining which of these proposals are truly foundational and which are more suitably developed by communities of interest. Web services have also inspired a high level of cooperation among technology vendors. For example, Microsoft and IBM have developed a proposal for a Global XML Web Services Architecture (GXA).

Support for Web services is further reflected in vendors' product strategies. Virtually all major hardware and software vendors have committed to introducing Web services Technology into their offerings.

While enterprises are carefully weighing all their technology investments, the pace of Web services implementations is beginning to gain speed. As Forrester Research notes, "early adopters", such as Ford are using Web services today to solve problems that traditional EAI and B2Bi products can't tackle cost-effectively: lower-volume integration between departmental apps and automated exchanges with small suppliers.

1.6 Familiar Challenges with Some New Twists

Despite the rapid progress being made in developing Web services technology, security remains a pressing concern. Security is the number one Inhibitor of Web services adoption. Perhaps, this should come as no surprise, given that many enterprises have only made modest progress in securing their existing e-business environments.

In many respects, the security challenges posed by Web services are similar to those presented by the first generation of Web technology. At the highest level, the objective is the same: to create a convenient and trusted e-business environment, where enterprises and individuals can conduct communications, transactions and other business processes.

The underlying challenges are also the same:

- How do you scale security to protect a widening set of resources against increasing points of vulnerability.

- How do you manage user identities and verify who is on the other end of a network connection.

- How do you exert fine-grained control over user access to sensitive resources.

- How do you ensure the confidentiality and integrity of transactions and communications.

In addressing these familiar challenges of distributed computing, there are major new twists, related to the ever-increasing scale and complexity of e-business.

1.7 Scalability and Control of Security Capabilities

Today, most large enterprises support dozens of major applications, each with its own unique security implementation. The familiar result: multiple schemes for authenticating users and managing access to Web applications. Inefficient and costly, these "islands of security"

Also increase risk by forcing organizations to apply security policies in a patchwork manner. In the future, as Web services gain wide acceptance adding hundreds or even thousands more discrete services to the enterprise IT environment this already unwieldy approach will be unsustainable.

In addition, the distributed nature of Web services makes it more difficult than ever to enforce security policies. As research points out, "With CORBA and DCE, IT Bosses could ensure security by exercising control: Only a small set of security-savvy developers could touch the code.

But with [new tools] novices are easily building and deploying Web services interfaces to critical data and unknowingly exposing their firms to security risks. Though it won't happen tomorrow, Web services will eventually make it possible to address these challenges by creating a unified security infrastructure that functions much like a utility, delivering security services to be consumed by other Web services across the enterprise and between trading partners. As a result, proven capabilities such as authentication, authorization and encryption services will be applied in new ways to protect a much wider set of resources than is possible today.

1.7.1 Authentication and Authorization

In this domain, there are several challenges that need to be addressed.

• In today's enterprise environments, the existence of multiple authentication schemes which typically result in multiple identities for one individual create major interoperability problems while also thwarting efforts to centrally manage user identities.

• Current authentication and authorization solutions focus largely on human-machine interactions. The growth of machine-to-machine interactions requires mechanisms for Web services to establish trust with each other, with support for multiple methods, including public key, SAML and perhaps Kerberos-based authentication and authorization. [Ref 1]

• The demand for federated identity and single sign-on (SSO) solutions creates an additional set of challenges.

1.7.2 Confidentiality and Integrity

Current technologies, most notably cryptography and PKI, offer well-developed mechanisms for protecting whole documents as they cross public and private networks. In addition to safeguarding confidentiality through encryption, these solutions enable recipients to authenticate the sender, via digital signatures, and verify text integrity to ensure a document has not been tampered with. [Ref 1]

Because Web services can easily aggregate information from multiple sources, they present more complex security challenges. For example, a transaction may require that digital signatures and encryption to be applied by multiple parties to different informational components within a service and at different points within a multi-tier system.

1.8 Security Mechanisms for Web Services

With the first generation of e-business technologies, many enterprises made the decision to "implement now and secure later." Wiser now, enterprises realize that addressing fundamental security challenges is one of the prerequisites for the long-term success of Web services and e-business.

For this reason, they are urging vendors to accelerate the development of security standards into the Web services framework and vendors are responding. As of mid-year 2002, approximately 30 key standards relating to security and reliability were in various stages of development, review and approval. In addition, major vendors have pledged support for open security standards and promised to build security into their mainstream products. Some of these core security standards are summarized below.

1.9 Reduce Risk by Providing Network Security

Enterprises that are considering a Web services implementation need to make security an integral part of their planning efforts. One way to reduce the risk and complexity of deploying this new technology is to start with an application that can be easily secured using existing capabilities. For example, point-to-point interactions between two companies can be authenticated and protected using VPN or SSL sessions.

From a security perspective, the next level of complexity involves three-way communication in a peered environment. An example would be a portal or business exchange, where data needs to be passed from one trading partner to another, through a third party. Because information crosses multiple security domains, point-to-point security is inadequate; beyond the first "hop," information will be exposed. At a minimum, an enterprise will want to implement WS-Security to ensure the privacy and integrity of communications as they cross multiple nodes.

With experience gained from these types of deployments, an enterprise will be prepared to address more sophisticated security challenges. For example, with a SAML-based security application, a company can readily transfer information from its diverse authentication systems to its Web access management system, facilitating centralized management of user access privileges. Similarly, a SAML-based system enables trading partners to federate user identity and authentication information for the purpose of providing single sign-on.

1.10 Work with a Trusted Provider

Another way to help ensure the success of a Web services implementation is to work with a trusted provider: one who combines a proven track record for implementing e-business security solutions with a strong knowledge of emerging Web services security technologies. RSA Security is uniquely qualified in this respect. RSA Security is the most trusted name in e-security. The company offers a comprehensive and well-proven set of solutions including authentication, Web access management and developer solutions for organizations that need to create trusted business processes in traditional e-business environments.

RSA Security is moving quickly to make these same capabilities available in the Web services environment. For example, RSA Company has been a significant contributor in the development of security standards for Web services, most notably, the SAML specification for exchanging authentication and authorization information. To speed the adoption of SAML, the company has made two of its patents available on a non-exclusive, royalty-free basis. Further, RSA Security has pledged its support for the WS-Security specification.

As these and other standards for Web services security evolve, RSA Security is fully committed to incorporating them into the company's offerings. In turn, enterprises can create an environment of authenticated, private and legally binding electronic transactions and communications in the new Web services world.

Further, through its involvement in the Liberty Alliance, RSA Security is helping to facilitate the practical application of Web services security capabilities in commercial environments.

The company provides the Alliance with expertise in the critical areas of user authentication, access rights management, data privacy and transaction integrity.

1.11 Summary

Many people feel the Internet is not secure because it is a public network. It is true that connecting an internal network to the Internet can expose the internal systems of a company to very considerable risks. The transmission of a document between two parties over the Internet may result in the document passing through half a dozen networks, each managed by different organizations.

In order to address security properly, it is necessary to build it into the integration solution from the beginning. Security issues must be considered seriously whenever information is sent or received from enterprise systems and when interfaces to the systems are built.

Security concerns can be divided into concerns about access control, and concerns about information and transaction security. Access control mechanisms, such as password protection, encrypted smart cards, biometrics, and firewalls, ensure that only valid users and applications get access to information resources such as user accounts, files and databases.

Information and transaction security schemes such as secret key encryption and public key encryption are used to ensure the privacy, integrity and confidentiality of business Transactions and messages.

CHAPTER TWO
SECURE SOCKETS LAYERS AND TRANSPORT LAYER SECURITY

2.1 Overview

Understanding SSL requires an understanding of cryptographic algorithms, message digest functions (also known as one-way or hash functions), and digital signatures.
SSL is a protocol using different cryptographic algorithms to implement security using authentication with certificates, session key exchange algorithms, encryption and integrity check. It is a common protocol, often used to ensure that the communication between WWW-server and WWW-client is secure.

SSL v3.0 is the de facto standard and the Internet Engineering Task Force, IETF, has a group working to develop SSL v3.0 further and the work formed the basis for the IETF's TLS standard. This working group is named Transport Layer Security (TLS). SSL v3.0 has been developed by Netscape with opinions from several industrial companies and developers.

2.2 Cryptographic Algorithms

Suppose Alice wants to send a private message to her bank to transfer some money. Alice would like the message to be private, since it will include information such as her bank account number and transfer amount. One solution is to use a cryptographic algorithm, a technique that would transform her message into an encrypted form, unreadable except by those it is intended for. Once in this form, the message may only be interpreted through the use of a secret key. Without the key the message is useless: good cryptographic algorithms make it so difficult for intruders to decode the original text that it isn't worth their effort.

There are two categories of cryptographic algorithms: conventional and public key.

2.2.1 Conventional Cryptography

Conventional cryptography, also known as symmetric cryptography, requires the sender and receiver to share a key: a secret piece of information that may be used to encrypt or decrypt a message. If this key is secret, then nobody other than the sender or receiver may read the message. If Alice and the bank know a secret key, then they may send each other private messages. The task of privately choosing a key before communicating, however, can be problematic.

2.2.2 Public Key Cryptography

Public key cryptography, also known as asymmetric cryptography, solves the key exchange problem by defining an algorithm which uses two keys, each of which may be used to encrypt a message. If one key is used to encrypt a message then the other must be user to decrypt it. This makes it possible to receive secure messages by simply publishing one key (the public key) and keeping the other secret (the private key). Anyone may encrypt a message using the public key, but only the owner of the private key will be able to read it. In this way, Alice may send private messages to the owner of a key-pair (the bank), by encrypting it using their public key. Only the bank will be able to decrypt it.

2.3 Certificates

Although Alice could have sent a private message to the bank, signed it and ensured the integrity of the message, she still needs to be sure that she is really communicating with the bank. This means that she needs to be sure that the public key she is using corresponds to the bank's private key. Similarly, the bank also needs to verify that the message signature really corresponds to Alice's signature.

If each party has a certificate which validates the other's identity, confirms the public key, and is signed by a trusted agency, then they both will be assured that they are communicating with whom they think they are. Such a trusted agency is called a Certificate Authority (CA), and certificates are used for authentication.

2.4 Public Key Infrastructure (PKI)

A PKI (public key infrastructure) enables users of a basically unsecured public network such as the Internet to securely and privately exchange data and money through the use of a public and a private cryptographic key pair that is obtained and shared through a trusted authority. The public key infrastructure provides for a digital certificate that can identify an individual or an organization and directory services that can store and, when necessary, revoke the certificates. Although the components of a PKI are generally understood, a number of different vendor approaches and services are emerging. Meanwhile, an Internet standard for PKI is being worked on.

The public key infrastructure assumes the use of public key cryptography, which is the most common method on the Internet for authenticating a message sender or encrypting a message. Traditional cryptography has usually involved the creation and sharing of a secret key for the encryption and decryption of messages. This secret or private key system has the significant flaw that if the key is discovered or intercepted by someone else, messages can easily be decrypted. For this reason, public key cryptography and the public key infrastructure is the preferred approach on the Internet. (The private key system is sometimes known as symmetric cryptography and the public key system as asymmetric cryptography).

A public key infrastructure consists of:

1. A certificate authority (CA) that issues and verifies digital certificate. A certificate includes the public key or information about the public key

2. A registration authority (RA) that acts as the verifier for the certificate authority before

3. Digital certificate is issued to a requestor

4. One or more directories where the certificates (with their public keys) are held

5. A certificate management system

2.4.1 Who Provides the Infrastructure

A number of products are offered that enable a company or group of companies to implement a PKI. The acceleration of e-commerce and business-to-business commerce over the Internet has increased the demand for PKI solutions. Related ideas are the virtual private network (VPN) and the IP Security (IPsec) standard. Among PKI leaders are:

1. RSA, which has developed the main algorithms used by PKI vendors

2. Verisign, which acts as a certificate authority and sells software that allows a company to create its own certificate authorities

3. GTE CyberTrust, which provides a PKI implementation methodology and consultation service that it plans to vend to other companies for a fixed Price

4. Xcert, whose Web Sentry product that checks the revocation status of certificates on a server, using the Online Certificate Status Protocol (OCSP)

5. Netscape, whose Directory Server product is said to support 50 million objects and process 5,000 queries a second; Secure E-Commerce, which allows a company or extranet manager to manage digital certificates; and Meta-Directory, which can connect all corporate directories into a single directory for security management.

2.5 Certificate Contents

A certificate associates a public key with the real identity of an individual, server, or other entity, known as the subject. As shown in the table below, information about the subject includes identifying information (the distinguished name), and the public key. It also includes the identification and signature of the Certificate Authority that issued the certificate, and the period of time during which the certificate is valid. It may have

additional information (or extensions) as well as administrative information for the Certificate Authority's use, such as a serial number.

Subject:	Distinguished Name, Public Key
Issuer:	Distinguished Name, Signature
Period of Validity:	Not Before Date, Not After Date
Administrative Information:	Version, Serial Number
Extended Information:	Basic Constraints, Netscape Flags, etc.

Table1: Certificate Authority

2.5.1 Distinguished Name

A distinguished name is used to provide an identity in a specific context - for instance, an individual might have a personal certificate as well as one for their identity as an employee. Distinguished names are defined by the X.509 standard, which defines the fields, field names, and abbreviations used to refer to the fields (see below).

DN Field	Abbreviation	Description	Example
Common Name	CN	Name being certified	CN=Joe Average
Organization or Company	O	Name is associated with this organization	O=University of Bern
Organizational Unit	OU	Name is associated with this organizational unit, such as a department	OU=Institute of Computer Science
City/Locality	L	Name is located in this city	L=Bern
State/Province	ST	Name is located in this state/province	ST=Bern
Country	C	Name is located in this country	C=CH

		(ISO code)	

Table2: Distinguished Name

A Certificate Authority may define a policy specifying which distinguished field names are optional, and which are required. It may also place requirements upon the field contents, as may users of certificates. As an example, a Netscape browser requires that the Common Name for a certificate representing a server has a name which matches a wildcard pattern for the domain name of that server, such as *.unibe.ch.

2.5.2 ASN.1 notation, BER

The binary format of a certificate is defined using the ASN.1 notation. This notation defines how to specify the contents, and encoding rules define how this information is translated into binary form. The binary encoding of the certificate is defined using Distinguished Encoding Rules (DER), which are based on the more general Basic Encoding Rules (BER). For those transmissions which cannot handle binary, the binary form may be translated into an ASCII form by using Base64 encoding (MIME

2.6 Certificate Authorities

By first verifying the information in a certificate request before granting the certificate, the Certificate Authority assures the identity of the private key owner of a key-pair. For instance, if Alice requests a personal certificate, the Certificate Authority must first make sure that Alice really is the person the certificate request claims.

2.6.1 Certificate Chains

A Certificate Authority may also issue a certificate for another Certificate Authority. When examining a certificate, Alice may need to examine the certificate of the issuer, for each parent Certificate Authority, until reaching one which she has confidence in. She may decide to trust only certificates with a limited chain of issuers, to reduce her risk of a "bad" certificate in the chain.

2.6.2 Creating a Root-Level CA

As noted earlier, each certificate requires an issuer to assert the validity of the identity of the certificate subject, up to the top-level Certificate Authority (CA). This presents a problem: Since this is who vouches for the certificate of the top-level authority, which has no issuer? In this unique case, the certificate is "self-signed", so the issuer of the certificate is the same as the subject. As a result, one must exercise extra care in trusting a self-signed certificate. The wide publication of a public key by the root authority reduces the risk in trusting this key - it would be obvious if someone else publicized a key claiming to be the authority. Browsers are preconfigured to trust well-known certificate authorities.

It is also possible to create your own Certificate Authority. Although risky in the Internet environment, it may be useful within an Intranet where the organization can easily verify the identities of individuals and servers.

2.6.3 Certificate Management

Establishing a Certificate Authority is a responsibility which requires a solid administrative, technical, and management framework. Certificate Authorities not only issue certificates, they also manage them - that is, they determine how long certificates are valid, they renew them, and they keep lists of certificates that have already been issued but are no longer valid (Certificate Revocation Lists, or CRLs). Say Alice is entitled to a certificate as an employee of a company. Say too, that the certificate needs to be revoked when Alice leaves the company. Since certificates are objects that get passed around, it is impossible to tell from the certificate alone that it has been revoked. When examining certificates for validity, therefore, it is necessary to contact the issuing Certificate Authority to check CRLs - this is not usually an automated part of the process.

2.7 SSL Protocol

SSL is placed on top of the TCP/IP layer and below the application layer; this layer is sometimes called secure transport layer. To set up a SSL connection a TCP/IP connection must be established first as SSL uses the primitives of TCP/IP. The SSL connection can

be seen as a secure channel within the TCP/IP connection in which all traffic between the application peers goes encrypted.

Application (e.g., HTTP)
Secure transport layer (SSL/TLS)
TCP
IP
Subnet

Table3: SSL Protocol

These capabilities address fundamental concerns about communication over the Internet and other TCP/IP networks:

- **SSL server authentication** allows a user to confirm a server's identity. SSL- enabled client software can use standard techniques of public-key cryptography to check that a server's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the client's list of trusted CAs. This confirmation might be important if the user, for example, is sending a credit card number over the network and wants to check the receiving server's identity.

- **SSL client authentication** allows a server to confirm a user's identity. Using the same techniques as those used for server authentication, SSL-enabled server software can check that a client's certificate and public ID are valid and have been issued by a certificate authority (CA) listed in the server's list of trusted CAs. This confirmation might be important if the server, for example, is a bank sending confidential financial information to a customer and wants to check the recipient's identity.

- **An encrypted SSL connection** requires all information sent between a client and a server to be encrypted by the sending software and decrypted by the receiving software,

thus providing a high degree of confidentiality. Confidentiality is important for both parties to any private transaction. In addition, all data sent over an encrypted SSL connection is protected with a mechanism for detecting tampering - that is, for automatically determining whether the data has been altered in transit. All the calls from the application layer to the TCP layer are replaced with calls to the SSL layer, and the SSL layer will take care of the communication with the TCP layer. SSL itself is actually divided again into two different layers as showed below.

Application layer		
Change Cipher Specification Protocol	Alert Protocol	Handshake protocol
SSL record layer		
TCP layer		
IP layer		

Table4: SSL is divided into two different layers

The SSL record protocol defines the format used to transmit data. The SSL handshake protocol involves using the SSL record protocol to exchange a series of messages between an SSL-enabled server and an SSL- enabled client when they first establish an SSL connection. This exchange of messages is designed to facilitate the following actions:

- Authenticate the server to the client.
- Allow the client and server to select the cryptographic algorithms, or ciphers, that they both support.
- Optionally authenticate the client to the server.
- Use public-key encryption techniques to generate shared secrets.

- Establish an encrypted SSL connection.

In the SSL Record layer the messages are divided/combined to fit within the payload of SSL records, a MAC is created and then the whole record layer message is encrypted with the secret key algorithm agreed upon. The substance of the SSL records will be: content type, protocol version number, length, data payload, message authentication code (MAC, provides for data integrity). The messages are the sent to the other peer through a TCP/IP connection.

SSL record format:

Record Type	Version MSB	Version LSB	Length MSB	Length LSB
Ciphertext (*length* bytes)				
MAC (0, 16 or 20 bytes)				

Table5; SSL record format

- Implicit sequence number, cipher-state

- Types: handshake, alert (close), data, rekey

2.8 Change Cipher Specification Protocol

The change cipher specification protocol exists to signal transitions in ciphering strategies. The protocol consists of a single message, which is encrypted and compressed under the current (not the pending) CiperSpec. The message consists of a single byte of value 1. The change cipher specification message is sent by both the client and server to notify the receiving party that subsequent records will be protected under the just-negotiated CipherSpec and keys.

2.9 Alert Protocol

One of the content types supported by the SSL Record layer is the alert type. Alert message convey the severity of the message and a description of the alert. Alert messages

with a level of "fatal" result in the immediate termination of the connection. In this case, other connection corresponding to the session may continue, but the session identifier must be invalidated, preventing the failed session from being used to establish new connections. Like other messages, alert messages are encrypted and compressed, as specified by the current connection state.

2.10 Ciphers Used with SSL

The SSL protocol supports the use of a variety of different cryptographic algorithms, or ciphers, for use in operations such as authenticating the server and client to each other, transmitting certificates, and establishing session keys. Clients and servers may support different cipher suites, or sets of ciphers, depending on factors such as the version of SSL they support, company policies regarding acceptable encryption strength, and government restrictions on export of SSL-enabled software. Among its other functions, the SSL handshake protocol determines how the server and client negotiate which cipher suites they will use to authenticate each other, to transmit certificates, and to establish session keys.

The cipher suite descriptions that follow refer to these algorithms:

1. **DES**. Data Encryption Standard, an encryption algorithm used by the U.S. Government.

2. **DSA**. Digital Signature Algorithm, part of the digital authentication standard used by the U.S. Government.

3. **KEA**. Key Exchange Algorithm, an algorithm used for key exchange by the U.S. Government.

4. **MD5**. Message Digest algorithm developed by Rivest.

5. **RC2 and RC4**. Rivest encryption ciphers developed for RSA Data Security.

6. **RSA**. A public-key algorithm for both encryption and authentication. Developed by Rivest, Shamir, and Adleman.

7. **RSA key exchange**. A key-exchange algorithm for SSL based on the RSA algorithm.

8. **SHA-1**. Secure Hash Algorithm, a hash function used by the U.S. Government.

9. **SKIPJACK**. A classified symmetric-key algorithm implemented in FORTEZZA-compliant hardware used by the U.S. Government. (For more information, see FORTEZZA Cipher Suites.)

10. **Triple-DES**. DES applied three times.

Key-exchange algorithms like KEA and RSA key exchange govern the way in which the server and client determine the symmetric keys they will both use during an SSL session. The most commonly used SSL cipher suites use RSA key exchange. The SSL 2.0 and SSL 3.0 protocols support overlapping sets of cipher suites. Administrators can enable or disable any of the supported cipher suites for both clients and servers. When a particular client and server exchange information during the SSL handshake, they identify the strongest enabled cipher suites they have in common and use those for the SSL session.

Decisions about which cipher suites a particular organization decides to enable depend on trade-offs among the sensitivity of the data involved, the speed of the cipher, and the applicability of export rules.

organizations want to disable the weaker ciphers to prevent SSL connections with weaker encryption. However, due to government restrictions on products that support anything stronger than 40-bit encryption, disabling support for all 40-bit ciphers effectively restricts access to network browsers that are available only in the United States (unless the server involved has a special Global Server ID that permits the international client to "step up" to stronger encryption).

To serve the largest possible range of users, it's administrators may wish to enable as broad a range of SSL cipher suites as possible. That way, when a domestic client or server is dealing with another domestic server or client, respectively, it will negotiate the use of the strongest ciphers available. And when an domestic client or server is dealing with an international server or client, it will negotiate the use of those ciphers that are permitted under U.S. export regulations.

2.11 Cipher Suites with RSA Key Exchange

Unless otherwise indicated, all ciphers listed in the table are supported by both SSL 2.0 and SSL 3.0. Cipher suites are listed from strongest to weakest; for more information about the way encryption strength is measured, see Key Length and Encryption Strength. Cipher suites supported by the SSL protocol that use the RSA key-exchange algorithm

Strongest cipher suite: This cipher suite is appropriate for banks and other institutions that handle highly sensitive data.

- Triple DES, which supports 168-bit encryption, with SHA-1 message authentication. Triple DES is the strongest cipher supported by SSL, but it is not as fast as RC4. Triple DES uses a key three times as long as the key for standard DES. Because the key size is so large, there are more possible keys than for any other cipher - approximately 3.7 * 1050.

Strong cipher suites: These cipher suites support encryption that is strong enough for most business or government needs.

- RC4 with 128-bit encryption and MD5 message authentication. Because the RC4 and RC2 ciphers have 128-bit encryption, they are the second strongest next to Triple DES (Data Encryption Standard), with 168-bit encryption. RC4 and RC2 128-bit encryption permits approximately 3.4 * 1038 possible keys, making them very difficult to crack. RC4 ciphers are the fastest of the supported ciphers.

- RC2 with 128-bit encryption and MD5 message authentication. Because the RC4 and RC2 ciphers have 128-bit encryption, they are the second strongest next to Triple DES

(Data Encryption Standard), with 168-bit encryption. RC4 and RC2 128-bit encryption permits approximately 3.4 * 1038 possible keys, making them very difficult to crack. RC2 ciphers are slower than RC4 ciphers.
This cipher suite is supported by SSL 2.0 but not by SSL 3.0.

• DES, which supports 56-bit encryption, with SHA-1 message authentication. DES is stronger than 40-bit encryption, but not as strong as 128-bit encryption. DES 56-bit encryption permits approximately 7.2 * 1016 possible keys. Æ Both SSL 2.0 and SSL 3.0 support this cipher suite, except that SSL 2.0 uses MD5 rather than SHA-1 for message authentication.

Exportable cipher suites: These cipher suites are not as strong as those listed above, but may be exported to most countries (note that France permits them for SSL but not for S/MIME). They provide the strongest encryption available for exportable products. Note that for RC4 and RC2 ciphers, the phrase "40-bit encryption" means the keys are still 128 bits long, but only 40 bits have cryptographic significance.

• RC4 with 40-bit encryption and MD5 message authentication. RC4 40-bit encryption permits approximately 1.1 * 1012 (a trillion) possible keys. RC4 ciphers are the fastest of the supported ciphers.

• RC2 with 40-bit encryption and MD5 message authentication. RC2 40-bit encryption permits approximately 1.1 * 1012 (a trillion) possible keys. RC2 ciphers are slower than the RC4 ciphers.

Weakest cipher suite: This cipher suite provides authentication and tamper detection but no encryption. Server administrators must be careful about enabling it, however, because data sent using this cipher suite is not encrypted and may be accessed by eavesdroppers.

- No encryption, MD5 message authentication only. This cipher suite uses MD5 message authentication to detect tampering. It is typically supported in case a client and server have none of the other ciphers in common.

2.12 FORTEZZA Cipher Suites

The list below lists additional cipher suites supported by Netscape products with FORTEZZA for SSL 3.0. It provides a hardware implementation of two classified ciphers developed by the federal government: FORTEZZA KEA and SKIPJACK. FORTEZZA ciphers for SSL use the Key Exchange Algorithm (KEA) instead of the RSA key-exchange algorithm mentioned in the preceding section, and use FORTEZZA cards and DSA for client authentication. Strong FORTEZZA cipher suites: Permitted for deployments within the United States only. These cipher suites support encryption that is strong enough for most business or government needs.

- RC4 with 128-bit encryption and SHA-1 message authentication. Like RC4 with 128-bit encryption and MD5 message authentication, this cipher is one of the second strongest ciphers after Triple DES. It permits approximately 3.4 * 10^38 possible keys, making it very difficult to crack. This cipher suite is supported by SSL 3.0 but not by SSL 2.0.

- RC4 with SKIPJACK 80-bit encryption and SHA-1 message authentication. The SKIPJACK cipher is a classified symmetric-key cryptographic algorithm implemented in FORTEZZA-compliant hardware. Some SKIPJACK implementations support key escrow using the Law Enforcement Access Field (LEAF). The most recent implementations do not. This cipher suite is supported by SSL 3.0 but not by SSL 2.0. - Weakest FORTEZZA cipher suite: This cipher suite provides authentication and tamper detection but no encryption. Server administrators must be careful about enabling it, however, because data sent using this cipher suite is not encrypted and may be accessed by eavesdroppers.

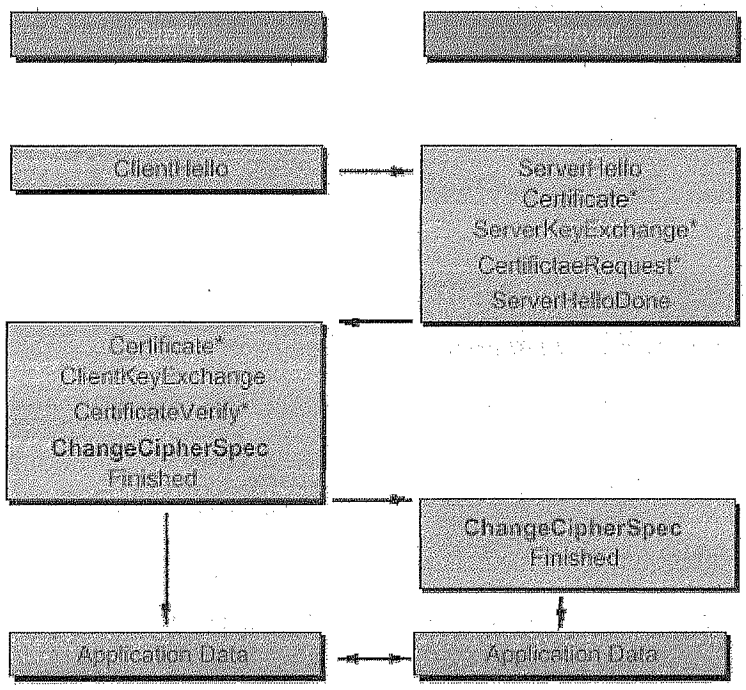
- No encryption, SHA-1 message authentication only. This cipher uses SHA-1 message authentication to detect tampering.

2.13 Handshake Protocol

The SSL protocol uses a combination of public-key and symmetric key encryption. Symmetric key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques. An SSL session always begins with an exchange of messages called the SSL handshake. The handshake allows the server to authenticate itself to the client using public-key techniques, then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows. Optionally, the handshake also allows the client to authenticate itself to the server. During the handshake there is a negotiation of what Cipher Suite to use during data transfer, establishment and sharing of the session key between the client and server and optionally authentication of the peers. The Cipher Suite consists of what methods to use for Key Exchange, Data Transfer and creation of Message Authentication Code, MAC.

1. The client sends the server the client's SSL version number, cipher settings, randomly generated data, and other information the server needs to communicate with the client using SSL.
2. The server sends the client the server's SSL version number, cipher settings, randomly generated data, and other information the client needs to communicate with the server over SSL. The server also sends its own certificate and, if the client is requesting a server resource that requires client authentication, requests the client's certificate.
3. The client uses some of the information sent by the server to authenticate the server (see Server Authentication for details). If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client goes on to Step 4.
4. Using all data generated in the handshake so far, the client (with the cooperation of the server, depending on the cipher being used) creates the premaster secret for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in Step 2), and sends the encrypted premaster secret to the server.

5. If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case the client sends both the signed data and the client's own certificate to the server along with the encrypted premaster secret.
6. If the server has requested client authentication, the server attempts to authenticate the client (see Client Authentication for details). If the client cannot be authenticated, the session is terminated. If the client can be successfully authenticated, the server uses its private key to decrypt the premaster secret, then performs a series of steps (which the client also performs, starting from the same premaster secret) to generate the master secret. After the master secret has been generated, the premaster secret will be immediately erased from memory.
7. Both the client and the server use the master secret to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity - that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection.
8. The client sends a message to the server informing it that future messages from the client will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.
9. The server sends a message to the client informing it that future messages from the server will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.
10. The SSL handshake is now complete, and the SSL session has begun. The client and the server use the session keys to encrypt and decrypt the data they send to each other and to validate its integrity.



(*)_Optional_messages

Fig1. Handshake Protocol [Ref 6]

It's important to note that both client and server authentication involve encrypting some piece of data with one key of a public-private key pair and decrypting it with the other key:

- In the case of server authentication, the client encrypts the premaster secret with the server's public key. Only the corresponding private key can correctly decrypt the secret, so the client has some assurance that the identity associated with the public key is in fact the server with which the client is connected. Otherwise, the server cannot decrypt the premaster secret and cannot generate the symmetric keys required for the session, and the session will be terminated.

- In the case of client authentication, the client encrypts some random data with the client's private key - that is, it creates a digital signature. The public key in the client's certificate can correctly validate the digital signature only if the corresponding private

key was used. Otherwise, the server cannot validate the digital signature and the session is terminated.

2.13.1 SSL Handshake Protocol in Deep [Ref 5]

Hello Request: The hello request message may be sent by the server at any time, but will be ignored by the client if the handshake protocol is already underway. It is a simple notification that the client should begin the negotiation process new by sending a client hello message when convenient.

Client Hello: When a client first connects to a server it is required to send the client hello as its first message. The client can also send a client hello response to a hello request or on its own initiative in order to renegotiate the security parameters in an existing connection. The structure of the client hello is as follows.

```
struct {
        ProtocolVersion client_version;
        Random random;
        SessionID session_id;
        CipherSuite cipher_suites<2..2^16-1>;
        CompressionMethod compression_methods<2..2^8-1>;
} ClientHello;
```

Server Hello: The server processes the client hello message and responds with either a handshake_failure alert or server hello message.

```
struct {
        ProtocolVersion server_version;
        Random random;
        SessionID session_id;
        CipherSuite cipher_suites<2..2^16-1>;
```

```
        CompressionMethod compression_methods<2..2^8-1>;
    } ServerHello;
```

Server Certificate: If the server is to be authenticated (which is generally the case), the server sends its certificate immediately following the server hello message. The certificate type must be appropriate for the selected cipher suite's key exchange algorithm, and is generally an X.509.v3 certificate (or a modified X.509 certificate in the case of FORTEZZA). The same message type will be used for the client's response to a certificate request message.

```
opaque ASN.1Cert<1..2^24-1>;

    struct {
        ASN.1Cert certificate_list<1..2^24-1>;
    } Certificate;
```

Server key exchange message: The server key exchange message is sent by the server if it has no certificate, has a certificate only used for signing, or FORTEZZA KEA key exchanges is used. This message is not used if the server certificate contains Difiie-Hellman parameters.

```
    struct {
        select (KeyExchangeAlgorithm) {
        case diffie_hellman:
        ServerDHParams params;
        Signature signed_params;
        case rsa:
        ServerRSAParams params;
        Signature signed_params;
        case fortezza_kea:
        ServerFortezzaParams params;
```

```
        };
    } ServerKeyExchange;
```

Certificate Request: A non-anonymous server can optionally request a certificate from the client, if appropriate for the selected cipher suite. If is fatal handshake failure alert for an anonymous server to request client identification.

```
struct {
        ClientCertificateType certificate_types<1..2^8-1>;
        DistinguishedName certificate_authorities<3..2^16-1>;
} CertificateRequest;
```

Server Hello done: The server hello done message is sent by the server to indicate the end of the server hello and associated messages. After sending this message the server will wait for a client response: struct { } ServerHelloDone;

Client Certificate: This is the first message the client can send after receiving a server hello done message. This message is only sent if the server requests a certificate. If no suitable certificate is available, the client should send a no_certificate alert instead. This alert is only a warning, however the server may respond with a fatal handshake failure alert if client authentication is required. The client certificate message is similar to the server certificate message above.

Client key exchange message: The choice of messages depends on which public key algorithm(s) has (have) been selected. See also Server key exchange message.

```
struct {
  select (KeyExchangeAlgorithm) {
    case rsa: EncryptedPreMasterSecret;
    case diffie_hellman: ClientDiffieHellmanPublic;
    case fortezza_kea: FortezzaKeys;
```

```
    } exchange_keys;
} ClientKeyExchange;
```

Certificate Verify: This message is used to provide explicit verification of a client certificate. This message is only sent following any client certificate that has signing capability.

```
struct {
    Signature signature;
} CertificateVerify;
```

Finished: A finished message is always sent immediately after a change cipher specs message to verify that the key exchange and authentication processes were successful. The finished message is the first protected with the just-negotiated algorithms, keys, and secrets. No acknowledgment of the finished message is required; parties may begin sending encrypted data immediately after sending the finished message. Recipients of finished messages must verify that the contents are correct.

```
struct {
    opaque md5_hash[16];
    opaque sha_hash[20];
} Finished;
```

2.14 Server Authentication

SSL always requires server authentication, or cryptographic validation by a client of the server's identity. As explained in Step 2 of The SSL Handshake, the server sends the client a certificate to authenticate itself. The client uses the certificate in Step 3 to authenticate the identity the certificate claims to represent. An SSL-enabled client goes through these steps to authenticate a server's identity:

1. **Is today's date within the validity period?** The client checks the server certificate's validity period. If the current date and time are outside of that range, the

authentication process won't go any further. If the current date and time are within the certificate's validity period, the client goes on to Step 2.

2. **Is the issuing CA a trusted CA?** Each SSL-enabled client maintains a list of trusted CA certificates. This list determines which server certificates the client will accept. If the distinguished name (DN) of the issuing CA matches the DN of a CA on the client's list of trusted CAs, the answer to this question is yes, and the client goes on to Step 3. If the issuing CA is not on the list, the server will not be authenticated unless the client can verify a certificate chain ending in a CA that is on the list.
3. **Does the issuing CA's public key validate the issuer's digital signature?** The client uses the public key from the CA's certificate (which it found in its list of trusted CAs in step 2) to validate the CA's digital signature on the server certificate being presented. If the information in the server certificate has changed since it was signed by the CA or if the CA certificate's public key doesn't correspond to the private key used by the CA to sign the server certificate, the client won't authenticate the server's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the client has determined that the server certificate is valid. It is the client's responsibility to take Step 4 before Step 5.
4. **Does the domain name in the server's certificate match the domain name of the server itself?** This step confirms that the server is actually located at the same network address specified by the domain name in the server certificate. Although step 4 is not technically part of the SSL protocol, it provides the only protection against a form of security attack known as a Man-in-the-Middle attack. Clients must perform this step and must refuse to authenticate the server or establish a connection if the domain names don't match. If the server's actual domain name matches the domain name in the server certificate, the client goes on to Step 5.
5. The server is authenticated. The client proceeds with the SSL handshake. If the client doesn't get to step 5 for any reason, the server identified by the certificate cannot be authenticated, and the user will be warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server requires client authentication, the server performs the steps described in Client Authentication.

6. After the steps described here, the server must successfully use its private key to decrypt the premaster secret the client sends in Step 4 of The SSL Handshake. Otherwise, the SSL session will be terminated. This provides additional assurance that the identity associated with the public key in the server's certificate is in fact the server with which the client is connected.

2.14.1 Man-in-the-Middle Attack

As suggested in Step 4 above, the client application must check the server domain name specified in the server certificate against the actual domain name of the server with which the client is attempting to communicate. This step is necessary to protect against a man-in the middle attack, which works as follows.

The "man in the middle" is a rogue program that intercepts all communication between the client and a server with which the client is attempting to communicate via SSL. The rogue program intercepts the legitimate keys that are passed back and forth during the SSL handshake, substitutes its own, and makes it appear to the client that it is the server, and to the server that it is the client.

The encrypted information exchanged at the beginning of the SSL handshake is actually encrypted with the rogue program's public key or private key, rather than the client's or server's real keys. The rogue program ends up establishing one set of session keys for use with the real server, and a different set of session keys for use with the client. This allows the rogue program not only to read all the data that flows between the client and the real server, but also to change the data without being detected. Therefore, it is extremely important for the client to check that the domain name in the server certificate corresponds to the domain name of the server with which a client is attempting to communicate - in addition to checking the validity of the certificate by performing the other steps described in Server Authentication.

2.15 Client Authentication

SSL-enabled servers can be configured to require client authentication, or cryptographic validation by the server of the client's identity. When a server configured this way requests client authentication (see Step 6 of The SSL Handshake), the client sends the server both a certificate and a separate piece of digitally signed data to authenticate itself. The server uses the digitally signed data to validate the public key in the certificate and to authenticate the identity the certificate claims to represent.

The SSL protocol requires the client to create a digital signature by creating a one- way hash from data generated randomly during the handshake and known only to the client and server. The hash of the data is then encrypted with the private key that corresponds to the public key in the certificate being presented to the server.

An SSL-enabled server goes through these steps to authenticate a user's identity:

1. **Does the user's public key validate the user's digital signature?** The server checks that the user's digital signature can be validated with the public key in the certificate. If so, the server has established that the public key asserted to belong to John Doe matches the private key used to create the signature and that the data has not been tampered with since it was signed. At this point, however, the binding between the public key and the DN specified in the certificate has not yet been established. The certificate might have been created by someone attempting to impersonate the user. To validate the binding between the public key and the DN, the server must also complete Step 3 and Step 4.
2. **Is today's date within the validity period?** The server checks the certificate's validity period. If the current date and time are outside of that range, the authentication process won't go any further. If the current date and time are within the certificate's validity period, the server goes on to Step 3.
3. **Is the issuing CA a trusted CA?** Each SSL-enabled server maintains a list of trusted CA certificates. This list determines which certificates the server will accept. If the DN of the issuing CA matches the DN of a CA on the server's list of trusted CAs, the answer to this question is yes, and the server goes on to Step 4. If the issuing CA is

not on the list, the client will not be authenticated unless the server can verify a certificate chain ending in a CA that is on the list (see CA Hierarchies for details). Administrators can control which certificates are trusted or not trusted within their organizations by controlling the lists of CA certificates maintained by clients and servers.

4. **Does the issuing CA's public key validate the issuer's digital signature?** The server uses the public key from the CA's certificate (which it found in its list of trusted CAs in Step 3) to validate the CA's digital signature on the certificate being presented. If the information in the certificate has changed since it was signed by the CA or if the public key in the CA certificate doesn't correspond to the private key used by the CA to sign the certificate, the server won't authenticate the user's identity. If the CA's digital signature can be validated, the server treats the user's certificate as a valid "letter of introduction" from that CA and proceeds. At this point, the SSL protocol allows the server to consider the client authenticated and proceed with the connection as described in Step 6. Netscape servers may optionally be configured to take Step 5 before Step 6.
5. **Is the user's certificate listed in the LDAP entry for the user?** This optional step provides one way for a system administrator to revoke a user's certificate even if it passes the tests in all the other steps. The Netscape Certificate Server can automatically remove a revoked certificate from the user's entry in the LDAP directory. All servers that are set up to perform this step will then refuse to authenticate that certificate or establish a connection. If the user's certificate in the directory is identical to the user's certificate presented in the SSL handshake, the server goes on to step 6.
6. **Is the authenticated client authorized to access the requested resources?** The server checks what resources the client is permitted to access according to the server's access control lists (ACLs) and establishes a connection with appropriate access. If the server doesn't get to step 6 for any reason, the user identified by the certificate cannot be authenticated, and the user is not allowed to access any server resources that require authentication.

2.16 Vulnerabilities

SSL v3.0 fixes vulnerabilities in v2.0: The SSLv2 protocol has two known flaws. The downgrade attack allows an active attacker (one who can change bits in the messages between client and server) to force an SSLv2 session to use weaker cipher suites than would ordinarily be negotiated. Since the SSLv2 handshake packets between the client and server are not integrity checked, there's no way for the server to tell if the cipher suite list has been modified. Once the attacker has forced the SSLv2 session into a weak cipher suite, the attacker can use brute-force techniques to crack the small key. The truncation attack allows an attacker to stop the SSLv2 session without the server or client knowing that the session was stopped by an attacker instead of by the other party. If the attacker knows something about the structure of the message and how it is sent in the SSLv2 packets, he can use the truncation attack to change the meaning of the message. For example, Alice the client is sending an order to buy stock to Bob the stock broker's secure server. Mark the attacker can change bits in the communications between Alice and Bob, and Mark wants Alice to lose money. He knows that Alice is sending a buy order, and he knows the format that the buy order will take (simple to figure out, he only has to look as the HTML for the form on Bob's web site). Alice sends "buy 10,000 shares of PBGX". Mark truncates her message to "buy 10,000 shares of PBG". Not being able to tell that Alice's message was truncated, Bob will execute the order to buy 10,000 shares of PBG. If Alice is unlucky, the price of PBG will be much greater than that of PBGX, and Alice will have to go on margin to cover it. At the least, she'll be out the commission and the opportunity cost of not buying PBGX when she wanted to do so.

2.17 Transport Layer Security (TLS)

Internet commerce requires secure communications. To order goods, a customer typically sends credit card details. To order life insurance, the customer might have to supply confidential personal data. Internet users would like to know that such information is safe from eavesdropping or alteration.

Many Web browsers protect transmissions using the protocol SSL (Secure Sockets Layer). The client and server machines exchange nonces and compute session keys from

them. The latest version of the protocol is called TLS (Transport Layer Security) it closely resembles SSL 3.0. Is TLS really secured? My proofs suggest that it is, but one should draw no conclusions without reading the rest of this paper, which describes how the protocol was modeled and what properties were proved. I have analyzed a much simplified form of TLS; I assume hashing and encryption to be secure.

2.17.1 Overview of TLS

TLS is the successor to the Secure Sockets Layer, and is nearly identical to SSL except that it implements

- an open, standards-based solution,
- more non-proprietary ciphers,
- better error reporting, and
- HMAC digests instead of MD5.

TLS and SSL are not interoperable. The TLS protocol does contain a mechanism that allows TLS implementation to back down to SSL. The most recent browser versions support TLS. The TLS Working Group, continues to work on the TLS protocol and related applications.

Internet browsers use security protocols to protect confidential messages. An inductive analysis of TLS, has been performed using the theorem prove Isabelle. Proofs are based on higher-order logic and make no assumptions concerning beliefs or finiteness. All the obvious security goals can be proved; session resumption appears to be secure even if old session keys have been compromised. The proofs suggest minor changes to simplify the analysis.

TLS, even at an abstract level, is much more complicated than most protocols that researchers have verified. Session keys are negotiated rather than distributed, and the protocol has many optional parts. Nevertheless, the resources needed to verify TLS are modest: six man-weeks of effort and three minutes of processor time.

A TLS handshake involves a client, such as a World Wide Web browser, and a Web server. Below, I refer to the client as A ('Alice') and the server as B ('Bob'), as is customary for authentication protocols, especially since C and S often have dedicated meanings in the literature.

At the start of a handshake, A contacts B, supplying a session identifier and nonce. In response, B sends another nonce and his public-key certificate (my model omits other possibilities). Then A generates a *pre-master-secret*, a 48-byte random string, and sends it to B encrypted with his public key. A optionally sends a signed message to authenticate herself. Now, both parties calculate the *master-secret* M from the nonces and the pre-master-secret, using a secure pseudo-random-number function (PRF). They calculate session keys from the nonces and master-secret.

Each session involves a pair of symmetric keys; A encrypts using one and B encrypts using the other. Before sending application data, both parties exchange finished messages to confirm all details of the handshake and to check that clear text parts of messages have not been altered.

A full handshake is not always necessary. At some later time, A can resume a session by quoting an old session identifier along with a fresh nonce. If B is willing to resume the designated session, then he replies with a fresh nonce. Both parties compute fresh session keys from these nonces and the stored master-secret, M. Both sides confirm this shorter run using finished messages.

TLS is highly complex. this version leaves out many details for the sake of simplicity:
1. Record formats, field widths, cryptographic algorithms, etc. are irrelevant in an abstract analysis.
2. Alert and failure messages are unnecessary because bad sessions can simply be abandoned.
3. The server key exchange message allows anonymous sessions among other things, but it is not an essential part of the protocol.

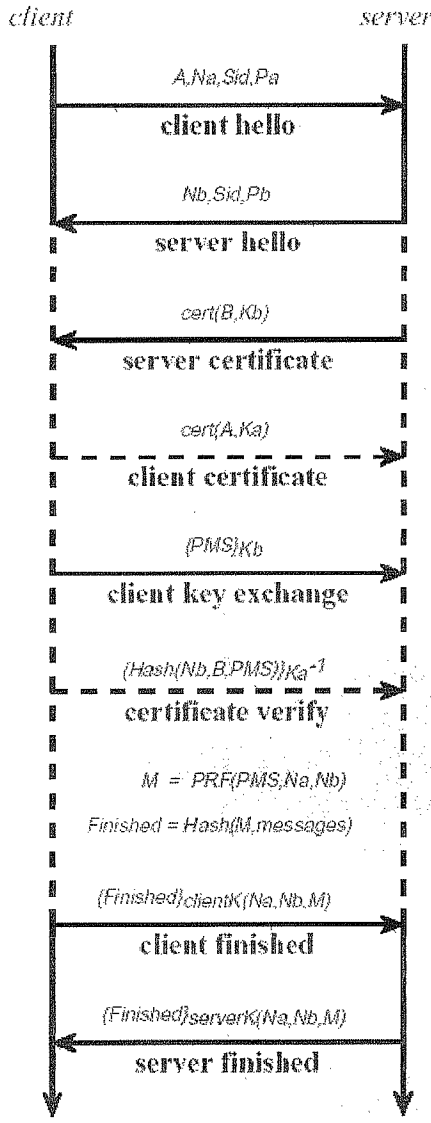


Fig3.1. The TLS Handshake Protocol as Modeled [Ref 6]

The inductive method has many advantages. Its semantic framework, based on the actions agents can perform, has few of the peculiarities of belief logics. Proofs impose no limits on the number of simultaneous or resumed sessions. Isabelle's automatic tools allow the proofs to be generated with a moderate effort, and they run fast. The full TLS proof script runs in 150 seconds on a 300Mhz Pentium.

2.18 Summary

To transfer information privately and securely across the Internet, the Secure Socket Layer (SSL) protocol was developed. This paper examines the SSL protocol and details how to improve the performance and scalability of Web sites by off-loading SSL processing and object delivery from Web servers.

CHAPTER THREE

RIVEST-ADI SHAMIR-LEONARD ADLEMAN (RSA) ENCRYPTION

3.1 Overview

Encryption is the act of encoding text so that others not privy to the decryption mechanism (the "key") cannot understand the content of the text. Encryption has long been the domain of spies and diplomats, but recently it has moved into the public eye with the concern of the protection of electronic transmissions and digitally stored data.

3.2 Public Key Cryptography

One of the biggest problems in cryptography is the distribution of keys. Suppose you live in the United States and want to pass information secretly to your friend in Europe. If you truly want to keep the information secret, you need to agree on some sort of key that you and he can use to encode/decode messages. But you don't want to keep using the same key, or you will make it easier and easier for others to crack your cipher. But it's also a pain to get keys to your friend. If you mail them, they might be stolen. If you send them cryptographically, and someone has broken your code, that person will also have the next key. If you have to go to Europe regularly to hand-deliver the next key, that is also expensive. If you hire some courier to deliver the new key, you have to trust the courier, et cetera.

3.2.1 Trap-Door Ciphers

But imagine the following situation. Suppose you have a special method of encoding and decoding that is "one way" in a sense. Imagine that the encoding is easy to do, but decoding is very difficult. Then anyone in the world can encode a message, but only one person can decode it. Such methods exist, and they are called "one way ciphers" or "trap door ciphers".

Here's how they work. For each cipher, there is a key for encoding and a different key for decoding. If you know the key for decoding, it is very easy to make the key for encoding, but it is almost impossible to do the opposite to start with the encoding key and work out the decoding key.

So to communicate with your friend in Europe, each of you has a trap door cipher. You make up a decoding key **Da** and generate the corresponding encoding key **Ea** your friend does exactly the same thing, but he makes up a decoding key **Db** and generates the corresponding encoding key **Eb** You tell him **Ea** (but not **Da**_) and he tells you **Eb** _ (but not **Db**) Then you can send him messages by encoding using **Eb**_ _ (which only he can decode) and vice-versa—he encodes messages to you using **Ea** (which only you can decode, since you're the only person with access to **Da**).

Now if you want to change to a new key, it is no big problem. Just make up new pairs and exchange the encoding keys. If the encoding keys are stolen, it's not a big deal.
The person who steals them can only encode messages—they can't decode them. In fact, he encoding keys (sometimes called "public keys") could just be published in a well-known location. It's like saying, "If you want to send me a private message, encode it using this key, and I will be the only person in the world who can read it." But be sure to keep the decoding key (the "private key") secret.

3.2.2 Certification

There is, of course, a problem with the scheme above. Since the public keys are really public, anyone can "forge" a message to you. So your enemy can pretend to be your friend and send you a message just like your friend can—they both have access to the public key. Your enemy's information can completely mislead you. So how can you be certain that a message that says it is from your friend is really from your friend? Here is one way to do it, assuming that you both have the public and private keys **Ea Eb Da Db** and as discussed in the previous section. Suppose I wish to send my friend a message that only he can read, but in such a way that he is certain that the message is from me. Here's how to do it. I will take my name, and pretend that it is an encoded message, and decode

it using Da I am the only person who can do this, since I am the only person who knows **Da** Then I include that text in the real message I wish to send, and I encode the whole mess Using **Eb** which only my friend knows how to decode. When he receives it, he will decode it using **Db** and he will have a message with an additional piece of what looks to him like junk characters. The junk characters are what I got by "decoding" my name. So he simply encodes the junk using my public key **Ea** and makes certain that it is my name. Since I am the only one who knows how to make text that will encode to my name, he knows the message is from me.

We can encode any text for certification, and in fact, you should probably change it with each message, but it's easy to do. Your message to your friend would look like this: "Attack at dawn. Here is my decoding of 'ABCDEFG': 'JDLEODK'." To assure privacy, for each message, change the "ABCDEFG" and the corresponding "JDLEODK".

3.3 RSA Encryption

In the previous section we described what is meant by a trap-door cipher, but how do you make one? One commonly used cipher of this form is called "RSA Encryption",
It is based on the following idea:

It is very simple to multiply numbers together, especially with computers. But it can be very difficult to factor numbers. For example, if I ask you to multiply together 34537 and 99991, it is a simple matter to punch those numbers into a calculator and 3453389167. But the reverse problem is much harder [Ref 2]. Suppose I give you the number 1459160519. I'll even tell you that I got it by plying together two integers. Can you tell me what they are? This is a very difficult problem. A computer can factor that number fairly quickly, but (although there are some tricks) it basically does it by trying most of the possible combinations. For any size number, the computer has to check something that is of the order of the size of the square-root of the number to be factored. In this case, that square-root is roughly 38000.

Now it doesn't take a computer long to try out 38000 possibilities, but what if the number to be factored is not ten digits, but rather 400 digits? The square-root of a number with 400 digits is a number with 200 digits. The lifetime of the universe is approximately 10/18 seconds – an 18 digit number. Assuming a computer could test one million factorizations per second, in the lifetime of the universe it could check 10/24 possibilities. But for a 400 digit product, there are 10/200 possibilities. This means the computer would have to run for 10/176 times the life of the universe to factor the large number.
It is, however, not too hard to check to see if a number is prime in other words to check to see that it cannot be factored. If it is not prime, it is difficult to factor, but if it is prime, it is not hard to show it is prime.

So RSA encryption works like this. I will find two huge prime numbers, p and q _ that have 100 or maybe 200 digits each. I will keep those two numbers secret (they are my private key), and I will multiply them together to make a number N=pq That number N is basically my public key. It is relatively easy for me to get N I just need to multiply my two numbers. But if you know N it is basically impossible for you to find p and q To get them, you need to factor N, which seems to be an incredibly difficult problem.

But exactly how is N used to encode a message, and how are p and q used to decode it? Below is presented a complete example, but I will use tiny prime numbers so it is easy to follow the arithmetic. In a real RSA encryption system, keep in mind that the prime numbers are huge.

In the following example, suppose that person A wants to make a public key, and that person B wants to use that key to send A a message. In this example, we will suppose that the message A sends to B is just a number. We assume that A and B have agreed on a method to encode text as numbers. Here are the steps:

1. Person A selects two prime numbers. We will use p=23 and q=41 for this example, but keep in mind that the real numbers person A should use should be *much* larger.

2. Person A multiplies p and q together to get pq= (23)(41)=943 _ is the "public key", which he tells to person B (and to the rest of the world, if he wishes).

3. Person A also chooses another number which must be relatively prime to (p-1)(q-1) In this case, (p-1)(q-1)=(22)(40)=880 so e=7 is fine. E is also part of the public key, so B also is told the value of e.

4. Now B knows enough to encode a message to A. Suppose, for this example, that the message is the number M=35

5. B calculates the $C=M^e \pmod N = 35^7 \pmod{943}$ value of $35^7 = 64339296875$ and $6433929875 \pmod{943} = 545$.
the number 545 is the encoding that B sends to A

7. Now A wants to decode 545. To do so, he needs to find a number d such that $ed = 1 \pmod{(p-1)(q-1)}$ or in this case, such that $7d = 1 \pmod{880}$.

A solution is d=503, since $7*503=3521=4(880)+1=1 \pmod{880}$.

8. To find the decoding, A must calculate $C^d \pmod N = 545^{503} \pmod{943}$ This looks like it will be a horrible calculation, and at first it seems like it is, but notice that 503=256+128+64+32+16+4+2+1 (this is just the binary expansion of 503). So this means that $545^{503} = 545^{256+128+64+32+16+4+2+1} = 545^{256}\ 545^{128} \ldots 545^{1}$.

But since we only care about the result (mod943), we can calculate all the partial results in that modulus, and by repeated squaring of 545, we can get all the exponents that are powers of 2. For example, $545^2 \pmod{943} = 545.545 = 297025 \pmod{943} = 923$

Then square again:

$545^4 \pmod{943} = 545^{2^2} \pmod{943} = 923.923 = 851929 \pmod{943} = 400$

and so on. We obtain the following table:

$545^1 \pmod{943} = 545$

$545^2 \pmod{943} = 923$

$545^4 \pmod{943} = 400$

$545^8 \pmod{943} = 633$

$545^{16} \pmod{943} = 857$

$545^{32} (\text{mod} 943) = 795$

$545^{64} (\text{mod} 943) = 215$

$545^{128} (\text{mod} 943) = 18$

$545^{256} (\text{mod} 943) = 324$

So the result we want is:

$545^{503} (\text{mod} 943) = 324.18.215.795.400.923.545 (\text{mod} 943) = 35.$

Using this tedious (but simple for a computer) calculation, A can decode B's message and obtain the original message N=35

3.3.1 Simple Explanation of RSA

Let p and q be distinct large primes and let n be their product. Assume that we also computed two integers, d (for decryption) and e (for encryption) such that d * e 1 (mod ø(n)) where $ø(n)$ is the number of positive integers smaller than n that have no factor except 1 in common with n The integers n and e are made public, while p, q, and d are kept secret. Let m be the message to be sent, where m is a positive integer less than and relatively prime to n. A plaintext message is easily converted to a number by using either the alphabet position of each letter (a=01, b=02, ..., z=26) or using the standard ASCII table. If necessary (so that $m<n$), the message can be broken into several blocks.

The encoder computes and sends the number

m' = m^e mod n

To decode, we simply compute

e^d mod n

Now, since both n and e are public, the question arises: can we compute from them d? The answer: it is possible, if n is factored into prime numbers. The security of RSA depends on the fact that it takes an impractical amount of time to factor large numbers.

3.4 Summary

RSA developed to help ensure internet security. Simply put, a cryptosystem is an algorithm that can convert input data into something unrecognizable (encryption), and convert the unrecognizable data back to its original form (decryption).

In a public-key cryptosystem, the sender encrypts a message with a private key. The sender's public key is posted (usually in a table). The recipient looks up the senders public key and uses it and her/his own private key to unlock the message. Private and public keys are associated by a function. In the RSA cryptosystem, the private and public keys are linked by the factorization of prime numbers.

CHAPTER FOUR

RSA ENCRYPTION APPLICATION USING JAVA

4.1 Overview

With the rapid growth of the Internet and E-Commerce, there has come a widespread need for strong encryption. RSA encryption has helped to fill that need by providing an easy to understand implementation of public key encryption while providing security that has stood the test of time.

4.2 Program Explanation

Below is a Java applet (Appendix A) that allows a user to create RSA keys and encrypt and decrypt text or numbers.

For starters, is it designed by a first time Java programmer, it also uses basic psuedo-random number generation.

4.3 Using The Program

The interface (Appendix B) is not very explanatory, so you'll probably need to follow these directions the first time through. The applet can be used for three tasks - key generation, encryption, and decryption.

Key generation

To use the RSA cipher for public key cryptography, two sets of keys are required. There is a public key which consists of an encryption value 'e' and a modulus 'n'. There is also a private key made up of 'd', your decryption value and the same 'n' modulus. To start, first enter two prime numbers into the 'p' and 'q' text fields. Or, you can click on the "Generate p and q" button and the applet will create two prime numbers that are any size you select greater than 3 bits long. Bigger numbers are, in general, more secure (512 will create an 'n' of size 1024, which is standard) but numbers too large will take a while to generate. So remember this: generating large primes p and q (greater than about 1024

bits) may take a long time - do so at your own risk of freezing your browser! Okay, now you've been warned. Next, click on the "Generate n" button. It multiplies 'p*q' and will give you 'n.' Now click on "Generate e" to come up with a 'e' value that is whatever size you want (32 bits usually okay). And finally click on "Calculate d" to get your 'd' value.

Now, to use these values to do public key cryptography make your 'e' and 'n' values available to anyone that wants to send you an encrypted message. Keep a copy of the 'n' value and keep the 'd' value secret (so only you can decrypt the messages).

Encryption

With the 'n' and 'e' values entered, you can now type a text message in the "Plain text message" area. Since RSA only encrypts numbers, we convert the message to number with the "Convert to number" button. You'll notice encryption is done with buttons on the left side and flows downward, decryption is done with buttons on the right and flows upward. The last step is encrypting the message by clicking on the "Encrypt" button. The encrypted message is left in numerical form because if it was converted back to text (ascii), some characters would not show up and you could not copy them properly.

Decryption

After making sure the proper 'd' and 'n' values are entered, place the encrypted numeric message into the bottom text box labeled "Encoded numerical message." Use the "Decrypt" button to decode the message, then click on the "Convert to text" button.

4.3 Summary

This paper deals with the generation of RSA keys and their security. It also touches on how RSA encryption can be used for digital signatures and the new weaker restrictions on encryption export to some countries. The paper then goes into depth on how you can construct short RSA keys while still maintaining security. These short RSA keys can be used to reduce the overall storage requirements of keys for a group of users. Short RSA keys can also be used to store publicly accessible information such as user ID's, name, address, or any other short piece of information you wish to be made publicly available.

GLOSSARY

Algorithm

A formula or set of steps for solving a particular problem. To be an algorithm, a set of rules must be unambiguous and have a clear stopping point.

Certificate Authorities

A trusted third-party organization or company that issues digital certificates used to create digital signatures and public-private key pairs. The role of the CA in this process is to guarantee that the individual granted the unique certificate is, in fact, who he or she claims to be.

Content Distribution Network "CDN"

The physical network infrastructure of connecting global locations together which allows for Web content to be distributed to the edges of the entire network infrastructure.

Cryptographic algorithms

A cryptographic system that uses two keys - a public key known to everyone and a private or secret key known only to the recipient of the message. An important element to the public key system is that the public and private keys are related in such a way that only the public key can be used to encrypt messages and only the corresponding private key can be used to decrypt them. Moreover, it is virtually impossible to deduce the private key if you know the public key.

Decrypt

The process of decoding data that has been encrypted into a secret format. Decryption requires a secret key or password.

Digital Certificate

An attachment to an electronic message used for security purposes. The most common use of a digital certificate is to verify that a user sending a message is who he or she claims to be, and to provide the receiver with the means to encode a reply.
An individual wishing to send an encrypted message applies for a digital certificate from a Certificate Authority.

Encrypt

The translation of data a secret code. Encryption is the most effective way to achieve data security.
Unencrypted data is called plain text encrypted data is referred to as cipher text.

HTTP

Short for Hypertext Transfer Protocol, the underlying protocol used by the World Wide Web. HTTP defines how messages are formatted and transmitted, and what actions Web servers and browsers should take in response to various commands.

HTTPS

By convention, Web pages that require an SSL connection start with https: instead of http.

IP

Abbreviation of Internet Protocol, pronounced as two separate letters. IP specifies the format of packets, also called datagrams, and the addressing scheme. Most networks combine IP with a higher-level protocol called Transport Control Protocol (TCP) which establishes a virtual connection between a destination and a source.

ISP

Short for Internet Service Provider. A company that provides connection and services on the Internet, such as remote dial-in access, DSL connections and Web hosting services.

OSI

Short for Open System Interconnection, an ISO standard for worldwide communications that defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer in one station, proceeding to the bottom layer, over the channel to the next station and back up the hierarchy.

PIN

Short for Personal Identification Number. Typically PIN's are assigned by financial institutes to validate the identity of a person during a transaction.

RSA

An public key encryption technology developed by RSA Data Security, Inc. The acronym stands for Rivest, Shamir, and Adelman, the inventors of the technique. The RSA algorithm is based on the fact that there is no efficient way to factor very large numbers. Deducing an RSA key, therefore, requires an extraordinary amount of computer processing power and time.

SSL

Short for Secure Sockets Layer, a protocol developed by Netscape for transmitting private documents via the Internet. SSL works by using a private key to encrypt data that's transferred over the SSL connection.

SSL performs a negotiation between the two parties who are exchanging information, the negotiation process involves understanding the key pairs, the protocols, and the type of data request.

TCP

Abbreviation of Transmission Control Protocol, and pronounced as separate letters. TCP is one of the main protocols in TCP/IP networks. Whereas the IP protocol deals only with packets, TCP enables two hosts to establish a connection and exchange streams of data. TCP guarantees delivery of data and also guarantees that packets will be delivered in the same order in which they were sent.

UDP

Short for User Datagram Protocol, a connectionless protocol that, like TCP, runs on top of IP networks.

Unlike TCP/IP, UDP/IP provides very few error recovery services, offering instead a direct way to send and receive datagram's over an IP network. It's used primarily for broadcasting messages over a network.

Web

A system of Internet servers that support specially formatted documents. The documents are formatted in with HTML (Hypertext Markup Language) that supports links to other documents, as well as graphics, audio, and video files.

Web Server Accelerators

A system that services Web servers by offloading TCP/IP connections, responding to Web Client requests, replicating the Web content for availability and surge protection, and enhancing Web server performance.

Also known as a Web accelerator or a reverse proxy server.

Web Sever

A computer that delivers (serves up) Web pages. Every Web server has an IP address and possibly a domain name. For example, if you enter the URL http://www.neu.edu.tr in your browser, this sends a request to the server whose domain name is *.neu.edu.tr* The server then fetches the page named index.html and sends it to your browser.

CONCLUSION

This thesis has described the SECURE SOCKETS LAYERS based network security and a JAVA based program has been developed by me to test the various security algorithms.

For more safety and security I allowed a user to create RSA keys and encrypt and decrypt text or numbers.

Fortunately, web services offer new mechanisms for securing Internet transactions, leveraging proven technologies such as authentication, authorization and encryption solutions. The technology also offers a long term potential for delivering pervasive, cost effective security services across the enterprise.

We discussed that SSL has become the universal standard for authenticating web sites to web browsers, and for encrypting communications between web browsers and web servers. However, because SSL is highly CPU-intensive and degrades web server performance, organizations have deployed SSL in a limited fashion.

By integrating SSL processing with its server accelerators, Companies can now apply SSL to more content without degrading the performance of their Web sites.

Those embarking on a web services initiative need to make security planning an integral part of their efforts, progressing from straightforward security implementations to more complex deployments. RSA Security stands ready to assist enterprises in ensuring that security is fully reflected in their web services strategies and implementations.

REFERENCES

1. [BELL96] Bellare, M., Canetti, R., Krawczyk, H., "Keying Hash Functions for Message Authentication", Preprint.
2. [RSA] R. Rivest, A. Shamir, and L. M. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, v. 21, n. 2, Feb 1978, pp. 120-126.
3. [X509] CCITT Recommendation X.509 (1988), "The Directory - Authentication Framework".
4. Computer Networks. A systems approach [Larry L. Peterson & Bruces Dave. 2000{ISBN 1-55860-577-0}]
5. Commerce and security: overview of SSL. Jet Treuhaft [http://www.developer.netscape.com]
6. "Secure Sockets Layers Prorocol and Application" [Allan Schiffman: Tersia Systems.Inc{http://www.tersia.com}]
7. Hull, Scott. Content Delivery Networks: Web Switching for Security, Availability, Speed McGraw-Hill, New York 2002
8. [PKCS-7] RSA Data Security, Inc. "Cryptographic Message Syntax Standard", PKCS-7, Nov 1, 1993.
9. [HAST86] Hastad, J., "On Using RSA With Low Exponents in a Public Key Network," Advances in Cryptology-CRYPTO 95 Proceedings, Springer-Verlag, 1986.
10. [FIPS-46-1] Federal Information Processing Standards Publication (FIPS PUB) 46-1, Data Encryption Standard, Reaffirmed 1988 January 22
11. [RFC-1422] Kent, S. "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management", RFC1422, Feb 1993.
12. [RFC-1510] Kohl, J., and Neuman, C., "The Kerberos Authentication Service (V5)", RFC1510, September 1993.
13. [SCH] B. Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C, Published by John Wiley & Sons, Inc. 1994.

14. [SHA] NIST FIPS PUB 180-1, "Secure Hash Standard," National Institute of Standards and Technology, U.S. Department of Commerce, DRAFT, 31 May 1994.

15. [X509] CCITT. Recommendation X.509: "The Directory - Authentication Framework". 1988.

16. [DH1] W. Diffie and M. E. Hellman, "New Directions in Cryptography," IEEE Transactions on Information Theory, V. IT-22, n. 6, Jun 1977, pp. 74-84.

17. [DES] ANSI X3.106, "American National Standard for Information Systems-Data Link Encryption," American National Standards Institute, 1983.

18. DSS] NIST FIPS PUB 186, "Digital Signature Standard," National Institute of Standards and Technology, U.S. Department of Commerce, 18 May 1994.

19. [PKCS7] RSA Laboratories, "PKCS #7: RSA Cryptographic Message Syntax Standard," version 1.5, November 1993.

20. KRAW] H. Krawczyk, IETF Draft: Keyed-MD5 for Message Authentication, November 1995.

21. [FOR] NSA X22, Document # PD4002103-1.01, "FORTEZZA: Application Implementers Guide," April 6, 1995.

APPENDIX A

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.math.BigInteger;
import java.util.Random;

public class RSAEncryption extends Applet implements ActionListener {
//Building the graphical user interface
Label pqLabel;
TextArea pTextArea;
Label pLabel;
TextArea qTextArea;
Label qLabel;
Button generate_pqButton;
Label generate_pq_sizeLabel;
TextField generate_pq_sizeField;
Label nLabel;
TextArea nTextArea;
Label nSpacer;
Button calculate_nButton;
Label eLabel;
TextArea eTextArea;
Label eSpacer;
Button generate_eButton;
Label generate_e_sizeLabel;
TextField generate_e_sizeField;
Label dLabel;
TextArea dTextArea;
Label dSpacer;
Button calculate_dButton;
```

```
Label useSpacer;
TextArea plainTextArea;
Button convert_to_numberButton;
Label convertButtonSpacer;
Button convert_to_textButton;
TextArea mTextArea;
Button encryptButton;
Label cryptButtonSpacer;
Button decryptButton;
TextArea cTextArea;

// This is the certainty that the bigInteger class will generate a prime number.  It's
currently
// set at 20 which means the odds of being a prime is 1-2^20 or about one in a million
of it being
// non-prime.  If this number is increased the time to generate a prime goes up.

int prime_certainty = 20;
public void init()
{
    //The white background matches the web page background
    setBackground (Color.white);

    //The flowLayout is left justified.
    setLayout(new FlowLayout(0));


    pqLabel = new Label("Enter prime 'p' and 'q' values or use the button below to
generate them:");
    add(pqLabel);
```

```
// setting the 'P' label and TextArea
Panel pPanel = new Panel();
pLabel = new Label("p:");
pPanel.add(pLabel);
pTextArea = new TextArea("", 2, 50, 1);
pPanel.add(pTextArea);
add(pPanel);

// setting the 'q' label and TextArea
Panel qPanel = new Panel();
qLabel = new Label("q:");
qPanel.add(qLabel);
qTextArea = new TextArea("", 2, 50, 1);
qPanel.add(qTextArea);
add(qPanel);

// setting the 'Generate' label and TextBox
Panel generate_pqPanel = new Panel();
generate_pqButton = new Button("Generate p and q");
generate_pqPanel.add(generate_pqButton);
generate_pq_sizeLabel = new Label("which are of average bit size:");
generate_pqPanel.add(generate_pq_sizeLabel);
generate_pq_sizeField = new TextField("16", 5);
generate_pqPanel.add(generate_pq_sizeField);
add(generate_pqPanel);

// setting the 'n' label and TextArea
Panel nTextPanel = new Panel();
nLabel = new Label("n:");
nTextPanel.add(nLabel);
nTextArea = new TextArea("", 4, 50, 1);
```

```
nTextPanel.add(nTextArea);
add(nTextPanel);
Panel nPanel = new Panel();
calculate_nButton = new Button("Calculate n");
nPanel.add(calculate_nButton);
nSpacer = new Label("                                                    ");
nPanel.add(nSpacer);
add(nPanel);

// setting the 'e' label and TextArea
Panel eTextPanel = new Panel();
eLabel = new Label("e:");
eTextPanel.add(eLabel);
eTextArea = new TextArea("", 2, 50, 1);
eTextPanel.add(eTextArea);
add (eTextPanel);
Panel ePanel = new Panel();
generate_eButton = new Button("Generate e");
ePanel.add(generate_eButton);
generate_e_sizeLabel = new Label("which is of bit size:");
ePanel.add(generate_e_sizeLabel);
generate_e_sizeField = new TextField("8", 5);
ePanel.add(generate_e_sizeField);
eSpacer = new Label("                                    ");
ePanel.add(eSpacer);
add(ePanel);

// setting the 'd' label and TextArea
Panel dTextPanel = new Panel();
dLabel = new Label("d:");
dTextPanel.add(dLabel);
```

```
        dTextArea = new TextArea("", 2, 50, 1);
        dTextPanel.add(dTextArea);
        add(dTextPanel);
        Panel dPanel = new Panel();
        calculate_dButton = new Button("Calculate d");
        dPanel.add(calculate_dButton);
        dSpacer = new Label("                                                    ");
        dPanel.add(dSpacer);
        add(dPanel);

        useSpacer = new Label("Enter text, numbers or encoded numbers below.
");
        add(useSpacer);

        plainTextArea = new TextArea("Plain text message", 4, 54, 1);
        add(plainTextArea);

        // setting the 'Convert to Number' label and TextArea
        Panel convertButtonPanel = new Panel();
        convert_to_numberButton = new Button("Convert to Number");
        convertButtonPanel.add(convert_to_numberButton);
        convertButtonSpacer = new Label("                                    ");
        convertButtonPanel.add(convertButtonSpacer);
        convert_to_textButton = new Button("Convert to Text");
        convertButtonPanel.add(convert_to_textButton);
        add(convertButtonPanel);

        mTextArea = new TextArea("Numerical message", 4, 54, 1);
        add(mTextArea);

        // setting the 'Encrypt' label and TextArea
```

```
    Panel cryptButtonPanel = new Panel();
    encryptButton = new Button("Encrypt") ;
    cryptButtonPanel.add(encryptButton);
    cryptButtonSpacer = new Label("
");
    cryptButtonPanel.add(cryptButtonSpacer);

    // setting the 'decrypt' label and TextArea
    decryptButton = new Button("Decrypt") ;
    cryptButtonPanel.add(decryptButton);
    add(cryptButtonPanel);

    cTextArea = new TextArea("Encoded numerical message", 4, 54, 1);
    add(cTextArea);

    // End of Building the GUI
    // Relating ActionListeners to the Buttons

    generate_pqButton.addActionListener(this);
    calculate_nButton.addActionListener(this);
    generate_eButton.addActionListener(this);
    calculate_dButton.addActionListener(this);
    convert_to_numberButton.addActionListener(this);
    convert_to_textButton.addActionListener(this);
    encryptButton.addActionListener(this);
    decryptButton.addActionListener(this);
}
public void actionPerformed(ActionEvent event) {
    //If the "generate pq" button is pushed then get the desired size of p and q
    // then let the BigInteger class generate the prime.
```

```
    // The size of p and q
    // is offset so that we can guarantee that p and q will not be too close
    // to each other. This make guessing p and q by searching values next to the
    // square root of n more difficult.

    if (event.getSource() == generate_pqButton) {
            int pq_size = new Integer(generate_pq_sizeField.getText()).intValue();
            if (pq_size >= 4) {
                    qTextArea.setText(new BigInteger(pq_size + 1, prime_certainty,
new Random()).toString());
                    pTextArea.setText(new BigInteger(pq_size - 1, prime_certainty,
new Random()).toString());
            }
            else {
                    pTextArea.setText("Enter larger p and q size.");
            }
    }
    //Send the data from p and q to the calculate n function

    else if (event.getSource() == calculate_nButton) {
            nTextArea.setText(calculate_n(new BigInteger(pTextArea.getText()),
                            new BigInteger(qTextArea.getText()))
                            .toString());
    }
    //Send the data from p, q and the size of e to the generate_e function

    else if (event.getSource() == generate_eButton) {
            eTextArea.setText(generate_e(new BigInteger(pTextArea.getText()),
                    new BigInteger(qTextArea.getText()),
                    new Integer(generate_e_sizeField.getText()).intValue())
                    .toString());
```

```
    }
    //Send data from p, q and e to the calculate_d function

    else if (event.getSource() == calculate_dButton) {
            dTextArea.setText(
                    calculate_d(new BigInteger(pTextArea.getText()),
                            new BigInteger(qTextArea.getText()),
                            new BigInteger(eTextArea.getText()))
                            .toString());
    }
    //This converts the plain text string into a number by reading the string in,
    //converting to bytes (ascii), then converting these bytes into a BigInteger.
    //The opposite happens in the next function.

    else if (event.getSource() == convert_to_numberButton) {
            mTextArea.setText(new
BigInteger(plainTextArea.getText().getBytes()).toString());
    }

    else if (event.getSource() == convert_to_textButton) {
            plainTextArea.setText(new String(new
BigInteger(mTextArea.getText()).toByteArray()));
    }
    //Send the data to be encrypted and decrypted.

    else if (event.getSource() == encryptButton) {
            cTextArea.setText(encrypt(new BigInteger(mTextArea.getText()),
                            new BigInteger(eTextArea.getText()),
                    new BigInteger(nTextArea.getText()))
                            .toString());
    }
```

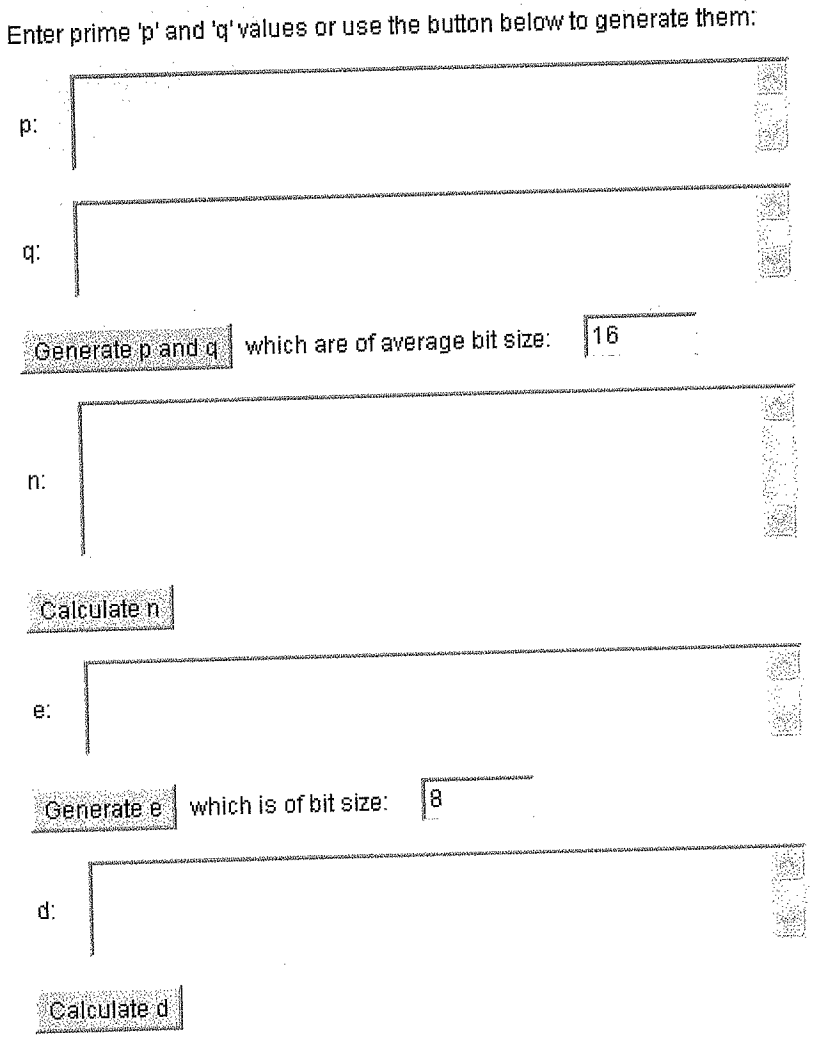
```
    else if (event.getSource() == decryptButton) {
            mTextArea.setText(decrypt(new BigInteger(cTextArea.getText()),
                                      new BigInteger(dTextArea.getText()),
                                      new BigInteger(nTextArea.getText()))
                                      .toString());
    }
}
//To generate e first phi(pq) (which is equal to phi(n)) is calculated.
// This is equal to (p-1)*(q-1).  Then the loop searches for psudo-randomly
// generated e of a specified size until one is found that is relatively
// prime to phi(pq) (gdc(e,phi_pq) = 1).
//The last line is to ensure it does not go into a infinite loop
// if a e cannot be found for that bit size.

BigInteger generate_e(BigInteger p, BigInteger q, int bitsize) {
    BigInteger e, phi_pq;
    e = new BigInteger("0");
    phi_pq = q.subtract(new BigInteger("1"));                    // q - 1
    phi_pq = phi_pq.multiply(p.subtract(new BigInteger("1")));// (p -1)*(q-1)

    int i = 0;

    // Generate e Randomly
    do {
            e = (new BigInteger(bitsize, 0, new Random())).setBit(0);
            i = i + 1;
    } while( i<100 && (e.gcd(phi_pq).compareTo(new BigInteger("1")) != 0));


    // To break the infinite loop
```

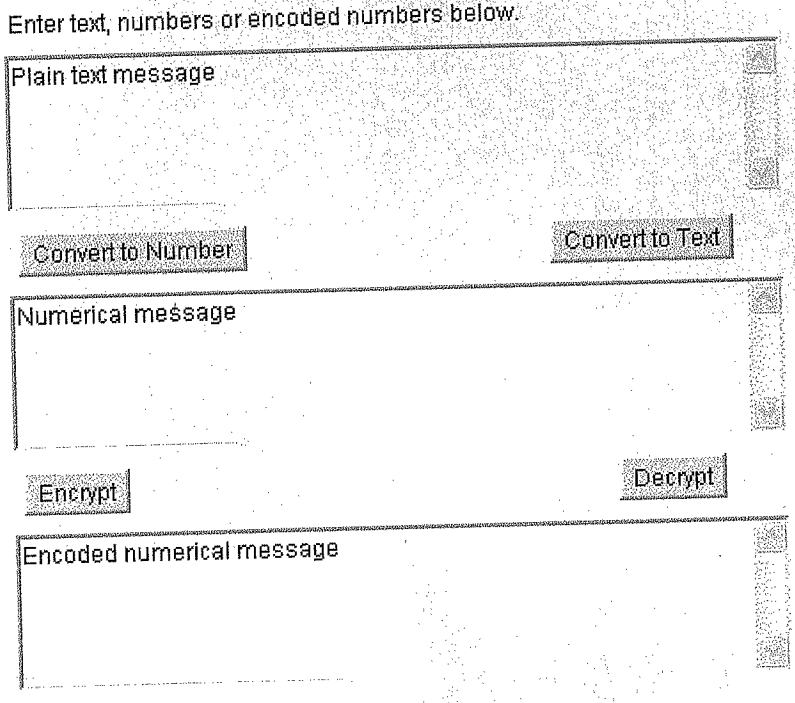
```
    return e;
}
 // I calculate phi_pq ((p-1)*(q-1)) and then let the modInverse function
// do the hard part of finding [ e^(-1) mod phi_pq ] =  [ ed = 1 [mode{p-1}{q-1}]
BigInteger calculate_d(BigInteger p, BigInteger q, BigInteger e) {
    BigInteger d, phi_pq;

    phi_pq = q.subtract(new BigInteger("1"));                    //{q-1}
    phi_pq = phi_pq.multiply(p.subtract(new BigInteger("1")));//{p-1}{q-1}

    d = e.modInverse(phi_pq);    // e^(-1) mod{p-1}{q-1}
    return d;
}
//Returns n=p*q

BigInteger calculate_n(BigInteger p, BigInteger q) {
    return p.multiply(q);
}

//This is a little tricky because the user is allowed to choose
// the size of n.  The value to be encrypted must be less than n.
// So first I find the bit size of n, then subract one, and that is the
// size of the message that I will encrypt at one time.  This ensures
// the message chunk that is encrypted is smaller than n.  I use a
// mask to take one chunk of message at a time.  Then the chunk is
// encrypted and placed in the result c.  During the next iteration
// the message is shifted right and the result is shifted left and combined
// with c.  The encrypted chunk must be the same bit size as n so that no
// data is lost.
// Encryption is done using the modPow function provide by the BigInt class.
```

```
BigInteger encrypt(BigInteger m, BigInteger e, BigInteger n) {
    BigInteger c, bitmask;
    c = new BigInteger("0");
    int i = 0;
    bitmask = (new BigInteger("2")).pow(n.bitLength()-1).subtract(new
BigInteger("1")); // [ 2^(length(n)-1) ] - 1
    while (m.compareTo(bitmask) == 1) {
            c = m.and(bitmask).modPow(e,n).shiftLeft(i*n.bitLength()).or(c);
            m = m.shiftRight(n.bitLength()-1);
            i = i+1;
    }
    c = m.modPow(e,n).shiftLeft(i*n.bitLength()).or(c);
    return c;
}

//Decryption is done just as encryption above, only now the data is read in
// in chunks the same size as n, and the result, if correct, will be one bit
// less than the size of n (because that was the original chuck size).

BigInteger decrypt(BigInteger c, BigInteger d, BigInteger n) {
    BigInteger m, bitmask;
    m = new BigInteger("0");
    int i = 0;
    bitmask = (new BigInteger("2")).pow(n.bitLength()).subtract(new
BigInteger("1"));
    while (c.compareTo(bitmask) == 1) {
            m = c.and(bitmask).modPow(d,n).shiftLeft(i*(n.bitLength()-1)).or(m);
            c = c.shiftRight(n.bitLength());
            i = i+1;
    }
    m = c.modPow(d,n).shiftLeft(i*(n.bitLength()-1)).or(m);
```

```
        return m;
    }

}
```

APPENDIX B


Enter prime 'p' and 'q' values or use the button below to generate them:
p:
q:
Generate p and q
which are of average bit size:
16
n:
Calculate n
e:
Generate e
which is of bit size:
8
d:
Calculate d


Enter text, numbers or encoded numbers below.
Plain text message
Convert to Number
Convert to Text
Numerical message
Encrypt
Decrypt
Encoded numerical message