Secure WWW Server for Lotus Notes

by

Dinko M. Zidaric-Sudovacki

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

February 7, 1996

Author_____
Department of Electrical Engineering and Computer Science
February 7, 1996

Certified by _____
Ronald L. Rivest
Thesis Supervisor

Accepted by_____
F.R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Secure WWW Server for Lotus Notes

by

Dinko M. Zidaric-Sudovacki

Submitted to the

Department of Electrical Engineering and Computer Science

February 7, 1996

In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

A major shortcoming of the InterNotes WebPublisher (the WWW server for Lotus Notes) is the lack of security mechanisms. Security mechanisms are present during transactions between native Notes clients and servers. However, they are not present during transactions between the WebPublisher and Web browsers.
This study proposes three designs that enable secure communications between Web browsers and the WebPublisher. In addition, it surveys current state-of-the-art of computer security and analyzes existing Lotus Notes and Web security mechanisms.

Thesis Supervisor: Ronald Linn Rivest, Ph.D.
Title: Associate Director, Laboratory for Computer Science

Table of Contents

Chapter 1: Introduction

1.1 Problem Statement

A major shortcoming of the InterNotes WebPublisher (the Lotus Notes gateway to the Web) is the lack of security mechanisms. Security mechanisms are present during transactions between native Notes clients and servers. However, they are not present during transactions between the WebPublisher and Web browsers.

The main purpose of this study was to analyze the existing Notes and emerging Web security protocols and to propose several designs that would enable secure communications between Web clients and the InterNotes WebPublisher.

1.2 Subgoals

To accomplish the major goal of designing a system which would allow secure transactions for the InterNotes WebPublisher we identified two other subgoals:

1. To research computer and network security

   This goal was identified very early in the study because we realized that computer security was an unfamiliar area to even very knowledgeable computer professionals. This study includes a survey of current state-of-the-art of computer security with an emphasis on issues related to Lotus.

2. To analyze Lotus Notes and Web security mechanisms

   An analysis of existing security mechanisms was considered a necessary prerequisite for this study. To effectively augment existing security mechanisms, and especially to be able to integrate them with Web security, it was considered crucial to have in-depth knowledge of both Notes and Web security issues.

1.3 Context

1.3.1 Lotus Notes

During the past decade, Lotus Notes has emerged as a groupware standard enabling groups of users to share data easily across various applications, OS platforms, and local and global area networks. Having superb built-in security mechanisms, Notes has provided its users with two-way authentication of both servers and clients, data encryption, and high fidelity access control capabilities.

1.3.2 World Wide Web

In the past few years, the role of the Internet has dramatically changed. For many years, the Internet was known only to a small subset of the general population and was used primarily by scientific, academic, and military communities. Today, businesses are becoming interested in exploring the Internet as a way to enhance their productivity, establish their market presence, perform commerce, and easily obtain and send information worldwide. The predominant business application using the Internet has been the World Wide Web, which provides easy and powerful ways for businesses to share data over long distances across different applications and OS platforms. Unfortunately, in contrast to Notes, most of today's Web servers and clients do not provide strong authentication mechanisms, rarely offer encryption, and have primitive or nonexistent access control mechanisms. Because of this, the information made available to the Internet through the Web servers can be easily compromised through unauthorized access.

1.3.3 InterNotes WebPublisher

To bridge the gap between Notes and the Internet, Lotus formed the InterNotes group whose main goal was to provide the means of connecting Lotus Notes technology with the rapidly growing technologies of the Internet. One result is the InterNotes WebPublisher which transforms Notes data, primarily documents and forms, into the Web's native HTML format. Thus, it provides a two-way link between Notes systems and the Web.

1.4 Organization

This study is organized in several chapters. This chapter defines the main problem of the study together with its major goals. It also presents the context in which the study has been performed and its organization. The next chapter gives a background on computer and network security issues. The third chapter presents and analyzes two major existing authentication protocols: The Lotus Notes Authentication Protocol developed by Lotus Development Corporation, and Secure Sockets Layer (SSL) Protocol developed by Netscape Communications Corporation. The fourth chapter presents and evaluates three designs providing different levels of security for the InterNotes WebPublisher.

1.4.1 Background on Computer and Network Security

This chapter gives definitions and examples of many issues connected with computer and network security. First, it defines security, its objectives and mechanisms. Second, it gives an overview of cryptography from both a theoretical and practical standpoints. It introduces secret and public key cryptography, hash functions, notion of digital signatures, and authentication protocols. It addition, it discusses certificates, certifying authorities and certificate hierarchies.

1.4.2 Existing Security Protocols

In order to effectively leverage the advantages of Notes as well as the Web security mechanisms in the design of the security mechanisms for the InterNotes WebPublisher, we performed an in-depth analysis of the Notes and Secure Sockets Layer authentication protocols. This analysis proved later to be crucial because all our designs of security mechanisms were at least partially based on either Notes or the SSL security protocols.

1.4.3 Design of Security Mechanisms for the InterNotes WebPublisher

In its current implementation, the InterNotes WebPublisher does not provide any mechanisms for user authentication or authorization; therefore preventing people from publishing sensitive information on the Web. Notes, on the other hand, offers very secure client and server authentication mechanisms allowing fully encrypted transactions to be

performed over the network. In this chapter we present several designs of authentication mechanisms for the WebPublisher which are based on Notes and Web security protocols. They range from simple name and password directory schemes to full two-way authentication protocols based on Notes and SSL authentication protocols.

Chapter 2: Background on Computer & Network Security

This chapter on computer and network security presents general definitions of security, its goals and mechanisms. It defines cryptography, cryptographic functions and techniques, notion of authentication and certificates. The material presented in this section is based in part on the lectures given by Prof. Ronald L. Rivest at the Massachusetts Institute of Technology during the fall term of 1995. All other materials and authors are referenced separately.

2.1 Security

Before we discuss security of any computer system, we have to define and outline what we mean by security. This is usually accomplished by specifying *security policy*, that is a set of desired goals. Once goals have been identified, we can define *security mechanisms* which are the tools that make sure that the desired goals have been met. To construct effective security mechanisms, it is useful to define general *security principles*.

2.1.1 Goals

While the specific security goals of a computer system vary depending on its purpose, they may be grouped in three broad categories:

- Secrecy: Assets of a computer system should be accessible only by authorized parties. Secrecy implies "read-type" access: reading, viewing, printing or knowing the existence of a particular object.

- Integrity: Assets of a computer system should be modified only by authorized parties. Integrity implies "write-type" access: writing, modifying, creating and deleting of computer assets.
- Availability: Assets of a computer system should be available to authorized users. A *denial of service* attack is an attack that infringes the availability of the system. A very simple system that preserves secrecy might prevent anyone from reading a particular object. Although this system would achieve secrecy, it would be useless because the availability of the system would not be preserved.

2.1.2 Mechanisms

Security mechanisms are tools that insure that the security goals are met. Very simple security mechanisms involve user awareness of possible attacks, physical protection of computer assets. More sophisticated ones involve cryptography, access control, and auditing.

2.1.3 Principles

Security principles establish guidelines for designing effective security mechanisms. For example:

- Principle of least privilege: A user should be given only privileges needed to perform a task. Giving less privileges will not allow successful accomplishment of the task, and giving more privileges might introduce security risks.
- Minimize amount of trusted components: Precisely identify what components need to be trusted and keep those components as small as possible.
- Do not aim for perfection: Perfection is impossible to achieve. Instead build the system which can detect problems and can also recover from attacks.

2.1.4 Security Threats

Threats to computer systems are circumstances that have the potential of causing loss or harm. There are four main classes of security threats which are explained below.

- Interruption: During interruption, a computer asset becomes lost, unavailable, or unusable. The erasure of files, the crash of an operating system, or the inability to find a file on a disk, are all examples of an interruption.
- Interception: Interception involves unauthorized parties gaining access to computer assets. These parties include other people, computers, or program threads. Reading other people's mail, sniffing packets on the Internet, or copying programs are prime examples of this kind of attack.
- Modification: Tampering with an asset so that the asset has been changed in some way represents a security threat called modification. Many different examples of modifications are available. Modifying data in a database, modifying a program so that it performs additional computations, or modifying packets going over the network represent different kinds of modification threats.
- Fabrication: During fabrication, an unauthorized party creates or fabricates counterfeit objects for a computer system. Fabrication might involve creation of new documents in a database, insertion of packets on the network, or creation of data structures in computer memory.

2.2 Cryptography

The word cryptography means hidden writing, and it is usually referred to as the art and science of secret writing. Cryptography uses encryption to conceal the text. Cryptanalysis is the study of encryption and encrypted messages, with the goal of finding the hidden meanings of the text. Cryptology encompasses both cryptography and cryptanalysis and is the research into and study of encryption and decryption.

2.2.1 Encryption & Decryption

Encryption is a process of transforming a message into a form which does not reveal the meaning of the message. Using encryption we can attain a secure channel of communication over an insecure one. Decryption is a reverse process which transforms an encrypted message into its original form.

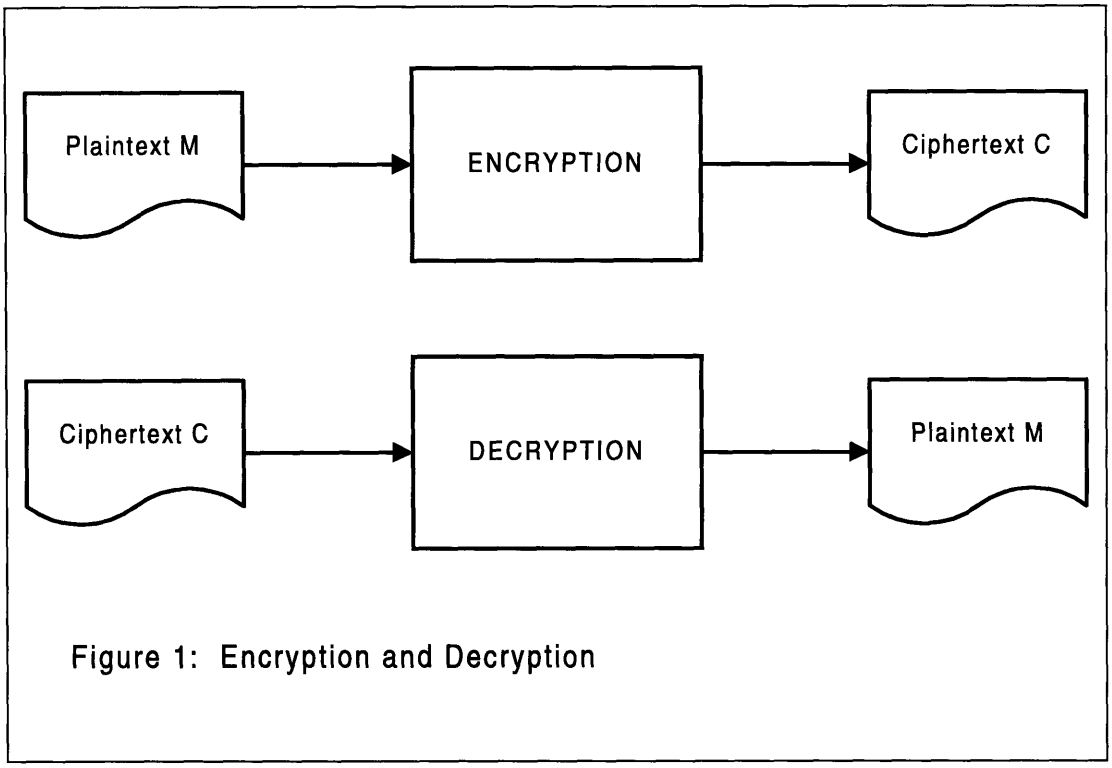

Figure 1: Encryption and Decryption

As shown in Figure 1 the original form of the message is usually called the plaintext, and the encrypted form is referred to as the ciphertext. In the literature, the plaintext message *M* is represented as a sequence of characters $m_1,m_2,...,m_n$, and the ciphertext message *C* as a sequence of characters $c_1,c_2,...,c_n$.

2.3 Secret Key Cryptography

As shown in Figure 2, secret key cryptography uses a single key S_K for both encryption and decryption. If only Alice and Bob share a secret key S_K , only they can encrypt and decrypt messages with that key. In addition, if Bob receives a message encrypted with S_K then he also knows that the message came from Alice, i.e. Alice's message is authenticated. Problems with secret key cryptography arise when the secret key is compromised because the interceptors can immediately decrypt all encrypted messages they have. Secret key cryptography also requires a large number of secret keys to be produced and securely distributed. The major advantage of secret key cryptography is

that its algorithms are usually fast and efficient. One of the most popular secret key algorithms is Data Encryption Standard (DES) introduced in the next section.

2.3.1 Data Encryption Standard (DES)

The Data Encryption Standard (DES) has been developed by the United States government for public use, and many hardware and software systems have been designed using the DES. Although it has been considered adequate for commercial use for many years, its security is recently becoming questionable.

The DES encrypts blocks of 64 bits of data producing 64-bit blocks of ciphertext. The main cryptographic techniques it uses are substitution and permutation, and the algorithm applies these two techniques one after the other for a total of 16 cycles. Although the key is 64 bits long, only 56 bits of the key are actually used.

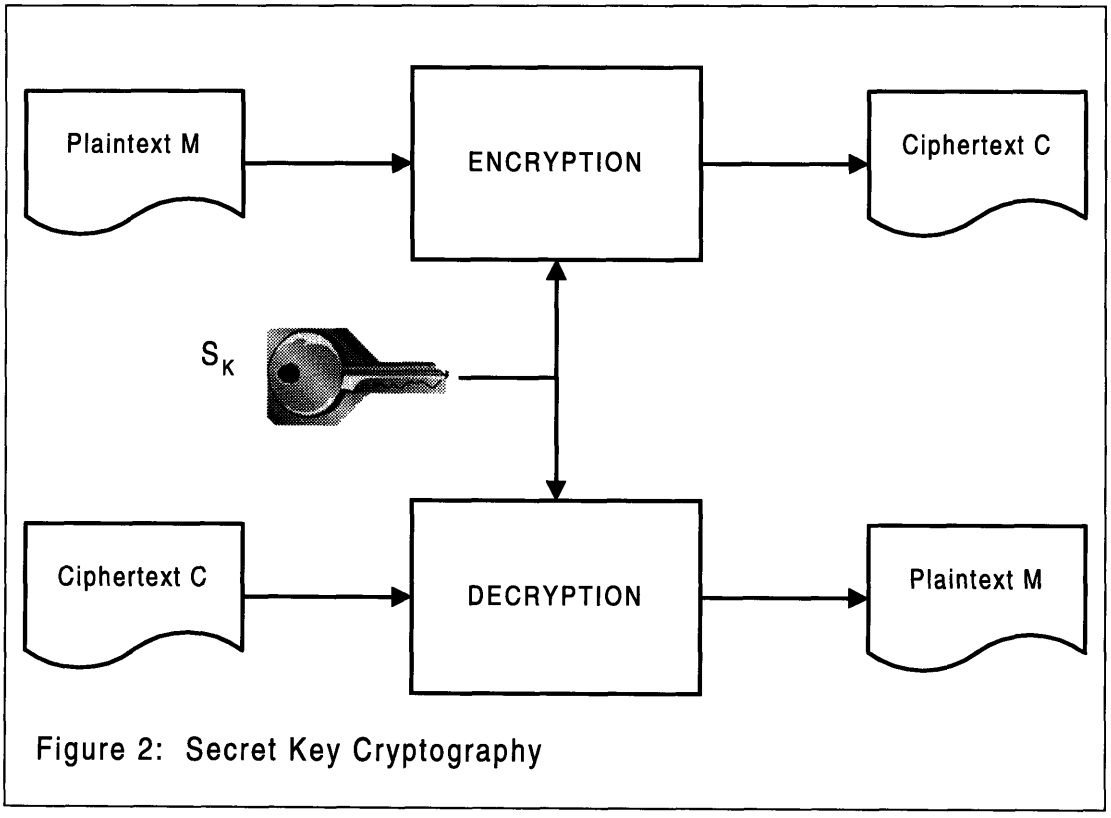

Figure 2: Secret Key Cryptography

2.4 Public Key Cryptography

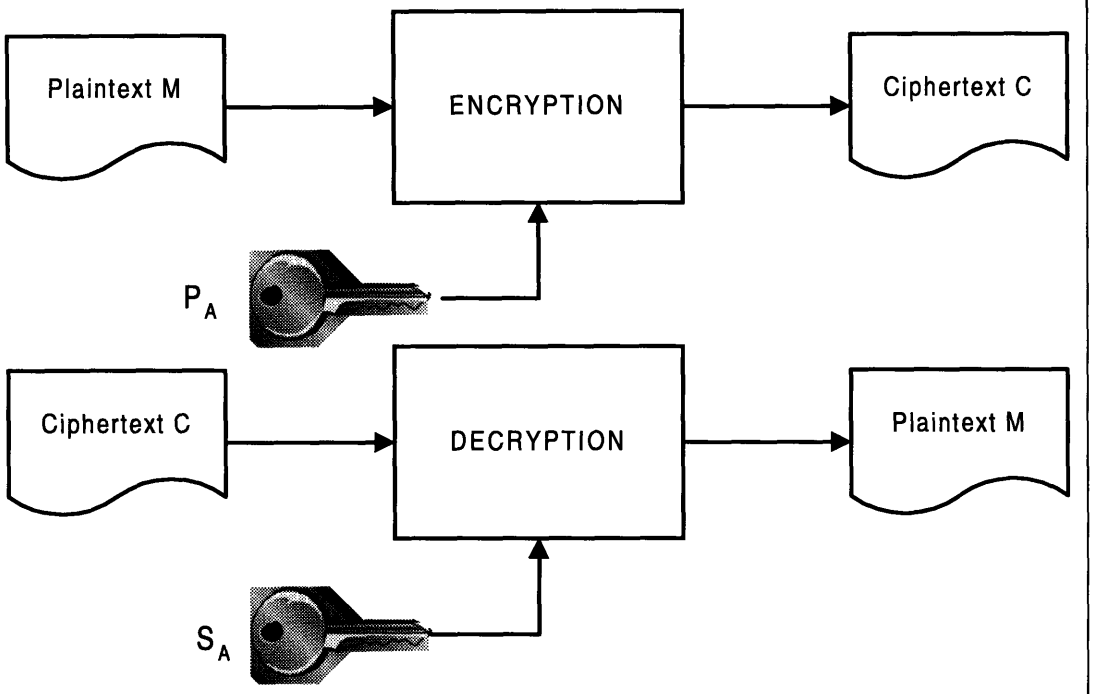

Figure 3: Public Key Cryptography

The main idea behind public key (asymmetric) cryptography is that the encryption and decryption capabilities are decoupled, and it is shown in Figure 3. Alice can have two keys P_A and S_A which are her public and secret keys, respectively. Alice can now give Bob (as well as everybody else) her public key P_A, and Bob can use it to encrypt a message M producing ciphertext $C=E(P_A,M)$. Although Bob encrypted the message for Alice, he nor anybody else can decrypt it. Alice, in possession of her private key, is the only party that can do it. Alice takes her private key S_A and decrypts the ciphertext C producing the original message M. $M=D(S_A,C)=D(S_A,E(P_A,M))$

Note that the secrecy of P_A is not required. Anyone can encrypt messages for Alice. In general, it should not be possible to figure out how to decrypt, even if you know everything how to encrypt, though *dictionary attacks* are still possible. Dictionary attacks involve searching through the space of possible plaintexts, encrypting all the candidates, and comparing the resulting ciphertexts against the ciphertext we are trying to decrypt.

One of the ways to protect against these kind of attacks is through an encryption technique called *Randomized Public Key Encryption*. The ciphertext C of a message M is equal to $(R \text{ XOR } M, E(P_A,R))$, where R is a random number. For longer messages a hybrid between public and secret key cryptography is used most often. The ciphertext is a pair $(DES(K,M), E(P_A,K))$. This scheme offers the advantages of both public key cryptography (flexibility) and of classical cryptography (speed).

2.4.1 Rivest-Shamir-Adelman (RSA) Encryption

RSA algorithm was invented at MIT in 1978 by Rivest, Shamir and Adelman [9]. It is based on modular arithmetic and the assumption that it is hard to factor a product of two large prime numbers. Although it has been a subject of an extensive cryptanalysis effort since its invention, no serious security flaws have been found.

2.4.1.1 RSA Algorithm

First, two large (usually 512 bits) prime numbers p and q are generated, and they are multiplied producing number $n = p * q$ (1024 bits long). A number e, relatively prime to $\varphi(n) = (p-1)(q-1)$ and a number d solution to $d * e = 1 \bmod \varphi(n)$ are generated. A pair of values (n, e) becomes a public key P_K and a pair of values (d, e) becomes a secret key S_K. A message M is encrypted, producing a ciphertext C by $C = M^e \bmod n$. A ciphertext C, producing the original message M is decrypted by $M = C^d \bmod n$.

2.4.1.2 RSA Validity

RSA algorithm performs modular arithmetic mod n, where $n = p * q$. Since the party producing the keys knows p and q, it can calculate $\varphi(n) = (p-1)(q-1)$ and can also find a number e which is relatively prime to $\varphi(n)$. In addition, it can calculate a number d which is a solution to $d * e = 1 \bmod \varphi(n)$. Therefore, for any x, $x^{d*e} = x \bmod n$. Since RSA encryption produces the ciphertext C by raising M to the *e-th* power *mod n*, and decryption raises the ciphertext C to the *d-th* power, we get $(M^e)^d \bmod n = M^{(e*d)} \bmod n = M$; RSA decryption reverses RSA encryption.
More in-depth analysis of the RSA algorithm is given in Appendix A.

2.4.1.3 RSA Security

The major premise behind RSA's security is that it is hard to factor a product of two large prime numbers. With today's best known methods it would take 30,000 MIPS-years to factor a 512-bit product of two prime numbers.

It is quite easy to see if we can factor n of the RSA's public *key (n,e),* we can deduce secret key *(d,e).* If we know p and q which are the prime factors of n, we can calculate $\varphi(n)$ and find d which is a multiplicative inverse of e, i.e. we can find the secret key *(d,e).*

It is unknown whether RSA algorithm can be broken using some method other than factoring n.

2.5 Types of Attacks on Cryptographic Functions

Depending what information is available to the cryptanalyst, four basic attacks on cryptographic functions and systems are: ciphertext only, known plaintext, chosen plaintext, and chosen ciphertext attacks.

2.5.1 Ciphertext Only Attack

In the ciphertext only attack, an adversary has seen and stored ciphertext messages being transmitted over insecure channels. Although the adversary does not know what the plaintext of a message might be, the adversary can search through all possible keys until something resembling the plaintext message is found. This attack is sometimes referred to as a recognizable plaintext attack.

2.5.2 Known Plaintext Attack

Known plaintext attack assumes that the adversary has in its possession a list of pairs of plaintexts and their corresponding ciphertexts. This attack can be very successful when the list of words used is rather limited. For example, if the attacker knows the ciphertexts corresponding to messages "Buy" and "Sell", he or she will be able to decrypt all subsequent messages having the same ciphertexts.

2.5.3 Chosen Plaintext Attack

During a chosen plaintext attack on a cryptosystem, an adversary can produce specific plaintext messages and have them encrypted by the cryptosystem under the attack. For example, an attacker of an ATM might request several financial transactions to be processed through the ATM and then tap the communications lines between the ATM and the bank. In this way, he or she can obtain the ciphertext of the requested transactions.

2.5.4 Chosen Ciphertext Attack

In the chosen ciphertext attack , an adversary can choose ciphertext messages and have them decrypted by the cryptosystem being attacked.

2.6 Authentication

The main goal of authentication is to verify someone's or something's identity. Authentication establishes a framework for achieving two main security goals: secrecy and integrity. By verifying someone's identity, authentication can prevent unauthorized parties from accessing a resource; therefore, protecting the secrecy of the resource. In addition, authentication can verify whether a resource has been tampered; therefore, protecting the integrity of the resource. Examples of authentication are numerous. Users entering their names and passwords, digital signatures of electronic checks, and mutual verification of identities of two Web servers are all examples of authentication.

In this section we present two examples of data authentication methods: the message authentication code method and the digital signature method.

2.6.1 Message Authentication Code (MAC)

Message authentication codes are used for authentication of data. Having a message M and a key K we produce a message authentication code $MAC(M,K)$ with the following properties:

- for any message M and key K, it should be easy to compute $MAC(M,K)$, that is the MAC should be computable in polynomial time
- it should be computationally infeasible to find $MAC(M,K)$ without knowing K

- given $MAC(M,K)$, it should not be possible to find the original message M

From the above properties and under the assumption that Alice and Bob share a secret key K Bob can easily verify the authenticity of Alice's message M. Bob computes $MAC(M,K)$ and compares it against the one received from Alice. If they agree, Bob concludes that Alice sent the message M.

2.6.2 Digital Signatures

Digital signatures are the public key solution to authentication of data. Digital signatures reverse the roles of public and private keys. Assuming that S_A can be used to encrypt as well as decrypt (as in RSA), $Signature_A(M) = E(S_A,M)$. To verify the signature, anybody can use Alice's public key P_A to check whether message M is equal to $D(P_A,Signature_A(M))$. Digital signatures are electronic equivalent for handwritten signatures for electronic checks, contracts, etc. Digital signatures can also be combined with the public key encryption schemes in a way that a message can be first signed, by using the secret key of the sender, and then sealed by using the public key of the receiver of the message.

2.6.3 Certificates

In our previous discussion about digital signatures, we have assumed that Bob was given Alice's public key P_A. Now we should discuss how Bob can be assured that the key he has is definitely Alice's key and not of an impostor. One simple way is to get it through a secure channel (e.g. directly from Alice). Another way is by having a *Certification Authority* (CA) which digitally signs a *certificate* saying that Alice's public key really is P_A.

2.6.3.1 Certificate Format & Content

Although there are many certificate formats proposed containing different information, majority of the formats include the following information [2]:

- Certificate serial number.

  Every certificate has a unique serial number which is used to efficiently track

certificates. In particular certificate serial number becomes very useful when checking validity of a certificate against certificate revocation lists (see below).

- Subject distinguished name.

  Subject distinguished name is the name of the entity to whom the certificate and hence the public key belongs to.
- Subject digital signature algorithm identifier.

  This identifier specifies which digital signature scheme the subject uses to sign documents (e.g. RSA, DSS, El Gamal).
- Subject public key and parameters.
- Certificate validity period.

  All certificates are usually issued only for a limited length of time. After the certificate validity period expires, the certificate becomes invalid.
- Issuer (trusted party) distinguished name.
- Issuer digital signature algorithm identifier.

  This identifier specifies which digital signature scheme has been used for signing of this certificate (e.g. RSA, DSS, El Gamal).
- Issuer public key and parameters
- Issuer digital signature

2.6.3.2 Certificate Revocation

Although certificates are useful for easy and efficient verification of public keys, there revocation is quite cumbersome. The main mechanism for certificate nullification are *certificate revocation lists* which are issued on a regular basis by the certifying authorities. Every time a certificate is used to verify somebody's public key, a certificate revocation list of the certificate issuer is checked as well. If a certificate has been compromised in some way, its name will appear on a certificate revocation list, and it will be no longer accepted as valid.

2.6.3.3 Certification Hierarchy

Although Bob can now verify Alice's public key through a certificate, he still needs to verify the certificate itself. If he does know the public key of the certificate issuer, he can

use it to verify the certificate. However, if he does not know the issuer's public key, he needs to obtain and verify the issuer's public key, and so on. This process continues recursively until Bob obtains a certificate of the CA whose public key he has and trusts. At this point he should be able to verify the chain of the certificates back to Alice, and at the end verify Alice's public key.

From the above description, it is should be obvious that CAs need to be organized in some manner, so that the certificate chain always leads to a CA whose public key is known and trusted by everybody. There are several different approaches present in the literature, but most of them organize CAs into a tree, that is they form a hierarchy.

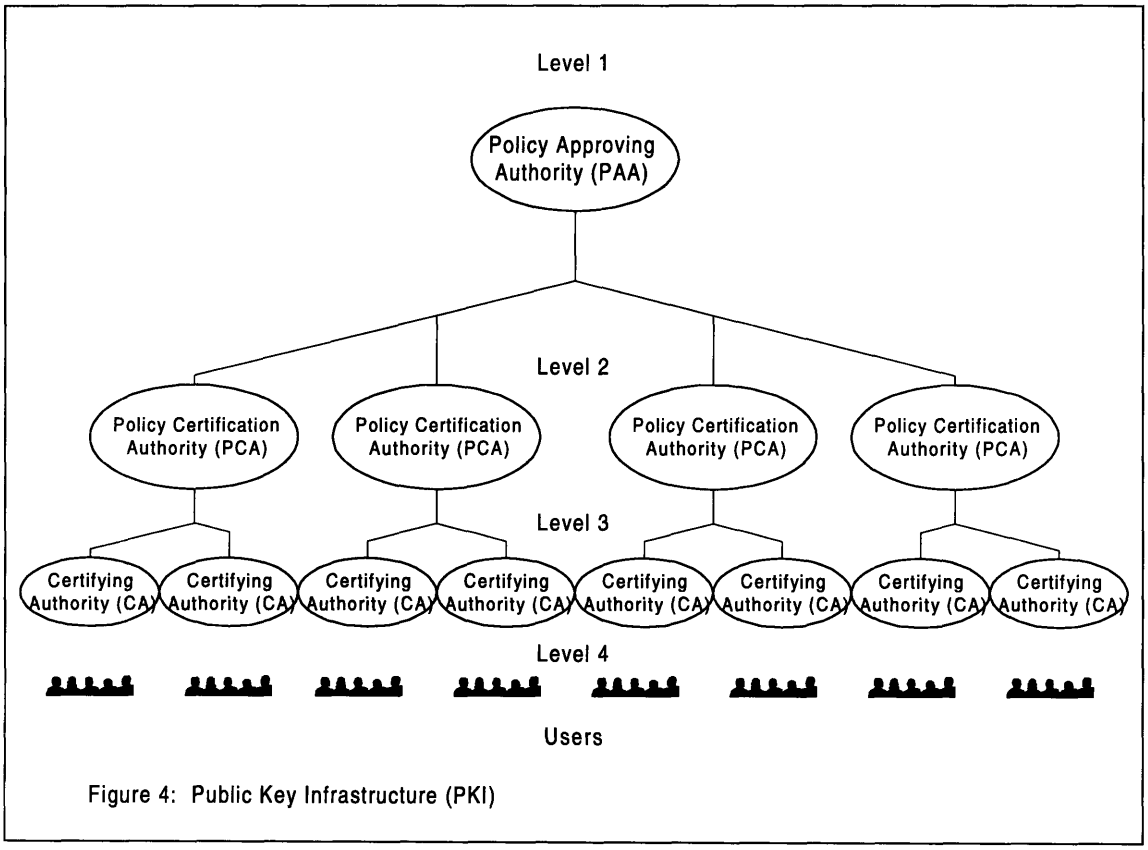

Figure 4: Public Key Infrastructure (PKI)

As an example of certification hierarchy, we will discuss *Public Key Infrastructure* (PKI) proposed by Santosh Chokhani [2]. As depicted in Figure 4, Chokhani proposes four level hierarchy based on the Internet model. At the level 1 is the Policy Approving Authority (PAA) whose public key is known to everybody. The PAAs main responsibility is to approve the certificate issuance policies developed by lower level nodes. There has to be sound basis for certification of a new level node, such as its size, organization and

uniqueness of the certificate issuance policy. The second level node is the Policy Certification Authority (PCA). It establishes the certificate issuance policy for the community it will serve and issues certificates to level 3 nodes. The level 3 nodes are named Certifying Authorities (CAs), and they actually issue certificates to users. The leafs of the tree, that is the fourth level nodes are the users themselves. There also might be present an Organization Registration Authority (ORA) between third and fourth levels which does not issue certificates but vouches for the identity and perhaps the affiliation of the certificate requesters.

2.6.4 Authentication Protocols

The goal of authentication protocols is to provide communicating parties with some assurance that they know each other's true identities. Most of the authentication protocols we discuss are two-party protocols. Depending whether the authentication protocol is one-way or a mutual authentication protocol, one or both party can conclude that the other party is legitimate. In addition, most of the authentication protocols are also authenticated key exchange protocols. These protocols, in addition to performing authentication, produce a secret key known only to two parties being authenticated.

Chapter 3: Existing Security Systems

The major goal of this chapter is to familiarize us with two existing security systems: the one developed by Lotus for Lotus Notes and the one developed by Netscape for the security of its Web products. In particular, we will present and discuss the Lotus Notes authentication protocol, and the Secure Sockets Layer (SSL) authentication protocol. Although those two protocols accomplish the same goal, i.e., the authentication of communicating parties, they were designed and implemented differently.

Although some of the material presented in the next section about the Notes authentication protocol has appeared previously in [6] and [12], a lot of the material is new. Most of the new material is based on information gathered from principal architects and implementers of the Notes authentication protocol. In addition, in our research we used some protocol design documents and even code itself, but none of the information being presented is proprietary.

Most of material about the Netscape's SSL protocol is based on the Netscape's proposal from June, 1995 [13] and the answers which were gathered through the mailing list "ssl-talk@netscape.com" during the period from August to December, 1995. Since the SSL protocol is a nonproprietary protocol, we are able to present and discuss it great detail.

3.1 Lotus Notes Security

Lotus Notes security provides two-way authentication of clients and servers and establishment of a shared session key. It also provides for electronic mail security through encryption and digital signatures. The main storage of security related information about

users comes from NotesID files which are generated and assigned to every Notes user. Lotus Notes security uses flat and hierarchical certificates, and the encryption algorithms involved are RSA, MD2, RC2, and RC4.

3.1.1 NotesID files

NotesID files contain user name, public and private keys, certificates, and the license information. All information contained in NotesID files is encrypted with the user's password. Notes security system based on NotesID files is more secure than a regular name / password system, since it requires users to posses physical entities (NotesID files) in addition to having knowledge of their passwords. Also, it is more advantageous than a directory based scheme, since the security information does not have to be somewhere on the network. In case of network failures, local security protocols can still proceed uninterrupted.

3.1.2 Lotus Notes Authentication Protocol

Lotus Notes authentication protocol is an end-to-end protocol which mutually authenticates two Notes entities (e.g., Notes client and a Notes server, two Notes servers). In order to verify the authenticated parties, it uses certificates issued by private certifying authorities of a particular Notes installation. At the end of the authentication protocol, both authenticating parties share a secret key known only to them which they can use to communicate securely in subsequent transactions.

3.1.2.1 Functional Description (Notes Authentication Protocol Flow)

The Lotus Notes authentication protocol consists of four distinct phases: the Hello Phase, the Certificate Exchange Phase, the Long-term Secret Production Phase, and the Session Key Production Phase. In all our description of the Notes protocol we assume that Alice initiates the authentication and Bob responds (i.e. Alice represents a client and Bob represents a server in client to server transactions, and Alice represents an initiating server and Bob represents a responding server in server to server transactions). The functional description of four phases follows:

1. The Hello Phase is used to initiate the authentication. Alice starts the authentication by sending a message to Bob containing her name, her public key, and a random challenge which will be used later to authenticate Bob. Upon receiving Alice's message, Bob responds with the similar message containing his name, his public key, and his random challenge.
2. The Certificate Exchange Phase allows both authenticating parties to receive a certificate from a certifying authority (CA) they trust. Since it is assumed that both parties have multiple certificates issued by different certifying authorities, Alice, for example, can send names of the CAs she trusts to Bob, and Bob can send one of his certificates to Alice. He chooses a certificate issued by one of the CAs Alice trusts. This phase is skipped if the parties have communicated in the past and have remembered their *long-term secret* (i.e., a large random number).
3. The Long-term Secret Production Phase produces a random value called a *long-term secret* which both parties are expected to save and use to reduce the number of authentication message exchanged in their future transactions. Bob creates, encrypts, and signs the long-term secret, and he sends it to Alice. This phase is skipped also if the parties have communicated in the past and have cached their long-term secret.
4. The Session Key Production Phase is a mandatory phase of every run of the authentication protocol. It produces a *session key* used as the secret key for the symmetric encryption of all messages of the current session.

3.1.2.2 Detailed Description

Lotus Notes authentication protocol is given in Figure 5. During the Hello Phase, i.e. in first two steps of the protocol, Alice and Bob send their public keys P_A and P_B, and their challenges (random numbers) R_A and R_B. The Certificate Exchange Phase follows, and in the third and the fourth steps, Alice provides Bob with the names of the Certifying Authorities (CAs) she trusts, and Bob provides Alice with the names of the CAs he trusts. In steps 5 and 6, Alice sends Bob a certificate that Bob will trust, and Bob sends Alice a certificate that Alice will trust. Both of them check validity of the received certificates and

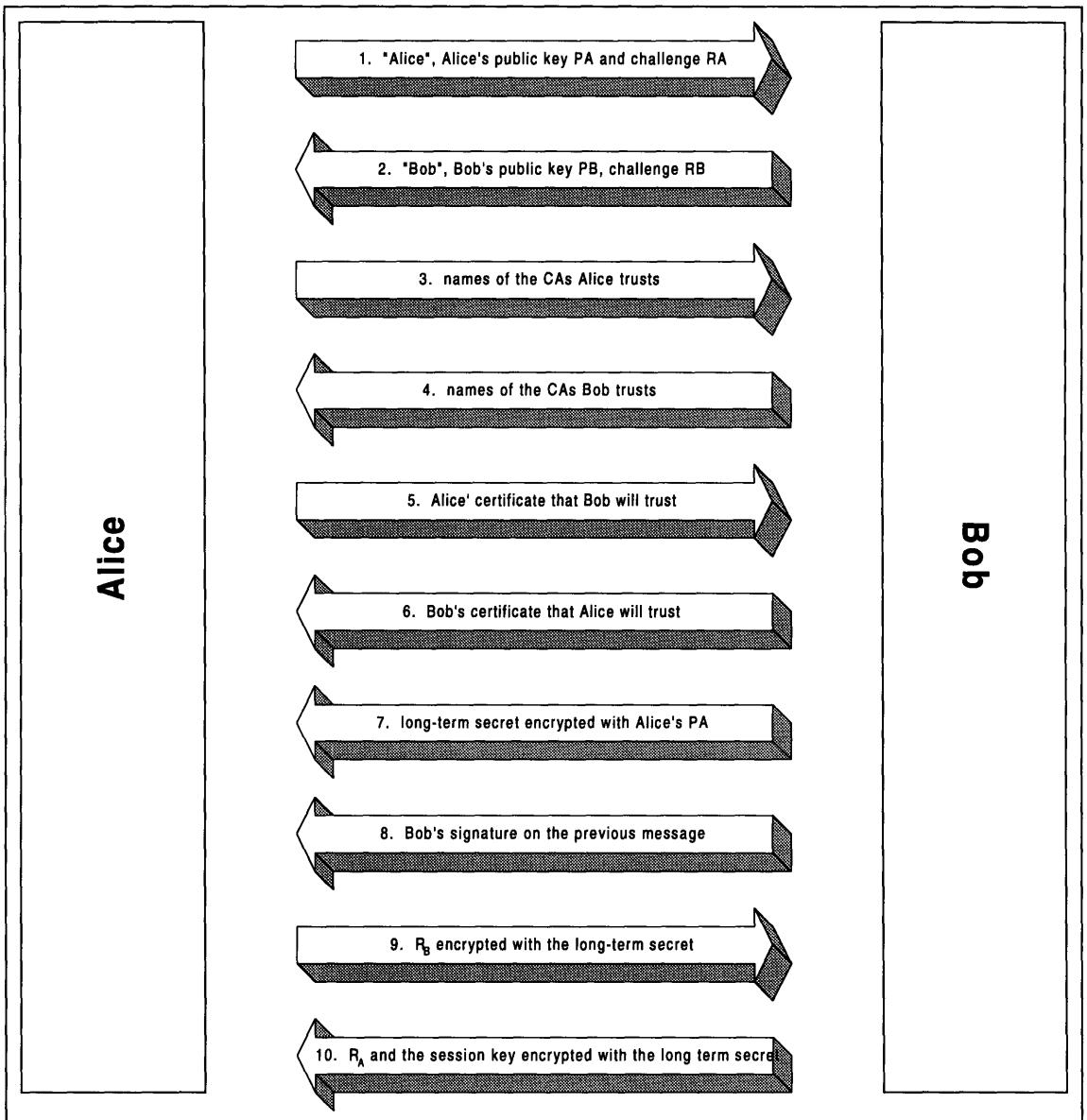

Figure 5: Lotus Notes Authentication Protocol

if the certificates provide satisfactory information (see section on certificates) continue to the next phase. At the beginning of the Long-term Secret Production Phase Bob creates a *long-term secret* (a random number) which will be used for future transactions between Alice and him. In step 7 he sends the *long-term secret* encrypted with Alice's public key P_A and in step 8 his signature on that message. Upon receipt, Alice decrypts long-term secret using her private key and checks Bob's signature on that message. The Session-key Production Phase starts with step 9; Alice uses the *long-term secret* to encrypt Bob's

challenge R_B and sends it to Bob. Bob decrypts the message, checks the correctness of his random challenge, and sends his last message to Alice. This message contains Alice's challenge R_A and the *session key*, both encrypted with the *long-term secret*. Alice then decrypts the message, checks the validity of the challenge R_A, and keeps the *session key*.

At the end of Lotus Notes authentication protocol, both Alice and Bob are assured that they are truly communicating with each other. In addition, they possess a *session key* known only to them which they can use in all subsequent transactions of that session, and a *long-term secret* which they can use to cut the number of messages needed for future runs of the authentication protocol.

3.1.2.3 Variations

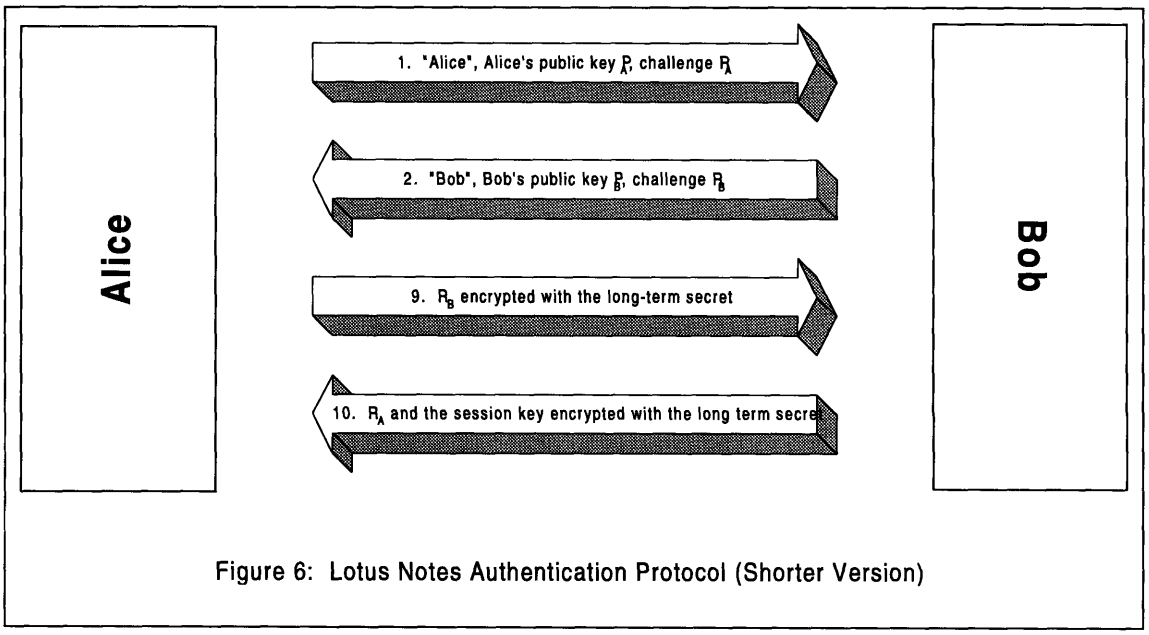

Figure 6: Lotus Notes Authentication Protocol (Shorter Version)

The Notes authentication protocol presented in the previous section rarely performs all 10 steps. More often shorter, but still secure, version of the protocol is executed. All 10 steps are necessary when Alice and Bob want to establish a connection for the first time. If they have communicated in the past, have exchanged and verified each other's certificates, and have kept their *long-term secret*, they can skip steps 3 through 8; therefore, performing a much shorter protocol presented in Figure 6. If in step 9 Bob receives his challenge R_B from step 2 encrypted with the long-term secret, he doesn't know which certifying authority vouched for Alice, but he does know that he trusted some

CA in the past and has created a *long-term secret* for Alice. Therefore, he knows that he is talking to Alice. Similarly, if Alice has kept the *long-term secret* for Bob, she doesn't need any information from steps 3 through 8 either. If in step 10 she receives her current *challenge* R_A from step 1 and the *session key* both encrypted with the *long-term secret*, she knows that the message came from Bob (because of the *long-term secret*), and also that the message is current (because of the *challenge* R_A from step 1)

Protocol also changes slightly when certificates for hierarchical names are used instead of flat names. If both Alice and Bob have hierarchical names, steps 3 and 4 are skipped, and steps 5 and 6 are augmented. In those steps Alice and Bob do not send single certificates, but they send a chain of certificates from their common root down to them. (see the section on certificates)

3.1.2.4 Discussion

Although there are no security reasons why Bob cannot send the data contained in steps 7 and 8 in a single step, there is a performance gain if he does. Both Alice and Bob can perform in parallel expensive operations of RSA decryption (Alice's decryption of the long-term secret) and of RSA signing (Bob's signature on that message).

In step 9 Alice authenticates herself to Bob. Alice had to know her secret key in order to be able to decrypt the long-term secret produced by Bob. Therefore, if the decrypted value of Bob's random challenge R_B corresponds to the original value sent by Bob in step 2, Bob knows that he communicated with Alice.

Similarly, in step 10 Bob authenticates himself to Alice, and in addition he creates and sends her a session-key. Since Bob has produced the long-term secret key and has provided his signature for it (in step 8), Alice knows that Bob created the long-term secret. If in step 10 she receives the correct value of her random challenge R_A from step 1, she knows that the message came from Bob, and that she can trust the session-key specified in the same message.

3.1.3 Certificates

There are two different kinds of certificates present in Lotus Notes: *flat certificates* issued for flat names, and *hierarchical certificates* issued for hierarchical names.

Since by definition, flat names don't have any structure, use of flat certificates is unstructured as well. If Alice wants Bob to accept her identity, she has to provide Bob with certificates he trusts. It is Alice's responsibility to acquire and store as many certificates as she needs. Since Notes stores certificates in users NotesID files, size of these files grow linearly in the number of certifying authorities producing large NotesID files.

To reduce the number of required certificates and to make their use more efficient, the hierarchical certificates are employed. Alice is given a certificate by the certifying authority (CA) responsible for that portion of the naming hierarchy. In addition, Alice's NotesID file holds a chain of certificates from the root CA down to her CA. In order to accept Alice's certificate, Bob needs a chain of certificates from a common ancestor of Bob and Alice down to Alice.

If a common ancestor does not exist, a *cross certificate* has to be created. The cross certificate is created by some ancestor of Bob creating a certificate for some ancestor of Alice. In this case, Notes does not keep the cross certificates in ID files, instead they are kept in the directory service (in Notes called the *Name and Address Book*). When Bob wants to authenticate Alice, he searches the Name and Address Book for a cross certificate from one of his ancestors to one of Alice's ancestors. If he finds none, he cannot authenticate Alice.

3.2 The Secure Sockets Layer (SSL) Protocol

The SSL authentication protocol (a protocol developed by Netscape) is an end-to-end protocol providing privacy and integrity between two communicating applications. It is built on top of the TCP network layer and underneath the application layer offering security for different network applications (e.g. HTTP, FTP, TELNET). Although both applications can act as clients and servers, for reasons connected with performance and functionality, one application is usually referred to as a client, and the other one as a server. The SSL protocol involves mandatory authentication of the server, and optional

authentication of the client. The designers of the protocol [13] state that the SSL protocol provides channel security with three basic properties:

1. The channel is private. After a simple handshake and the establishment of a secret key, all messages going over the network are encrypted. The SSL uses symmetric cryptography for data encryption (e.g., DES, RC4).
2. The channel is authenticated. Using public key cryptography the SSL protocol ensures that server authentication and optional client authentication.
3. The channel is reliable. The message authentication code (MAC) is included with every network message ensuring integrity and reliability. MD2 and MD5 are used for MAC computation.

3.2.1 Functional Description (SSL Handshake Protocol Flow)

The SSL Handshake protocol negotiates security parameters used in later transactions. It consists of five phases:

1. The Hello Phase is used to establish client and server capabilities in terms of the SSL version, encryption and key exchange algorithms which they support. In addition, this phase also allows the discovery of a *session-ID* which is used to reduce the number of steps necessary for the handshake protocol when the client and the server have kept information from communications in the past.
2. The Key Exchange Phase facilitates the exchange of key material between the client and the server. At the end of this phase, both parties share a *master key*.
3. The Session Key Production Phase produces the actual session keys for the current communication session.
4. When the RSA key exchange algorithm is used, Server Verify Phase is used to verify the server's discovery of the *master key*. If the RSA key exchange is not used, this phase is skipped.
5. The optional Client Authentication phase is used for client authentication, but only for the key exchange algorithms that do not already authenticate the client. In the current implementation this is only the RSA key exchange algorithm.

6. The Finished Phase involves both parties exchanging their finished messages acknowledging that they are satisfied with the authentication messages other party has provided.

3.2.2 Detailed description

In this section we adopt a "C-like" notation for specifying protocol messages.

For example, a message of the form

```
char CIPHER-SPECS-LENGTH-MSB
char CIPHER-SPECS-LENGTH-LSB
char CIPHER-SPECS-DATA[CIPHER-SPECS-LENGTH-MSB<<8 | CIPHER-SPECS-LENGTH-LSB]
```

specifies a message containing two 8-bit values and an array of 8-bit values. The length of the array is a 16-bit number produced by concatenating first two 8-bit values (MSB and LSB values).

3.2.2.1 Hello Phase:

The Hello Phase involves the exchange of CLIENT-HELLO and SERVER-HELLO messages in order to determine enhancement capabilities of the client and the server.

CLIENT-HELLO Message (sent in clear)

```
char MSG-CLIENT-HELLO
char CLIENT-VERSION-MSB
char CLIENT-VERSION-LSB
char CIPHER-SPECS-LENGTH-MSB
char CIPHER-SPECS-LENGTH-LSB
char SESSION-ID-LENGTH-MSB
char SESSION-ID-LENGTH-LSB
char CHALLENGE-LENGTH-MSB
char CHALLENGE-LENGTH-LSB
char CIPHER-SPECS-DATA[(CIPHER-SPECS-LENGTH-MSB<<8) | CIPHER-SPECS-LENGTH-LSB]
char SESSION-ID-DATA[(SESSION-ID-LENGTH-MSB<<8) | SESSION-ID-LENGTH-LSB]
char CHALLENGE-DATA[(CHALLENGE-LENGTH-MSB<<8 | CHALLENGE-LENGTH-LSB]
```

When the client initiates a connection with a server, it has to send the CLIENT-HELLO message. The message contains the SSL version (CLIENT-VERSION) and the cipher specifications the client supports (CIPHER-SPECS-DATA). The session identifier (SESSION-ID-DATA) is included if the client has had a communication with that server

in the past and has kept the session identifier in its cache. The challenge data (CHALLENGE-DATA) is mandatory and is used to authenticate the server.

SERVER-HELLO Message (sent in clear):

char MSG-SERVER-HELLO
char SESSION-ID-HIT
char CERTIFICATE-TYPE
char SERVER-VERSION-MSB
char SERVER-VERSION-LSB
char CERTIFICATE-LENGTH-MSB
char CERTIFICATE-LENGTH-LSB
char CIPHER-SPECS-LENGTH-MSB
char CIPHER-SPECS-LENGTH-LSB
char CONNECTION-ID-LENGTH-MSB
char CONNECTION-ID-LENGTH-LSB
char CERTIFICATE-DATA [CERTIFICATE-LENGTH-MSB<<8 | CERTIFICATE-LENGTH-LSB]
char CIPHER-SPECS-DATA [CIPHER-SPECS-LENGTH-MSB<<8 | CIPHER-SPECS-LENGTH-LSB]
char CONNECTION-ID-DATA [CONNECTION-ID-LEN-MSB<<8 | CONNECTION-ID-LEN-LSB]

When the server receives the CLIENT-HELLO message and the client has not specified the session identifier in its message, the server verifies that it can support the client version and one of the client ciphers. The server edits the cipher specs list removing all the ciphers which it cannot support and returns the new cipher specs list as a part of the SERVER-HELLO message. In addition, the server encloses its certificate, certificate type and the connection identifier.

If the client has specified a session identifier in the CLIENT-HELLO message, and the server has an identical session identifier for that client in its cache, the server returns SESSION-ID-HIT and omits the cipher specs and the server certificate information.

3.2.2.2 Key-Exchange Phase:

The Key-Exchange Phase is required if the client and the server have not found the correct session identifier in their caches, that is the SESSION-ID-HIT has not been asserted in the SERVER-HELLO message. The main purpose of the Key-Exchange Phase is to establish a MASTER-KEY which is a shared secret between the client and the server. The exact flavor of the exchanged messages depends on the key exchange algorithm chosen by the server. The CLIENT-MASTER-KEY message is used for an RSA key exchange algorithm, and the CLIENT-DH-KEY message is used for Diffie-Hellman style key exchanges (e.g., DH and FORTEZZA).

CLIENT-MASTER-KEY Message (sent partially in clear):

char MSG-CLIENT-MASTER-KEY
char CIPHER-KIND[3]
char CLEAR-KEY-LENGTH-MSB
char CLEAR-KEY-LENGTH-LSB
char ENCRYPT-KEY-LENGTH-MSB
char ENCRYPT-KEY-LENGTH-LSB
char KEY-ARG-LENGTH-MSB
char KEY-ARG-LENGTH-LSB
char CLEAR-KEY-DATA [CLEAR-KEY-LENGTH-MSB<<8 | CLEAR-KEY-LENGTH-LSB]
char ENCRYPT-KEY-DATA [ENCRYPT-KEY-LENGTH-MSB<<8 | ENCRYPT-KEY-LENGTH-LSB]
char KEY-ARG-DATA [KEY-ARG-LENGTH-MSB<<8 | KEY-ARG-LENGTH-LSB]

The client chooses randomly a value of the MASTER-KEY and divides it into two parts. One part becomes a public portion and the other part becomes a secret portion of the MASTER-KEY.

The CLIENT-MASTER-KEY message contains the clear and encrypted portions of the MASTER-KEY. It also contains the specifications of the cipher chosen by the client from the CIPHER-SPECS list.

If the chosen encryption algorithm needs an additional argument such as an initialization vector used by some encryption algorithms (e.g., CBC DES), it is provided in the KEY-ARG-DATA.

3.2.2.3 Session Key Production Phase:

The Session Key Production Phase produces the session keys for current communication session between the client and the server. If the non-token key-exchange algorithms are used, this phase does not require the exchange of messages between the client and the server. In contrast, since the tokens do not allow the keys to leave the tokens unencrypted, the client must create and communicate to the server the encrypted value of the session keys by sending to the server an additional CLIENT-SESSION-KEY message.

Session Key Production for Non-Token Algorithms:

After the client and the server have in their possession the MASTER-KEY, they can produce their session keys in the following way:

1. They create KEY-MATERIAL-0 and KEY-MATERIAL-1 values using the MD5 hash function.

   KEY-MATERIAL-0 := MD5 [MASTER-KEY, "0", CHALLENGE, CONNECTION-ID]

   KEY-MATERIAL-1 := MD5 [MASTER-KEY, "1", CHALLENGE, CONNECTION-ID]

2. The define their read and write keys:

   CLIENT-READ-KEY := SERVER-WRITE-KEY := KEY-MATERIAL-0 [0-15]

   CLIENT-WRITE-KEY := SERVER-READ-KEY := KEY-MATERIAL-1 [0-15]

Where KEY-MATERIAL-0 [0-15] means the first 16 bytes of the KEY-MATERIAL-0 data, with KEY-MATERIAL-0 [0] becoming the most significant byte of the CLIENT-READ-KEY and the SERVER-WRITE-KEY.

Session Key Production for Token Algorithms -- CLIENT-SESSION-KEY Message (sent partially in clear):

```
char MSG-CLIENT-SESSION-KEY
char CIPHER-KIND[3]
char CLEAR-KEY1-LENGTH-MSB
char CLEAR-KEY1-LENGTH-LSB
char ENCRYPT1-KEY-LENGTH-MSB
char ENCRYPT1-KEY-LENGTH-LSB
char CLEAR-KEY2-LENGTH-MSB
char CLEAR-KEY2-LENGTH-LSB
char ENCRYPT2-KEY-LENGTH-MSB
char ENCRYPT2-KEY-LENGTH-LSB
char KEY-ARG-LENGTH-MSB
char KEY-ARG-LENGTH-LSB
char RANDOM-NUMBER-LENGTH-MSB
char RANDOM-NUMBER-LENGTH-LSB
char CLEAR-KEY1-DATA [CLEAR-KEY1-LENGTH-MSB<<8 | CLEAR-KEY1-LENGTH-LSB]
char ENCRYPT-KEY1-DATA [ENCRYPT-KEY1-LEN-MSB<<8 | ENCRYPT-KEY1-LEN-LSB]
char CLEAR-KEY2-DATA [CLEAR-KEY2-LENGTH-MSB<<8 | CLEAR-KEY2-LENGTH-LSB]
char ENCRYPT-KEY2-DATA [ENCRYPT-KEY2-LEN-MSB<<8 | ENCRYPT-KEY2-LEN-LSB]
char KEY-ARG-DATA [KEY-ARG-LENGTH-MSB<<8 | KEY-ARG-LENGTH-LSB]
char RANDOM-NUMBER-DATA [RANDOM-NUM-LEN-MSB<<8 | RANDOM-NUM-LEN-LSB]
```

The client sends this message to the server in order to communicate one or two session keys to the server. Each session key contains two pieces: the CLEAR-KEY-DATA and the ENCRYPTED-KEY-DATA; the final value of the session key is the concatenation of the value of the CLEAR-KEY-DATA and the decrypted value of the ENCRYPTED-KEY-DATA.

If two keys are used then the KEY1 becomes the CLIENT-WRITE-KEY and the SERVER-READ-KEY; the KEY2 becomes the CLIENT-READ-KEY and the SERVER-WRITE-KEY. MASTER-KEY from the previous message is used to encrypt the ENCRYPTED-KEY-DATA, and the RANDOM-NUMBER-DATA is encrypted using KEY2. The RANDOM-NUMBER-DATA is used as the SECRET for MAC computations; the KEY-ARG-DATA is required when some kind of initialization data is needed by the encryption algorithms.

3.2.2.4 Server Verify Phase:

SERVER-VERIFY Message (sent encrypted):

char MSG-SERVER-VERIFY
char CHALLENGE-DATA[N-1]

The SERVER-VERIFY message is sent by the server in order to authenticate itself to the client. The CHALLENGE-DATA contains the encrypted value of the challenge sent by the client in the CLIENT-HELLO message. The key used for encryption of the CHALLENGE-DATA is the SERVER-WRITE-KEY which has been derived in the previous step from the MASTER-KEY. Only the legitimate server which has the secret key corresponding to the public key specified in its certificate could have decrypted the client message containing the MASTER-KEY and therefore created the correct SERVER-WRITE-KEY. Thus only the server and the client know the SERVER-WRITE-KEY, and if the value of the decrypted CHALLENGE-DATA is identical to the CHALLENGE-DATA sent by the client in the CLIENT-HELLO message, the client is assured that the server is the legitimate one.

3.2.2.5 Client Authentication Phase:

Client authentication messages are sent if the server wants to authenticate the client, and the key exchange algorithm did not already perform the client authentication (e.g., RSA key exchange). The first message (REQUEST-CERTIFICATE message) is sent by the server to request the certificate from the client. The second message (CLIENT-CERTIFICATE message) is the client's response to the server.

REQUEST-CERTIFICATE Message (sent encrypted):

char MSG-REQUEST-CERTIFICATE
char AUTHENTICATION-TYPE
char CERTIFICATE-CHALLENGE-DATA[N-2]

The server creates a random number challenge and sends the encrypted REQUEST-CERTIFICATE message to the client. The message contains the AUTHENTICATION-TYPE the server wants to use (e.g., MD5 hash function with RSA encryption) and some challenge data.

CLIENT-CERTIFICATE Message (sent encrypted):

char MSG-CLIENT-CERTIFICATE
char CERTIFICATE-TYPE
char CERTIFICATE-LENGTH-MSB
char CERTIFICATE-LENGTH-LSB
char RESPONSE-LENGTH-MSB
char RESPONSE-LENGTH-LSB
char CERTIFICATE-DATA [CERTIFICATE-LENGTH-MSB<<8 | CERTIFICATE-LENGTH-LSB]
char RESPONSE-DATA [RESPONSE-LENGTH-MSB<<8 | RESPONSE-LENGTH-LSB]

The client responds with the CLIENT-CERTIFICATE message after it has received the REQUEST-CERTIFICATE message from the server. The message contains the client certificate with its type, and the response data. Depending on the authentication type requested by the server, the response data contains different values. If the MD5 digital signatures are used with the RSA encryption, the client produces the RESPONSE-DATA in the following way:

1. ANSWER := MD5 [KEY-MATERIAL-0, KEY-MATERIAL-1, KEY-MATERIAL-2 (optional), CERTIFICATE-CHALLENGE-DATA, the server's signed certificate from the Server-Hello message).
2. RESPONSE-DATA := encrypted ANSWER using the CLIENT-WRITE-KEY.

3.2.2.6 Finished Phase:

CLIENT-FINISHED Message (sent encrypted):

char MSG-CLIENT-FINISHED
char CONNECTION-ID-LENGHT-MSB
char CONNECTION-ID-LENGTH-LSB
char HANDSHAKE-HASH-LENGTH-MSB
char HANDSHAKE-HASH-LENGTH-LSB
char CONNECTION-ID-DATA [CONNECTION-ID-LEN-MSB<<8 | CONNECTION-ID-LEN-LSB]

char HANDSHAKE-HASH-DATA [HAND-HASH-LEN-MSB<<8 | HAND-HASH-LEN-LSB]

The CLIENT-FINISHED message is sent by the client when the client is satisfied with the information provided by the server. The entire message is encrypted using the CLIENT-WRITE-KEY. The message contains the connection identifier the server has sent to the client in the SERVER-HELLO message, and the handshake hash. The handshake hash contains the hash value of all messages sent by the client during the entire handshake protocol. The client initializes the hash function at the beginning of the handshake communication and feeds into it every SSL record sent to the server.

SERVER-FINISHED Message (sent encrypted):

char MSG-SERVER-FINISHED
char SESSION-ID-LENGTH-MSB
char SESSION-ID-LENGTH-LSB
char HANDSHAKE-HASH-LENGTH-MSB
char HANDSHAKE-HASH-LENGHT-LSB
char SESSION-ID-DATA [SESSION-ID-LENGTH-MSB<<8 | SESSION-ID-LENGTH-LSB]
char HANDSHAKE-HASH-DATA [HAND-HASH-LEN-MSB<<8 | HAND-HASH-LEN-LSB]

The SERVER-FINISHED message is sent to the client when the server is satisfied with the client security handshake and it is ready to proceed with the transmission and the reception of the higher level protocol data. The server creates a new session identifier, stores it in its cache and includes it as a part of the SERVER-FINISHED message. The client and the server are supposed to store the session identifier and all other relevant information, so that they can reconstruct all security parameters needed to successfully and securely communicate in the future without going through the entire handshake protocol again.

Similar to the handshake hash of the CLIENT-FINISHED message, the handshake hash of this message is produced from all the messages sent by the server to the client.

3.2.3 Examples and discussion of SSL protocol runs

In this section we present and discuss two examples of SSL protocol executions running under different assumptions. In Example 1, our assumptions are that the client and the server do not have a common session-identifier in their caches, and that they use the RSA key exchange algorithm . In addition, in Example 1 they do not perform client

authentication. In Example 2, our assumptions are different, and we assume that they do have a session-identifier kept from a previous authentication session, and that they do perform client authentication.

3.2.3.1 Example 1: Assuming no session-identifier and the RSA key exchange:

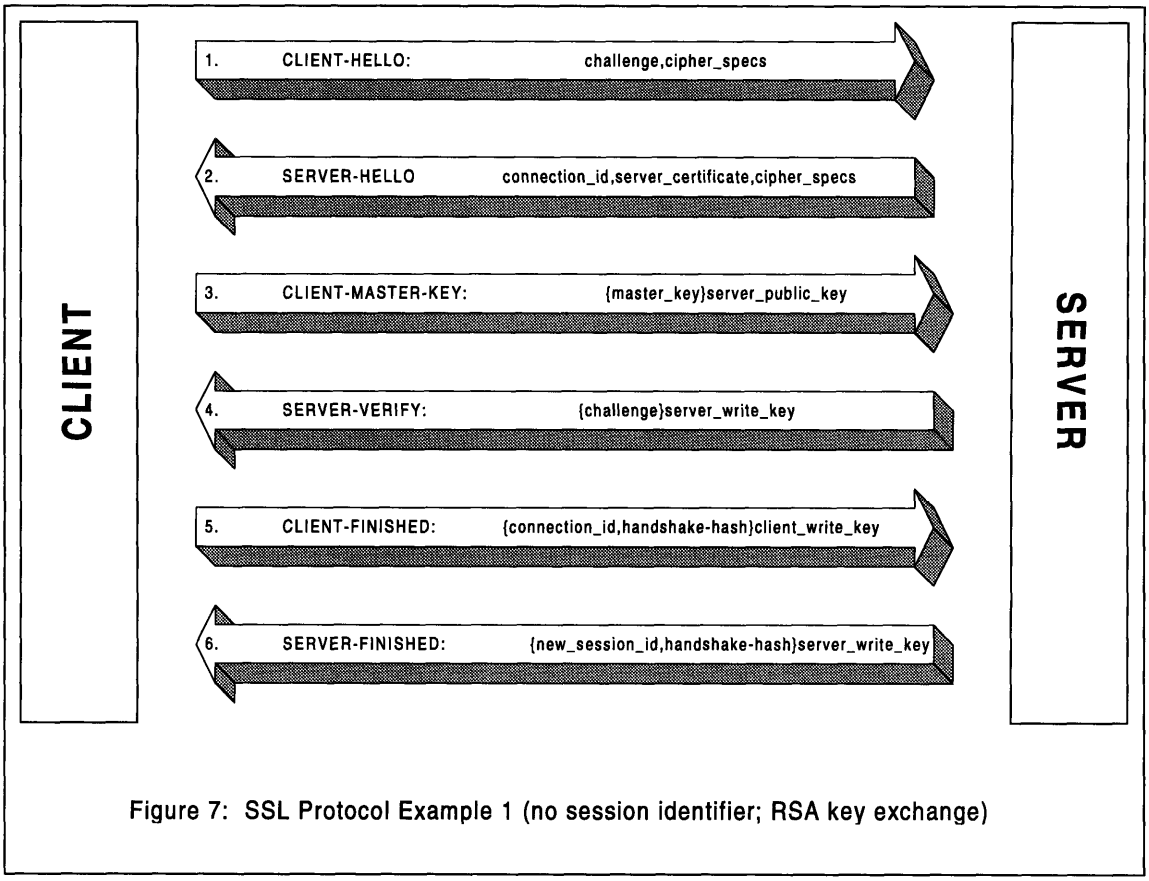

Figure 7: SSL Protocol Example 1 (no session identifier; RSA key exchange)

In Example 1 of the SSL protocol depicted in Figure 7, we assume that the client and the server have not communicated in the past or have no records kept from their previous communications. In addition, we assume that they use the RSA key exchange algorithm.

In this run of the protocol CLIENT-HELLO message contains a random number (*challenge*) and the list of key exchange and encryption algorithms it supports (*cipher_specs*). The server records the *challenge* and analyzes the *cipher_specs*. It removes the key exchange and encryption algorithms it does not support from the *cipher_specs* list, and if the list is not empty, it returns the list to the client as a part of the SERVER-HELLO message. In addition, the SERVER-HELLO message contains the *connection_id* and the server certificate. The client verifies the server certificate and

extracts the server's public key from the certificate. Then the client composes the CLIENT-MASTER-KEY message by encrypting a randomly selected *master_key* using the server's public key. After the server receives this message, both the server and the client share a *master_key* known only to them. At this point, the server can create the *server_write_key* and the *client_read_key* from the *master_key*; the client can create the *client_write_key* and the *server_read_key* also from the *master_key*.

To verify the server, the client has to receive the SERVER-VERIFY message from the server. It contains the client's *challenge* encrypted with the *server_write_key*; therefore only an authentic server will be able to decrypt it with the *server_read_key*. The client's last message CLIENT-FINISHED contains the *connection_id* and the *handshake-hash* encrypted with the *client_write_key* (known only to the client). The last message SERVER-FINISHED sent by the server contains *new_session_id* used for future transactions together and the *handshake-hash* both encrypted with the *server_write_key*.

3.2.3.2 Example 2: Assuming a session-identifier found and client authentication used

In Example 2 of the SSL protocol run presented in Figure 8, we assume that the client and the server have communicated in the past, and that they have stored their common *session_id*.

In addition to the *challenge* and the *cipher_specs* present in Example 1, the CLIENT-HELLO message of this example contains the *session_id* associated with the communication between this client and the server it tries to connect. Since the server has a record of communication with this client from the past, it returns only the *connection_id* and the *session_id_hit* flag. Server certificate is not necessary because the client has verified the server certificate in the past, and it also has the server's public key on file. The SERVER-VERIFY message in this example is the same as the one in Example 1. The server proves to the client that it knows the *server_write_key* by encrypting the client's challenge using that key. The CLIENT-FINISH message is identical to the one in Example 1, since the client still has to return the *connection_id* and the *handshake-hash* encrypted with the *client_write_key*.

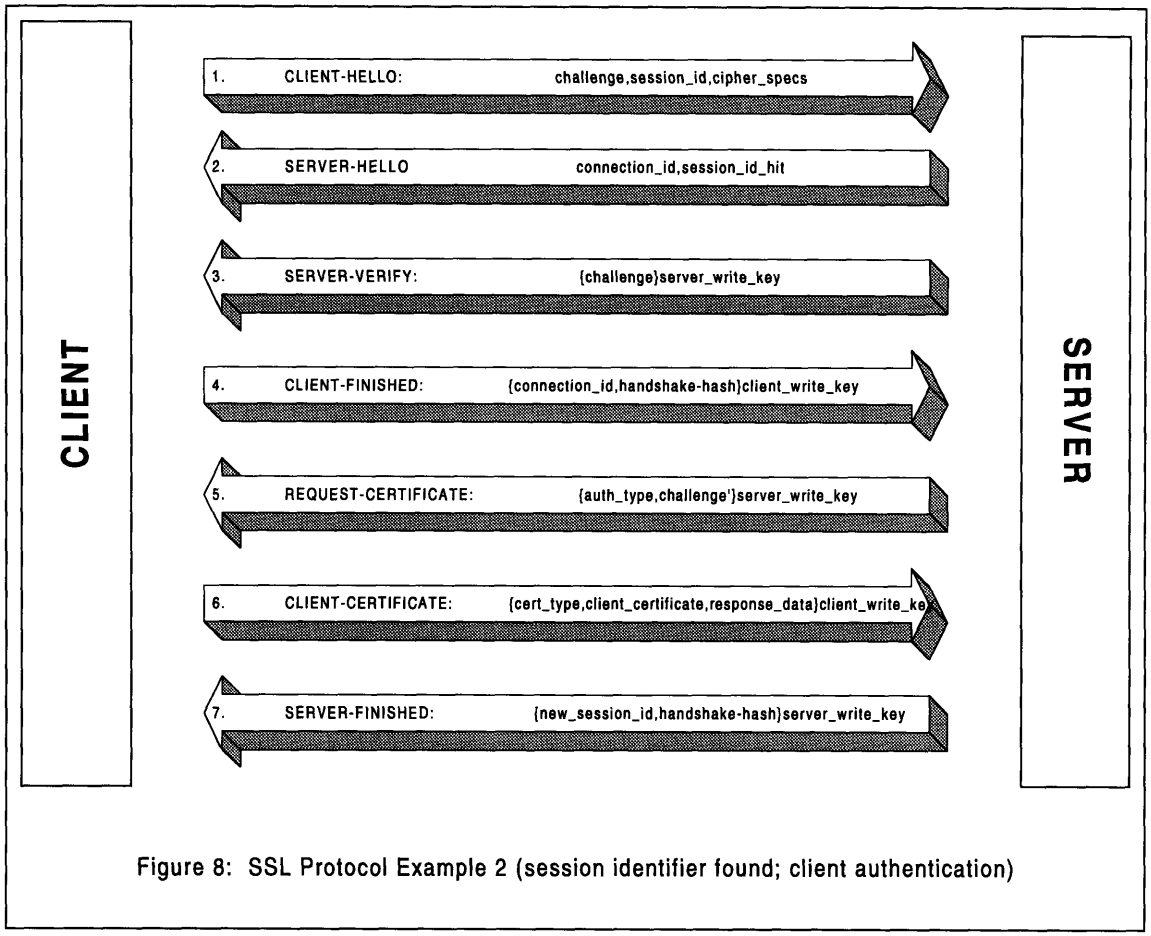

Figure 8: SSL Protocol Example 2 (session identifier found; client authentication)

REQUEST-CERTIFICATE and CLIENT-CERTIFICATE messages are needed for client authentication which was not present in Example 1. REQUEST-CERTIFICATE message contains the *auth_type* and the *challenge'* encrypted with the *server_write_key*. The *auth_type* specifies the client authentication protocol server wants to execute; the *challenge'* is the random number used to authenticate the client. If the client supports requested client authentication protocol, it returns the CLIENT-CERTIFICATE message. The CLIENT-CERTIFICATE message contains the client certificate with its type and some additional data. The entire message is encrypted with the *client_write_key*. When the server receives the CLIENT-CERTIFICATE message, it decrypts it using the *client_read_key* and verifies the client certificate. If correct, it returns the SERVER-FINISH message containing the *new_session_id* and the *handshake-hash* both encrypted with the *server_write_key*.

3.2.4 Strengths and Weaknesses of the SSL protocol and its current implementation(s)

The SSL protocol has several important strengths. It is flexible, it is an open standard, and it is an end-to-end protocol. However, some of its current implementations have weaknesses as well. The SSL protocol implementations often lack client authentication and require proprietary and flat certificates.

3.2.4.1 Flexibility

The SSL protocol allows great flexibility because applications are able to choose security parameters used in any particular run of the protocol. In the first couple of steps, applications can negotiate between several key-exchange and encryption algorithms. In version 3 of the protocol, supported key-exchange algorithms are RSA, Diffie-Hellman and KEA, and encryption algorithms are RC4, RC2, IDEA, DES. In addition, applications can decide whether client authentication needs to be performed.

Flexibility allows different parties to develop only a subset of the protocol they want and still be able to communicate with others. For example, they can choose which encryption algorithm to use based on the required level of security (RC4 vs. RC2), export controls (RC4 vs. DES), and the existence of patents and licensing agreements. The major problem of flexibility of the SSL protocol arises when two different communicating applications do not have any key-exchange or encryption algorithms in common and therefore cannot successfully execute the rest of the protocol.

3.2.4.2 Open Standard

Being a nonproprietary (open standard) protocol is a big advantage of the SSL protocol as well. Similar to flexibility, being an open standard protocol allows different parties to develop functionally similar or even identical applications without the need to license the protocol. Therefore, it allows any developer or user of the protocol to have an equal opportunity in successfully developing or using the protocol. However, the developers of an open standard protocol still have an advantage by being the first to implement it. And often the speed of implementation rather than its quality is the deciding factor in the success of a new product.

3.2.4.3 End-to-end

End-to-end security characteristic of an authentication protocol design is usually considered a very good characteristic because it is secure and efficient. All security related activities are performed at the end points of communication, and all communication points in between are not concerned with security. Although the design of end-to-end protocols assumes that the end points of communication are secure, the architecture of the Web sometimes cannot support that assumption. Servers and browsers themselves are often not very secure, and an adversary might choose to attack them instead of the communication channel between them. In this case, protocols allowing security of stored data in addition to security of transmitted data might be more applicable. One example of a protocol that provides security of stored as well as transmitted data is the S-HTTP protocol, but a discussion of this protocol is outside the scope of this study.

3.2.4.4 No Client Authentication

Although the SSL protocol has been designed to allow client authentication, the Netscape Commerce Server and the Netscape Navigator do not provide client authentication in their current implementations. A more primitive encrypted name/password authentication scheme is present, but it is a much weaker scheme than the one proposed by SSL.

3.2.4.5 Proprietary Certificates

The current implementation of the SSL protocol for the Netscape Commerce Server and Navigator requires that public keys of all certifying authorities are "hard wired" in both the client code (for server authentication) and the server code (for client authentication). This requirement does not allow organizations to create their own certifying authorities and provide certificates for their internal installations. In addition, it gives a monopoly to certain CAs, and it disallows people to choose which CAs they trust.

3.2.4.6 Flat Certificates

In addition to being proprietary, certificates produced by the Netscape endorsed CAs are flat. As explained in Chapter 2, the creation and management of flat certificates is not easy for systems with a large number of users, although the usage of flat certificates is

simple. A single certifying authority itself has to issue all certificates and keep track of all the revoked certificates as well. On the other hand, systems with hierarchical certificates can distribute the load of certificate issuance and management to different CAs in the certificate hierarchy based on a user as already discussed in the section concerning hierarchical certificates

3.3 Differences between Notes Authentication and Secure Sockets Layer Protocols

Notes authentication and SSL protocols have differences in three main areas. First, they differ in security levels and authentication goals. Second, they differ in the number of parties involved in authentication. And third, they differ in the number of certifying authorities they can support.

3.3.1 Security Levels and Authentication Goals

Both the Notes protocol and the SSL protocol are secure, and they accomplish identical security goals: the authentication of communicating parties and the establishment of a secure channel for data transmission. But there is a slight difference between their authentication goals and thereby their security levels. Notes and the SSL authentication protocols guarantee to each party (e.g., Alice) that the messages received and the included data (e.g., session key, long-term session key) came from another legitimate party (e.g., Bob). However, the SSL authentication protocol also guarantees that the other party is aware of the other's party trust (e.g., Bob knows that Alice trusts the authenticity of data received from him). Through the Finish Phase messages, the SSL protocol notifies parties involved that the other party accepted their authentication messages.

3.3.2 Mutual vs. One-side Authentication

The Notes authentication protocol mutually authenticates both communicating parties. In contrast, the SSL protocol requires the authentication of the server, but it leaves optional the authentication of the client. For Notes systems, it is important that both Notes clients and Notes servers are authenticated primarily because Notes users have extensive capabilities in terms of modifying data stored on the Notes servers. On the other hand, most Web users tend to only retrieve public data from Web servers and prefer to keep their anonymity; Web users cannot usually modify the data on the Web server. Server authenticity is much more important because it vouches for the authenticity of data on the server as well.

3.3.3 Single vs. Multiple Certifying Authorities

Notes infrastructure assumes that a single user can have multiple certificates issued by different certifying authorities. This allows a user to authenticate and receive data from sources which trust different certifying authorities. In contrast, the SSL protocol assumes that the client has a single certificate and that certificate will be either accepted or not by all servers on the Internet.

Notes allows *local universes of trust* to exist in a global space, while the SSL protocol assumes only a single *global universe of trust.*

Notes *local universes of trust* are useful for Notes systems because at times they play a role in authorization in addition to authentication of Notes users. Sometimes Notes gives public access to Notes databases requiring only that users are authenticated by a specific certifying authority; performing *implicit authorization.* In contrast, the SSL protocol never plays a role in *implicit authorization*; therefore, it can assume that a single certifying authority can be sufficient to authenticate a user.

Chapter 4: Design

In this chapter we accomplish the major goal of the study: the design of security mechanisms for the InterNotes WebPublisher. We present three designs of security mechanisms providing a range of security levels and having different levels of complexity. In the first design we take advantage of Notes database technology and its friendly user interface to construct a simple directory based scheme for user authentication. In the second design we use the Notes authentication protocol and the NotesID files to augment existing WebPublisher. In this design the WebPublisher performs strong authentication and authorization of its users. And finally, in the third design we exploit both the Notes and the SSL authentication protocols to design the SSL protocol for Lotus Notes.

4.1 Notes as the storage of security related information

In this design we are proposing a simple directory scheme for user authentication based on Notes database technology. All security related information is stored in a Notes database which is tightly coupled with a WebPublisher and a Web server. Although authentication protocols based on this design are much weaker than usual challenge-response based protocols with public key encryption, this design is expected to be easy and fast to implement. It requires very simple modifications to the WebPublisher and is largely independent of the Web server used.

4.1.1 Requirements

The main requirement of this design is to provide Web level security for the WebPublisher, i.e. *basic authentication* specified in the HTTP 2.0 protocol specifications. Since the WebPublisher does not provide any authentication nor authorization mechanisms at the present time, it would be beneficial to quickly implement a simple authentication scheme

providing a level of security of the Web itself. It is evident that this design would not have a long-lasting value; instead, it would be used as an exercise in building security modules for the WebPublisher and the Web, in addition to being a temporary solution for WebPublisher's authentication before more secure mechanisms are put in place.

4.1.2 Modules and their functions

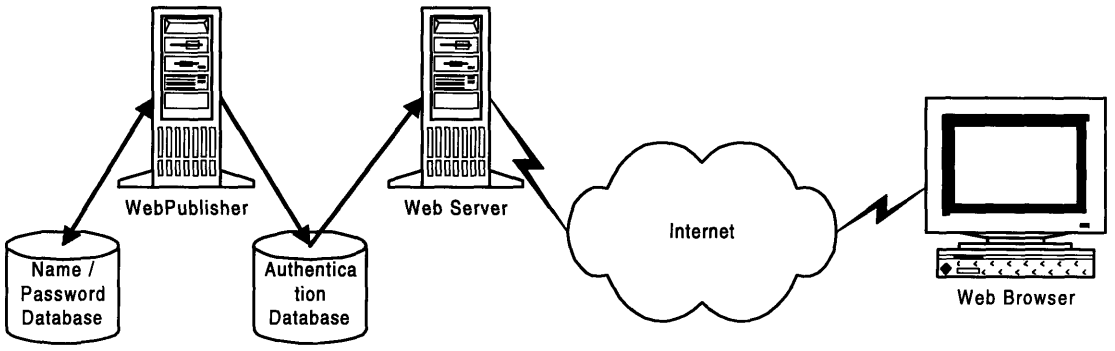

Figure 9: Notes as a storage of security related information

As shown in Figure 9 there are three functional and two storage modules present in the design. The functional modules are the Web server, the Web browser, and the WebPublisher, and the storage modules are the Notes Name and Password Database, and the Web Server Authentication Database

The Web server acts as a communication channel between the Web browser and the WebPublisher. The Web browser requests Notes documents from the WebPublisher; the WebPublisher transforms Notes documents into the HTML format (Web's native format) and serves them to the Web browser.

The Notes Name and Password Database is a Notes database containing information about users accessing Notes databases through the WebPublisher. This highly user-friendly database is maintained by the administrator of the WebPublisher, and it is also read by the WebPublisher. The WebPublisher takes information from this database to create entries in the Web Server Authentication Database. The Web Server Authentication Database is used by the Web server in order to authenticate users

requesting Notes documents through the WebPublisher. A more in-depth explanation of the modules follows.

4.1.2.1 Notes Name and Password database

A Notes Name and Password Database contains information about all users accessing Notes databases through the WebPublisher. Every user has an entry in the database containing his / her name, password (either in clear or hashed) and a list of databases with the corresponding views a particular user can access. A unique Name and Password Database is created together with the WebPublisher Configuration Database for every copy of the WebPublisher.

4.1.2.2 Web Server Authentication Database

Almost all of the Web servers which provide some kind of authentication and authorization mechanisms have simple databases (usually just text files) containing lists of user names, passwords, and *realms* (i.e., parts of the server a user can access). The entries in the Web Server Authentication Database are created from the information in the Name and Password Database by the WebPublisher.

4.1.2.3 Secure WebPublisher

The non-secure version of the WebPublisher creates HTML files corresponding to Notes databases. The HTML files are organized into directories, based on the Notes database, and subdirectories, based on the Notes *view* of the database from which they were created. In addition to the HTML files produced by the non-secure version of the WebPublisher, the Secure WebPublisher of this design creates the Web Server Authentication Database for a specific Web server out of the Name and Password Database.

4.1.2.4 Web Server

In this design a Web server has to perform both authentication and authorization of users. Therefore, the level of security of the entire design largely depends on the level of sophistication of the Web server employed. If the Web server only supports *basic authentication* as specified in the HTTP 2.0 specification, then all authentication information will be transmitted in clear which provides extremely low security. If the Web

server can establish a secure (i.e. encrypted) connection with a Web client, then the security level will be significantly higher because the passwords will be transmitted encrypted.

4.1.2.5 Web Browser

The Web browser needed for this scheme has to perform at least *basic authentication*, but similarly to above, the entire system becomes much more secure if the client and the server are able to establish an encrypted connection between each other.

4.1.3 Information flow (typical runtime of the entire system)

An administrator of the WebPublisher creates a Name and Password Database and an entry in the database (i.e. a Notes *document*) for every user. The document contains user name, password, and a list of databases and views a user can access. Upon the first startup of the WebPublisher, the WebPublisher takes user information out of the Name and Password Database and creates a custom Web Server Authentication Database. In particular, the WebPublisher transforms the lists of authorized databases and views into realms (as defined by the syntax of the Web server employed).

When a user tries to access Notes documents from the WebPublisher, it connects to the Web server, and the server and the user's browser perform user's authentication and authorization. The Web browser prompts the user with a dialog box asking from the user to enter his /her user name and a password. This information is transmitted back to the Web server (either in clear or encrypted) where it is checked against the Web Server Authentication Database. If the information user entered is correct, the Web server contacts the WebPublisher. It produces the requested documents which are then transmitted back to the browser. If the user information is incorrect, the server returns *not authorized* server error.

4.1.4 Security Analysis

It is clear that the level of security of this design is quite low for several reasons. First, the design allows transmission of secret data (e.g., passwords, Notes documents) in clear. Anybody who can eavesdrop the communication channel between the Web server and the

browser can obtain both authentication information and the sensitive data itself. Second, even when the server and the browser establish secure channel of communication, and the passwords are transmitted encrypted over the network, transmission of passwords still presents a security threat. If a user does not change passwords frequently, an adversary can mount a successful ciphertext-only attack on the communication channel between the server and the browser and perhaps recover the user's password. Once the password is compromised, it has to be changed in every Name and Password Database in which it exists.

4.2 Client Authentication

Although secure Web servers and browser providing secure channels of communication and server authentication are available on the market, strong client authentication is very often omitted. The main reason for its lack is the shortage of necessary infrastructure for its successful employment. The client authentication requires issuance and maintenance of client certificates, and this requirement is hard to achieve on a large scale.

In this design we propose a strong client authentication mechanism for the WebPublisher based on Notes authentication technology. Notes authentication technology has already in place powerful infrastructure for management of certificates. We also assume that the server authentication and the establishment of a secure channel have been accomplished through other means (e.g., SSL protocol).

4.2.1 Requirements

The main requirement of this design is the strong client authentication. Our definition of strong client authentication requires from a protocol to be a challenge-response protocol based on public key cryptography. In addition, we require that no private information about a user (e.g., user's password) is transmitted over the network; all public user information is extracted and verified via a user's certificate.

4.2.2 Modules and their functions

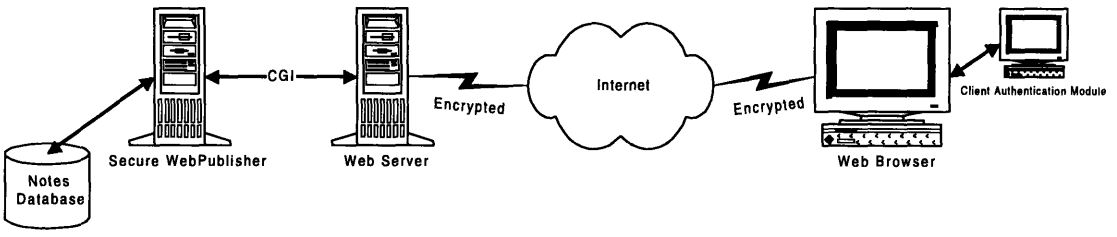

Figure 10: Client Authentication

As shown in Figure 10, in addition to the Web browser, the Web server and the WebPublisher, this design also requires a Client Authentication Module on the browser side which is needed for client authentication. There is a secure (encrypted) channel between the Web server and the browser, and an unencrypted CGI channel between the server and the WebPublisher. The WebPublisher receives both Notes data and authentication and authorization information from standard Notes databases. More detailed discussion of every module is given in following section.

4.2.2.1 Web Browser

A Web browser (client) has to have two main features. First, it has to be able to establish a secure channel with a Web server, and second, it has to support *client extensions*. Client extensions are add-in programs which extend capabilities of Web browsers. They usually receive data from clients, perform some computation, and return results back to clients.

4.2.2.2 Client Authentication Module

Client Authentication Module is a client extension module which is invoked by a Web browser when the server requests authentication of the client. The module receives authentication messages from the Web client, process them, and returns them to the client. It also extracts information about the user (e.g., user name, password, public and private keys, user's certificates) from user's NotesID file.

4.2.2.3 Web Server

Similarly to a Web browser, a Web server has to be able to establish a secure channel and to perform server authentication. In addition, it has to support *Common Gateway Interface (CGI)* standard for server extensions. The server receives requests from Web browsers in an encrypted form, decrypts them, and hands them over to the WebPublisher for further processing. In addition, it receives data from the WebPublisher, encrypts the data and transmits the encrypted data to appropriate clients.

4.2.2.4 Secure WebPublisher

It addition to creating HTML files corresponding to Notes databases, the Secure WebPublisher performs client authentication and authorization. It follows the Client Authentication Protocol presented in the next section to authenticate the client. If client authentication is successful, it performs client authorization using Notes *Access Control Lists* (ACLs) and returns appropriate documents or signals authorization errors.

4.2.3 Client Authentication Protocol

Client authentication protocol authenticates the client using already established secure channel between the client and the server. It is a challenge-response application level protocol, and it is similar to both Notes and SSL authentication protocols.

4.2.3.1 Functional Description

The client authentication protocol consists of two distinct phases: the Secure Channel Establishment Phase, and the Client Authentication Phase. The former establishes a secure channel of communication between the client and the server and authenticates the server; the latter authenticates the client.

Although the Secure Channel Establishment Phase is important, it is not further specified in this design. As discussed earlier, there are protocols, such as the SSL protocol which successfully accomplish the task of authenticating the server and establishing a secure link between the client and the server.

The Client Authentication Phase consists of two messages: a REQUEST-CERTIFICATE message and a CLIENT-CERTIFICATE message.

REQUEST-CERTIFICATE Message:

With this message a server requests a certificate from a client. It specifies the certifying authorities it trusts and gives a random challenge to the client.

CLIENT-CERTIFICATE Message:

After receiving the REQUEST-CERTIFICATE message, the client checks whether it has one of the certificates requested by the server, and if it does, it sends the certificate to the server in the CLIENT-CERTIFICATE message. In addition, it encrypts the challenge received from the server using its private key and sends it to the server.

After the server receives the CLIENT-CERTIFICATE message, it verifies the client's certificate and it extracts from it the public key of the client. It decrypts the random challenge using the client's public key, and if the decrypted value is equal to the one it has sent to the client in the REQUEST-CERTIFICATE message, the server completes the client authentication protocol, and it accepts the client as valid.

4.2.4 Information flow (typical runtime of the entire system)

When a user tries to access Notes documents which require client authentication and authorization, the client authentication protocol is initiated. First, the user's Web browser authenticates the server, and the server and the browser establish a secure channel between them. Second, the WebPublisher sends a message (through the Web server) to the Web browser forcing the browser to run the Client Authentication Module. Client Authentication Module and the Secure WebPublisher exchange necessary authentication messages and thereby authenticate the client. The WebPublisher associates the secure channel (established by the client and the server) with the newly authenticated user; any subsequent messages received through that channel are associated with that user. User requests for Notes documents are checked against Notes ACLs, and if the user is authorized to access those documents, they are returned to the user.

4.2.5 Security analysis

Security of the client authentication protocol depends on one main assumption: The underlying protocol (e.g., the SSL protocol) that performs server authentication and

establishes a secure channel has to be secure. If the channel is not secure, the client authentication is not secure either.

If the above assumption holds, we can informally deduce the security of the protocol as follows: If the server receives from the client a certificate which was issued by one of the certifying authorities the server trusts, the server is assured that the client's public key is valid. Since the server always creates a new random challenge for every CERTIFICATE-REQUEST message, and only a valid client has a secret key associated with the public key from the client's certificate, only a valid client can successfully encrypt the server's challenge. Therefore, when the server decrypts the response to its random challenge, and the decrypted value equals to the challenge value sent by the server in the previous message, the server knows it is talking to a valid client.

Our design is safe against man-in-the-middle attacks primarily because of the structure of the underlying protocol. Since the underlying protocol requires that one side is always authenticated and be called a server, an adversary cannot use messages received as a client in order to pretend to be a server. For example, let's assume that Alice is a server and is trying to authenticate Bob (a client). An adversary Charlie is trying to mount a man-in-the-middle attack. Charlie receives a challenge from Alice (pretending to be Bob) and would like Bob to encrypt it with Bob's private key. Bob will not encrypt the challenge for Charlie because Bob must first verify he is talking to Alice, and the verification will fail. The underlying protocol that will try to authenticate Charlie as a server will not succeed because Charlie does not know Alice's private key needed to perform authentication. Therefore, Charlie will not be able to continue and mount a man-in-the-middle attack.

4.3 SSL Protocol for Lotus Notes

Although Lotus Notes authentication protocol is a secure and efficient protocol used by Notes clients and servers, it is a proprietary protocol specifically tailored for Notes architecture.

Since Notes indirectly supports the Hyper Text Markup Language (HTML) and the Hyper Text Transfer Protocol (HTTP) through InterNotes WebPublisher, and we also assume

that it will natively support HTTP protocol in its future versions, we are proposing a design for a SSL protocol for Lotus Notes. Because the general SSL protocol architecture is flexible, and it allows to be extended, we are able to map the Notes authentication protocol onto the SSL protocol.

4.3.1 Requirements

The main requirement of this design is that Notes servers natively support the SSL protocol. In addition, the implementation of the design should involve the smallest amount of modifications of Notes servers and existing Notes infrastructure.

4.3.2 Design Issues

Although someone might think that correctness and efficiency of the Notes SSL protocol might be the biggest design issue, that is not the case. Since we are adapting the original SSL protocol "as-is", we are not concerned with arguing it being correct or efficient. Others have already performed that cumbersome task.

One the other hand, there are issues when thinking about using and porting Notes infrastructure on the Web and most of them are connected with certificates. In this section we present two main issues: the issue concerning the type and the format of certificates, and the issue concerning the availability of multiple certificates in Notes for a single Notes entity.

4.3.2.1 Certificate Types and Formats

Although Notes certificates contain the same information as the X.509 certificates, Notes certificate format is proprietary and does not conform to any public standards. The Version 3 of the SSL protocol supports X.509 (see Appendix B) and the PKCS 7 certificate formats. Since most of the certificates issued by the CAs providing certificates for the Web are in X.509 format, the Notes SSL protocol should use X.509 certificate format as well. This implies that all NotesIDs used by the Notes Web servers should contain X.509 certificates as well, and that utilities for automatic or semiautomatic recertification need to be provided also.

4.3.2.2 Multiplicity of Available Certificates

Notes authentication protocol allows parties to choose which certificates they trust in case multiple ones are available. For example, when a client wants to authenticate a server, the client sends the list of CAs which it trusts, and if the server has a certificate for at least one of them, it sends a single one to the client, thereby authenticating itself to the client.
On the other hand, the SSL protocol assumes that a single certificate chosen by the party being authenticated will either be sufficient to authenticate the party, or the authentication will fail. For example, if the client does not trust a certifying authority *C* because it does not have the CA's public key or considers the CA is not trustworthy, any certificate provided by servers certified by *C* will not be accepted. The server cannot ask in advance which CA the client trusts. It can only "guess", and if the guess is wrong the server authentication will fail.

4.3.3 Design Specs

In Figure 11 we present the most extensive version of the Notes SSL protocol which includes both server and client authentication and assumes that the client and the server have not communicated in the past. In the next few sections we give a detailed specification of each of the steps.

4.3.3.1 CLIENT-HELLO Message:

As explained in the section concerning the SSL protocol, the CLIENT-HELLO message contains data relevant to the SSL version the client supports, its cipher specs, the challenge data, and the session identifier (if known). Although we are designing the SSL protocol particularly for Lotus Notes, we should expect to receive other relevant messages from generic Web browsers. Therefore, it is useful to present and explain them all.

The most general version of the CLIENT-HELLO message has the following format:

char MSG-CLIENT-HELLO
char CLIENT-VERSION-MSB
char CLIENT-VERSION-LSB
char CIPHER-SPECS-LENGTH-MSB
char CIPHER-SPECS-LENGTH-LSB
char SESSION-ID-LENGTH-MSB

char SESSION-ID-LENGTH-LSB
char CHALLENGE-LENGTH-MSB
char CHALLENGE-LENGTH-LSB
char CIPHER-SPECS-DATA[(CIPHER-SPECS-LENGTH-MSB<<8) | CIPHER-SPECS-LENGTH-LSB]
char SESSION-ID-DATA[(SESSION-ID-LENGTH-MSB<<8) | SESSION-ID-LENGTH-LSB]
char CHALLENGE-DATA[(CHALLENGE-LENGTH-MSB<<8 | CHALLENGE-LENGTH-LSB]

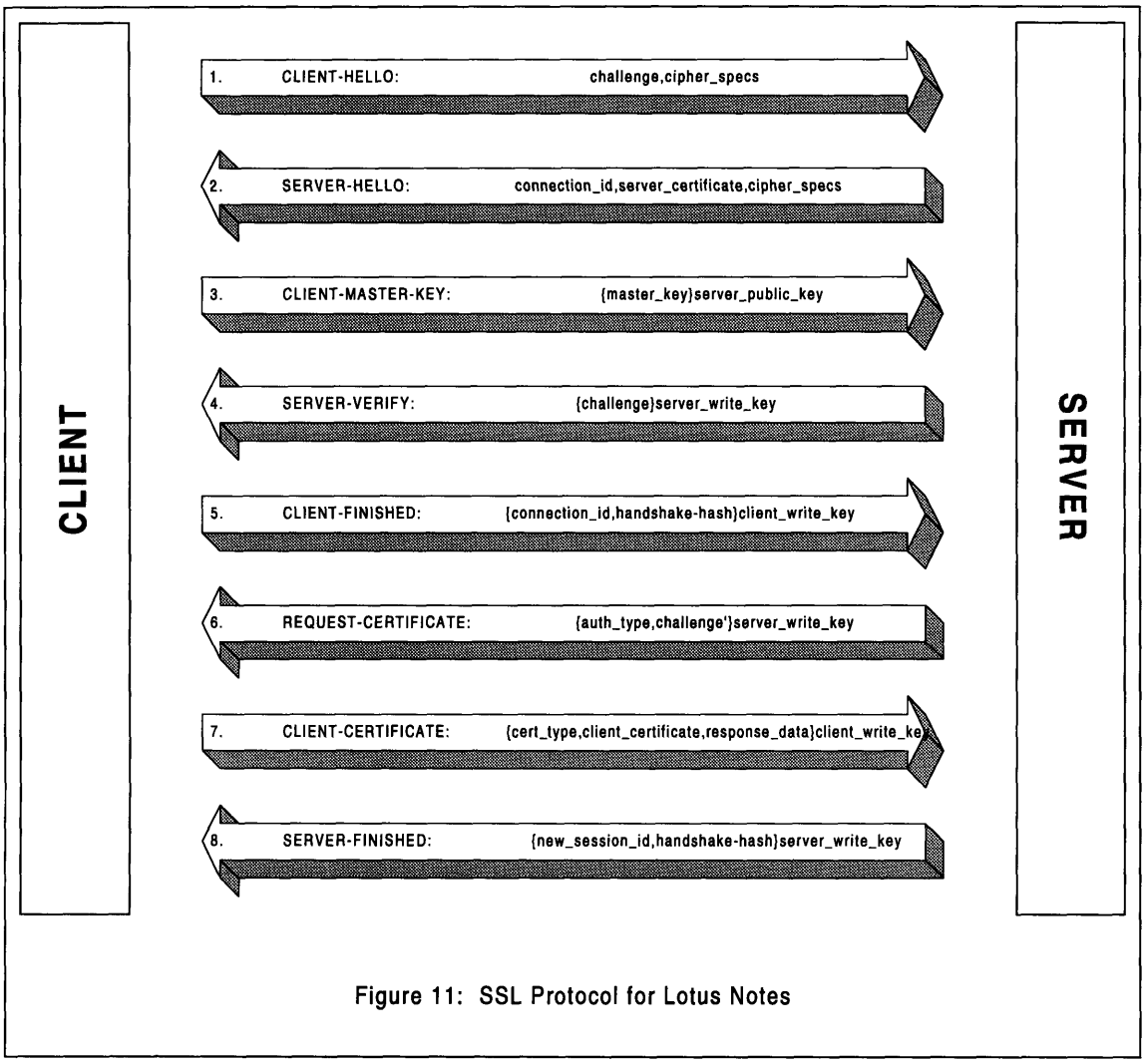

Figure 11: SSL Protocol for Lotus Notes

The syntax and values of the MSG-CLIENT-HELLO and CLIENT-VERSION parameters are given in Appendix C. If the client knows the SESSION-ID value, it will specify it, and the value will be 16 bytes long. The CHALLENGE-DATA is a randomly generated string between 16 and 32 bytes long. The server has to remember it and return it to the client in the SERVER-VERIFY message.

The CIPHER-SPECS-DATA defines the kind of security algorithms the client supports and is a list of SESSION-CIPHER-SPECs. Every SESSION-CIPHER-SPEC is 3 bytes long and has the following form:

char CIPHER-KIND-0
char CIPHER-KIND-1
char CIPHER-KIND-2

Although the SSL protocol supports several CIPHER-KIND values, we present ones we choose to support, namely the ones Notes already supports natively:

SSL_CK_RC4_128_WITH_MD5
SSL_CK_RC4_128_EXPORT40_WITH_MD5
SSL_CK_RC2_128_CBC_WITH_MD5
SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5
SSL_CK_NULL_WITH_MD5

The other CIPHER-KIND values defined by the SSL might be included in the messages received from generic Web clients, but since we will not support them they have no direct value for us and we choose not to discuss them in this context.

4.3.3.2 SERVER-HELLO Message:

char MSG-SERVER-HELLO
char SESSION-ID-HIT
char CERTIFICATE-TYPE
char SERVER-VERSION-MSB
char SERVER-VERSION-LSB
char CERTIFICATE-LENGTH-MSB
char CERTIFICATE-LENGTH-LSB
char CIPHER-SPECS-LENGTH-MSB
char CIPHER-SPECS-LENGTH-LSB
char CONNECTION-ID-LENGTH-MSB
char CONNECTION-ID-LENGTH-LSB
char CERTIFICATE-DATA [CERTIFICATE-LENGTH-MSB<<8 | CERTIFICATE-LENGTH-LSB]
char CIPHER-SPECS-DATA [CIPHER-SPECS-LENGTH-MSB<<8 | CIPHER-SPECS-LENGTH-LSB]
char CONNECTION-ID-DATA [CONNECTION-ID-LEN-MSB<<8 | CONNECTION-ID-LEN-LSB]

The SERVER-HELLO message is sent to the Web client after the server receives the CLIENT-HELLO message. Most of the content of a SERVER-HELLO message depends whether the server received a SESSION-ID value from the client and whether the server has exact SESSION-ID value in its cache. If this is not the case, the server sets the SESSION-ID-HIT flag to zero, packages up its certificate, its cipher spec list and the connection identifier. The cipher spec list is an updated version of the cipher spec list

received from the client. The server removes from the list all algorithms which it does not want to support. The connection identifier is a string of randomly selected bytes between 16 and 32 bytes long.

If the server and the client have correct SESSION-ID value in their caches, the server sets the SESSION-ID-HIT flag to a positive value and does not send its certificate nor the cipher spec list.

4.3.3.3 CLIENT-MASTER-KEY Message

The CLIENT-MASTER-KEY message is a part of the Key Exchange Phase presented in the section about the general SSL protocol. With this message the client communicates to the server the value of the MASTER-KEY which is a shared secret between the client and the server. The message contains following values:

char MSG-CLIENT-MASTER-KEY
char CIPHER-KIND[3]
char CLEAR-KEY-LENGTH-MSB
char CLEAR-KEY-LENGTH-LSB
char ENCRYPT-KEY-LENGTH-MSB
char ENCRYPT-KEY-LENGTH-LSB
char KEY-ARG-LENGTH-MSB
char KEY-ARG-LENGTH-LSB
char CLEAR-KEY-DATA [CLEAR-KEY-LENGTH-MSB<<8 | CLEAR-KEY-LENGTH-LSB]
char ENCRYPT-KEY-DATA [ENCRYPT-KEY-LENGTH-MSB<<8 | ENCRYPT-KEY-LENGTH-LSB]
char KEY-ARG-DATA [KEY-ARG-LENGTH-MSB<<8 | KEY-ARG-LENGTH-LSB]

The CIPHER-KIND field indicates which cipher the client has chosen from the server's CIPHER-SPECS list. The CLEAR-KEY-DATA contains the public part of the MASTER-KEY created by the client, and the ENCRYPTED-KEY-DATA contains the encrypted value of the secret part of the MASTER-KEY. The client uses the server's public key to encrypt the secret part of the MASTER-KEY. The KEY-ARG-DATA field is not used.

For brevity we include the Session Key Production Phase here as well, although there are no messages exchanged during that phase.

After the client and the server have in their possession the MASTER-KEY, they can produce their session keys in the following way:

1. They create KEY-MATERIAL-0 and KEY-MATERIAL-1 values using the MD5 hash function.

   KEY-MATERIAL-0 := MD5 [MASTER-KEY, "0", CHALLENGE, CONNECTION-ID]

   KEY-MATERIAL-1 := MD5 [MASTER-KEY, "1", CHALLENGE, CONNECTION-ID]

2. They define their read and write keys:

   CLIENT-READ-KEY := SERVER-WRITE-KEY := KEY-MATERIAL-0 [0-15]

   CLIENT-WRITE-KEY := SERVER-READ-KEY := KEY-MATERIAL-1 [0-15]

Where KEY-MATERIAL-0 [0-15] means the first 16 bytes of the KEY-MATERIAL-0 data, with KEY-MATERIAL-0 [0] becoming the most significant byte of the CLIENT-READ-KEY and the SERVER-WRITE-KEY.

4.3.3.4 SERVER-VERIFY Message:

char MSG-SERVER-VERIFY
char CHALLENGE-DATA[N-1]

In the CLIENT-HELLO message, the client sent N byte long CHALLENGE-DATA to the server. Now the server returns the value of the CHALLENGE-DATA (without one-byte header) encrypted with the SERVER-WRITE-KEY.

4.3.3.5 CLIENT-FINISHED Message:

char MSG-CLIENT-FINISHED
char CONNECTION-ID-LENGHT-MSB
char CONNECTION-ID-LENGTH-LSB
char HANDSHAKE-HASH-LENGTH-MSB
char HANDSHAKE-HASH-LENGTH-LSB
char CONNECTION-ID-DATA [CONNECTION-ID-LEN-MSB<<8 | CONNECTION-ID-LEN-LSB]
char HANDSHAKE-HASH-DATA [HAND-HASH-LEN-MSB<<8 | HAND-HASH-LEN-LSB]

The client sends this message when it is satisfied with the authentication messages server provided. The CONNECTION-ID value is a number from the SERVER-HELLO message. The HANDSHAKE-HASH is a 128-bit MD5 hash value produced by the client. The client computes the hash value by feeding the hash function with all the messages it has sent to the server. The server checks the received hash value against the hash value produced from the messages it has received from the client.
The entire message is encrypted using CLIENT_WRITE_KEY.

4.3.3.6 REQUEST-CERTIFICATE Message:

char MSG-REQUEST-CERTIFICATE
char AUTHENTICATION-TYPE
char CERTIFICATE-CHALLENGE-DATA[N-2]

Although the Client Authentication Phase consisting of REQUEST-CERTIFICATE and CLIENT-CERTIFICATE messages is optional for the standard execution of the SSL protocol, we make it a required phase of the Notes SSL protocol. The REQUEST-CERTIFICATE message sent by the server contains the authentication type server wants to pursue, i.e. SSL_AT_MD5_WITH_RSA_ENCRYPTION and a random challenge. The entire message is encrypted with the SERVER_WRITE_KEY.

4.3.3.7 CLIENT-CERTIFICATE Message:

char MSG-CLIENT-CERTIFICATE
char CERTIFICATE-TYPE
char CERTIFICATE-LENGTH-MSB
char CERTIFICATE-LENGTH-LSB
char RESPONSE-LENGTH-MSB
char RESPONSE-LENGTH-LSB
char CERTIFICATE-DATA [CERTIFICATE-LENGTH-MSB<<8 | CERTIFICATE-LENGTH-LSB]
char RESPONSE-DATA [RESPONSE-LENGTH-MSB<<8 | RESPONSE-LENGTH-LSB]

The CLIENT-CERTIFICATE message is a client response message to the REQUEST-CERTIFICATE message. Since we choose to support only X.509 certificate format and SSL_AT_MD5_WITH_RSA_ENCRYPTION authentication type, the RESPONSE-DATA is produced in the following way :

1. ANSWER := MD5 [KEY-MATERIAL-0, KEY-MATERIAL-1, KEY-MATERIAL-2(optional) CERTIFICATE-CHALLENGE-DATA, the server's signed certificate from the Server-Hello message).
2. RESPONSE-DATA := encrypted ANSWER using the CLIENT-WRITE-KEY.

The entire message is encrypted using CLIENT_WRITE_KEY.

4.3.3.8 SERVER-FINISHED Message:

char MSG-SERVER-FINISHED
char SESSION-ID-LENGTH-MSB
char SESSION-ID-LENGTH-LSB
char HANDSHAKE-HASH-LENGTH-MSB
char HANDSHAKE-HASH-LENGHT-LSB
char SESSION-ID-DATA [SESSION-ID-LENGTH-MSB<<8 | SESSION-ID-LENGTH-LSB]

char HANDSHAKE-HASH-DATA [HAND-HASH-LEN-MSB<<8 | HAND-HASH-LEN-LSB]

The server sends this message when it is satisfied with the authentication messages it has received from the client. It creates a new SESSION-ID which will be used in future runs of the authentication protocol with the same client. Similar to the CLIENT-FINISHED message, the SERVER-FINISHED message contains a HANDSHAKE-HASH which was produced from all the messages sent by the server to the client, and the HANDSHAKE-HASH has to be verified by the client through comparison with the hash value produced from all the messages client received from the server.
The entire message is encrypted using SERVER_WRITE_KEY.

4.4 Comparison of the Proposed Designs

In this section we compare the proposed designs in three major areas: in the area of security (i.e. the security level they provide), in the area of cost (i.e. the expected cost of their implementations), and in the area of importance (i.e. the importance of the problem they try to solve).

4.4.1 Security

The first design presented in section 4.1 provides very low security since it allows private information to be sent in clear over public networks. In contrast, two other designs provide a much higher level of security. They both use encryption and provide authentication of both servers and clients.

4.4.2 Cost

Although the first design provides low level security, it is very easy and cheap to implement. It requires minor modifications to the WebPublisher and can work with almost any Web server and browser.
On the other hand, the cost of implementations of the second and third designs are much higher. Client Authentication Design requires extensions to Web browsers which have not yet been standardized; i.e., it requires separate implementations for every type of a Web browser. It also requires understanding of Notes core code which deals with NotesID

files and Access Control Lists (ACLs). In addition, if not properly implemented, it can open security holes in Notes and bring bad publicity to Notes and Lotus.

Similar to the Client Authentication Design, the SSL Protocol for Lotus Notes Design is quite expensive to implement as well. It requires understanding of the entire Notes security code, in particular how Notes security fits in the entire Notes system. It requires knowledge how Notes handles encryption and authentication. In addition, understanding of the SSL protocol is essential.

4.4.3 Importance

All three designs have the same importance for Lotus, but for generally different reasons. The first design is important because it can bring a temporary solution to customers fast. If customers just need a way to identify people who are using the WebPublisher and restrict non-malicious users from seeing some WebPublisher data, they can use this design. The second design is important because it emphasize quality of Notes technology, in particular it shows how Notes technology can be used to handle client certificates and authentication in a non-Notes environment. This design provides security level of Notes for Web users and enables Notes users to use NotesIDs on the Web.

The third design is important because it emphasizes Lotus' commitment to the Internet through making Notes "speak" natively a Web protocol. Therefore, it makes the Web just another environment in which Notes lives.

Chapter 5: Conclusions

The main goal of the study to design systems which allow secure transactions for the InterNotes WebPublisher has been successfully accomplished. In addition, two other subgoals of researching computer and network security and analyzing Lotus Notes and Web security mechanisms have been achieved also.

5.1 Research of Computer and Network Security

The research of computer and network security was important because it provided means for accomplishing the main goal of the study successfully. In addition, it allowed us to produce a self standing document of current state-of-the-art of computer and network security which we presented in Chapter 2. This document can be useful to anybody interested in basics of computer and network security.

5.2 Analysis of Lotus Notes and Web Security Mechanisms

Similar to above, an analysis of Lotus Notes and Web security mechanisms was essential for accomplishing the main goal of the study. However, the analysis we have performed and presented in Chapter 3 can be used outside the context of this study as well. Many people are becoming interested in learning about and developing security mechanisms for the Web. Chapter 3 provides an analysis of one of the most popular proposal (the SSL protocol proposal) and compares it against already existing security mechanisms developed for Lotus Notes.

5.3 Design of Security Mechanisms for the InterNotes WebPublisher

In Chapter 4 we presented three different designs which provided security mechanisms for the InterNotes WebPublisher. The simplest design used Notes database technology to implement a simple directory scheme for user authentication; another design produced a specifications for the SSL protocol for Lotus Notes, and the most complex design added client authentication to the WebPublisher using Notes authentication technology.

As we discussed in Chapter 4, all three designs have good and not so good characteristics, such as high and low level security, low and high cost. If a design provides a good level of security, then it usually costs a lot to implement, and vice versa.

We expect that the WebPublisher development team can take this study as a guide and a design document in actual implementation of security mechanisms for the WebPublisher.

Appendix

Appendix A: Mathematical Background of the RSA Algorithm

Let's define group $Z_n = \{0, 1, ..., n-1\}$, and the order of the group Z_n^* to be $\varphi(n)$. Recalling Euler's Theorem, we have:

For all n and all a such that gcd(a,n)=1

$$a^{\varphi(n)} \equiv 1 \bmod n$$

If n is prime, then $\varphi(n) = n-1$, and we get a special case of the Euler's Theorem, Fermat's Theorem:

If p is prime then for all a such that gcd(a,p)=1

$$a^{p-1} \equiv 1 \bmod p$$

Furthermore, if $n = p * q$, where p and q are both primes,

$$\varphi(n) = (p-1)(q-1)$$

If we want to encrypt a message M so that $C = M^e \bmod n$, we need to select a number e, so that its inverse d is easy to compute. Because e and d are inverses $\bmod \varphi(n)$,

$$e * d \equiv 1 \bmod \varphi(n)$$

or

$$e * d = k * \varphi(n) + 1, \text{ for some integer } k.$$

From Fermat's theorem,

$$M^{p-1} \equiv 1 \bmod p,$$

and since $(p-1)$ is a factor of $\varphi(n)$,

$$M^{k*\varphi(n)} \equiv 1 \bmod p$$

Multiplying both sides by M we get,

$$M^{k*\varphi(n)+1} \equiv M \bmod p,$$

We can use the same argument for q, and we get

$$M^{k*\varphi(n)+1} \equiv M \bmod q$$

Combining last two results from above with $e * d = k * \varphi(n) + 1$, we get

$$\begin{aligned}(M^e)^d &= M^{e*d} \\ &= M^{k*\varphi(n)+1} \\ &\equiv M \bmod p \\ &\equiv M \bmod q\end{aligned}$$

Using Chinese Remainder Theorem

$$(M^e)^d \equiv M \bmod n$$

and e and d are inverse operations.

Appendix B: ASN.1 Syntax for X.509 Certificates

```
X.509-Certificate ::= SEQUENCE {        certificateInfo         CertificateInfo,
                                        signatureAlgorithm      AlgorithmIdentifier,
                                        signature               BIT STRING
                                }

CertificateInfo ::=     SEQUENCE {      version [0]             Version DEFAULT
v1988,
                                        serialNumber
CertificateSerialNumber,
                                        signature               AlgorithmIdentifier,
                                        issuer                  Name,
                                        validity                Validity
                                        subject                 Name,
                                        subjectPublicKeyInfo    SubjectPublicKeyInfo
                                }

Version         ::=     INTEGER         {       v1988(0)
                                        }

CertificateSerialNumber ::=             INTEGER

Validity        ::=     SEQUENCE        {       notBefore               UTCTime,
                                                notAfter                UTCTime
                                        }

SubjectPublicKeyInfo ::= SEQUENCE       {       algorithm               AlgorithmIdentifier,
                                                subjectPublicKey        BIT STRING
                                                }

AlgorithmIdentifier     ::= SEQUENCE    {       algorithm               OBJECT
IDENTIFIER
                                                parameters ANY DEFINED BY
ALGORITHM
OPTIONAL
                                                }
```

Appendix C: Protocol Constant Values

The SSL protocol specification [13] has been used as a reference for the preparation of this Appendix.

C.1 Protocol Version Codes

(missing from the June, 1995 version of the SSL specifications)

C.2 Protocol Message Codes

The following values define the message codes that are used by version 2 of the SSL Handshake Protocol:

SSL_MT_ERROR	:=	0
SSL_MT_CLIENT_HELLO	:=	1
SSL_MT_CLIENT_MASTER_KEY	:=	2
SSL_MT_CLIENT_FINISHED_V2	:=	3
SSL_MT_SERVER_HELLO	:=	4
SSL_MT_SERVER_VERIFY	:=	5
SSL_MT_SERVER_FINISHED_V2	:=	6
SSL_MT_REQUEST_CERTIFICATE	:=	7
SSL_MT_CLIENT_CERTIFICATE	:=	8

The following are the new messages added by version 3 of the SSL protocol:

SSL_MT_CLIENT_DH_KEY	:=	9
SSL_MT_CLIENT_SESSION_KEY	:=	10
SSL_MT_CLIENT_FINISHED	:=	11
SSL_MT_SERVER_FINISHED	:=	12

C.3 Error Message Codes

The following values define the error codes used by the ERROR message:

SSL_PE_NO_CIPHER	:=	0x0001
SSL_PE_NO_CERTIFICATE	:=	0x0002
SSL_PE_BAD_CERTIFICATE	:=	0x0004
SSL_PE_UNSUPPORTED_CERTIFICATE_TYPE	:=	0x0006

C.4 Cipher Kind Values

The following values define the CIPHER-KIND codes used in the CLIENT-HELLO and SERVER-HELLO messages:

SSL_CK_RC4_128_WITH_MD5	:=	0x01, 0x00, 0x80
SSL_CK_RC4_128_EXPORT40_WITH_MD5	:=	0x02, 0x00, 0x80
SSL_CK_RC2_128_CBC_WITH_MD5	:=	0x03, 0x00, 0x80
SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5	:=	0x04, 0x00, 0x80
SSL_CK__IDEA_128_CBC_WITH_MD5	:=	0x05, 0x00, 0x80
SSL_CK_DES_64_CBC_WITH_MD5	:=	0x06, 0x00, 0x40
SSL_CK_DES_192_EDE3_CBC_WITH_MD5	:=	0x07, 0x00, 0xC0

The following are new for the SSL version 3:

SSL_CK_NULL_WITH_MD5	:=	0x00, 0x00, 0x00
SSL_CK_DES_64_CBC_WITH_SHA	:=	0x06, 0x01, 0x40
SSL_CK_DEC_192_EDE3_CBC_WITH_SHA	:=	0x07, 0x01, 0xC0
SSL_KEA_RSA	:=	0x10, 0x00, 0x00
SSL_KEA_RSA_TOKEN_WITH_DES	:=	0x10, 0x01, 0x00
SSL_KEA_RSA_TOKEN_WITH_DES_EDE3	:=	0x10, 0x01, 0x01
SSL_KEA_RSA_TOKEN_WITH_RC4	:=	0x10, 0x01, 0x02
SSL_KEA_DH	:=	0x11, 0x00, 0x00
SSL_KEA_DH_TOKEN_WITH_DES	:=	0x11, 0x01, 0x00
SSL_KEA_DH_TOKEN_WITH_DES_EDE3	:=	0x11, 0x01, 0x01
SSL_KEA_DH_ANON	:=	0x12, 0x00, 0x00
SSL_KEA_DH_TOKEN_ANON_WITH_DES	:=	0x12, 0x01, 0x00
SSL_KEA_DH_TOKEN_ANON_WITH_DES_EDE3	:=	0x12, 0x01, 0x01
SSL_KEA_FORTEZZA	:=	0x13, 0x00, 0x00
SSL_KEA_FORTEZZA_ANON	:=	0x14, 0x00, 0x00
SSL_KEA_FORTEZZA_TOKEN	:=	0x15, 0x00, 0x00
SSL_KEA_FORTEZZA_TOKEN_ANON		

Any cipher types whose first byte of 0xFF are considered private and can be used for defining local/experimental algorithms. Interoperability of such types is a local matter.

C.5 Certificate Type Codes

The following values define the certificate type codes used in the SERVER-HELLO and CLIENT-CERTIFICATE messages:

SSL_CT_X509_CERTIFICATE	:=	0x01
SSL_CT_PKCS7_CERTIFICATE	:=	0x02

C.6 Authentication Type Codes

The following values define authentication type codes used in the REQUEST-CERTIFICATE message.

SSL_AT_MD5_WITH_RSA_ENCRYPTION	:=	0x01

C.7 Escape Type Codes

The following values define the escape type codes used by the SSL record protocol:

SSL_ET_OOB_DATA	:=	0x01

C.8 Upper / Lower Bounds

The following values define upper / lower bounds for various protocol parameters:

SSL_MAX_MASTER_KEY_LENGTH_IN_BITS	:=	256
SSL_MAX_SESSION_ID_LENGTH_IN_BYTES	:=	16
SSL_MIN_RSA_MODULUS_LENGTH_IN_BYTES	:=	64
SSL_MAX_RECORD_LENGTH_2_BYTE_HEADER	:=	32767
SSL_MAX_RECORD_LENGTH_3_BYTE_HEADER	:=	16383

References

[1] Burrows, M. Abadi, M and Needham, R., "A Logic of Authentication", *ACM Transactions on Computer Systems*, Vol. 8, No.1, February 1990, pp.18-36.

[2] Chokhani, S., "Toward a National Public Key Infrastructure", *IEEE Communications Magazine*, Vol. 32, No. 9, September 1994, pp. 70-74.

[3] Diffie, W. and Hellman, M. E., "New Directions In Cryptography", *IEEE Transactions on Information Theory*, Vol IT-22, 1976, pp. 644-654.

[4] Diffie, W. Van Oorschot, C.V. and Wiener, M.J., "Authentication and Authenticated Key Exchanges", *Designs, Codes, and Cryptography*, Vol. 2, No. 2, June 1992, pp. 107-125.

[5] El Gamal, T., "A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. IEEE Transactions on Information Theory, Vol. 31, No. 4, 1985, pp. 469-472.

[6] Kaufman, C. Perlman, R. and Speciner, M., *Network Security: Private Communication in a Public World*, Prentice Hall, New Jersey 1995.

[7] Lampson, B. Abadi, M. Burrows, M. and Wobber, E., "Authentication in Distributed Systems: Theory and Practice", *Theory of Computer Science*, Vol. 10, No.4, November 1992, pp. 265-310.

[8] Pfleeger, C. P., *Security in Computing*, Prentice Hall, 1989.

[9] Rivest, R.L. Shamir, A. and Adleman, L., "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of ACM*, Vol. 21, 1978, pp. 120-126.

[10]Shannon, C.E., "Communications Theory of Secrecy Systems" *Bell System Technical Journal*, Vol. 28, No. 4, October 1949, pp. 656-715.

[11]*Certificate Services: An RSA White Paper*, RSA Data Security, Inc., January 1994.

[12]*Notes Internals: Security*, Lotus Development Corp., May 1995.

[13]*Standards Documentation: The Secure Sockets Layer (SSL) Protocol*, Netscape Communications Corp., June 1995.

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