



Using of Security Protocols

by

Mr. Sirisak Sirisawatdiwat

A Final Report of the Three-Credit Course
IC 6997 E-Commerce Practicum

Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Internet and E-Commerce Technology
Assumption University

November 2003

St. Gabriel's Library, Au

Using of Security Protocols

by
Mr. Sirisak Sirisawatdiwat

A Final Report of Three-Credit Course
IC 6997 E-Commerce Practicum

Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Science
in Internet and E-Commerce Technology
Assumption University

November 2003

Project Title Using of Security Protocol for Payment


Name Mr. Sirisak Sirisawatdiwat


Project Advisor Rear Admiral Prasart Sribhadung


Academic Year November 2003

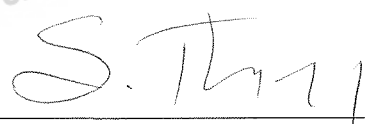
The Graduate School of Assumption University has approved this final report of the three-credit course, IC 6997 E-Commerce Practicum, submitted in partial fulfillment of the requirements for the degree of Master of Science in Internet and E-Commerce Technology.

Approval Committee:


(Rear Admiral Prasart Sribhadung)
Dean and Advisor


(Prof. Dr. Srisakdi Charmonman)
Chairman


(Dr. Ketchayong Skowratananont)
Member


(Assoc. Prof. Somchai Thayarnhyong)
CHE Representative

November 2003

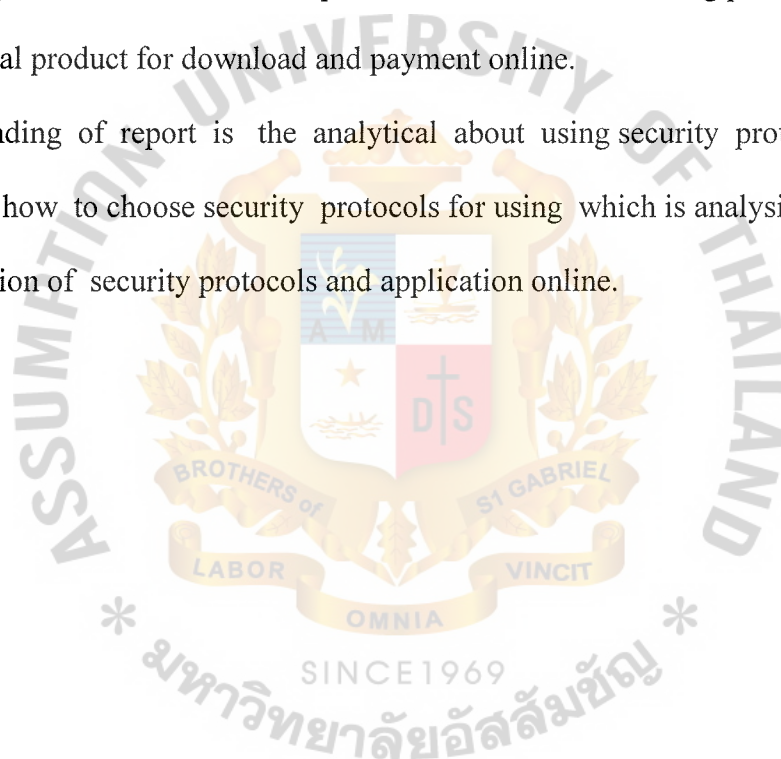
ABSTRACT

This project studies the feasibility of security protocols including type and explanation of security protocols, using security protocols for applications online, comparison of security protocols per each application.

In this report studies in 3 types of security protocols which are TLS protocol, SSL protocol and SET protocol and differential properties in each protocol.

For application online in this report studies in term of sending password, sending E-mail, digital product for download and payment online.

The ending of report is the analytical about using security protocols for each application, how to choose security protocols for using which is analysis from details and explanation of security protocols and application online.



ACKNOWLEDGEMENTS

I am indebted to the following people and organizations. Without them, this project would not have been possible.

I wish to express sincere gratitude to my advisor and dean of IEC, Rear Admiral Prasart Sribhadung. His patient assistance, guidance, and constant encouragement have led me from the inception to the completion of this research, and insights into the field of Computer Security, which is of utmost importance for our academic.

I wish to thank the entire faculty of the Graduate School, and Rear Admiral Prasart Sribhadung, a lecturer in IEC class for providing building blocks to the knowledge of E-Commerce Security Course.

Special appreciation is due to my family for their fervent and continuous encouragement and patience during this period of time. Above all, I am forever grateful to my parents whose willingness to invest in my future has enabled me to achieve my education goal.

TABLE OF CONTENTS

<u>Chapter</u>	<u>Page</u>
I. INTRODUCTION	1
1.1 Objective	
1.2 Scope	
1.3 Deliverable	
1.4 Research Methodology	
II. REVIEW	2
2.1 Background and Impact of Electronic Commerce	2 – 6
2.2 Shopping and Payment on Electronic Commerce	7 - 11
III. SECURITY PROTOCOLS	12
3.1 Transport Layer Security (TLS)	12 - 39
3.2 Secure Sockets Layer (SSL)	40 - 66
3.3 Secure Electronic Transaction (SET)	67 - 118
IV. USING OF SECURITY PROTOCOLS FOR APPLICATION	119
4.1 Sending Password to user	119 - 122
4.2 Sending E-mail	123 - 125
4.3 Sending Digital Product Online	126 - 130
4.4 Payment Online	131 - 135
V. CONCLUSION	136 - 137
BIBLIOGRAPHY	138

LIST OF FIGURES

<u>Figure</u>	<u>Page</u>
3.1 Message Flow in TLS Handshake Protocol	18
3.2 Compare the Performance of Secure Web Server	21
3.3 Timing Analysis of TLS Handshake Protocol	26
3.4 Crypto Overhead in Data Transfer	27
3.5 TLS Handshake with Cached Server Certificate	30
3.6 Message Flow in Optimized TLS Handshake Protocol	32
3.7 Estimated Latencies With and Without Certificate Caching	34
3.8 Secret-Key Cryptography	67
3.9 Public-Key Cryptography	68
3.10 Encryption Overview	73
3.11 Hierarchy of Trust	80
3.12 Detailed Diagrams	83
3.13 Guide to Diagrams, continued	84
3.14 Cardholder Registration	88
3.15 Cardholder Initiates Registration	89
3.16 Certificate Authority Sends Response	89
3.17 Cardholder Receives Response and Request Registration Form	91
3.18 Certificate Authority Processes Request and Sends Registration Form	92
3.19 Cardholder Receives Registration Form and Request Certificate	94
3.20 Certificate Authority Process Request and Creates Certificate	96
3.21 Cardholder Receives Certificate	97
3.22 Merchant Registration	98
3.23 Merchant Request Registration Form	99

<u>Figure</u>	<u>Page</u>
3.24 Certificate Authority Processes Request and Send Registration Form	100
3.25 Merchant Receives Registration Form and Request Certificate	101
3.26 Certificate Authority Processes Request and Create Certificate	103
3.27 Merchant Receives Certificate	104
3.28 Purchase Request	105
3.29 Cardholder Initiates Request	106
3.30 Merchant Send Certificate(s)	106
3.31 Cardholder Receives Response and Send Request	108
3.32 Merchant Processes Request Message	109
3.33 Cardholder Receives Purchase Response	110
3.34 Payment Authorization	111
3.35 Merchant Requests Authorization	112
3.36 Payment Gateway Processes Authorization Request	114
3.37 Merchant Processes Response	115
3.38 Payment Capture	116
3.39 Merchant Request Payment	117
3.40 Merchant Receives Response	118
4.1 SSL Payment	132
4.2 SET Payment	133

I. INTRODUCTION

1.1 Objective

- (a) For analysis “Co-worker between Security Protocols and Application on E-commerce.

1.2 Scope

This project is for research in term of security protocol on E-commerce. The core detail of this project is How to use security protocol for Application E-commerce such as how to transmit application E-commerce, how to security payment on E-commerce included the Analysis of How to security protocol for Application Payment.

1.3 Deliverable

- (a) Full report.
- (b) Presentation – Power Point

1.4 Research Methodology

The research emphasizes document research. The way of research is find Information in Text book and Internet information and then analysis conclusion.

II. REVIEW

2.1 Background and Impact of Electronic Commerce

There is no question that electronic commerce, as exemplified by the popularity of The Internet is going to have an enormous impact on the financial services industry. No financial institution will be left unaffected by the explosion of electronic commerce.

The number of payment card purchases made through this medium will grow as Internet based online ordering systems are created.

(a) Many banks are planning to support this new form of electronic commerce by offering card authorizations directly over the Internet.

(b) Several trials with electronic currency and digital cash are already under way.

(i) Projected use With more than 30 million users today, and 90 million Projected To come on board in the next two years, the Internet is a new way for Businesses to establish computer based resources that can be accessed by consumers as well as business partners around the world.

(ii) Internet The Internet is changing the way we access and purchase information, communicate and pay for services, and acquire and pay for goods. Financial services such as bill payment, brokerage, insurance, and home banking are Now or soon will be available over the Net. Any organization can become a global Publisher by establishing an information site on the Internet's World Wide Web.

(iii) World Wide Web The Web - or other interactive transport mechanism can display text, sound, images, and even video, allowing merchants to transmit information directly to potential consumers around the world and around the clock.

(iv) Consumer payment devices With open networks, consumer-driven devices will make payments increasingly. As advanced technologies become more practical and affordable, the marketplace will move from "brick and mortar" to more convenient

locations such as the home or office. As financial services evolve, consumers will consolidate their payment needs into one multifunctional relationship product that enables widespread, around-the-clock access.

(v) Publicity Recently, an explosion of publicity has heralded the growth of the Internet and the possibilities for consumers and merchants to create a new type of shopping called *electronic commerce*. The publicity has focused on three areas:

- (a) Marketing opportunities to develop new ways to browse, select, and pay for goods and services to on-line consumers,
- (b) New products and services, and
- (c) Security risks associated with sending unprotected financial information across Public networks.

All three areas must be addressed to facilitate the future growth of payment card transaction volume in the electronic marketplace.

(vi) Role of payment systems Payment systems and their financial institutions Will play a significant role by establishing open specifications for payment card transactions that:

- (a) Provide for confidential transmission,
- (b) Authenticate the parties involved,
- (c) Ensure the integrity of payment instructions for goods and services order data,
- (d) Authenticate the identity of the cardholder and the merchant to each other.

(vii) Procedures needed Because of the anonymous nature of communications networks, procedures must be developed to substitute for existing procedures used in face-to-face or mail order/telephone order (MOTO) transactions including the authentication of the cardholder by the merchant. There is also a need for the cardholder to authenticate that the merchant accepts SET transactions and is authorized to accept

payment cards.

(viii) Use of payment card products Financial institutions have a strong interest in accelerating the growth of electronic commerce. Although electronic shopping and Ordering do not require electronic payment, a much higher percentage of these transactions use payment card products instead of cash or checks. This will hold true in both the consumer marketplace and the commercial Marketplace. To meet these needs, the SET Secure Electronic Transaction protocol uses cryptography to:

- (a) Provide confidentiality of information,
- (b) Ensure payment integrity, and
- (c) Authenticate both merchants and cardholders.

This specification will enable greater payment card acceptance, with a level of security that will encourage consumers and businesses to make wide usage of payment card products in this emerging market.

(a) Objectives

(i) Motivation Primary: motivations for the payment card brands to provide specifications for secure payments are to:

- (a) Encourage the payment card community to take a leadership Position in establishing a secure payment specification and, in so doing, to avoid costs associated with future reconciliation of implemented approaches,
- (b) Respect and preserve the relationship between merchants and Acquirers and between cardholders and Issuers,
- (c) Facilitate rapid development of the marketplace,
- (d) Respond quickly to the needs of the financial services market,
- (e) Protect the integrity of payment card brands.

- (ii) **Payment security:** The objectives of payment security are to:
- (a) Authenticate cardholders, merchants, and acquirers,
 - (b) Provide confidentiality of payment data,
 - (c) Preserve the integrity of payment data, and
 - (d) Define the algorithms and protocols necessary for these security services.
- (iii) **Interoperability:** The objectives of interoperability are to:
- (a) Clearly define detailed information to ensure that applications developed by various vendors will interoperate,
 - (b) Create and support an open payment card standard,
 - (c) Define exportable technology throughout, to encourage globally interoperable software,
 - (e) Build on existing standards where practical,
 - (f) Ensure compatibility with and acceptance by appropriate standards bodies, and
 - (g) Allow for implementation on any combination of hardware and software platforms, such as PowerPC, Intel, Sparc, UNIX, MS-DOS, OS/2, Windows, and Macintosh.
- (iv) **Market acceptance:** The objectives of market acceptance are to:
- (a) Achieve global acceptance via ease of implementation and minimal impact on merchant and cardholder end users,
 - (b) Allow for “bolt-on” implementation of the payment protocol to existing client applications,
 - (c) Minimize change to the relationship between acquirers and merchants, and cardholders and issuers,

- (d) Allow for minimal impact to existing merchant, acquirer, and payment system applications and infrastructure, and
- (e) Provide a protocol that will be efficient for financial institutions.



2.2 Shopping and Payment on Electronic Commerce

(a) Electronic shopping experience

The electronic shopping experience can be divided into several distinct stages.

Stage Description

- (i) The cardholder browses for items in a variety of ways, such as:
 - (a) Using a browser to view an online catalog on the merchant's World Wide Web page.
 - (b) viewing a catalog supplied by the merchant on a CD-ROM; or
 - (c) looking at a paper catalog.
- (ii) The cardholder selects items to be purchased.
- (iii) The cardholder is presented with an order form containing the list of items, Their prices, and a total price including shipping, handling, and taxes.

This order form may be delivered electronically from the merchant's server or created on the cardholder's computer by electronic shopping software. Some online merchants may also support the ability for a cardholder to negotiate for the price of items (such as by presenting frequent shopper identification or information about a competitor's pricing).
- (iv) The cardholder selects the means of payment. This specification focuses on the case when a payment card is selected.
- (v) The cardholder sends the merchant a completed order along with payment instructions.

In this specification, cardholders who possess certificates digitally sign the order And the payment instructions.

- (vi) The merchant requests payment authorization from the cardholder's financial institution.

(vii) The merchant sends confirmation of the order.

(viii) The merchant ships the goods or performs the requested services from the order.

(ix) The merchant requests payment from the cardholder's financial institution.

Although these stages are listed in a specific order, variations are possible; many Such variations are described later in this specification.

This specification focuses on stages 5, 6, 7, and 9 when the cardholder chooses to use A payment card as the means of payment.

(b) Within the scope

The following are within the scope of this specification:

- (i) Application of cryptographic algorithms (such as RSA and DES)
- (ii) Certificate message and object formats
- (iii) Purchase messages and object formats
- (iv) Authorization messages and object formats
- (v) Capture messages and object formats
- (vi) Message protocols between participants

(c) Outside the scope

The following are outside the scope of this specification:

- (i) Message protocols for offers, shopping, delivery of goods, etc.
- (ii) Operational issues such as the criteria set by individual financial institutions for The issuance of cardholder and merchant certificates.
- (iii) Screen formats including the content, presentation and layout of order entry forms as defined by each merchant
- (iv) General payments beyond the domain of payment cards
- (v) Security of data on cardholder, merchant, and payment gateway systems

including protection from viruses, Trojan horse programs, and hackers

Note: This is only a partial list of categories of things that are outside the scope of the SET specification.

(d) Kinds of Shopping

Cardholders will shop in many different ways, including the use of online catalogs And electronic catalogs. The SET protocol supports each of these shopping experiences And should support others as they are defined.

(e) Online catalogs

The growth of electronic commerce is attributed largely to the popularity of the World Wide Web. Merchants can tap into this popularity by creating virtual storefronts on the Web that contain online catalogs. They can quickly update these catalogs as their product offerings change for seasonal promotions or other reasons.

A cardholder can visit these Web pages and select items to order. When the Cardholder finishes shopping and submits a request, the merchant's Web server can send the cardholder a completed order form to review and approve. Once the cardholder approves the order and designates a payment card, the SET protocol enables the cardholder to transmit payment instructions by a secure means, while enabling the merchant to obtain authorization and receive payment.

(f) Electronic catalogs

A growing number of merchants are distributing their catalogs via electronic media such as diskette or CD-ROM. This approach allows the cardholder to browse through merchandise off-line. With an on-line catalogue, the merchant has to be concerned about bandwidth and may choose to include fewer graphics or reduce the resolution of the graphics. By providing an off-line catalogue, such constraints are significantly reduced.

In addition, the merchant may provide a custom shopping application tailored to The merchandise in the electronic catalogue. Cardholders will shop by browsing Through the catalogue and selecting items to include on an order. Once the cardholder approves the order and chooses to use a payment card, an electronic message using the SET protocol can be sent to the merchant with the order and payment instructions. This Message can be delivered on-line, such as to the merchant's Web page, or sent via a store-and-forward mechanism, such as electronic mail.

(g) Payment System Participants Interaction of participants

E-commerce changes the way that participants in a payment system interact. In a face-to-face retail transaction or a mail order transaction, electronic processing begins with the merchant or the Acquirer. However, in a SET transaction, the electronic processing begins with the cardholder.

(h) Cardholder

In the electronic commerce environment, consumers and corporate purchasers Interact with merchants from personal computers. A cardholder uses a payment card that has been issued by an Issuer. SET ensures that in the cardholder's interactions with the merchant, the payment card account information remains confidential.

(i) Issuer

An Issuer is a financial institution that establishes an account for a cardholder and Issues the payment card. The Issuer guarantees payment for authorized transactions Using the payment card in accordance with payment card brand regulations and local legislation.

(j) Merchant

A merchant offers goods for sale or provides services in exchange for payment. With SET, the merchant can offer its cardholders secure electronic interactions. A

merchant that accepts payment cards must have a relationship with an Acquirer.

(k) Acquirer

An Acquirer is the financial institution that establishes an account with a merchant And processes payment card authorizations and payments.

(l) Payment gateway

A payment gateway is a device operated by an Acquirer or a designated third party That processes merchant payment messages, including payment instructions from cardholders.

(m) Brand

Financial institutions have founded payment card brands that protect and advertise the brand, establish and enforce rules for use and acceptance of their payment cards, and provide networks to interconnect the financial institutions. Financial services companies that advertise the brand, and establish and enforce rules for use and acceptance of their payment cards own other brands. These brands combine the roles of Issuer and Acquirer in interactions with cardholders and merchants.

(n) Third parties *

Issuers and Acquirers sometimes choose to assign the processing of payment card Transactions to third-party processors. This document does not distinguish between the financial institution and the processor of the transactions.

III. SECURITY PROTOCOLS

3.1 Transport Layer Security (TLS Protocols)

(a) Introduction

Security is important on the Internet. Whether sharing financial, business, or Personal information, people want to know with whom they are communicating (authentication), they want to ensure that what is sent is what is received (integrity), and they want to prevent others from intercepting their communications (privacy). The Secure Sockets Layer (SSL) protocol provides one means for achieving these goals at the transport layer. It was designed and first implemented by Netscape Corporation as a security enhancement for their Web servers and browsers. Since then, almost all vendors and public domain software developers have integrated SSL in their security sensitive Client-server applications. At present, SSL is widely deployed on the intranet as well as over the public Internet in the form of SSL-capable servers and clients and has become the defacto standard for transport layer security. Recently, the Internet Engineering Task Force (IETF) has started an effort to standardize SSL as an IETF Standard under the name of Transport Layer Security (TLS) protocol. In the rest of this paper, we refer to SSL as the Transport Layer Security protocol, or simply TLS. One of the reasons that TLS has outgrown other transport and application layer security protocols such as SSH, SET, and SMIME in terms of deployment is that it is application protocol independent. Conceptually, any application that runs over TCP can also run over TLS. There are many examples of applications such as TELNET and FTP running transparently over TLS. However, TLS is most widely used as the secure transport layer below HTTP. A large number of Websites dealing with private and sensitive information, including all those engaged in electronic commerce; use TLS as the secure transport layer. This number is expected to grow exponentially as more and more

businesses and users embrace electronic commerce. As security becomes an integral feature of Internet applications and the use of TLS raises, its impact on the performance of the servers as well the clients is going to be increasingly important. The objective of these papers is to take a close and critical look at the TLS protocol with an eye on performance.

The TLS protocol is composed of two main components: the TLS Record Protocol responsible for data transfer, and the TLS Handshake Protocol responsible for Establishing TLS session states between communicating peers. At the lowest level, layered on top of some reliable transport protocol (e.g., TCP), is the TLS Record Protocol. The Record Protocol provides two basic security services: privacy and Message integrity. It uses data encryption using symmetric cryptography (e.g., DES, RC4, etc.) To provide privacy and a keyed message authentication digest (e.g. MD5, SHA1) to ensure message integrity. The TLS Record Protocol is used for encapsulation of various higher level protocol messages, including the TLS handshake protocol. The TLS Handshake Protocol is responsible for authenticating the server and the client to each other. It is also entrusted with the job of negotiating an encryption algorithm along with the required session keys before the application protocol transmits or receives its first byte of data. The Handshake protocol typically uses public key cryptography for exchanging secret information (e.g. RSA and Diffie-Hellman). TLS allows the session state to be cached for configurable amount of time. If a client needs to setup a new TLS session while its session state is cached at the server, it can skip the steps involving authentication and secret negotiation and reuse the cached session state to generate a set of keys for the new session. In the rest of the paper, We analyze the performance impact of TLS on HTTP and quantify the overhead associated with different components of TLS. To measure the performance impact of TLS on Web server performance, we have

modified the SPECweb96 benchmark to generate client workload for servers and clients running HTTP transactions over TLS. Using this modified SPECweb96 benchmark, we evaluated the performance of two different secure Web servers with varying degrees of session reuse. Our results show that depending on the degree of session reuse the overhead due to TLS can decrease the rate at which the server can process HTTP transactions by up to two orders of magnitude. To identify the overhead associated with different components of TLS, we have instrumented and traced the TLS Handshake Protocol and TLS Record Protocol using timers with sub-micro second granularity. Our results indicate that for a typical HTTP transaction (10-15 Kbytes); the bulk of the overhead comes from the TLS Handshake Protocol unless the session state is reused. For very large HTTP transactions (1 Mbytes or more), the overhead due to data encryption and authentication is significant. We also observed that TLS hand-shake protocol adds significant latency to Web transfers due its four-way handshake. In light of these observations, we propose two techniques to improve the performance of the TLS Handshake protocol, namely caching of server certificates by clients and a three-way handshake protocol. As discussed later in the paper, by caching server certificates at the client, it is possible to reduce the number of messages exchanged during TLS handshake and consequently a round trip time. Certificate caching also reduces computational overhead at the client and the volume of data transferred during handshake. The second scheme is designed to offload the computationally expensive private key operations from the server. The rest of the paper is organized as follows. In section II, we present the operational overview of the TLS protocol. Section III is devoted to the evaluation and analysis of TLS protocol performance and its impact on secure Web servers. In section IV, we discuss the proposed enhancements to improve the performance of TLS handshake protocol. We conclude this paper in section V.

(b) Transport Layer Security

TLS provides the ability to setup private communication channels in a public network. Broadly, the operation of TLS can be split up along two major axes. One is the cryptographic techniques that it uses to provide security and the other is the operation of the protocol itself. First we review a few basic cryptographic operations that are critical to TLS and then describe the protocol.

(i) Basic Mechanisms

TLS uses symmetric key encryption techniques, such as DES and RC4, to ensure privacy. In symmetric key encryption the sender and the receiver share a secret key which is used to encrypt or decrypt messages. However, this secret key must somehow be exchanged between the communicating parties before any secure communication can take place. During the TLS handshake process the client chooses a secret, which it then sends to the server. Public key cryptography is used to protect this exchange. Unlike symmetric key cryptography, public key cryptography uses a pair of keys, a public key and a private key. As the name suggests, the owner of the key pair publishes the public component of the key and keeps the private component secret. If the Public key is used to encrypt a message then only the private key can be used to decrypt it and vice versa. In TLS, the initiator of a session, typically the client generates the secret and encrypts it with the public key of the peer, typically the server. The server, upon receipt of this message, uses its private key to decrypt it.

Since the server is the only one who possesses the private key, from this point on, the client and the server share a secret that no one else knows. One of the main reasons why public key cryptography is used only to communicate the shared secret is the fact that it is computationally rather expensive and so in reality

it can only be used to encrypt a few bytes of data. So the key exchange problem is solved, provided the client knows the server's public key. The server can supply this, but the client has to be able to bind the public key with the true identity of the server. TLS makes use of X.509 certificates to associate a public key with the real identity of an individual, server, or other entity, known as the subject. A certificate is signed by a trusted agency, commonly referred to as a certificate authority (CA). The signature process again typically involves public key cryptography.

The signing entity computes a hash function of the data to be signed and encrypts that with its private key. Performing the corresponding decryption with the public component of the private key and then matching the result with the freshly computed hash of the data can verify the signature. Although encryption guarantees privacy, it does not ensure message integrity. An adversary may still alter the encrypted messages without the sender or receiver being aware of it. TLS ensures message integrity by sending a digest of the message to the receiver along with the original message. Digest algorithms, such as MD5 and SHA, are one-way hash functions that output a unique digest for each input message. It is relatively easy to verify a digest given the original message. However reproducing the message given the digest is impossible. TLS guarantees message integrity by keying the message digests with a secret key shared between the sender and the receiver. Any modification to the message will result in a mismatch between the digests computed by the sender and the receiver, thus enabling the receiver to detect a compromised message.

(ii) Protocol Overview

Rather than defining a completely new transport layer protocol, TLS is layered on top of an existing reliable transport protocol viz. TCP/IP. This naturally introduces inefficiency since the TLS negotiation cannot start until the TCP/IP handshake is completed. However, this clean separation between the transport and security operations enabled a fairly rapid prototyping effort and partly contributed to the wide popularity of TLS. A TLS connection involves two stages. First, the communicating parties optionally authenticate each other and then exchange session keys. This phase is known as the TLS handshake. Once the handshake is complete, the two parties share a secret which can be used to construct a secure channel over which application data can be exchanged. TLS is an asymmetric protocol. It differentiates between a client and a server. The TLS handshake sequence may vary, depending on whether the RSA or Diffie-Hellman key exchange is used. Although TLS handshake allows both the client and the server to be authenticated to each other, most commonly, it is only the server that is authenticated. Client authentication is optional and is omitted in most cases. A typical TLS session makes use of the RSA key exchange with only the server being authenticated. We only consider this case in this paper. Figure 1(a) shows the message flow required to establish a new session. The client initiates the communication by sending a *Hello* message to the server. The Hello message includes a random number that is used in the handshake to prevent replay attacks. In response to the client hello, the server replies With a Hello of its own, Followed by a certificate that contains the server's public key.

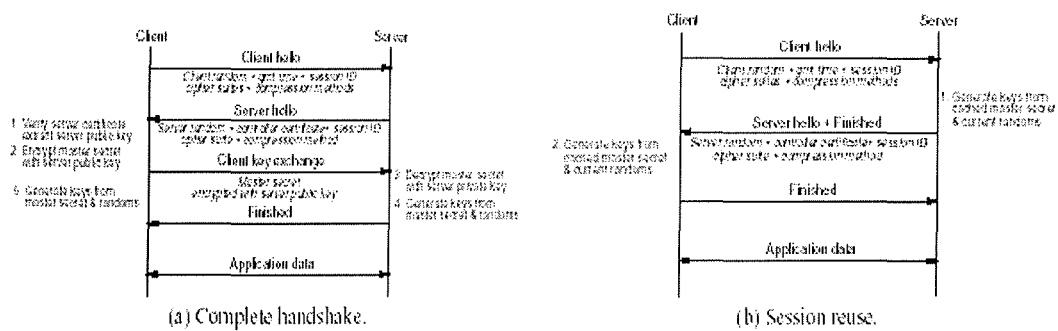


Fig. 1. Message flow in TLS handshake protocol.

Figure 3.1. Message flow in TLS handshake protocol.

Optionally, it may also send a chain of certificates belonging to the authorities in the certification hierarchy. The client verifies the certificate (or chain of certificates) by verifying the identity of the server and checking the validity of the CA's signature.

The client then generates a pre-master secret and encrypts it with the public key obtained from the server's certificate. This is sent to the server, which does a decryption using its private key, thus obtaining the pre-master secret. The pre-master secret is used to generate a master secret that is now shared between the client and the server. The master secret is then used to generate symmetric keys for encryption and message authentication. In other words the master secret is a shared state between the client and server and this constitutes a TLS session. This session can identify by the session ID that was included in the initial server Hello message. In contrast to the initial handshake protocol, the reestablishment of a cached TLS session is relatively simple. Figure 1(b) shows the messages exchanged to reestablish a TLS session. As shown in the figure, the client simply

specifies the session ID of the old or existing session it wishes to reuse when sending the Hello message. The server checks in its cache to determine if it has state associated with this session. If the session state still exists in the cache, it uses the stored master secret to create keys for the secure channel. The client repeats the same process and generates an identical set of keys. Note that multiple secure channels between the same pair of hosts can be established by reusing a single session state. This is a rather key feature of the TLS protocol that is particularly important in the context of the World Wide Web. A single secure web-page may be composed of multiple HTTP links and being able to reuse an existing session state to obtain the multiple links greatly reduces the latency and processing involved in setting up the secure channel.

(c) Performance Profile

Although TLS can be used with a variety of application protocols, such as TELNET and FTP, the most important and most common use of TLS has been to ensure privacy and authentication for HTTP transactions. All commercial web sites that require privacy and authentication use TLS. In this section, we benchmark the performance of secure Web servers and quantify the overheads of different components of TLS. We use the SPECweb96 benchmark as it attempts to capture real-world usage of a web-server and is based on the analysis of server logs from a few different Internet servers.

(a) Experimental Setup

Our tested consisted of a single IBM RS/6000 model 43P-200 running AIX 4.2, working as the server with multiple PCs working as clients. The server (model 43P-200) was equipped with a PowerPC 604e CPU running at 200 MHz with 32 KB on chip 4-way associative instruction and data caches, a 512 KB direct

mapped secondary cache, and 128 MB of RAM. The client machines were 266 MHz Pentium II PCs running Linux 2.0.35. A total of 8 client machines were directly attached to a Fast Ethernet Switch to which the server machine was also connected. This ensured that there were no bottlenecks due to network capacity during any of the experiments.

We have modified SPECweb96 to generate client workload for our secure web servers. The modified SPEC web clients make HTTP requests over TLS sessions. Since a typical web access results in several different links being fetched from the same web-server, there is bound to be some reuse of TLS session state when setting up subsequent connections. The amount of reuse is heavily coupled with the way web pages are setup, and we would like to investigate the server throughput with varying amounts of session reuse. Towards this end, we introduced a tunable knob that allows the SPEC web clients to control the degree of TLS session reuse.

For the experiments reported in this section, we did not modify the workload Generated by SPEC web. The workload generated by SPEC web is designed to mimic the workload on regular Web servers. More specifically, workload mix is built out of files in four classes: files less than 1KB account for 35% of all requests, files between 1KB and 10KB account for 50% of requests, 14% between 10KB and 100KB, and finally 1% between 100KB and 1MB. There are 9 discrete sizes within each class (e.g. 1 KB, 2 KB, on up to 9KB, then 10 KB, 20 KB, through 90KB, etc.), resulting in a total of 36 different files (9 in each of 4 classes). Accesses within a class are not evenly distributed; they are allocated using a Poisson distribution centered on the midpoint within the class. The resulting access pattern mimics the behavior where some files (such as “index. Html”) are

more popular when the rest, and some files (such as “mydog.gif”) are rarely requested.

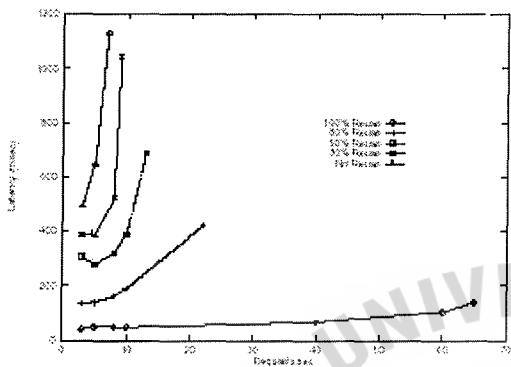


Fig. 2. SPECweb96 Performance of Netscape Enterprise Server.

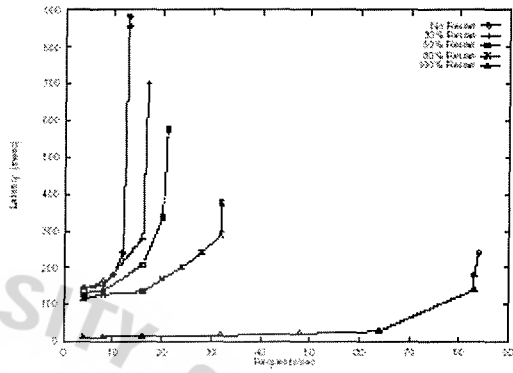


Fig. 3. SPECWeb96 Performance of Apache with SSLay.

Figure 3.2. Compare the performance of secure Web server.

Although, the “real-life” workloads for standard and secure servers are likely to be different, we chose to use the standard SPECweb workload for two reasons: (a) “real-life” workload for secure Web servers are not available at this time, and (b) our objective is to compare the performance of secure Web servers with that of non-secure servers and to analyze the performance impact of TLS. Using this modified SPECweb96 benchmark, we have evaluated two of the more popular web servers—Netscape Enterprise Server 3.5.1, and Apache 1.2.4 with SSLay 0.8.

(b) Benchmark Results

Figures 2 and 3 show the latency versus the number of HTTP (over TLS) Requests handled by the Netscape and Apache server respectively. The servers are

configured with certificates for 1024-bit keys. In all of these experiments, we used RC4 for data encryption and MD5 for message authentication, since these are the most widely used in real life.

Performance of other encryption and message authentication schemes are presented later in the section. We varied the degree of session reuse from 0-100%. When session reuse is 0% all TLS sessions setup between the server and the clients require a full handshake with the associated public and private key operations. When session reuse is 100%, only the first TLS session setup between the server and a client involves a full handshake. All subsequent connections reuse the already established session state between the server and the client. When the percentage of session reuse is in between 0 and 100, the clients reuse the same session for a certain number of times depending on the value of the reuse percentage. The way this is done is by maintaining a running counter of the number of connections that attempted to reuse session state. Whenever this counter drops below the desired fraction (reuse percentage) of total connections, the client attempts to reuse an existing session ID.

For example, when the reuse percentage is set to 50, the sessions setup by a SPECweb client takes the form NRNR : : : :::, where N stands for a new session and R stands for a reused session. From Figure 3 it is evident that the Apache server can handle, at most, 13 requests per second when there is no session reuse. For the same case, Figure 2 indicates that the Netscape server can only handle About 7 requests per second. At these operating points the latencies are extremely high in both cases with Apache coming in at around 300 msec and Netscape hovering above the 600 msec mark. In both the Figures we notice that as the amount of session reuse is increased the performance improves and with a 100%

reuse the latency is fairly low even when the rate of connection requests is quite high. The numbers for 100% reuse are only provided as a reference since in all practicality a web-server is unlikely to experience such a large amount of session reuse. In comparison, the SPECweb96 numbers for Netscape and Apache for regular web-pages on the same server are around 300 and 250 requests a second, respectively. The behavior of the Netscape server is fairly typical of what one would expect when the level of session reuse is varied. In Figure 2 we observe that the latency reduces and the sustainable throughput increases as the level of session reuse is increased. In contrast, with the Apache server at light loads there does not seem to be much difference in the latency results when the reuse is increased from 0 to 80%. This behavior may be a result of how session reuse is implemented in the Apache web server. Apache uses a process model in its web-server implementation. The web-server is composed of several, dynamically created server processes that serve web-requests. Rather than make a single entity responsible for dispatching the requests to each of the server processes, the creators of Apache chose to have each server process pick up a connection request and service it. This provides for some natural load-balancing features since a server process only picks up a request when it is free. When a TLS client wishes to reuse a session, it includes the session ID in the client Hello message. However at the time the connection is accepted by a server process, it has no knowledge of what the session ID will be since the Hello message is received only after the connection is accepted.

Unfortunately, with most flavors of Unix, once a connection request is accepted there is no way to rescind it and so the server process is forced to serve the request, whether or not it has the session ID in its cache. To get around this

problem, the Apache server runs a separate process which acts as the global cache (gcache) server. Whenever a server process gets a session reuse request from a client, it first searches its own local session cache. If the local session cache does not have an entry for the client, the server process contacts the gcache server. If the gcache server has the specified entry in its database, it returns the cached state to the server process and session reuse is performed. Otherwise, a full handshake is performed and the session state is added to both the local cache and the global session cache. Since Apache spawns several server processes for the purpose of efficiency, at light loads it is quite likely that a newly arriving reuse request will be sent to a different server process (say process B) than the original process (say process A) with which the session state was established. As a result of this B needs to obtain the session state from the gcache server before setting up the new connection. Now B will not get a response from the gcache server until the gcache process is scheduled and subsequently B is scheduled to run again. This can take quite a while and so, for light loads, there is hardly any apparent reduction in latency even when there is session reuse. In fact, we ran a separate experiment where we reused the same session state over and over again and noticed that after about 16 requests (the maximum number of server processes was limited to 16) the latency to establish a secure connection dropped down significantly to little over 3 ms. This is because by this time all the server processes have a copy of the session state in their local cache and so do not need to go to the global cache to obtain the session state. This effect can also be seen in Figure 3 for the case where we have a 100 % reuse of session state.

Since the same session is now being reused all the time, each of the server Processes has the session state in its local cache and so the latency is really low

even at fairly high rates of connection requests.

(c) Overhead Analysis

In the last section, we quantified the performance of secure Web servers and compared it with that of standard non-secure servers under the same workload, namely SEPCweb96. Our results show that the price we have to pay for security is rather large. In this section, we take a closer look at the performance overhead associated with different components of TLS. For this purpose, we have instrumented the TLS protocol stack for detailed profiling of various processing modules in the data path. The instrumented stack can be used to capture a sequential flow of time-stamped events on the data path. The time-stamps are of sub-microsecond granularity and are taken by reading a real time clock, which is an integral part of the PowerPC CPUs used in RS/6000s. We use a two-instruction assembly language routine to read two 32-bit clock registers with minimal overhead. In the following, we present a detailed analysis of the overhead associated with the TLS handshake protocol, and the performance impact of encryption and authentication during data transfer

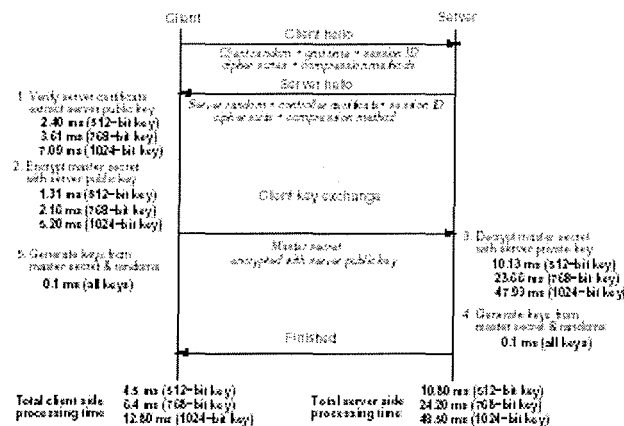


Fig. 4. Timing Analysis of TLS Handshake Protocol.

Figure 3.3. Timing Analysis of TLS Handshake Protocol.

(d) Session Setup Overhead

The TLS session setup overhead can be divided up into (1) an increase in Data volume due to additional data items, such as server certificates, and (2) computational overhead for crypto functions. Data items exchanged during TLS handshake increase the latency of HTTP transactions. When the server uses a self-signed certificate the amount of data sent by the server to the client during handshake is about 750 bytes.

When the server sends a chain of certificates to the client, each certificate adds about 750 bytes to the data sent by the server. The amount of data sent by the client is about 250 bytes. The relative overhead due to increase in data volume depends on the network connectivity.

For clients connected via dialup lines, the increase in latency due to increase in data volume may be significant. For clients connected via LANs, the overhead due to increase in data volume pales in comparison with the computational

overhead incurred by the crypto functions. Figure 4 shows the overheads involved in setting up a TLS session. There are 3 sets of numbers based on the size of the server's public key, i.e. 512, 768 or 1024-bit. As seen in Figure 4, the most expensive component in session setup is the private key operation at the server side. Verification of the server certificate(s) and generation and encryption of the master secret are the major operations performed on the client side.

Ironically, the most expensive of the crypto operations is performed at the server, resulting in rather low throughput numbers. Note that both server and client side operations are more expensive when the server uses longer private keys. For US domestic use 1024-bit server certificates are recommended and used. Reusing existing session state can greatly reduce the cost of connection setup. Since there are no public key operations involved, when the session state is being reused, the time taken to setup a secure channel is about 3 to 4 msec.

This is of the same order of magnitude as a TCP handshake as part of the latency is due to the round trip time from the client to the server and back.

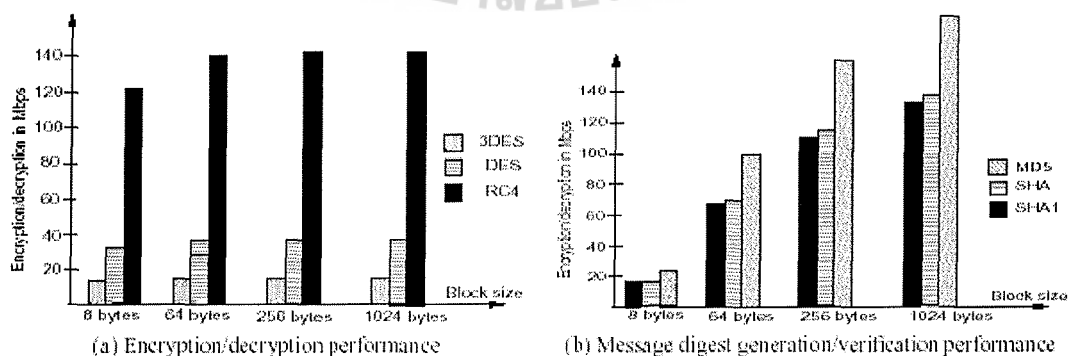


Fig. 5. Crypto overhead in data transfer.

Figure 3.4. Crypto overhead in data transfer.

(e) Data Transfer Overhead

Figure 5 shows the performance of crypto functions used in the data path to encrypt/decrypt message and to generate/verify message digests. Most web browsers are by default configured to use RC4 for data encryption. When higher level of security is required, DES is preferred. For the highest level of security, 3DES is the recommended encryption algorithm. Figure5 (a) shows the performance of RC4, DES, and 3DES for different data block sizes. When RC4 is used as the encryption/decryption algorithm, the server can encrypt/decrypt more than 120Mbps. The encryption rate for DES is between 20-40Mbps and is based on the block size. With 3DES the encryption/ decryption rate goes down to about 10-15Mbps. Note that in the results reported in Figures 3 and 2 we used RC4 for data encryption. The results in Figure 5(a) further confirms the fact that encryption/decryption is only small part of the overhead in TLS. As the figure shows, as the block size increases so does the performance. However, the performance improvement due to increase in data block size diminishes rapidly and almost flattens out beyond block sizes of 512 bytes.

Figure 5(b) shows the performance of the message digest generation and Verification algorithms. By default all web browsers use MD5 as the message digest algorithm. Browsers can also be configured to use SHA and SHA1, which are considered to be more secured. As the figure shows, MD5 can generate/verify message digest at a rate of 20Mbps for 8 byte messages and at more than 120Mbps for messages of size 1024 byte and more. SHA and SHA1 achieve comparable performance of about 20Mbps for small (8 byte) messages and up to 120Mbps for larger messages (1024 bytes and more). We should note here that typical web transfers are about 4 Kbytes or more. The results in Figure 5 (b) show that

generation and verification of message digests is clearly not the bottleneck in TLS.

(d) Optimizations to the handshake protocol

In this section we propose two techniques that can be used to speed up the TLS Handshake operation. One involves the simple caching of server certificates by clients. Thereby obviating the need for the server to send its certificate to the client. The second approach is more involved as it reverses the role of client and server in terms of the generation of the pre-master secret. This reversal reduces the customary two round-trips of a TLS handshake to a 3-way handshake. In fact, the client can send data after a single set of messages is exchanged, i.e. after one round-trip time.

Another important feature of this scheme is that it offloads the expensive private key operations from the server to the client. This should significantly increase the rate at which the server can accept secure connections.

(i) Certificate Caching

Clients often setup multiple TLS sessions to the same server. If successive Sessions to the same server are within a short time window, the same session state Can be reused. However, for security reasons, servers typically do not cache Session state for more than 30 minutes. Consequently, if a client needs to setup a TLS session after its cache state has been timed out, it has to undergo the complete handshake again. As shown in Figure 4 a full handshake involves exchange of four sets of messages and takes two round trip times.

If client s can make use of the server certificate obtained during past Exchanges with the same server it is possible to reduce the number and size of messages as well as the processing on the client side. In fact, in the context of the web, most people have bookmarks and access a few pages rather frequently. If the the web-browser were to store the server certificates in its bookmark file then

there is no need for the server to have to send a certificate back to the client. All that is required is for the client to include a fingerprint, which can be the result of applying a well-known hash function like SHA or MD5 on the certificate, in its hello message to the server. In the following, we outline in more detail the message flow involved when certificates are cached at the client.

(ii) Message Flow

Figure 6 shows the message flow for the modified handshake protocol Designed to take advantage of the cached certificates. As shown in the figure, the Client includes the pre-master secret encrypted in the public key of the server in the client hello message.

Additionally, it also includes the finger print of the cached server certificate, With the hello message. The server verifies the finger print contained in the client



Fig. 6. TLS Handshake with Cached Server Certificate.

Figure 3.5. TLS Handshake with Cached Server Certificate.

hello message with the fingerprint of its certificate. If the fingerprints match, the server decrypts the pre-master secret and proceeds to the finish message

directly. If the fingerprints do not match, either because the server certificate has changed or because the client has an incorrect certificate, the server discards the pre-master secret and proceeds with the server hello as depicted in Figure 4(a). Note that caching of certificates does not, in any way, impact the security functions of the TLS handshake protocol. Consequently, it does not have any implication on the security of TLS.

(iii) Performance Implications

Besides eliminating a round trip time from the handshake protocol, certificate caching can lead to significant savings both in terms of number and size of messages exchanged during the handshake. Since certificates can be quite large, and the server often sends multiple certificates, this saving may translate into significant reduction in connection setup latency, especially for clients connected over low bit rate links.

Table I presents numerical results quantifying the reduction in perceived Latency for different network scenarios. Certificate caching also helps eliminate The public key operations at the client end to verify the certificates sent by the server. Since the certificates are cached only after they have been verified, when cached certificates are reused they do not need to be verified again. If, for some reason, the certificate cached at the client is no longer in use by the server, the proposed modifications may inflict a slight penalty. Since, in this case the server ignores the encrypted pre-master secret sent by the client as part of client hello and proceeds with the standard handshake protocol, the number of messages exchanged and the round trip time taken remains the same. Consequently, the public key operations performed by the client in step 1 of Figure 6 is wasted. The client has to generate the pre-master secret again and send it as a part of client key

exchange message. However, the scenario where the finger print of the certificate cached at the client does not match the one maintained at the server is expected to be extremely rare as server certificates change rather infrequently.

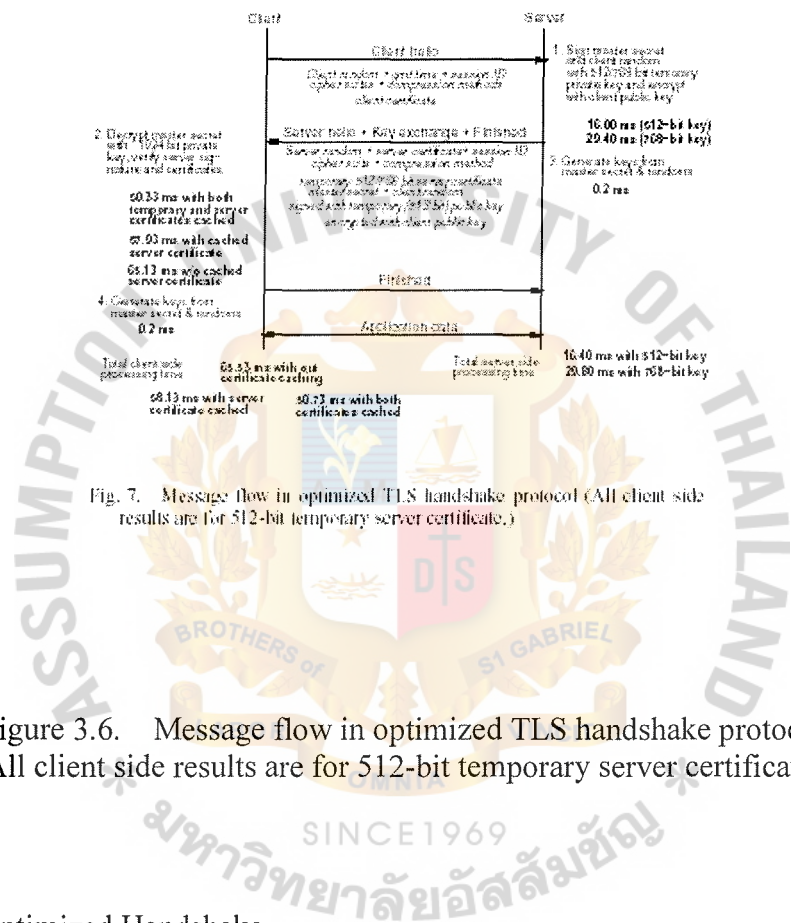


Fig. 7. Message flow in optimized TLS handshake protocol (All client side results are for 512-bit temporary server certificate.)

Figure 3.6. Message flow in optimized TLS handshake protocol (All client side results are for 512-bit temporary server certificate).

(iv) Optimized Handshake

Certificate caching and the associated changes to the handshake protocol Described in the previous section improve the connection setup latency. However, it does not reduce the computational burden on the server. As discussed earlier, the most expensive server side component in the handshake protocol is the private key operation and it is clearly the limiting factor on the critical path of the server side protocol processing. One of the reasons why this private key operation is necessary is due to the fact that it enables the authentication of the server. It is

however, possible to reduce the overhead by using a key of a smaller size. Since a smaller key is easier to break, it needs to be changed more frequently than a longer key. For example, a 1024 bit RSA key is considered safe for months, while a 512-bit RSA key is considered safe for days. Assume that a server possesses a 1024-bit key and the corresponding certificate is signed by a well-known CA.

Our objective is to reduce server side protocol processing overhead without Compromising security. For this purpose, we propose to use a certificate with a temporary 512-bit (or 768-bit) key for server authentication. This certificate is signed with the server's 1024 bit key.

The temporary key can be changed frequently (every half hour) to limit the Exposure caused by the smaller key. Another reason to change the server's key pair frequently is the recently discovered security exposure of TLS against a chosen cipher text attack. The exposure is a result of TLS's use of the PKCS#1 encoding of the pre-master secret prior to encryption with the server's public key and is described in.

In the current TLS handshake the (public) key used by the client to encrypt The shared pre-master secret, is also obtained from the server certificate. In the proposed scheme, we turn this around and have the server choose the pre-master secret and send that to the client encrypted with client's public key. A certificate containing the public key of the client can be sent to the server along with the client Hello message.

Note: The client public key is used only for the purpose of encryption and hence it is sufficient for the client to have a self-signed certificate.

Line Speed	RTT	Data Transfer Time		Round Trip Delay		Total	
		Cached	Standard	Cached	Standard	Cached	Standard
28.8 Kbps	5ms	4.5ms	416ms	5ms	10ms	62.5ms	486ms
28.8 Kbps	250ms	4.5ms	416ms	250ms	500ms	307.5ms	966ms
1.5 Mbps	5ms	0.1ms	8ms	5ms	10ms	57.6ms	78ms
1.5 Mbps	250ms	0.1ms	8ms	250ms	500ms	302.1ms	568ms

TABLE I
ESTIMATED LATENCIES WITH AND WITHOUT CERTIFICATE CACHING.

Figure 3.7. Estimated Latencies with and without Certificate Caching.

A self-signed certificate can be generated locally or can be included in the web-browser or other application program. In the following, we present the message flow of the proposed scheme and discuss its security and performance implications.

(v) Message Flow

Figure 7 shows the message flow in the optimized TLS handshake protocol. As shown in the figure, besides the information that is part of the standard TLS Client hello message, in our proposed scheme the client also includes its own certificate. If client authentication is not required, which is typically the case, this certificate can be self-signed. The server, upon receipt of the client hello message, generates the pre-master secret, signs the pre master secret and the client random with the temporary 512-bit key, and encrypts the signed pre-master secret and the client random with the public key of the client. It then sends the signed and encrypted pre-master secret to the client along with its certificate chain, i.e. the temporary as well as the original certificate. The client upon receipt of the signed and encrypted pre-master secret decrypts the pre-master secret with its private key and verifies the signature by extracting the 512-bit public key from the temporary

certificate. By verifying the signature on the pre master secret as well as the server certificates, the client authenticates the server. By signing the pre-master secret and the client random together, the server binds the pre-master secret with the current session. This ensures that a malicious third party can not launch a replay attack later. Once the client extracts the pre-master secret and verifies the signatures, it sends a Finished message to the server and proceeds to key generation and data exchange.

The modifications to the TLS handshake protocol can be used in conjunction with the certificate caching described in Section IV-A. The client can send the finger prints of the server certificate as well as the finger print of the temporary certificate (if un-expired) with the client hello if it has those cached at its end. If the same certificates are still in use by the server it can avoid transmitting the certificates to the client. The client also saves two public key operations since it does not need to verify the server certificates again. Notice, that the optimized scheme is significantly different from the standard TLS handshake. In the standard TLS handshake, the pre-master secret is generated by the client and is encrypted with the public key of the server. As a result, the server needs to perform the private key operation to extract the pre-master secret from the client key exchange message. In the proposed scheme, the server generates the pre-master secret. It is signed with a 512-bit temporary private key and is encrypted with the public key of the client.

Consequently, the client performs the expensive 1024-bit private key operation. By off-loading the expensive 1024-bit private key operation from the server, the proposed scheme improves server scalability in terms of number of new connections it can handle. In the following, we discuss the security and

performance implications of these modifications.

(v) Security Considerations

In this section, we argue that in terms of security, the proposed scheme is as secure as the standard TLS handshake protocol. The purpose of the TLS handshake protocol is to (1) authenticate the server to the client, (2) establish a shared secret between the server and the client, and (3) optionally authenticates the client to the server. Server authentication is performed using the server's original certificate, which is signed by a CA. In the original TLS handshake protocol, the client verifies the server certificate and uses the public key contained in the certificate to encrypt the pre-master secret it sent to the server. The server, by decrypting the pre-master secret using its private key, furnishes a proof that it is indeed the server that the client desires to communicate with. In the proposed protocol, the server authenticates itself by signing the client random and the pre-master secret with its 512-bit temporary private key. The client verifies this signature with the corresponding public key which itself the 1024-bit private key of the server signs. The public key corresponding to the 1024-bit private key is contained in a certificate which is signed by a well known CA.

Note: The strength of the server authentication scheme depends on a 512-bit signature which appears to be weaker than the 1024-bit signature required by the original TLS handshake protocol. However, this 512-bit key pair is changed frequently (once every half hour) to reduce the time window in which this key must be broken to compromise security.

Using a brute force technique, with a large number of distributed computers, it takes days to break a 512-bit RSA key. Hence, if the keys are refreshed every half hour, it is extremely difficult, if not impossible to break this key. The expiry

date and time in the temporary 512-bit certificate indicates the lifetime of the 512-bit key pair.

In our scheme, we use the client public key contained in the client certificate for encrypting the shared secret exchanged between the server and the client. The client decrypts the pre-master secret using its private key. Depending on the security needs of the client, it can use 512-bit, 768-bit or 1024-bit key pairs for this purpose. The refresh interval for the key can also be a function of its security needs.

(vi) Performance Implications

The objective of the proposed scheme is to off load some of the burdens of the TLS handshake protocol from the server. The proposed modification reduces the computational burden on the server at the expense of putting additional computational load on the clients. However, since the client's setup far less TLS sessions, the scheme proposed here makes the secure servers more scalable in terms of the number of TLS sessions they can setup per unit of time. It may even reduce the TLS handshake latency observed by the clients under similar load conditions. The proposed scheme may not be suitable for clients with low computing power, such as hand-held palm-tops. However, the preferable way to provide a secure channel to small palm-top computers is through the use of a security proxy. The client can establish a single session to the security proxy, thereby requiring very few handshake messages. In the original TLS handshake protocol, the client performs two public key operations, once in step 1 of Figure 4 to verify server certificate, and once in step 2 of Figure 4 to encrypt the pre-master secret with the server public key. The bulk of the computational overhead is on the server which has to perform a private key operation in step 3 of Figure 4 to

decrypt the pre-master secret. In contrast, in the proposed protocol, a client performs one private key operation and three public key operations.

All operations are performed in step 2 of Figure 7 which includes (1) decryption of the master secret using a 1024-bit private key, (2) verification of the server signature on the pre-master secret and client random using the public key (corresponding to the 512-bit temporary certificate, (3) verification of the signature on the 512-bit temporary certificate, and (4) verification of the signature in the server certificate. The server needs to perform a 512-bit private key operation to sign the pre-master secret and the client random and a public key operation to encrypt the pre-master secret. As shown in Figure 7 the total computational cost on the server side is 16.40 ms as compared to 48.50 ms in the original TLS handshake protocol. In contrast, the client side processing overhead goes up from 12.80 ms to 65.53 ms. Figure 7 also lists the computational costs for a 768-bit temporary key. When the modifications to the TLS handshake protocol are used in conjunction with certificate caching, the client saves on the processing need to verify the server's certificates.

As a result the processing overhead at the client goes down to 50.73 ms and 58.13 ms depending on whether both permanent and temporary or only the permanent server certificate is cached at the client.

(d) Conclusion

TLS is widely deployed on the intranet as well as over the public Internet in the form of TLS-capable servers and clients and has become the de facto standard for transport layer security. Although the security implications of TLS have been under the microscope ever since its inception, similar analysis of its performance has not been performed. In this paper, we have analyzed TLS from a performance perspective and

quantified its impact on applications protocols and servers, specifically HTTP and Web servers. Using a modified SPECweb96 benchmark, we evaluated the performance of Web servers running HTTP transactions over TLS. Our results show that the overhead due to TLS can decrease the number of HTTP transactions handled by up to two orders of magnitude. Given the rather significant growth in the use of TLS particularly in the burgeoning field of E-commerce, it is not clear how secure web-servers of today will keep pace with this growth. We analyzed the specific components responsible for this overhead and based on our observations we proposed two techniques, namely certificate caching and a three-way handshake to improve the performance of the TLS protocol. We are currently benchmarking Web servers running HTTP transactions on the optimized TLS protocol.



3.2 Secure Socket Layer (SSL PROTOCOL)

(a) Abstract

This document specifies the Secure Sockets Layer (SSL) protocol, a security protocol that provides privacy over the Internet. The protocol allows client/server applications to communicate in a way that cannot be eavesdropped. Servers are always authenticated and clients are optionally authenticated.

(b) Motivation

The SSL Protocol is designed to provide privacy between two communicating applications (a client and a server). Second, the protocol is designed to authenticate the server, and optionally the client. SSL requires a reliable transport protocol (e.g. TCP) for data transmission and reception.

The advantage of the SSL Protocol is that it is application protocol independent. A "higher level" application protocol (e.g. HTTP, FTP, TELNET, etc.) can layer on top of the SSL Protocol transparently. The SSL Protocol can negotiate an encryption algorithm and session key as well as authenticate a server before the application protocol transmits or receives its first byte of data. All of the application protocol data is transmitted encrypted, ensuring privacy.

The SSL protocol provides "channel security" which has three basic properties:

- (i) The channel is private. Encryption is used for all messages after a simple handshake is used to define a secret key.
- (ii) The channel is authenticated. The server endpoint of the conversation is always authenticated, while the client endpoint is optionally authenticated.

- (iii) The channel is reliable. The message transport includes a message integrity check (using a MAC).

(c) SSL Record Protocol Specification

(i) SSL Record Header Format

In SSL, all data sent is encapsulated in a **record**, an object that is composed of a header and some non-zero amount of data. Each record header contains a two or three byte length code. If the most significant bit is set in the first byte of the record length code then the record has no padding and the total header length will be 2 bytes, otherwise the record has padding and the total header length will be 3 bytes.

The record header is transmitted before the data portion of the record. Note That in the long header case (3 bytes total), the second most significant bit in the First byte has special meaning. When zero, the record being sent is a data record. When one, the record being sent is a security escape (there are currently no examples of security escapes; this is reserved for future versions of the protocol). In either case, the length code describes how much data is in the record. The Record length code does not include the number of bytes consumed by the record Header (2 or 3). For the 2 byte header, the record length is computed by (using a "C"-like notation):

$\text{RECORD-LENGTH} = ((\text{byte}[0] \& 0x7f) \ll 8) | \text{byte}[1]$; Where byte[0] represents the first byte received and byte[1] the second byte received. When the 3 byte header is used, the record length is computed as follows (using a "C"-like notation):

$$\text{RECORD-LENGTH} = ((\text{byte}[0] \& 0x3f) \ll 8) | \text{byte}[1];$$

$$\text{IS-ESCAPE} = (\text{byte}[0] \& 0x40) \neq 0;$$

PADDING = byte[2];

The record header defines a value called PADDING. The PADDING value specifies how many bytes of data were appended to the original record by the sender.

The padding data is used to make the record length be a multiple of the block ciphers block size when a block cipher is used for encryption. The sender of a "padded" record appends the padding data to the end of its normal data and then encrypts the total amount (which is now a multiple of the block cipher's block size). The actual value of the padding data is unimportant, but the encrypted form of it must be transmitted for the receiver to properly decrypt the record. Once the total amount being transmitted is known the header can be properly constructed with the PADDING value set appropriately. The receiver of a padded record decrypts the entire record data (sans record length and the optional padding) to get the clear data, then subtracts the PADDING value from the RECORD-LENGTH to determine the final RECORD-LENGTH. The clear form of the padding data must be discarded.

(ii) SSL Record Data Format

The data portion of an SSL record is composed of three components (transmitted and received in the order shown):

MAC-DATA[MAC-SIZE]

ACTUAL-DATA[N]

PADDING-DATA[PADDING]

ACTUAL-DATA is the actual data being transmitted (the message payload).

PADDING-DATA is the padding data sent when a block cipher is used and padding is needed. Finally, MAC-DATA is the *Message Authentication Code*.

When SSL records are sent in the clear, no cipher is used. Consequently the amount of PADDING-DATA will be zero and the amount of MAC-DATA will be zero. When encryption is in effect, the PADDING-DATA will be a function of the cipher block size.

The MAC-DATA is a function of the CIPHER-CHOICE (more about that later).

The MAC-DATA is computed as follows:

MAC-DATA = HASH[SECRET, ACTUAL - DATA, PADDING-DATA, SEQUENCE-NUMBER]

Where the SECRET data is fed to the hash function first, followed by the ACTUAL-DATA, which is followed by the PADDING-DATA which is finally followed by the SEQUENCE-NUMBER. The SEQUENCE-NUMBER is a 32 bit value which is presented to the hash function as four bytes, with the first byte being the most significant byte of the sequence number, the second byte being the next most significant byte of the sequence number, the third byte being the third most significant byte, and the fourth byte being the least significant byte (that is, in network byte order or "big endian" order).

MAC-SIZE is a function of the digest algorithm being used. For MD2 and MD5 the MAC-SIZE will be 16 bytes (128 bits).

The SECRET value is a function of which party is sending the message. If the client is sending the message then the SECRET is the CLIENT-WRITE-KEY (the server will use the SERVER-READ-KEY to verify the MAC). If the client is receiving the message then the SECRET is the CLIENT-READ-KEY (the server will use the SERVER-WRITE-KEY to generate the MAC).

The SEQUENCE-NUMBER is a counter which is incremented by both the

sender and the receiver. For each transmission direction, a pair of counters is kept (one by the sender, one by the receiver). Every time a message is sent by a sender the counter is incremented. Sequence numbers are 32 bit unsigned quantities and must wrap to zero after incrementing past 0xFFFFFFFF.

The receiver of a message uses the expected value of the sequence number as input into the MAC HASH function (the HASH function is chosen from the CIPHER-CHOICE). The computed MAC-DATA must agree bit for bit with the Transmitted MAC-DATA. If the comparison is not identity then the record is considered damaged, and it is to be treated as if an "I/O Error" had occurred (i.e. an unrecoverable error is asserted and the connection is closed).

A final consistency check is done when a block cipher is used and the protocol is using encryption. The amount of data present in a record (RECORD-LENGTH) must be a multiple of the cipher's block size. If the received record is Not a multiple of the cipher's block size then the record is considered damaged, and it is to be treated as if an "I/O Error" had occurred (i.e. an unrecoverable error is asserted and the connection is closed).

The SSL Record Layer is used for all SSL communications, including Handshake messages, security escapes and application data transfers. The SSL Record Layer is used by both the client and the server at all times.

For a two byte header, the maximum record length is 32767 bytes. For the Three byte header, the maximum record length is 16383 bytes. The SSL Handshake Protocol messages are constrained to fit in a single SSL Record Protocol record.

Application protocol messages are allowed to consume multiple SSL Record Protocol records. Before the first record is sent using SSL all sequence numbers

Are initialized to zero. The transmit sequence number is incremented after every message sent, starting with the CLIENT-HELLO and SERVER-HELLO messages.

(iii) SSL Handshake Protocol Specification

(a) SSL Handshake Protocol Flow

The SSL Handshake Protocol has two major phases. The first phase is used to establish private communications. The second phase is used for client authentication.

Phase 1 The first phase is the initial connection phase where both Parties communicate their "hello" messages. The client initiates the conversation by sending the CLIENT-HELLO message. The server receives the CLIENT-HELLO message and processes it responding with the SERVER-HELLO message. At this point both the client and server have enough information to know whether or not a new master key is needed. When a new master key is not needed, both the client and the server proceed immediately to phase 2.

When a new master key is needed, the SERVER-HELLO message will contain enough information for the client to generate it. This includes the server's signed certificate (more about that later), a list of bulk cipher specifications (see below), and a connection-id (a connection-id is a randomly generated value generated by the server that is used by the client and server during a single connection). The client generates the master key and responds with a CLIENT-MASTER-KEY message (or an ERROR message if the server information indicates that the client and server cannot agree on a bulk cipher).

It should be noted here that each SSL endpoint uses a pair of ciphers per connection (for a total of four ciphers). At each endpoint, one cipher is used for outgoing communications, and one is used for incoming communications. When the client or server generates a session key, they actually generate two keys, the SERVER-READ-KEY (also known as the CLIENT-WRITE-KEY) and the SERVER-WRITE-KEY (also known as the CLIENT-READ-KEY). The master key is used by the client and server to generate the various session keys (more about that later).

Finally, the server sends a SERVER-VERIFY message to the client after the master key has been determined. This final step authenticates the server, because only a server, which has the appropriate public key, can know the master key.

Phase 2 The second phase is the authentication phase. The client has already authenticated the server in the first phase, so this phase is primarily used to authenticate the client. In a typical scenario, the server will require something from the client and send a request. The client will answer in the positive if it has the needed information, or send an ERROR message if it does not. This protocol specification does not define the semantics of an ERROR response to a server request (e.g., an implementation can ignore the error, close the connection, etc. and still conform to this specification).

When a party is done authenticating the other party, it sends its **finished** message.

For the client, the CLIENT-FINISHED message contains the Encrypted form of the CONNECTION-ID for the server to verify. If the verification fails, the server sends an ERROR message.

Once a party has sent its **finished** message it must continue to listen to its peers messages until it too receives a **finished** message. Once a party has both sent a finished message and received its peers finished message, the SSL handshake protocol is done. At this point the application protocol begins to operate (Note: the application protocol continues to be layered on the SSL Record Protocol).

(b) Typical Protocol Message Flow

The following sequences define several typical protocol message flows For the SSL Handshake Protocol. In these examples we have two principals in the conversation: the client and the server. We use a notation commonly found in the literature [10]. When something is enclosed in curly braces "{something}key" then the something has been encrypted using "key".

(i) Assuming no session-identifier

Client-hello C -> S: challenge, cipher specs

Server-hello S -> C: connection-, server certificate,
cipher_specs

Client-master-key C -> S: {master_key}server_public_key

Client-finish C -> S: {connection-id}client_write_key

Server-verify S -> C: {challenge}server_write_key

Server-finish S -> C: {new_session_id}server_write_key

(ii) Assuming a session-identifier was found by both client & server

Client-hello C -> S: challenge, session_id, cipher specs

Server-hello S -> C: connection-id, session_id_hit

Client-finish C -> S: {connection-id}client_write_key

NO-CERTIFICATE-ERROR

When a REQUEST-CERTIFICATE message is sent, this error may be returned if the client has no certificate to reply with. This error is recoverable (for client authentication only).

BAD-CERTIFICATE-ERROR

This error is returned when a certificate is deemed bad by the receiving party. Bad means that either the signature of the certificate was bad or that the values in the certificate were inappropriate (e.g. a name in the certificate did not match the expected name). This error is recoverable (for client authentication only).

UNSUPPORTED-CERTIFICATE-TYPE-ERROR

This error is returned when a client/server receives a certificate type that it can't support. This error is recoverable (for client authentication only).

(d) SSL Handshake Protocol Messages

The SSL Handshake Protocol messages are encapsulated in the SSL Record Protocol and are composed of two parts: a single byte message type code, and some data. The client and server exchange messages until both ends have sent their "finished" message, indicating that they are satisfied with the SSL Handshake Protocol conversation. While one end may be finished, the other may not, therefore the finished end must continue to receive SSL Handshake Protocol messages until it too receives a "finished" message.

After each party has determined the pair of session keys, the message bodies are encrypted using it. For the client, this happens after it verifies the session-identifier or creates a new session key and has sent it to the server.

For the server, this happens after the session-identifier is found to be good, or the server receives the client's session key message.

The following notation is used for SSLHP messages:

Char MSG-EXAMPL

Char FIELD1

Char FIELD2

Char THING-MSB

Char THING-LSB

Char THING-DATA [(MSB<<8) |LSB];

This notation defines the data in the protocol message, including the Message type code. The order is presented top to bottom, with the top most element being transmitted first, and the bottom most element transferred last.

For the "THING-DATA" entry, the MSB and LSB values are actually THING-MSB and THING-LSB (respectively) and define the number of bytes of data actually present in the message. For example, if THING-MSB were zero and THING-LSB were 8 then the THING-DATA array would be exactly 8 bytes long. This shorthand is used below.

Length codes are unsigned values, and when the MSB and LSB are combined the result is an unsigned value. Unless otherwise specified lengths values are "length in bytes".

(e) Client Only Protocol Messages

There are several messages that are only generated by clients. These messages are never generated by correctly functioning servers. A client receiving such a message closes the connection to the server and returns an error status to the application through some unspecified mechanism.

CLIENT-HELLO (Phase 1; Sent in the 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 [(MSB<<8) |LSB]

Char SESSION-ID-DATA [(MSB<<8) |LSB]

Char CHALLENGE-DATA [(MSB<<8) |LSB]

When a client first connects to a server it is required to send the CLIENT-HELLO message. The server is expecting this message from the client as its first message. It is an error for a client to send anything else as its first message.

The client sends to the server its SSL version, its cipher specs (see below), some challenge data, and the session -identifier data. The session-identifier data is only sent if the client found a session-identifier in its cache for the server, and the SESSION-ID-LENGTH will be non-zero. When there is no session-identifier for the server SESSION-ID-LENGTH must be zero. The challenge data is used to authenticate the server. After the client and Server agree on a pair of session keys, the server returns a SERVER-VERIFY message with the encrypted form of the CHALLENGE-DATA.

Also note that the server will not send its SERVER-HELLO message until it has received the CLIENT-HELLO message. This is done so that the server can indicate the status of the client's session-identifier back to the client in the server's first message (i.e. to increase protocol efficiency and reduce the number of round trips required).

The server examines the CLIENT-HELLO message and will verify that it can support the client version and one of the client cipher specs. The server can optionally edit the cipher specs, removing any entries it doesn't choose to support. The edited version will be returned in the SERVER-HELLO message if the session-identifier is not in the server's cache.

The CIPHER-SPECS-LENGTH must be greater than zero and a multiple of 3.

The SESSION-ID-LENGTH must either be zero or 16.

The CHALLENGE-LENGTH must be greater than or equal to 16 and less than or equal to 32.

This message must be the first message sent by the client to the server. After the message is sent the client waits for a SERVER-HELLO message. Any other message returned by the server (other than ERROR) is disallowed.

CLIENT-MASTER-KEY (Phase 1; Sent primarily in the clear)

Char MSG-CLIENT-MASTER-KEY

Char CIPHER-KIND [3]

Char CLEAR-KEY-LENGTH-MSB

Char CLEAR-KEY-LENGTH-LSB

Char ENCRYPTED-KEY-LENGTH-MSB

Char ENCRYPTED-KEY-LENGTH-LSB

Char KEY-ARG-LENGTH-MSB

Char KEY-ARG-LENGTH-LSB

Char CLEAR-KEY-DATA [MSB<<8|LSB]

Char ENCRYPTED-KEY-DATA [MSB<<8|LSB]

Char KEY-ARG-DATA [MSB<<8|LSB]

The client sends this message when it has determined a master key for the server to use. Note that when a session-identifier has been agreed upon, this message is not sent.

The CIPHER-KIND field indicates which cipher was chosen from the server's CIPHER-SPECS.

The CLEAR-KEY-DATA contains the clear portion of the MASTER-KEY.

The CLEAR-KEY-DATA is combined with the SECRET-KEY-DATA (described shortly) to form the MASTER-KEY, with the SECRET-KEY-DATA being the least significant bytes of the final MASTER-KEY.

The ENCRYPTED-KEY-DATA contains the secret portions of the MASTER-KEY, encrypted using the server's public key. The encryption block is formatted using block type 2 from PKCS#1 [5].

The data portion of the block is formatted as follows:

Char SECRET-KEY-DATA [SECRET-LENGTH]

SECRET-LENGTH is the number of bytes of each session key that is being transmitted encrypted. The SECRET-LENGTH plus the CLEAR-KEY-LENGTH equals the number of bytes present in the cipher key (as defined by the CIPHER-KIND). It is an error if the SECRET-LENGTH found after decrypting the PKCS#1 formatted encryption block doesn't

match the expected value. It is also an error if CLEAR-KEY-LENGTH is non-zero and the CIPHER-KIND is not an export cipher.

If the key algorithm needs an argument (for example, DES-CBC's Initialization vector) then the KEY-ARG-LENGTH fields will be non-zero and the KEY-ARG-DATA will contain the relevant data.

For the SSL_CK_RC2_128_CBC_WITH_MD5,
SSL_CK_RC2_128_CBC_EXPORT40_WITH_MD5,
SSL_CK_IDEA_128_CBC_WITH_MD5,
SSL_CK_DES_64_CBC_WITH_MD5

And SSL_CK_DES_192_EDE3_CBC_WITH_MD5 algorithms the KEY-ARG data must be present and be exactly 8 bytes long.

Client and server session key production is a function of the CIPHER-CHOICE:

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_IDEA_128_CBC_WITH_MD5

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

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

CLIENT-READ-KEY = KEY-MATERIAL-0[0-15]

CLIENT-WRITE-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.

Data is fed to the MD5 hash function in the order shown, from left to right: first the MASTER-KEY, then the "0" or "1", then the CHALLENGE and then finally the CONNECTION-ID.

Note: The "0" means the ascii zero character (0x30), not a zero value. "1" means the ascii 1 character (0x31). MD5 produces 128 bits of output data which are used directly as the key to the cipher algorithm (The most significant byte of the MD5 output becomes the most significant byte of the key material).

SSL_CK_DES_64_CBC_WITH_MD5

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

CLIENT-READ-KEY = KEY-MATERIAL-0[0-7]

CLIENT-WRITE-KEY = KEY-MATERIAL-0[8-15]

For DES-CBC, a single 16 bytes of key material are produced using MD5. The first 8 bytes of the MD5 digest are used as the CLIENT-READ-KEY while the remaining 8 bytes are used as the CLIENT-WRITE-KEY. The initialization vector is provided in the KEY-ARG-DATA. Note that the raw key data is not parity adjusted and that this step must be performed before the keys are legitimate DES keys.

SSL_CK_DES_192_EDE3_CBC_WITH_MD5

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

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

KEY-MATERIAL-2 = MD5 [MASTER-KEY, "2", CHALLENGE, CONNECTION-ID]

CLIENT-READ-KEY-0 = KEY-MATERIAL-0[0-7]

CLIENT-READ-KEY-1 = KEY-MATERIAL-0[8-15]

CLIENT-READ-KEY-2 = KEY-MATERIAL-1[0-7]

CLIENT-WRITE-KEY-0 = KEY-MATERIAL-1[8-15]

CLIENT-WRITE-KEY-1 = KEY-MATERIAL-2[0-7]

CLIENT-WRITE-KEY-2 = KEY-MATERIAL-2[8-15]

Data is fed to the MD5 hash function in the order shown, from left to right: first the MASTER-KEY, then the "0", "1" or "2", then the CHALLENGE and then finally the CONNECTION-ID.

Note: The "0" means the ascii zero character (0x30), not a zero value. "1" means the ascii 1 character (0x31). "2" means the ascii 2 character (0x32).

A total of 6 keys are produced, 3 for the read side DES-EDE3 cipher and 3 for the write side DES-EDE3 function. The initialization vector is provided in the KEY-ARG-DATA. The keys that are produced are not parity adjusted. This step must be performed before proper DES keys are usable.

Recall that the MASTER-KEY is given to the server in the CLIENT-MASTER-KEY message. The client in the CLIENT-HELLO message gives the CHALLENGE to the server. The server in the SERVER-HELLO

message gives the CONNECTION-ID to the client. This makes the resulting cipher keys a function of the original session and the current session. Note that the master key is never directly used to encrypt data, and therefore cannot be easily discovered.

The CLIENT-MASTER-KEY message must be sent after the CLIENT-HELLO message and before the CLIENT-FINISHED message.

The CLIENT-MASTER-KEY message must be sent if the SERVER-HELLO message contains a SESSION-ID-HIT value of 0.

CLIENT-CERTIFICATE (Phase 2; 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 [MSB<<8|LSB]

Char RESPONSE-DATA [MSB<<8|LSB]

One sends an SSL client in response to a server REQUEST-CERTIFICATE message this message. The CERTIFICATE-DATA contains data defined by the CERTIFICATE-TYPE value. An ERROR message is sent with error code NO-CERTIFICATE-ERROR when this request cannot be answered properly (e.g. the receiver of the message has no registered certificate).

CERTIFICATE-TYPE is one of:

SSL_X509_CERTIFICATE

The CERTIFICATE-DATA contains an X.509 (1988) [3] signed certificate.

The RESPONSE-DATA contains the authentication response data. This data is a function of the AUTHENTICATION-TYPE value sent by the server.

When AUTHENTICATION-TYPE is SSL_AT_MD5_WITH_RSA_ENCRYPTION then the RESPONSE-DATA contains a digital signature of the following components (in the order shown):

the KEY-MATERIAL-0

the KEY-MATERIAL-1 (only if defined by the cipher kind)

the KEY-MATERIAL-2 (only if defined by the cipher kind)

the CERTIFICATE-CHALLENGE-DATA (from the REQUEST-CERTIFICATE message)

the server's signed certificate (from the SERVER-HELLO message)

The digital signature is constructed using MD5 and then encrypted using the client's private key, formatted according to PKCS#1's digital signature standard [5]. The server authenticates the client by verifying the digital signature using standard techniques. Note that other digest functions are supported. Either a new AUTHENTICATION-TYPE can be added, or the algorithm-id in the digital signature can be changed.

The client only in response to a REQUEST-CERTIFICATE message must send this message.

CLIENT-FINISHED (Phase 2; Sent encrypted)

Char MSG-CLIENT-FINISHED

Char CONNECTION-ID [N-1]

The client sends this message when it is satisfied with the server.

Note: The client must continue to listen for server messages until it receives a SERVER-FINISHED message. The CONNECTION-ID data is the original connection-identifier the server sent with its SERVER-HELLO message, encrypted using the agreed upon session key.

"N" is the number of bytes in the message that was sent, so "N-1" is the number of bytes in the message without the message header byte. For version 2 of the protocol, the client must send this message after it has received the SERVER-HELLO message. If the SERVER-HELLO message SESSION-ID-HIT flag is non-zero then the CLIENT-FINISHED message is sent immediately, otherwise the CLIENT-FINISHED message is sent after the CLIENT-MASTER-KEY message.

(f) Server Only Protocol Messages

There are several messages that are only generated by servers. The messages are never generated by correctly functioning clients.

SERVER-HELLO (Phase 1; Sent in the 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[MSB<<8|LSB]

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

char CONNECTION-ID-DATA[MSB<<8|LSB]

The server sends this message after receiving the clients CLIENT-HELLO message.

The server returns the SESSION-ID-HIT flag indicating whether or not the received session-identifier is known by the server (i.e. in the server's session-identifier cache).

The SESSION-ID-HIT flag will be non-zero if the client sent the server a session-identifier (in the CLIENT-HELLO message with SESSION-ID-LENGTH!=0) and the server found the client's session-identifier in its cache. If the SESSION-ID-HIT flag is non-zero then the CERTIFICATE-TYPE, CERTIFICATE-LENGTH and CIPHER-SPECS-LENGTH fields will be zero.

The CERTIFICATE-TYPE value, when non-zero, has one of the values described above (see the information on the CLIENT-CERTIFICATE message). When the SESSION-ID-HIT flag is zero, the server packages up its certificate, its cipher specs and a connection-id to send to the client. Using this information the client can generate a session key and return it to the server with the CLIENT-MASTER-KEY message.

When the SESSION-ID-HIT flag is non-zero, both the server and the Client compute a new pair of session keys for the current session derived from the MASTER-KEY that was exchanged when the SESSION-ID was created.

The SERVER-READ-KEY and SERVER-WRITE-KEY are derived from the original MASTER-KEY keys in the same manner as the CLIENT-READ-KEY and CLIENT-WRITE-KEY:

$$\text{SERVER-READ-KEY} = \text{CLIENT-WRITE-KEY}$$
$$\text{SERVER-WRITE-KEY} = \text{CLIENT-READ-KEY}$$

Note: when keys are being derived and the SESSION-ID-HIT flag is set and the server discovers the client's session-identifier in the server's cache, then the KEY-ARG-DATA is used from the time when the SESSION-ID was established. This is because the client does not send new KEY-ARG-DATA (recall that the KEY-ARG-DATA is sent only in the CLIENT-MASTER-KEY message).

The CONNECTION-ID-DATA is a string of randomly generated bytes used by the server and client at various points in the protocol. The CLIENT-FINISHED message contains an encrypted version of the CONNECTION-ID-DATA. The length of the CONNECTION-ID must be between 16 and than 32 bytes, inclusive.

The CIPHER-SPECS-DATA defines a cipher type and key length (in bits) that the receiving end supports. Each SESSION-CIPHER-SPEC is 3 bytes long and looks like this:

char CIPHER-KIND-0

char CIPHER-KIND-1

char CIPHER-KIND-2

Where CIPHER-KIND is one of:

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_IDEA_128_CBC_WITH_MD5

SSL_CK_DES_64_CBC_WITH_MD5

SSL_CK_DES_192_EDE3_CBC_WITH_MD5

This list is not exhaustive and may be changed in the future.

The SSL_CK_RC4_128_EXPORT40_WITH_MD5 cipher is an RC4 cipher where some of the session key is sent in the clear and the rest is sent encrypted (exactly 40 bits of it). MD5 is used as the hash function for production of MAC's and session key's. This cipher type is provided to support "export" versions (i.e. versions of the protocol that can be distributed outside of the United States) of the client or server.

An exportable implementation of the SSL Handshake Protocol will have secret key lengths restricted to 40 bits. For non-export implementations key lengths can be more generous (we recommend at least 128 bits). It is permissible for the client and server to have a non-intersecting set of stream ciphers. This, simply put, means they cannot communicate.

Version 2 of the SSL Handshake Protocol defines the SSL_CK_RC4_128_WITH_MD5 to have a key length of 128 bits. The SSL_CK_RC4_128_EXPORT40_WITH_MD5 also has a key length of 128 bits.

However, only 40 of the bits are secret (the other 88 bits are sent in the clear by the client to the server).

The SERVER-HELLO message is sent after the server receives the CLIENT-HELLO message, and before the server sends the SERVER-VERIFY message.

SERVER-VERIFY (Phase 1; Sent encrypted)

char MSG-SERVER-VERIFY

char CHALLENGE-DATA[N-1]

The server sends this message after a pair of session keys (SERVER-READ-KEY and SERVER-WRITE-KEY) have been agreed upon either by a session-identifier or by explicit specification with the CLIENT-MASTER-KEY message. The message contains an encrypted copy of the CHALLENGE-DATA sent by the client in the CLIENT-HELLO message.

"N" is the number of bytes in the message that was sent, so "N-1" is the number of bytes in the CHALLENGE-DATA without the message header byte. This message is used to verify the server as follows. A legitimate server will have the private key that corresponds to the public key contained in the server certificate that was transmitted in the SERVER-HELLO message. Accordingly, the legitimate server will be able to extract and reconstruct the pair of session keys (SERVER-READ-KEY and SERVER-WRITE-KEY). Finally, only a server that has done the extraction and

decryption properly can correctly encrypt the CHALLENGE-DATA. This, in essence, "proves" that the server has the private key that goes with the public key in the server's certificate.

The CHALLENGE-DATA must be the exact same length as originally sent by the client in the CLIENT-HELLO message. Its value must match exactly the value sent in the clear by the client in the CLIENT-HELLO message. The client must decrypt this message and compare the value received with the value sent, and only if the values are identical is the server to be "trusted". If the lengths do not match or the value doesn't match then the connection is to be closed by the client.

This message must be sent by the server to the client after either detecting a session-identifier hit (and replying with a SERVER-HELLO message with SESSION-ID-HIT not equal to zero) or when the server receives the CLIENT-MASTER-KEY message. This message must be sent before any Phase 2 messages or a SEVER-FINISHED message.

SERVER-FINISHED (Phase 2; Sent encrypted) *
char MSG-SERVER-FINISHED
char SESSION-ID-DATA[N-1]

The server sends this message when it is satisfied with the clients security handshake and is ready to proceed with transmission/reception of the higher level protocols data. The SESSION-ID-DATA is used by the client and the server at this time to add entries to their respective session-identifier caches. The session-identifier caches must contain a copy of the MASTER-KEY sent in the CLIENT-MASTER-KEY message as the master key is used for all subsequent session key generation.

"N" is the number of bytes in the message that was sent, so "N-1" is the number of bytes in the SESSION-ID-DATA without the message header byte.

This message must be sent after the SERVER-VERIFY message.

REQUEST-CERTIFICATE (Phase 2; Sent encrypted)

char MSG-REQUEST-CERTIFICATE

char AUTHENTICATION-TYPE

char CERTIFICATE-CHALLENGE-DATA[N-2]

A server may issue this request at any time during the second phase of the connection handshake, asking for the client's certificate. The client responds with a CLIENT-CERTIFICATE message immediately if it has one, or an ERROR message (with error code NO-CERTIFICATE-ERROR) if it doesn't.

The CERTIFICATE-CHALLENGE-DATA is a short byte string (whose length is greater than or equal to 16 bytes and less than or equal to 32 bytes) that the client will use to respond to this message.

The AUTHENTICATION-TYPE value is used to choose a particular means of authenticating the client. The following types are defined:

SSL_AT_MD5_WITH_RSA_ENCRYPTION

The SSL_AT_MD5_WITH_RSA_ENCRYPTION type requires that the client construct an MD5 message digest using information as described above in the section on the CLIENT-CERTIFICATE message. Once the digest is created, the client encrypts it using its private key (formatted according to the digital signature standard defined in PKCS#1). The server authenticates the client when it receives the CLIENT-CERTIFICATE

message. This message may be sent after a SERVER-VERIFY message and before a SERVER-FINISHED message.

(g) Client/Server Protocol Messages

These messages are generated by both the client and the server.

ERROR (Sent clear or encrypted)

char MSG-ERROR

char ERROR-CODE-MSB

char ERROR-CODE-LSB

This message is sent when an error is detected. After the message is sent, the sending party shuts the connection down. The receiving party records the error and then shuts its connection down.

This message is sent in the clear if an error occurs during session key negotiation. After a session key has been agreed upon, errors are sent encrypted like all other messages.

3.3 SET PROTOCOL

(a) Cryptography Protection of sensitive information

Cryptography has been used for centuries to protect sensitive information as it is transmitted from one location to another. In a cryptographic system, a message is encrypted using a key. The resulting cipher text is then transmitted to the recipient where it is decrypted using a key to produce the original message. There are two primary encryption methods in use today: secret-key cryptography and public-key cryptography. SET uses both methods in its encryption process.

(b) Secret-key cryptography

Secret-key cryptography, also known as symmetric cryptography, uses the same key to encrypt and decrypt the message. Therefore, the sender and the recipient of a message must share a secret, namely the key. A well known secret-key cryptography algorithm is the Data Encryption Standard (DES), which is used by financial institutions to encrypt PINs (personal identification numbers).

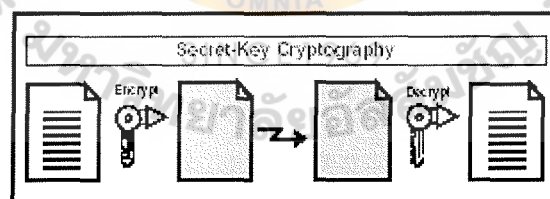


Figure 1: Secret-Key Cryptography

Figure 3.8. Secret-Key Cryptography.

(c) Public-key cryptography

Public-key cryptography, also known as asymmetric cryptography, uses two

keys: one key to encrypt the message and the other key to decrypt the message. The two keys are mathematically related so that data encrypted with either key can only be decrypted using the other. Each user has two keys: a *public key* and a *private key*. The user distributes the public key. Because of the relationship between the two keys, the user and anyone receiving the public key can be assured that data encrypted with the public key and sent to the user can only be decrypted when the user uses the private key. *This assurance is only maintained if the user ensures that the private key is not disclosed to anyone else.* Therefore, the key pair should be generated by the user. The best known public-key cryptography algorithm is RSA (Named after its inventors Rivest, Shamir, and Adleman).

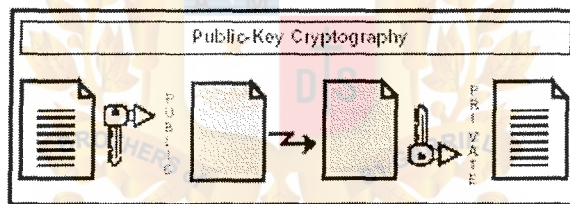


Figure 2: Public-Key Cryptography

Figure 3.9. Public-Key Cryptography.

Secret-key cryptography is impractical for exchanging messages with a large group of previously unknown correspondents over a public network. For a merchant to conduct transactions securely with millions of Internet subscribers, each consumer would need a distinct key assigned by that merchant and transmitted over a separate secure channel. On the other hand, by using public-key cryptography, that same merchant could create a public/private key pair and

publish the public key, allowing any consumer to send a secure message to that merchant.

(d) Encryption Relationship of keys

Confidentiality is ensured by the use of message encryption. When two users want to exchange messages securely, each of them transmits one component of their key pair, designated the public key, to the other and keeps secret the other component, and designated the private key. Because messages encrypted with the public key can only be decrypted using the private key, these messages can be transmitted over an insecure network without fear that an eavesdropper could use the key to read encrypted transmissions.

For example, Bob can transmit a confidential message to Alice by encrypting the message using Alice's public key. As long as Alice ensures that no one else has access to her private key, both she and Bob will know that only Alice can read the message.

(e) Use of symmetric key

SET will rely on cryptography to ensure message confidentiality. In SET, message data will be encrypted using a randomly generated symmetric encryption key. This key, in turn, will be encrypted using the message recipient's public key. This is referred to as the "digital envelope" of the message and is sent to the Recipient along with the encrypted message itself. After receiving the digital envelope, the recipient decrypts it using his or her private key to obtain the randomly generated symmetric key and then uses the symmetric key to unlock the original message.

Note: To provide the highest degree of protection, it is essential that the programming methods and random number generation algorithms generate keys in

a way that ensures that the keys cannot be easily reproduced using information about either the algorithms or the environment in which the keys are generated.

(f) Digital signatures

Integrity and authentication are ensured by the use of digital signatures.

(g) Relationship of keys

Because of the mathematical relationship between the public and private keys, data encrypted with either key can only be decrypted with the other. This allows the sender of a message to encrypt it using the sender's private key. Any recipient can determine that the message came from the sender by decrypting the message using the sender's public key. For example, Alice can encrypt a known piece of data, such as her telephone number, with her private key and transmit it to Bob. When Bob decrypts the message using Alice's public key and compares the result to the known data, he can be sure that the message could only have been encrypted using Alice's private key.

(h) Using message digests

When combined with *message digests*, encryption using the private key allows users to digitally sign messages. A message digest is a value generated for a message (or document) that is unique to that message.¹ A message digest is generated by passing the message through a one-way cryptographic function; that is, one that cannot be reversed. When the digest of a message is encrypted using the sender's private key and is appended to the original message, the result is known as the digital signature of the message.

The recipient of the digital signature can be sure that the message really came from the sender. And, because changing even one character in the message changes the message digest in an unpredictable way, the recipient can be sure that

the message was not changed after the message digest was generated.

(i) Example of the use of a digital signature

For example, Alice computes the message digest of a property description and encrypts it with her private key yielding a digital signature for the message. She transmits both the message and the digital signature to Bob. When Bob receives the message, he computes the message digest of the property description and decrypts the digital signature with Alice's public key. If the two values match, Bob knows that the message was signed using Alice's private key and that it has not changed since it was signed.

(j) Two key pairs

SET uses a distinct public/private key pair to create the digital signature. Thus, each SET participant will possess two asymmetric key pairs: a "key exchange" pair, which is used in the process of encryption and decryption, and a "signature" pair for the creation and verification of digital signatures.

Note: The roles of the public and private keys are reversed in the digital signature process where the private key is used to encrypt (sign) and the public key is used to decrypt (verify the signature).

(k) Certificates Need for authentication

Authentication is further strengthened by the use of certificates. Before two Parties use public-key cryptography to conduct business, each wants to be sure that the other party is authenticated. Before Bob accepts a message with Alice's digital signature, he wants to be sure that the public key belongs to Alice and not to someone masquerading as Alice on an open network. One way to be sure that the public key belongs to Alice is to receive it over a secure channel directly from Alice. However, in most circumstances this solution is not practical.

(a) Need for a trusted third party

An alternative to secure transmission of the key is to use a trusted third party to authenticate that the public key belongs to Alice. Such a party is known as a *Certificate Authority* (CA). The Certificate Authority authenticates Alice's claims according to its published policies. For example, a Certificate Authority could supply certificates that offer a high assurance of personal identity, which may be required for conducting business transactions; this Certificate Authority may require Alice to present a driver's license or passport to a notary public before it will issue a certificate.

Once Alice has provided proof of her identity, the Certificate Authority creates a message containing Alice's name and her public key. The Certificate Authority digitally signs this message, known as a certificate. It contains owner identification information, as well as a copy of one of the owner's public keys ("key exchange" or "signature"). To get the most benefit, the public key of the Certificate Authority should be known to as many people as possible. Thus, by trusting a single key, an entire hierarchy can be established in which one can have a high degree of trust. Because SET participants have two key pairs, they also have two certificates. Both certificates are created and signed at the same time by the Certificate Authority.

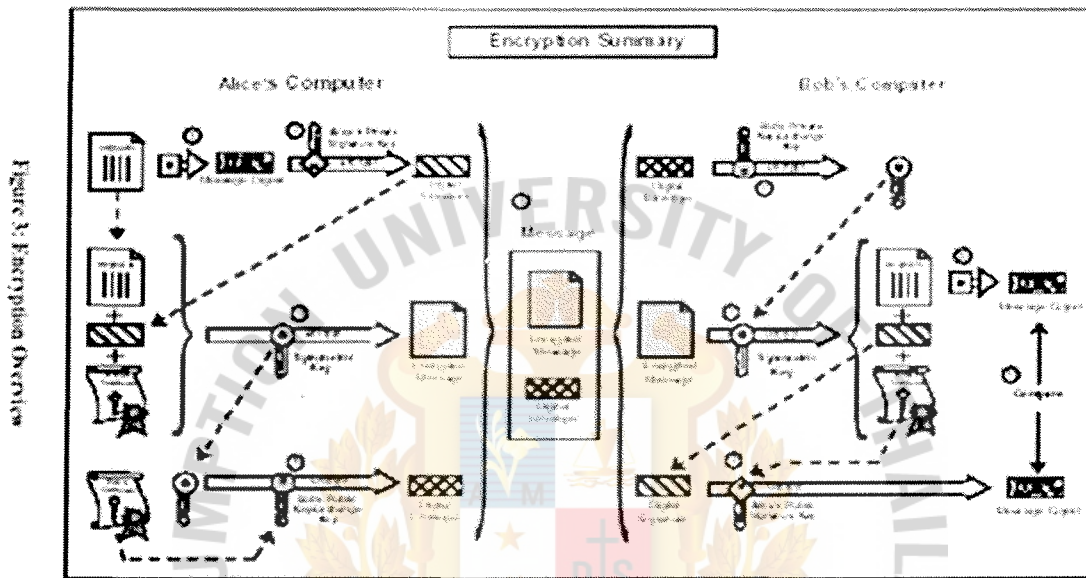
(m) SET authentication

The means that a financial institution uses to authenticate a cardholder or merchant is not defined by this specification. Each payment card brand and financial institution will select an appropriate method.

(n) Encryption summary

This diagram provides an overview of the entire encryption process when

Alice wishes to sign, for example, a property description and send it in an Encrypted message to Bob. The numbered steps in the diagram are explained on the following pages.



* Figure 3.10. Encryption Overview. *

Encryption The encryption process in Figure 3 consists of the following steps:

Step Description

- (i) Alice runs the property description through a one-way algorithm to produce a unique value known as the message digest. This is a kind of digital fingerprint of the property description and will be used later to test the integrity of the message.
- (ii) She then encrypts the message digest with her private signature key to

produce The digital signature.

- (iii) Next, she generates a random symmetric key and uses it to encrypt the Property description, her signature and a copy of her certificate, which Contains her public signature key. To decrypt the property description, Bob will require a secure copy of this random symmetric key.
- (iv) Bob's certificate, which Alice must have obtained prior to initiating secure communication with him, contains a copy of his public key-exchange key. To ensure secure transmission of the symmetric key, Alice encrypts it using Bob's public key-exchange key. The encrypted key, referred to as the digital envelope, will be sent to Bob along with the encrypted message itself.
- (v) Alice sends a message to Bob consisting of the following: the symmetrically encrypted property description, signature and certificate, as well as the asymmetrically encrypted symmetric key (the digital envelope).
- (vi) Bob receives the message from Alice and decrypts the digital envelope with His private key-exchange key to retrieve the symmetric key.
- (vii) He uses the symmetric key to decrypt the property description, Alice's signature, and her certificate.
- (viii) He decrypts Alice's digital signature with her public signature key, which he acquires from her certificate. This recovers the original message digest of the property description.
- (ix) He runs the property description through the same one-way algorithm used by Alice and produces a new message digest of the decrypted property description.
- (x) Finally, he compares his message digest to the one obtained from Alice's Digital signature. If they are exactly the same, he confirms that the message

content has not been altered during transmission and that it was signed using Alice's private signature key.

If they are not the same, then the message either originated somewhere else Or was altered after it was signed. In that case, Bob takes some appropriate action such as notifying Alice or discarding the message.

(o) Dual signature

SET introduces a new application of digital signatures, namely the concept of dual signatures. To understand the need for this new concept, consider the following scenario: Bob wants to send Alice an offer to purchase a piece of property and an authorization to his bank to transfer the money if Alice accepts the offer, but Bob doesn't want the bank to see the terms of the offer nor does he want Alice to see his account information. Further, Bob wants to link the offer to the Transfer so that the money is only transferred if Alice accepts his offer. He accomplishes all of this by digitally signing both messages with a single signature operation that creates a dual signature.

(p) Generating a dual signature

A dual signature is generated by creating the message digest of both messages, concatenating the two digests together, computing the message digest of the result and encrypting this digest with the signer's private signature key. The signer must include the message digest of the other message in order for the recipient to verify the dual signature. A recipient of either message can check its authenticity by generating the message digest on its copy of the message, concatenating it with the message digest of the other message (as provided by the sender) and computing the message digest of the result. If the newly generated digest matches the decrypted dual signature, the recipient can trust the authenticity

of the message.

Example: If Alice accepts Bob's offer, she can send a message to the bank indicating her acceptance and including the message digest of the offer. The bank can verify the authenticity of Bob's transfer authorization and ensure that the acceptance is for the same offer by using its digest of the authorization and the message digest presented by Alice of the offer to validate the dual signature. Thus the bank can check the authenticity of the offer against the dual signature, but the bank cannot see the terms of the offer.

(q) Use of dual signatures

Within SET, dual signatures are used to link an order message sent to the merchant with the payment instructions containing account information sent to the Acquirer. When the merchant sends an authorization request to the Acquirer, it includes the payment instructions sent to it by the cardholder and the message digest of the order information. The Acquirer uses the message digest from the merchant and computes the message digest of the payment instructions to check the dual signature.

(r) Import/export issues

A number of governments have regulations regarding the import or export of cryptography. As a general rule; these governments allow cryptography to be used when:

- (i) the data being encrypted is of a financial nature;
- (ii) the content of the data is well-defined;
- (iii) the length of the data is limited; and
- (iv) the cryptography cannot easily be used for other purposes.

The SET protocol is limited to the financial portion of shopping and the

content of the SET messages has been carefully reviewed to satisfy the concerns of governments. As long as software vendors can demonstrate that the cryptography used for SET cannot easily be put to other purposes, import and export licenses should be obtainable.

(s) Certificate Issuance Cardholder certificates

Cardholder certificates function as an electronic representation of the Payment card. Because a financial institution digitally signs them, they cannot be altered by a third party and can only be generated by a financial institution. A cardholder certificate does not contain the account number and expiration date. Instead the account information and a secret value known only to the cardholder's software are encoded using a one-way hashing algorithm. If the account number, expiration date, and the secret value are known, the link to the certificate can be proven, but looking at the certificate cannot derive the information.

Within the SET protocol, the cardholder supplies the account information And the secret value to the payment gateway where the link is verified. A certificate is only issued to the cardholder when the cardholder's issuing financial institution approves it. By requesting a certificate, a cardholder has indicated the intent to perform commerce via electronic means. This certificate is transmitted to merchants with purchase requests and encrypted payment instructions. Upon receipt of the cardholder's certificate, a merchant can be assured, at a minimum, that the account number has been validated by the card-issuing financial institution or its agent. In this specification, cardholder certificates are optional at the payment card brand's discretion.

(t) Merchant certificates

Merchant certificates function as an electronic substitute for the payment

brand decal that appears in the store window—the decal itself is a representation that the merchant has a relationship with a financial institution allowing it to accept the payment card brand. Because they are digitally signed by the merchant's financial institution, merchant certificates cannot be altered by a third party and can only be generated by a financial institution. These certificates are approved by the acquiring financial institution and provide assurance that the merchant holds a valid agreement with an Acquirer. A merchant must have at least one pair of certificates to participate in the SET environment, but there may be multiple certificate pairs per merchant. A merchant will have a pair of certificates for each payment card brand that it accepts.

(u) Payment gateway certificates

Payment gateway certificates are obtained by Acquirers or their processors for the systems that process authorization and capture messages. The gateway's encryption key, which the cardholder gets from this certificate, is used to protect the cardholder's account information. Payment gateway certificates are issued to the Acquirer by the payment brand.

(v) Acquirer certificates

An Acquirer must have certificates in order to operate a Certificate Authority that can accept and process certificate requests directly from merchants over public and private networks. Those Acquirers that choose to have the payment card brand process certificate requests on their behalf will not require certificates because they are not processing SET messages. Acquirers receive their certificates from the payment card brand.

(w) Issuer certificates

An Issuer must have certificates in order to operate a Certificate Authority

that can accept and process certificate requests directly from cardholders over public and private networks. Those Issuers that choose to have the payment card brand process certificate requests on their behalf will not require certificates because they are not processing SET messages. Issuers receive their certificates from the payment card brand.

(x) Hierarchy of trust

SET certificates are verified through a hierarchy of trust. Each certificate is linked to the signature certificate of the entity that digitally signed it. By following the trust tree to a known trusted party, one can be assured that the certificate is valid. For example, a cardholder certificate is linked to the certificate of the Issuer (or the Brand on behalf of the Issuer). The Issuer's certificate is linked back to a root key through the Brand's certificate.

The public signature key of the root is known to all SET software and may be used to verify each of the certificates in turn. The following diagram illustrates the hierarchy of trust.

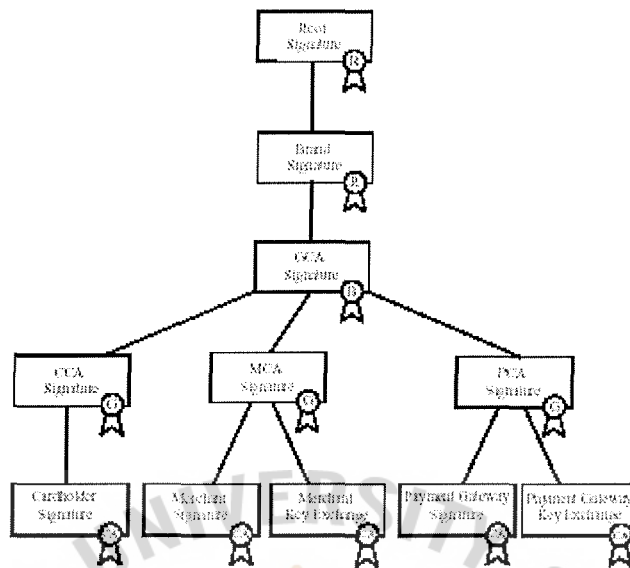


Figure 4: Hierarchy of Trust

Figure 3.11. Hierarchy of Trust.

The number of levels shown in this diagram is illustrative. A payment card Brand may not always operate a geopolitical Certificate Authority between itself and the financial Institutions.

Root key distribution

The root key will be distributed in a self-signed certificate. This root Key certificate will be available to software vendors to include with their software.

Root key validation

Software can confirm that it has a valid root key by sending an initiate Request to The certificate Authority that contains the hash of the root certificate.

In the event that the software does not have a valid root certificate, the

Certificate Authority will send one in the response.

Note: In this extremely unusual case where the software's root key is invalid, the user (cardholder or merchant) will have to enter a string that corresponds to the hash of the certificate. This confirmation hash must be obtained from a reliable source, such as the cardholder's financial institution.

Root key replacement

When the root key is generated, a replacement key is also generated. The replacement key is stored securely until it is needed. The self-signed Root certificate and the hash of the replacement key are distributed together. Software will be notified of the replacement through a message that contains a self-signed certificate of the replacement root and the hash of the next replacement root key.

Software validates the replacement root key by calculating its hash and comparing it with the hash of the replacement key contained in the root certificate.

(y) Payment Processing

(i) Overview

- (a) Purpose This chapter describes the flow of transactions as they are processed by various systems.
- (b) Protocol description In the event that the description of the processing in this book differs from that in Book 3: Formal Protocol Definition, the Formal Protocol Definition takes precedence.
- (c) Transactions described SET defines a variety of transaction protocols that use the cryptographic concepts introduced in Chapter 3 to securely conduct electronic commerce. This chapter describes the following

transactions:

- (d) Other transactions The additional transactions listed below are part of the SET specification, but are not described in this book. For more information about these transactions, see Book 2: Programmer's Guide. Certificate inquiry and status If the CA is unable to complete the Processing of a certificate request quickly, it will send a reply to the cardholder or merchant indicating that the requester should check back later. The cardholder or merchant sends the *Certificate Inquiry* message to determine the status of the certificate request and to receive the certificate if the request has been approved.

Purchase inquiry Allows the cardholder to check the status of the processing of an order after the purchase response has been received.

Note: This message does not include information such as the status of back ordered goods, but does indicate the status of authorization, capture and credit processing. Authorization reversal allows a merchant to correct previous authorization requests. If the order will not be completed, the merchant reverses the entire authorization. If part of the order will not be completed (such as when goods are back ordered), the merchant reverses part of the amount of the authorization. Capture reversal allows a merchant to correct errors in capture requests such as transaction amounts that were entered incorrectly by a clerk.

Credit allows a merchant to issue a credit to a cardholder's account such as when goods are returned or were damaged during shipping. Note that the SET *Credit* message is always initiated by the merchant, not the cardholder. All communications between the cardholder and merchant that

result in a credit being processed happen outside of SET. Credit reversal
Allows a merchant to correct a previously request credit. Payment gateway
Certificate request Allows a merchant to query the Payment Gateway and
Receive a copy of the gateway's current key - exchange and signature
Certificates . Batch administration allows a merchant to communicate
information to the Payment Gateway regarding merchant batches. Error
message indicates that a responder rejects a message because it fails format
or content verification tests.

(e) Guide to the diagrams

The following abbreviations are use in the detailed diagrams in this
chapter to indicate the participant who digitally signs a message or certificate.

Initial	Participant
C	Cardholder
M	Merchant
P	Payment Gateway
CA	Certificate Authority

Figure 3.12. Detailed Diagrams.

The following symbols are used in the detailed diagrams:

Symbol	Description
	<p>These are cryptographic keys.</p> <ul style="list-style-type: none"> * The “teeth” of the key indicate the key’s owner. * Keys with “PK” on the handle are public keys, and those with “PK” are private keys. Private keys are always known to their owner. * Keys with a diamond (◆) are signature keys and those with a small key (■) are key-exchange keys.
	<p>This is a digital signature. The initial indicates which private key was used to create the signature. For example, this signature was created by the merchant private signature key.</p>
	<p>This is a dual signature. The initial indicates which private key was used to create the signature. For example, this dual signature was created by the cardholder private signature key.</p>
	<p>These are certificates.</p> <ul style="list-style-type: none"> * The initial in the “seal” indicates which private key was used to sign the certificate. * The letter on the certificate indicates the public key being certified. * The diamond and key symbols distinguish signature certificates from key-exchange certificates. <p>The “CA” in these symbols indicates that these certificates were created by the Certificate Authority, and the “M” indicates they are merchant certificates.</p>
	<p>This is a symmetric key used to encrypt data. It will always be sent with the encrypted data in the digital envelope. The number following the key differentiates symmetric keys used in a transaction set.</p>
	<p>This is a payment card and is used to indicate when the cardholder’s account number is being transmitted in the digital envelope along with the symmetric encryption key.</p>
	<p>This is protected data. It is used to represent account information sent in the digital envelope of registration requests for merchants and payment gateways.</p>
	<p>This is an encrypted message including the digital envelope. The data in the shaded region has been encrypted using a randomly generated symmetric key (identified here as the second such key generated for this transaction set). The entity whose public key was used to encrypt the envelope is identified above the envelope (in this case, the Payment Gateway).</p> <p>Note that in this case the digital envelope includes both the symmetric key and the cardholder’s account number. Also note that the portion of the message encrypted using the symmetric key contains the cardholder’s signature certificate and was dual signed by the cardholder.</p>

Figure 3.13. Guide to Diagrams, continued.

(f) Certificate Authority functions

The primary functions of the Certificate Authority are to:

- (i) Receive registration requests,
- (ii) Process and approve/decline requests,
- (iii) Issue certificates. The processing flows describe these functions as though they are performed by a single entity, but they actually may be performed by one to three entities. Payment card brands and individual financial institutions will review their business needs for these functions to select a solution for implementation. The selected solution may be to implement a single - server device that provides the Certificate Authority functions or multiple devices that distribute the processing.

The following list suggests some *possible* arrangements with variations on distribution:

- (a) A company that issues proprietary cards may perform all three steps for its cardholders.
- (b) A financial institution may receive, process, and approve certificate requests for its cardholders or merchants, and forward the information to the appropriate payment card brand(s) to issue the certificates.
- (c) An independent *Registration Authority* that processes payment card certificate applications for multiple payment card brands may receive certificate requests and forward them to the appropriate financial institution (Issuer or Acquirer) for processing; the financial institution forwards approved requests to

the payment card brands to issue the certificates.

These scenarios simply suggest some possible arrangements. Payment card brands and financial institutions will select an appropriate solution based on their individual business needs.

Optional cardholder certificates

The diagrams and processing flows that follow describe the processing of the transactions when the cardholder is in possession of a signature certificate issued under the trust hierarchy of the payment card brand. Payment card brands at their option may allow cardholders to process transactions without a certificate as a temporary measure to facilitate implementation of this specification.

(i) Cardholder authentication

The SET protocol uses a cardholder signature certificate to confirm that a transaction is from a registered user of a payment card.

(ii) Strength of cardholder certificates

A cardholder certificate is not a guarantee of the identity of the cardholder. The strength of a cardholder certificate is wholly dependent on the methods used by the payment card brand and the payment card issuer to authenticate the cardholder prior to the certificate being issued.

(vii) No digital signature

When a cardholder does not possess a signature certificate, no digital signature is generated. In place of the digital signature, the cardholder generates the message digest of the data and inserts the message digest into

the digital envelope.

(a) Assurance of integrity

The recipient of data from the cardholder uses the message digest from the digital envelope to confirm the integrity of the data.

(b) Cardholder Registration

Figure 6 provides a high-level overview of the cardholder registration process, showing its seven fundamental steps. The detailed sections that follow describe each step. The icon to the left corresponds to Figure 6 and serves as a map to the scenario; it is repeated in the more detailed sections with a shaded region that indicates which step is being described.



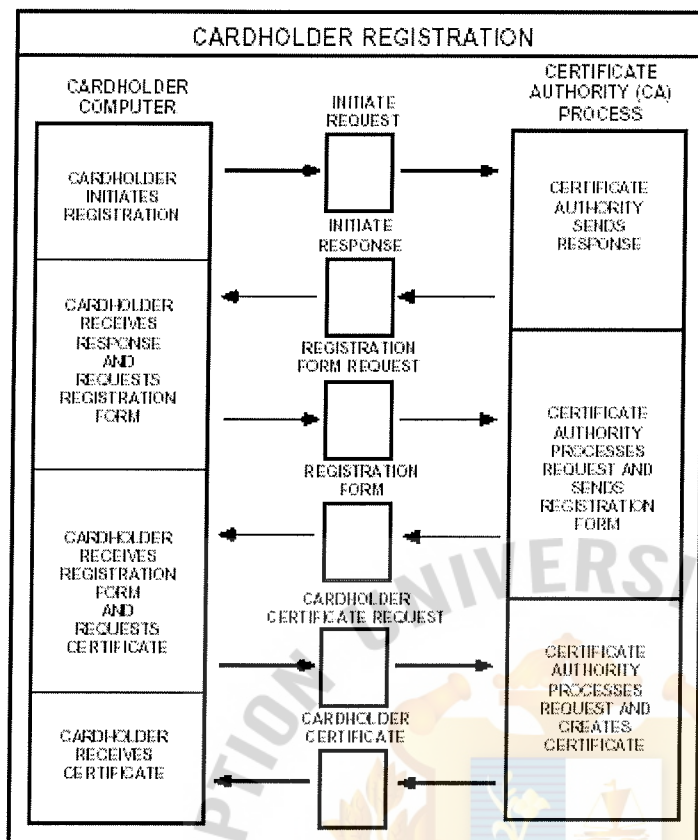


Figure 6: Cardholder Registration

Figure 3.14. Cardholder Registration.

Cardholders must register with a Certificate Authority (CA) Before they can send SET messages to merchants. In order to send SET Messages to the CA, the cardholder must have a copy of the CA public key - exchange key, which is provided in the CA key-exchange certificate.

The cardholder also needs a copy of the registration form from the cardholder's financial institution. In order for the CA to provide the registration form, the cardholder software must identify the issuing financial institution to the CA. Obtaining the registration form requires

two exchanges between the cardholder software and the CA. The registration process is started when the cardholder software requests a copy of the CA's key-exchange certificate.

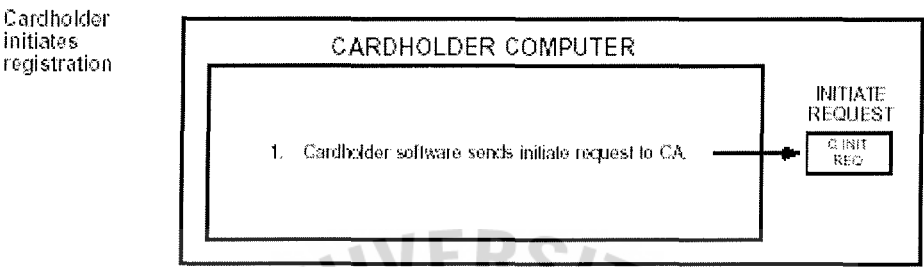


Figure 3.15. Cardholder initiates registration.

When the CA receives the request, it transmits its certificates to the cardholder. The CA key encryption certificate provides the cardholder software with the information necessary to protect the payment card account number in the registration form request.

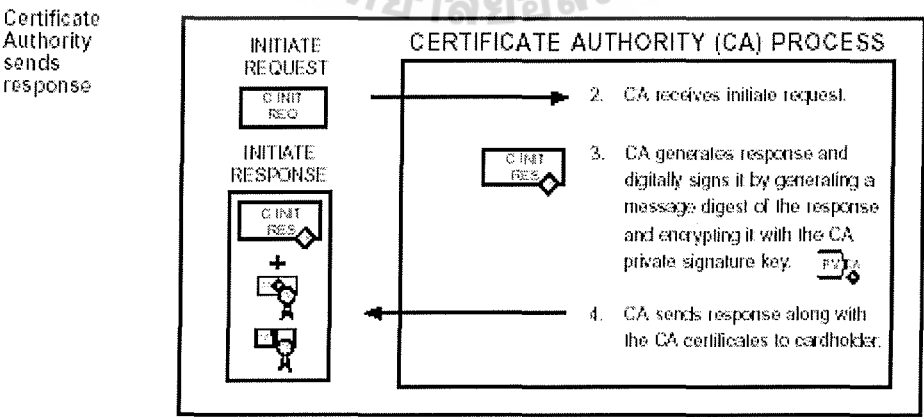


Figure 3.16. Certificate Authority sends response.

The cardholder software verifies the CA certificate by traversing the trust chain to the root key, as described in Section 3.3. The software must hold the CA certificates to use later during the registration process. Once the software has a copy of the CA key-exchange certificate, the cardholder can request a registration form. The cardholder software creates a registration form request message. Next the software generates a random symmetric encryption key. It uses this random key to encrypt the registration form request message. The random key is then encrypted along with the account number into the digital envelope using the CA public key-exchange key. Finally, the software transmits all of these components to the CA.

The cardholder software:

- (a) verifies the CA certificate by traversing the trust chain to the root key,
- (b) holds the CA certificates to use later during the registration process,
- (c) creates a registration form request message,
- (d) generates a random symmetric encryption key,
- (e) uses this random key to encrypt the registration form request message,
- (f) encrypts the random key along with the account number into the digital envelope using the CA public key-exchange key, transmits all of these components to the CA.

Cardholder receives response and requests registration form

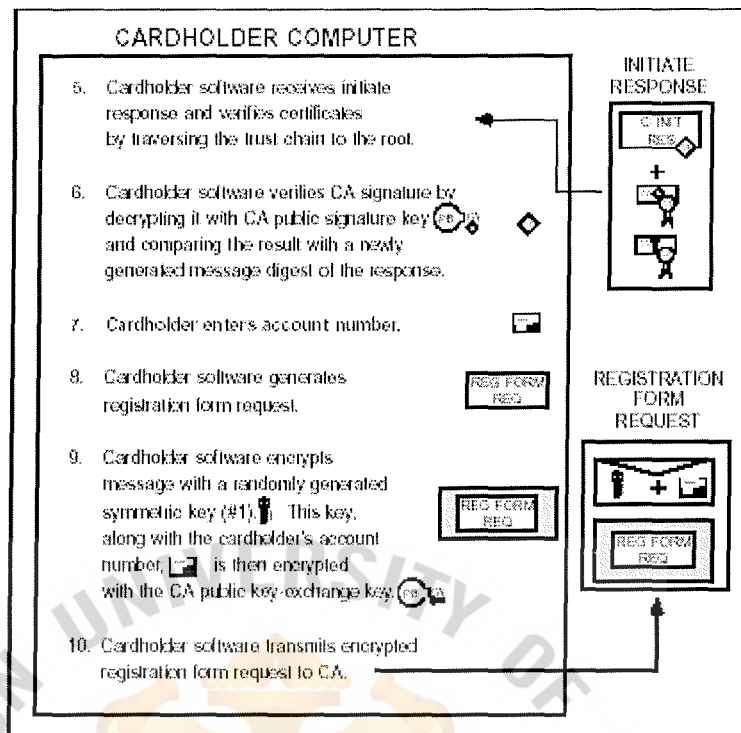


Figure 3.17. Cardholder receives response and requests registration form.

Cardholder Registration : The CA identifies the cardholder's Financial institution (using the first six to eleven digits of the account number) and selects the appropriate registration form. It digitally signs and then returns this registration form to the cardholder. In some cases, the CA may not have a copy of the registration form but can inform the cardholder software where the form can be obtained. For example, the cardholder's issuing financial institution may operate its own CA. In this event, the CA returns a referral response instead of the registration form. (This referral response is not shown in the diagram below.)

Certificate Authority processes request and sends registration form

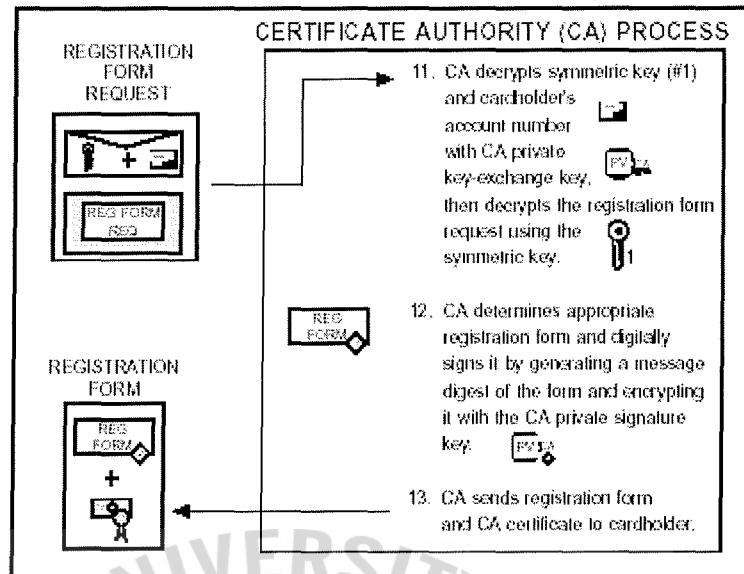


Figure 3.18. Certificate Authority processes request and sends registration form.

The cardholder software verifies the CA certificate by traversing the trust chain to the root key. The cardholder needs a signature public/private key pair for use with SET. The cardholder software generates this key pair if it does not already exist. To register an account, the cardholder fills out the registration form that was returned by the CA with information such as the cardholder's name, expiration date, account billing address, and any additional information the issuing financial institution deems necessary to identify the certificate requester as the valid cardholder. The cardholder software generates a random number that will be used by the CA in generating the certificate. The usage of this random number is described in the processing performed by the CA.

The cardholder software takes this registration information and

combines it with the public key in a registration message. The software digitally signs the registration message. Next the software generates two random symmetric encryption keys. The software places one random key inside the message; the CA will use this key to encrypt the response. It uses the other random key to encrypt the registration message. This random key is then encrypted along with the account number, expiration date, and the random number into the digital envelope using the CA public key-exchange key. Finally, the software transmits all of these components to the CA.

Note: If the CA returned a referral response as described earlier in the CA processing, the cardholder software will return to the beginning of the registration process communicating with the referral CA to receive that CA's certificates and the appropriate registration form.

Cardholder receives registration form and requests certificate

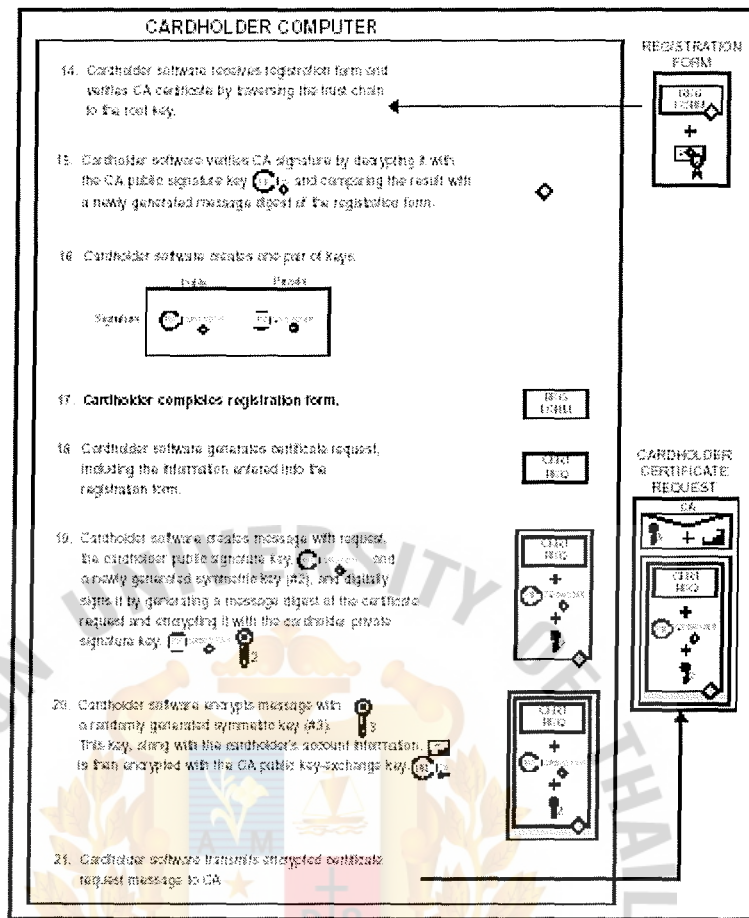


Figure 3.19. Cardholder receives registration form and requests certificate.

When the CA receives the cardholder's request, it decrypts the digital envelope to obtain the symmetric encryption key, the account information, and the random number generated by the cardholder software. It uses the symmetric key to decrypt the registration request. It then uses the signature key in the message to ensure the request was signed using the corresponding private signature key. If the signature is verified, the message processing continues; otherwise, the message is rejected and an appropriate response message is returned to the

cardholder. Next the CA must verify the information from the registration request using the cardholder's account information. The process by which the CA and the Issuer exchange information and the steps taken to verify the information in the registration request are outside the scope of this specification. As described in Section 4.1, there are several ways to configure the processing performed by the CA and the Issuer, such as having the payment card brand provide some or all of the functions on behalf of the Issuer or having the Issuer provide all of the functions. If the information in the registration request is verified, a certificate will be issued. First, the CA generates a random number that is combined with the random number created by the cardholder software to generate a secret value. This secret value is used to protect the account information in the cardholder certificate. The account number, expiration date, and the secret value are encoded using a one-way hashing algorithm. The result of the hashing algorithm is placed into the cardholder certificate. If the account number, expiration date, and the secret value are known, the link to the certificate can be proven, but looking at the certificate cannot derive the information.

Next, the CA creates and digitally signs the cardholder certificate. The validity period of this certificate will be determined by CA policy; often it will correspond to the expiration date of the payment card, but it may expire sooner. A response message containing the random number generated by the CA and other information (such as the brand logo) is then generated and encrypted using the symmetric key sent by

the cardholder software in the registration message. The response is then transmitted to the cardholder.

4.2 Cardholder Registration, continued

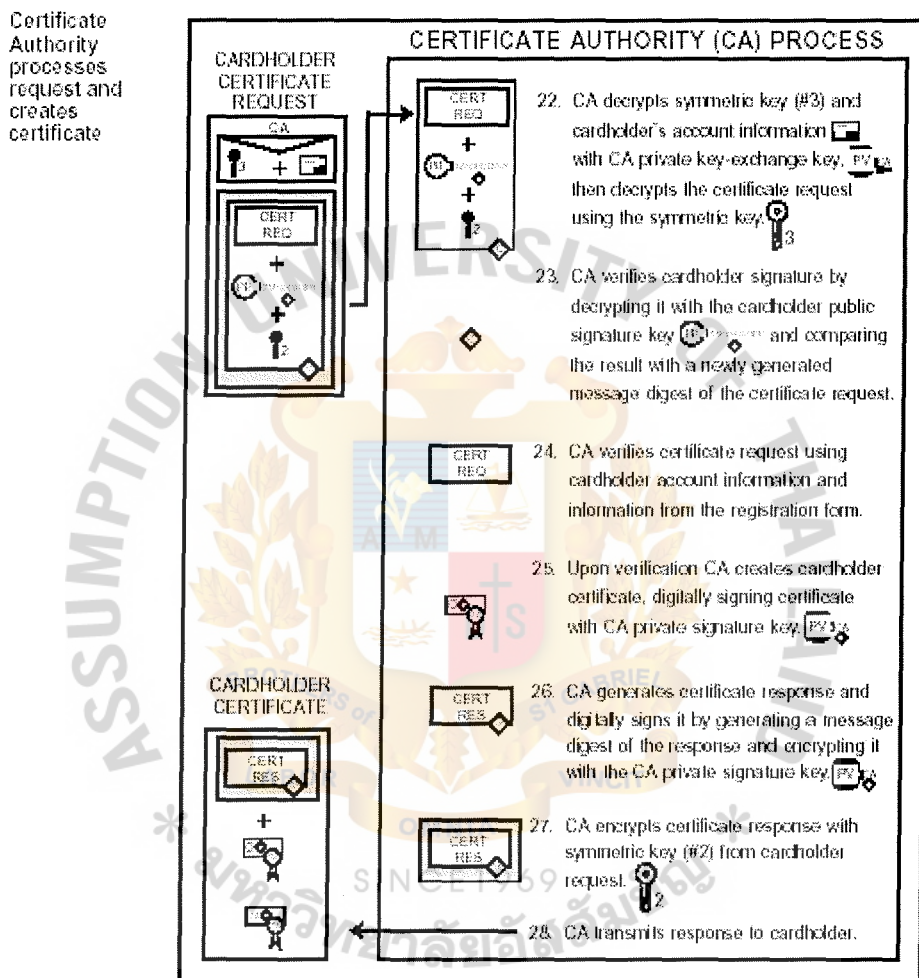


Figure 3.20. Certificate Authority processes request and creates certificate.

When the cardholder software receives the response from the CA, it verifies the certificate by traversing the trust chain to the root key, as described in Section 3.3. It stores the certificate on the cardholder's

computer for use in future electronic commerce transactions.

Next, the cardholder software decrypts the registration response using the symmetric encryption key that it sent to the CA in the Registration message. It combines the random number returned by the CA with the value that it sent in the registration message to determine the secret value. It then stores the secret value to use with the certificate. Cardholder software vendors will ensure that the certificate and related information is stored in a way to prevent unauthorized access.

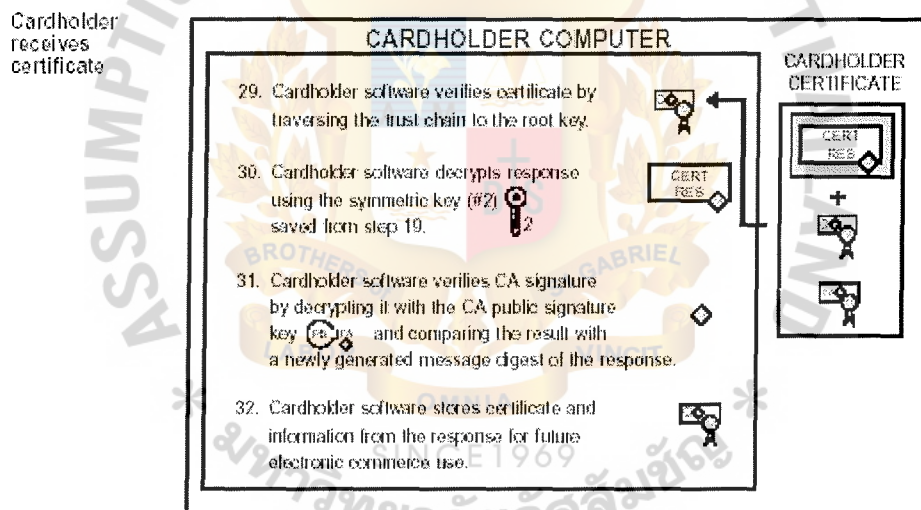


Figure 3.21. Cardholder receives certificate.

(c) Merchant Registration

Figure 7 provides a high-level overview of the merchant registration process, showing its five fundamental steps. The detailed sections that follow describe each step. The icon to the left corresponds

to Figure 7 and serves as a map to the scenario; it is repeated in the more detailed sections with a shaded region that indicates which step is being described.

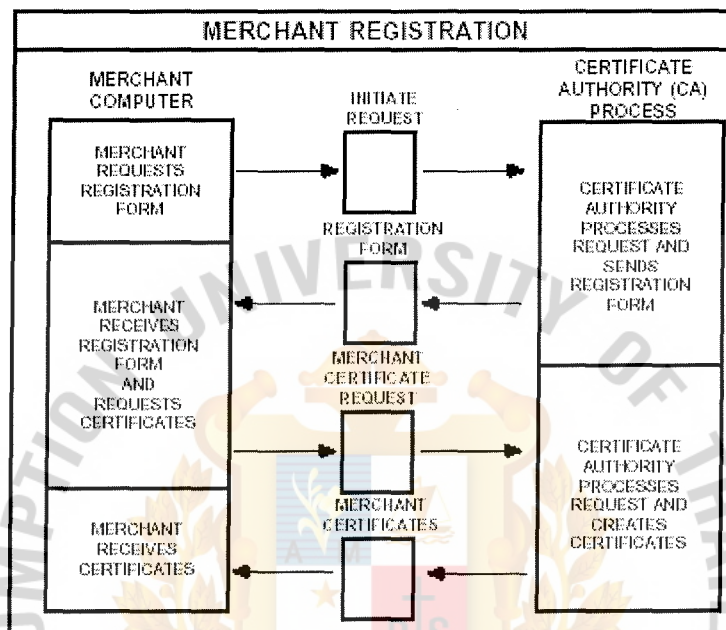


Figure 7: Merchant Registration

Figure 3.22. Merchant Registration.

Merchants must register with a Certificate Authority (CA) before they can receive SET payment instructions from cardholders or process SET transactions through a payment gateway. In order to send SET messages to the CA, the merchant must have a copy of the CA public key-exchange key, which is provided in the CA key-exchange certificate.

The merchant also needs a copy of the registration form from the

merchant's financial institution. The merchant software must identify the Acquirer to the CA. The registration process starts when the merchant software requests a copy of the CA's key exchange certificate and the appropriate registration form.

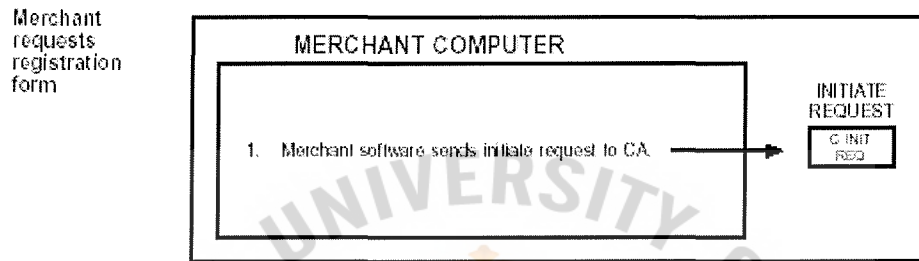


Figure 3.23. Merchant requests registration form.

The CA identifies the merchant's financial institution and selects the appropriate registration form. It returns this registration form along with a copy of its own key-exchange certificate to the merchant.

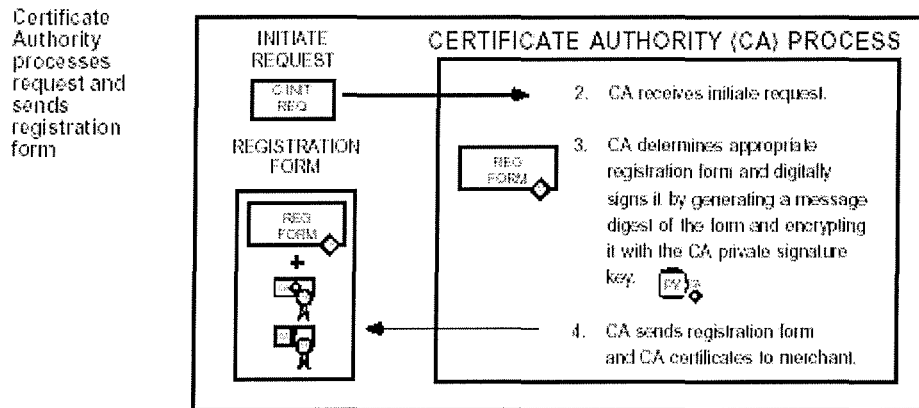


Figure 3.24. Certificate Authority processes request and sends registration form.

The merchant software verifies the CA certificate by traversing the trust chain to the root key, then holds the CA certificate to use later during the registration process. Once the software has a copy of the CA key-exchange certificate, the merchant can register to accept SET payment instructions and process SET transactions. The merchant must have a relationship with an Acquirer that processes SET transactions before a certificate request can be processed. The merchant needs two public/private key pairs for use with SET: key-exchange and signature.

The merchant software generates these key pairs if they do not already exist. To register, the merchant fills out the registration form on the screen with information such as the merchant's name, address, and merchant ID. The merchant software takes this registration information and combines it with the public keys in a registration message. The software digitally signs the registration message. Next the software generates a random symmetric encryption key. It uses this random key

private signature key. If the signature is verified, the message processing continues; otherwise, the message is rejected and an appropriate response message is returned to the merchant. Next the CA must verify the information from the registration request using known merchant information. The process by which the CA and the acquirer exchange information and the steps taken to verify the information in the registration request are outside the scope of this specification. As described in Section 4.1, there are several ways to configure the processing performed by the CA and the Acquirer, such as having the payment card brand provide some or all of the functions on behalf of the Acquirer or having the Acquirer provide all of the functions. If the information in the registration request is verified, the CA creates and digitally signs the merchant certificates.

The validity period of these certificates will be determined by CA policy; often it will correspond to the expiration date of the merchant's contract with the Acquirer, but it may expire sooner. The certificates are then encrypted using a new randomly generated symmetric key, which in turn is encrypted using the merchant public key-exchange key. The response is then transmitted to the merchant.

4.3 Merchant Registration, continued

Certificate Authority processes request and creates certificates

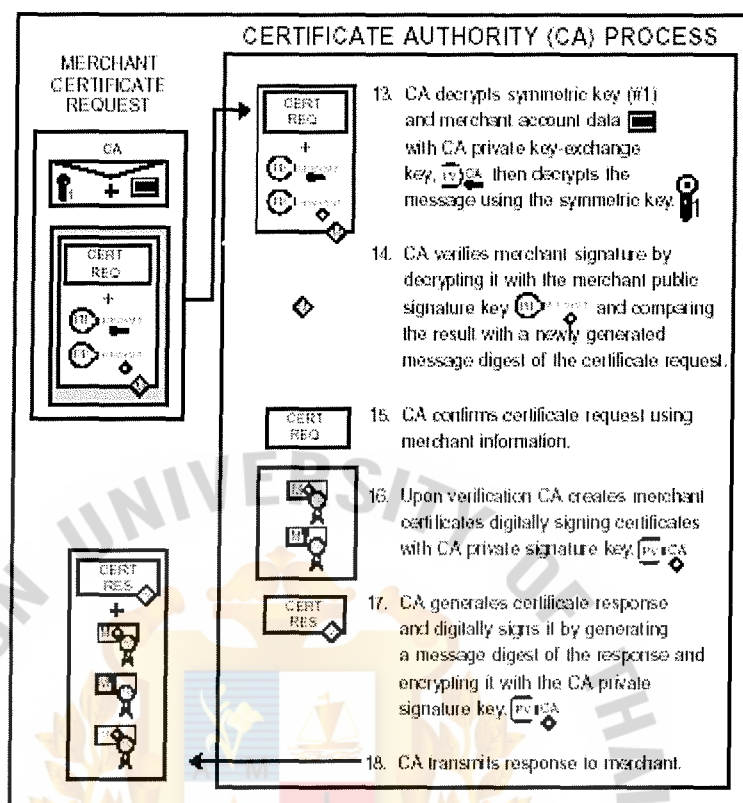


Figure 3.26. Certificate Authority processes request and creates certificates.

When the merchant software receives the response from the CA, it decrypts the digital envelope to obtain the symmetric encryption key. It uses the symmetric key to decrypt the registration response containing the merchant certificates.

After the merchant software verifies the certificates by traversing the trust chain to the root key, it stores the certificates on the merchant's computer for use in future electronic commerce transactions.

Merchant
receives
certificates

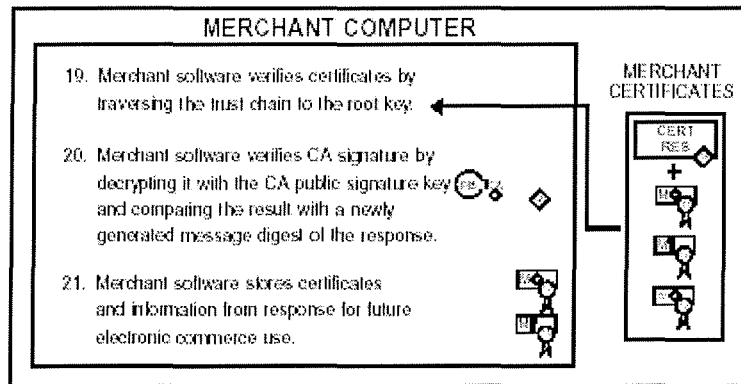


Figure 3.27. Merchant receives certificates.

(b) Purchase Request

Figure 8 provides a high level overview of the purchase request portion of a cardholder's order process, showing its five fundamental steps. The detailed sections that follow describe each step. The icon to the left corresponds to Figure 8 and serves as a map to the scenario; it is repeated in the more detailed sections with a shaded region that indicates which step is being described.

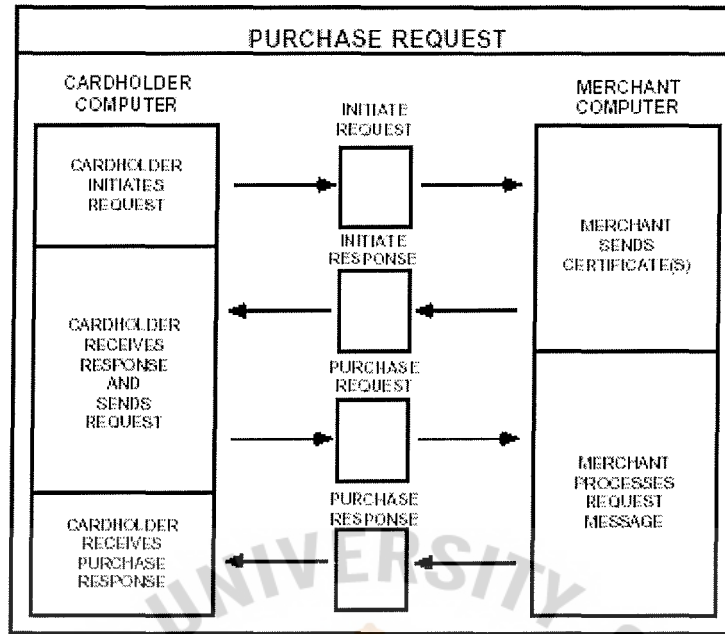


Figure 3.28: Purchase Request

Figure 3.28. Purchase Request.

The SET protocol is invoked after the cardholder has completed browsing, selection, and ordering. Before this flow begins, the cardholder will have been presented with a completed order form and approved its contents and terms, such as the number of installment payments if the merchant is billing for the transaction in installments. In addition, the cardholder will have selected a payment card as the means of payment.

In order to send SET messages to a merchant, the cardholder must have a copy of the Payment Gateway's key-exchange keys. The SET order process is started when the cardholder software requests a copy of the gateway's certificate. The message from the cardholder indicates which payment card brand will be used for the transaction.

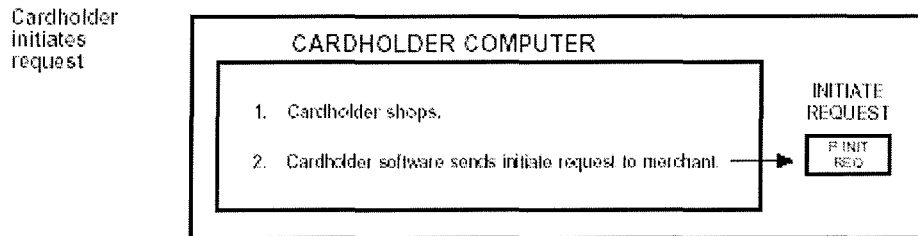


Figure 3.29. Cardholder initiates request.

When the merchant receives the request, it assigns a unique transaction identifier to the message. It then transmits the merchant and gateway certificates that correspond to the payment card brand indicated by the cardholder, along with the transaction identifier to the cardholder.

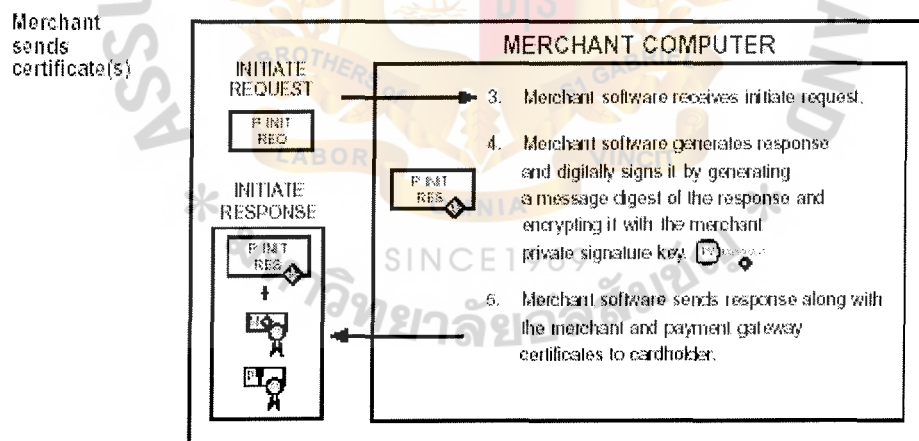


Figure 3.30. Merchant send certificate (s).

The cardholder software verifies the merchant and gateway certificates by traversing the trust chain to the root key, then holds these certificates to use later

during the ordering process. The cardholder software creates the Order Information (OI) and Payment Instructions (PI). The software places the transaction identifier assigned by the merchant in the OI and the PI; this identifier will be used by the Payment Gateway to link the OI and the PI together when the merchant requests authorization.

Note: The OI does not contain the order data such as the description of goods (the items and quantities) or the terms of the order (such as number of installment payments). This information is exchanged between the cardholder and merchant software during the shopping phase before the first SET message. The cardholder software generates a dual signature for the OI and the PI by computing the message digests of both, concatenating the two digests, computing the message digest of the result and encrypting that using the cardholder private signature key. The message digests of the OI and the PI are sent along with the dual signature. Next the software generates a random symmetric encryption key and uses it to encrypt the dual signed PI. The software then encrypts the cardholder account number as well as the random symmetric key used to encrypt the PI into a digital envelope using the Payment Gateway's key-exchange key. Finally, the software transmits a message consisting of the OI and the PI to the merchant.

Cardholder
receives
response and
sends request

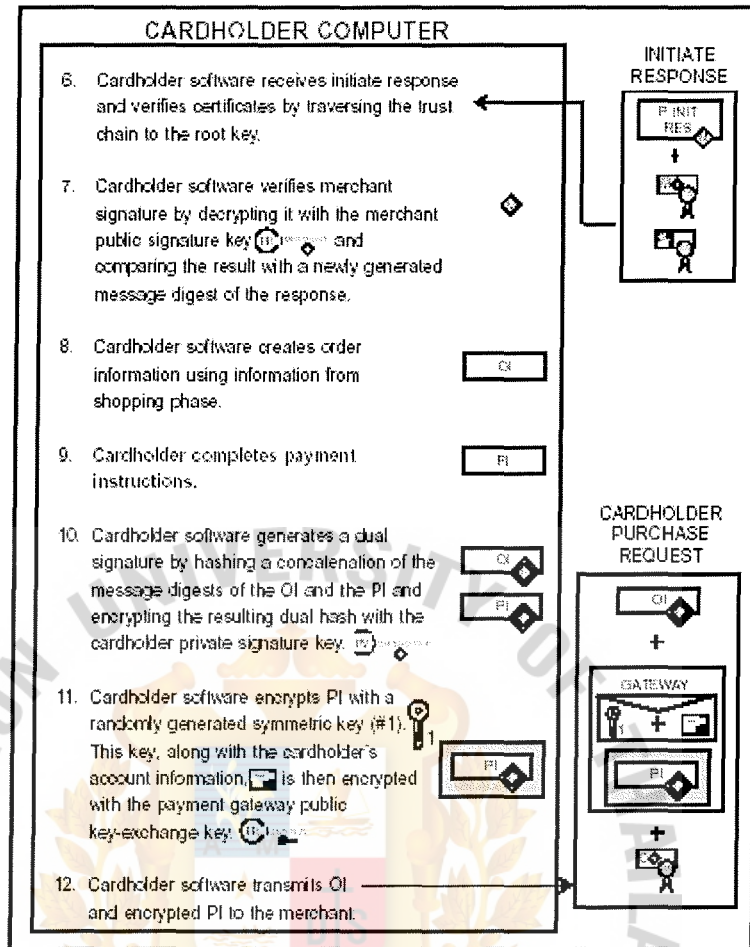


Figure 3.31. Cardholder receives response and send request.

When the merchant software receives the order, it verifies the cardholder signature certificate by traversing the trust chain to the root key. Next it uses the cardholder public signature key and the message digest of the PI (included with the OI) to check the digital signature to ensure that the order has not been tampered with in transit and that it was signed using the cardholder private signature key. The merchant software then processes the order including the payment authorization described in Section 4.5. Note: It is not necessary for the merchant to perform the authorization phase prior to sending a response to the

cardholder. The cardholder can determine later if the authorization has been performed by sending an order inquiry message. (The order inquiry flow is described in Book 2: Programmer's Guide.) After the OI has been processed, the merchant software generates and digitally signs a purchase response message, which includes the merchant signature certificate and indicates that the cardholder's order has been received by the merchant. The response is then transmitted to the cardholder. If the authorization response (see Section 4.5) indicates that the transaction was approved, the merchant will ship the goods or perform the services indicated in the order.

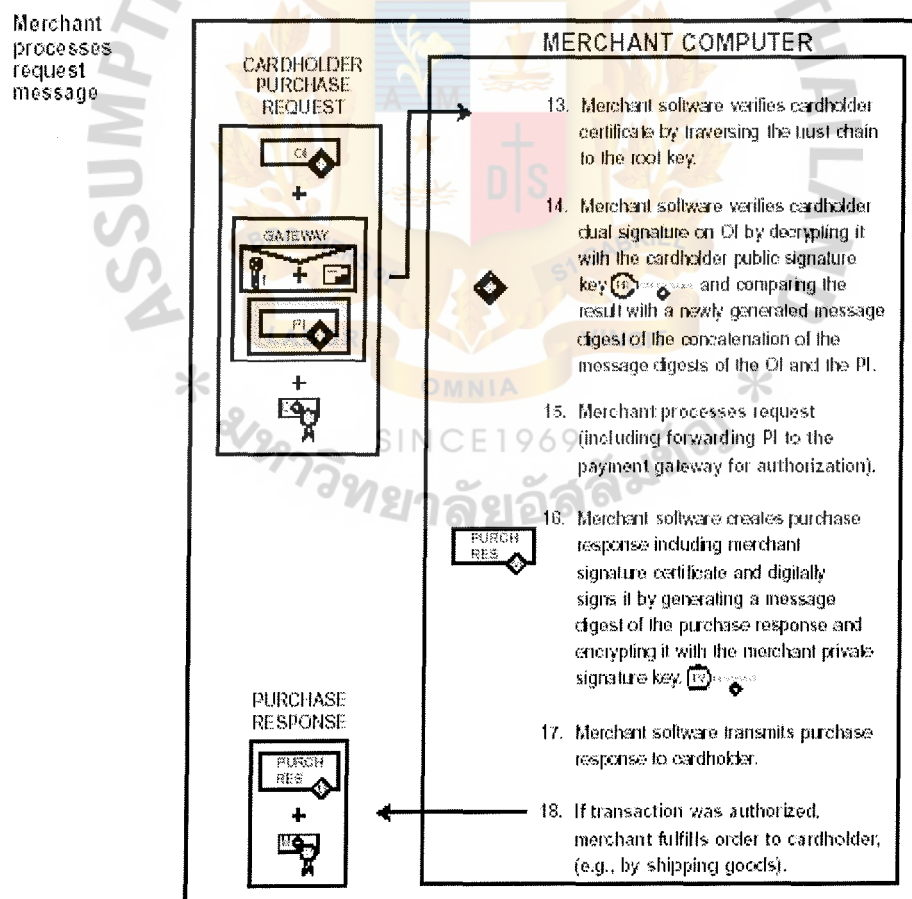


Figure 3.32. Merchant processes request message.

When the cardholder software receives the purchase response message from the merchant, it verifies the merchant signature certificate by traversing the trust chain to the root key. It uses the merchant public signature key to check the merchant's digital signature. Finally, it takes some action based on the contents of the response message, such as displaying a message to the cardholder or updating a database with the status of the order. The cardholder can determine the status of the order (such as whether it has been authorized or submitted for payment) by sending an order inquiry message.

This message is described in Book 2: Programmer's Guide.

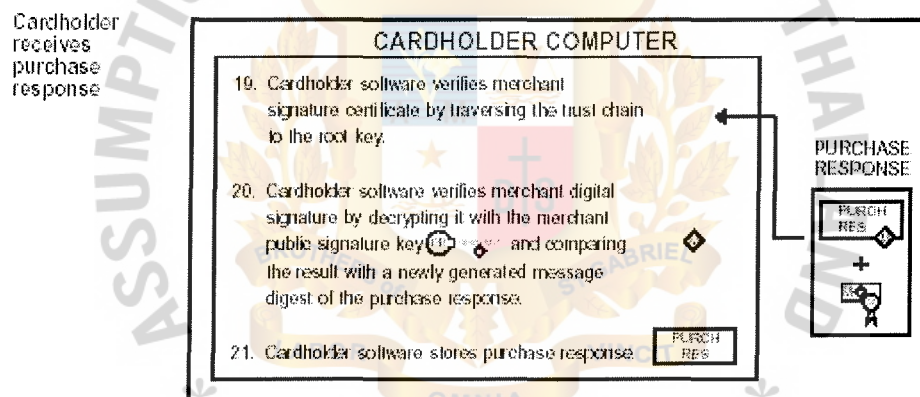


Figure 3.33. Cardholder receives purchase response.

(c) Payment Authorization

Figure 9 provides a high level overview of a merchant's payment Authorization process, showing its three fundamental steps. The detailed sections that follow describe each step. The icon to the left corresponds to Figure 9 and serves as a map to the scenario; it is repeated in the more detailed sections with a shaded region that indicates which step is being described.

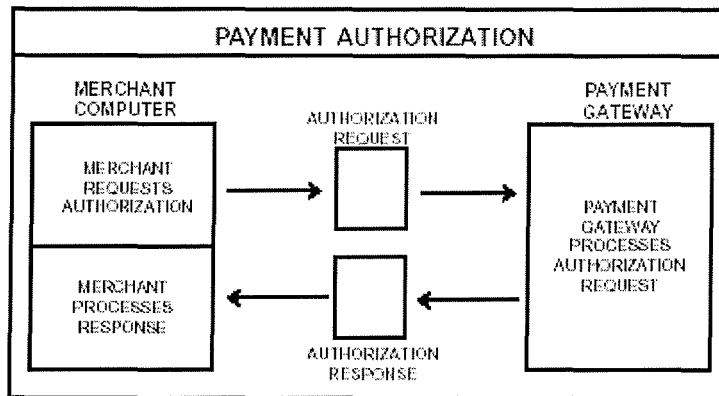


Figure 9: Payment Authorization

Figure 3.34. Payment Authorization.

During the processing of an order from a cardholder (see Section 4.4), the Merchant will authorize the transaction. The merchant software generates and digitally signs an authorization request, which includes the amount to be authorized, the transaction identifier from the OI, and other information about the transaction.

The request is then encrypted using a new randomly generated symmetric key, which in turn is encrypted using the public key-exchange key of the Payment Gateway. (This is the same key the cardholder used to encrypt the digital envelope of the payment instructions.) The authorization request and the cardholder payment instructions are then transmitted to the Payment Gateway.

Note: The SET protocol also includes a sales transaction that allows a merchant to authorize a transaction and request payment in a single message. While the sales message includes an additional block of data on the request from the merchant, it otherwise parallels the message flow described in this section. Details about the processing of a sales transaction are provided in Book 2: Programmer's Guide.

Merchant
requests
authorization

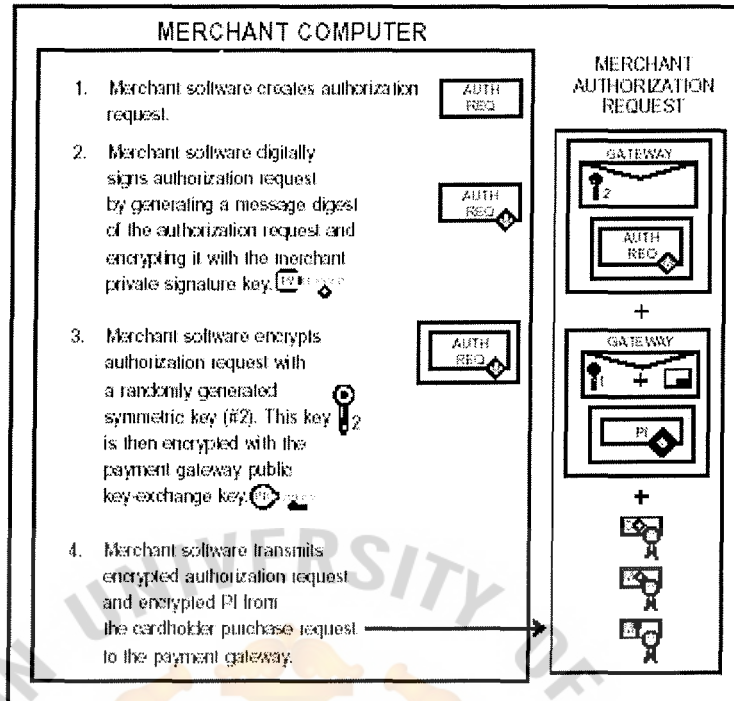


Figure 3.35. Merchant requests authorization.

When the Payment Gateway receives the authorization request, it decrypts the Digital envelope of the authorization request to obtain the symmetric encryption key.

It uses the symmetric key to decrypt the request. It then verifies the merchant signature certificate by traversing the trust chain to the root key; it also verifies that the certificate has not expired. It uses the merchant public signature key to ensure the request was signed using the merchant private signature key. Next the Payment Gateway decrypts the digital envelope of the Payment Instructions to obtain the symmetric encryption key and the account information. It uses the symmetric key to decrypt the PI. It then verifies the cardholder signature certificate by traversing the trust chain to the root; it also verifies that the

certificate has not expired. Next it uses the cardholder public signature key and the message digest of the OI (included in the PI) to check the digital signature to ensure that the PI has not been tampered with in transit and that it was signed using the cardholder private signature key. Next, the Payment Gateway verifies that the transaction identifier received from the merchant matches the one in the cardholder Payment Instructions. The Payment Gateway then formats and sends an authorization request to the Issuer via a payment system. Upon receiving an authorization response from the Issuer, the Payment Gateway generates and digitally signs an authorization response message, which includes the Issuer's response and a copy of the Payment Gateway signature certificate. The response also includes an optional capture token with information the Payment Gateway will need to process a capture request (see Section 4.6). The capture token is only included if required by the Acquirer. The response is then encrypted using a new randomly generated symmetric key, which in turn is encrypted using the merchant public key-exchange key. The response is then transmitted to the merchant.

Payment Gateway processes authorization request

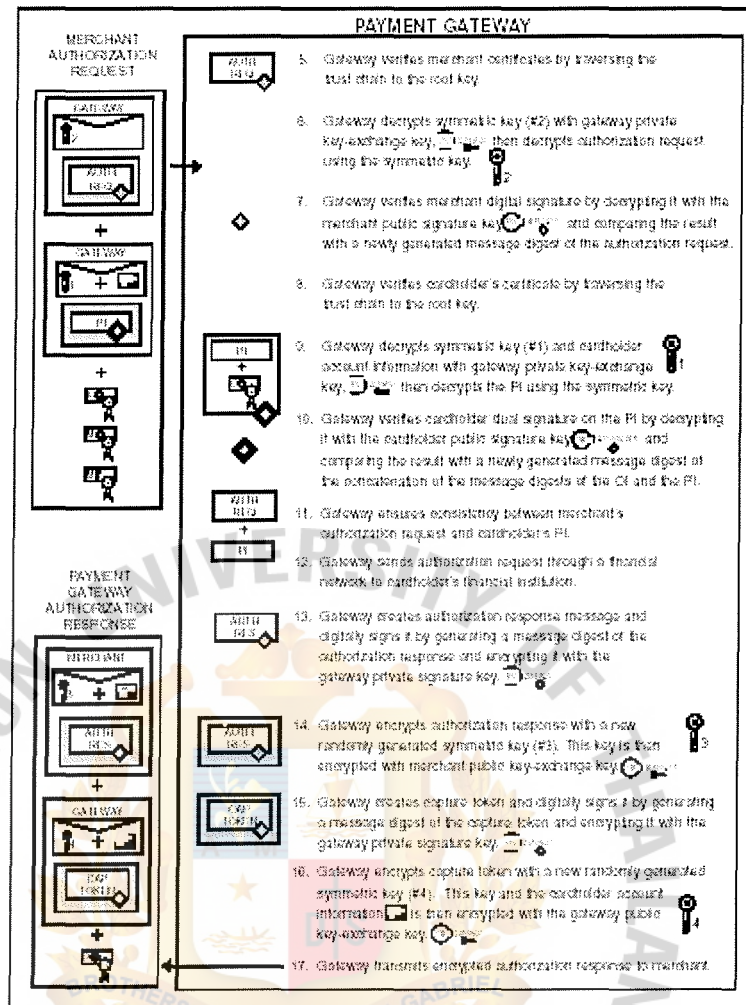


Figure 3.36. Payment Gateway processes authorization request.

When the merchant software receives the authorization response message From the Payment Gateway, it decrypts the digital envelope to obtain the Symmetric encryption key. It uses the symmetric key to decrypt the response message. It then verifies the Payment Gateway signature certificate by traversing the trust chain to the root key. It uses the Payment Gateway public signature key to check the Payment Gateway digital signature. The merchant software stores the authorization response and the capture token to be used when requesting payment

through a capture request (see Section 4.6). The merchant then completes processing of the cardholder's order (see Section 4.4) by shipping the goods or performing the services indicated in the order.

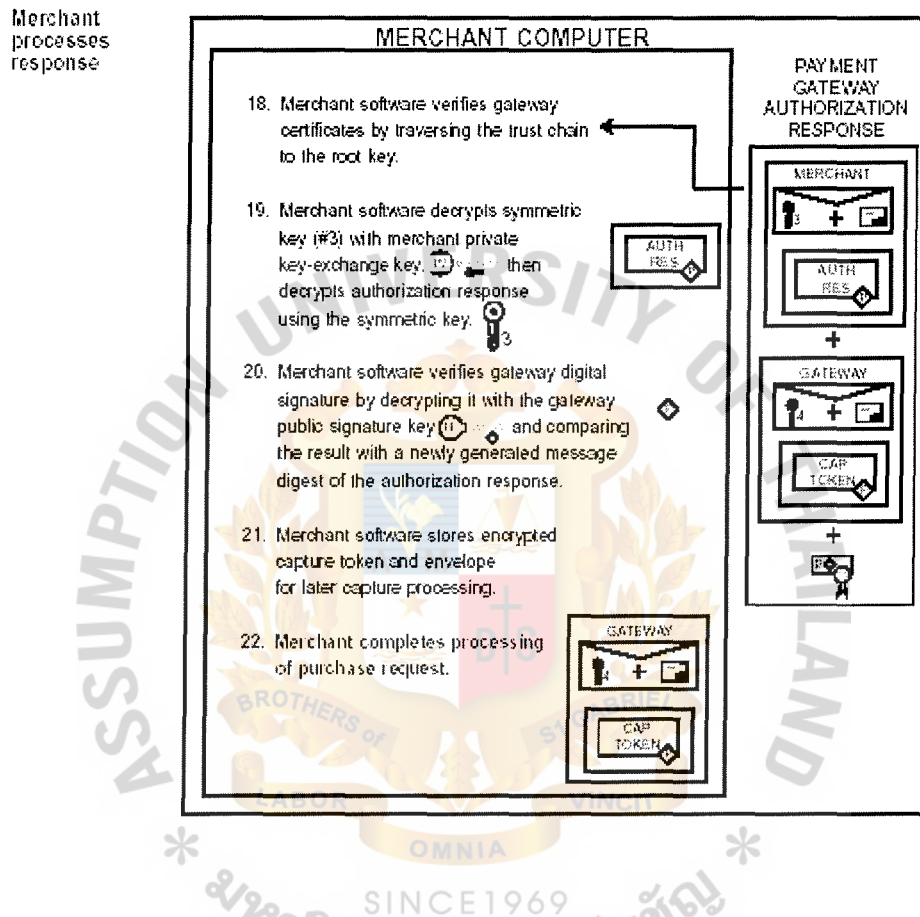


Figure 3.37. Merchant processes response.

(d) Payment Capture

Figure 10 provides a high level overview of a merchant's payment capture process, showing its three fundamental steps. The detailed sections that follow describe each step. The icon to the left corresponds to Figure 10 and serves as a map to the scenario; it is repeated in the more detailed sections with a shaded region that indicates which step is being described.

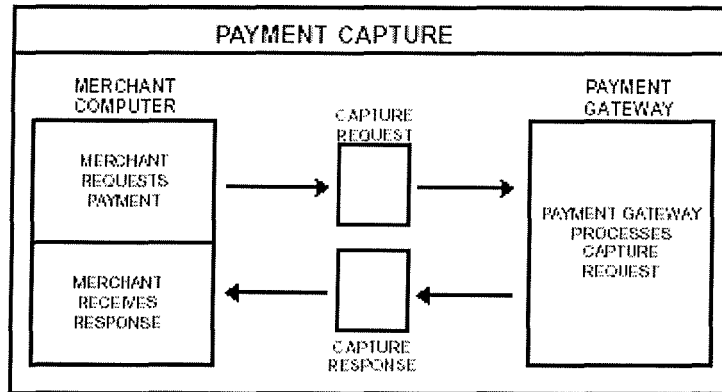


Figure 10: Payment Capture

Figure 3.38. Payment Capture.

After completing the processing of an order from a cardholder (see Section 4.4), The merchant will request payment. There will often be a significant time lapse between the message requesting authorization and the message requesting payment. The merchant software generates and digitally signs a capture request, which includes The final amount of the transaction, the transaction identifier from the OI, and other information about the transaction. The request is then encrypted using a new randomly generated symmetric key, which in turn is encrypted using the public key-exchange key of the Payment Gateway. The capture request and optionally the capture token if one was included in the authorization response (see Section 4.5) are then transmitted to the Payment Gateway.

Note: While the flow described here contains only a single capture request, the merchant software is permitted to batch multiple requests into a single message.

Merchant
requests
payment

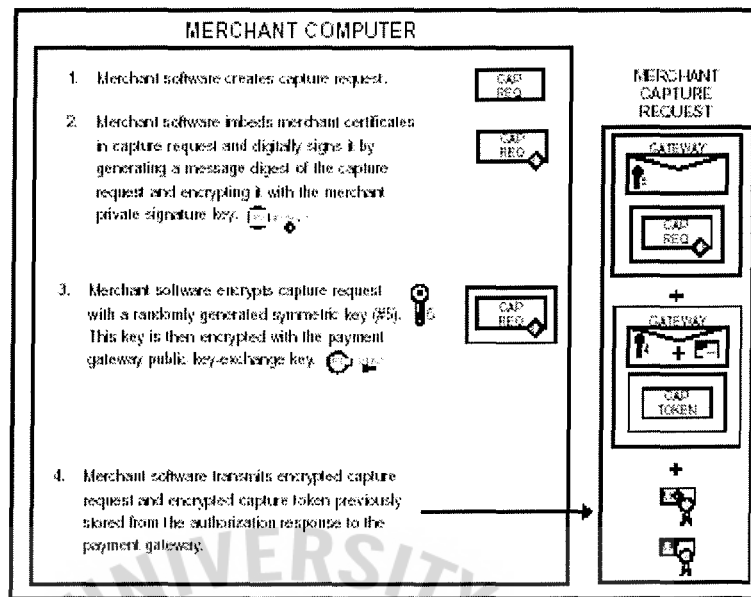


Figure 3.39. Merchant requests payment.

When the Payment Gateway receives the capture request, it decrypts the Digital envelope of the capture request to obtain the symmetric encryption key. It Uses the symmetric key to decrypt the request. It then uses the merchant public Signature key to ensure the request was signed using the merchant private signature key. The Payment Gateway decrypts the capture token (if present) and then uses the information from the capture request and the capture token to format a clearing request, which it sends to the Issuer via a payment card payment system. The Payment Gateway then generates and digitally signs a capture response message, which includes a copy of the Payment Gateway signature certificate. The response is then encrypted using a new randomly generated symmetric key, which in turn is encrypted using the merchant public key-exchange key. The response is then transmitted to the merchant.

Merchant
receives
response

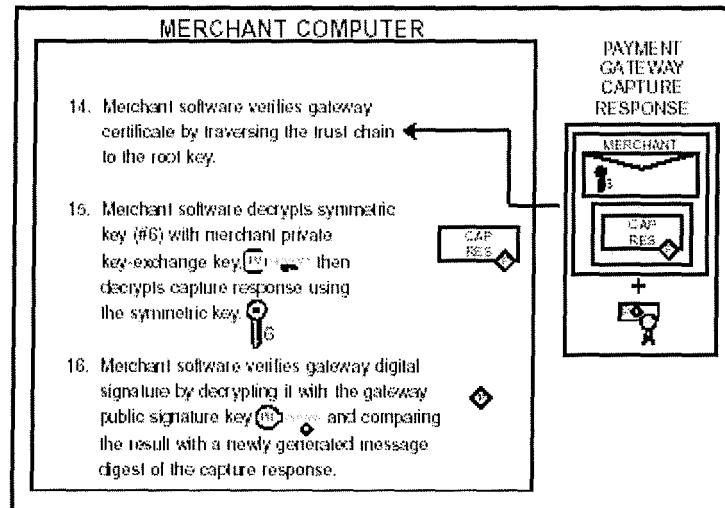


Figure 3.40. Merchant receives response.



IV. USING OF SECURITY PROTOCOLS FOR APPLICATION PAYMENT

4.1 Sending Password to user

(a) Meaning

Protected/private a set of unique bit patterns that are used to represent letters of an alphabet, decimal digits, punctuation marks, and other special signs and symbols used in grammar, business, and science, such as those displayed on conventional typewriter keyboards used to verify the identity of a user, user device, or other entity, or the integrity of data stored, transmitted, or otherwise exposed to unauthorized modification in an information system (IS), or establish the validity of a transmission, an identity or to authorize access to data.

(b) Using Security Protocol for Transferring

(i) TLS Protocol

TLS uses symmetric key encryption techniques, such as DES and RC4, to ensure privacy. In symmetric key encryption the sender and the receiver share a secret key which is used to encrypt or decrypt messages. However, this secret key must somehow be exchanged between the communicating parties before any secure communication can take place.

During the TLS handshake process the client chooses a secret, which it then sends to the server. Public key cryptography is used to protect this exchange. When sent password in this method must have secreted key for exchange and receive password and must show real owner for hand checking.

(ii) SSL Protocol

The SSL Handshake Protocol has two major phases. The first phase is Used to establish private communications. The second phase is used for client authentication. The first phase is the initial connection phase where

both parties communicate their "hello" messages.

The channel is private. Encryption is used for all messages after a simple handshake is used to define a secret key. When sent password in this method must have private key for receive password and must show real owner for hand checking.

(iii) SET Protocol

Encryption The encryption process in Figure 3 consists of the following steps:

Step Description

- (a) Alice runs the property description through a one-way algorithm to produce a unique value known as the message digest. This is a kind of digital fingerprint of the property description and will be used later to test the integrity of the message.
- (b) She then encrypts the message digest with her private signature key to produce the digital signature.
- (c) Next, she generates a random symmetric key and uses it to encrypt the property description, her signature and a copy of her certificate, which contains her public signature key. To decrypt the property description, Bob will require a secure copy of this random symmetric key.
- (d) Bob's certificate, which Alice must have obtained prior to initiating Secure communication with him, contains a copy of his public key-exchange key. To ensure secure transmission of the symmetric key, Alice encrypts it using Bob's public key-exchange key. The encrypted key, referred to as the digital

envelope, will be sent to Bob along with the encrypted message itself.

- (e) Alice sends a message to Bob consisting of the following: the Symmetrically encrypted property description, signature and certificate, as well as the asymmetrically encrypted symmetric key (the digital envelope).
- (f) Bob receives the message from Alice and decrypts the digital envelope with his private key-exchange key to retrieve the symmetric key.
- (g) He uses the symmetric key to decrypt the property description, Alice's signature, and her certificate.
- (h) He decrypts Alice's digital signature with her public signature key, which he acquires from her certificate. This recovers the original message digest of the property description.
- (i) He runs the property description through the same one-way algorithm used by Alice and produces a new message digests of the decrypted property description.
- (j) Finally, he compares his message digest to the one obtained from Alice's digital signature. If they are exactly the same, he confirms that the message content has not been altered during transmission and that it was signed using Alice's private signature key.

If they are not the same, then the message either originated

Somewhere else or was altered after it was signed. In that case, Bob takes some appropriate action such as notifying Alice or discarding the message.

(c) Analysis

When sent password by SET protocol has a lot of process for send and receives per each of time because SET protocol has test the integrity of the message. In the first step and then have digital signature for inspect the message again. And then have the symmetric for authenticate the message. But if processing use SSL and TLS protocols for sent password processing will faster than sending by SET protocol Because SSL and TLS uses symmetric key encryption techniques, such as DES and RC4, to ensure privacy. In symmetric key encryption the sender and the receiver share a secret key, Which is used to encrypt or decrypt messages. However, this secret key must Somehow be exchanged between the communicating parties before any secure communication can take place. During the TLS handshake process the client chooses a secret, which it then sends to the server. Public key cryptography is used to protect this exchange.

So who want to send password to protect data or information of them. They want fast processing for each process. So password sending use SSL and TLS protocols better than use SET protocol.

4.2 Sending E-mail

(a) E-mail Meaning

Messages automatically passed from one computer user to another, often through computer networks and/or via modems over telephone lines. A message, especially one following the common RFC 822 standard, begins with several lines of headers, followed by a blank line, and the body of the message. An increasing number of e-mail systems support the MIME standard which allows the message body to contain "attachments" of different kinds rather than just one block of plain ASCII text. It is conventional for the body to end with a signature.

(b) Using Security Protocol for Transferring

(i) TLS Protocol

TLS protocol (Transportation Layer Security) as RFC2246. It offers Both encryption of the communication (stopping eavesdropping) and strong authentication (making sure that both parties of a communication are correctly identified and that the communication cannot be altered).

TLS does not realize the TLS protocol itself; it rather uses the Open SSL package for this task. At the Open SSL WWW-site you can also find links to in-depth documentation of the protocol and its features, so that it is not necessary to included them here. (And, of course, there is no use of re-writing what other people already wrote down, it just introduces additional errors.)

(ii) SSL Protocol

SSL stands for Secure Sockets Layer encryption. SSL is a protocol that Utilizes public/private key encryption to negotiate a connection between the web server and your browser, or in this case, the mail server and your mail

client. SSL connections provide security because the communication between the mail server and your mail software is encrypted. This does not encrypt your e-mail, only the transmission that delivers the mail from the mail server to your computer. This lessens the chance that your password or mail could be read while in transit between the mail server and your computer.

(iii) SET Protocol

SET uses a system of locks and keys along with certified account IDs for both sender and receiver. Then, through a unique process of "encrypting" or scrambling the information exchanged between the sender and the receiver, SET ensures send and receives process that is convenient, private and most of all secure. Specifically, SET:

- (a) Establishes industry standards to keep your E-mail confidential.
- (b) Increases integrity for all transmitted data through encryption.
- (c) Provides authentication that a sender is a legitimate user of name account.
- (d) Provides authentication that a receiver can accept name transactions through its relationship with an acquiring institution.
- (e) Allows the use of the best security practices and system design techniques to protect all legitimate parties in an electronic mail transaction.

(c) Analysis

E-MAIL or Messages automatically passed from one computer user to another, for communication they want comfortable for sending per each time. When sender send e-mail to receiver may have security protocol for protect

Message form disturb or bandit data between the way. SSL stands for Secure Sockets Layer encryption. SSL is a protocol that utilizes public/private key Encryption to negotiate a connection between the web server and your browser, or in this case, the mail server and your mail client. SSL connections provide security because the communication between the mail server and your mail software is encrypted. TLS protocol (Transportation Layer Security) as RFC2246. It offers both encryption of the communication (stopping eavesdropping) and strong authentication (making sure that both parties of a communication are correctly identified and that the communication cannot be altered). SET: Establishes industry standards to keep your E-mail confidential.

Increases integrity for all transmitted data through encryption. Provides Authentication that a sender is a legitimate user of name account. Provides authentication that a receiver can accept name transactions through its relationship with an acquiring institution. Allows the use of the best security practices and system design techniques to protect all legitimate parties in an electronic mail. transaction So SET protocol have more protection more than SSL and TLS but sender and receiver want comfortable for reading or sending for each time. For SSL and TLS enough for sending and receiving E-mail.

4.3 Sending Digital Product Online

After you've finished creating the HTML files on your computer, you'll need to Transfer them onto your web server.

(a) What Is FTP?

FTP stands for *File Transfer Protocol*. It's a method of transferring files Over the Internet. This is usually done with an ftp program, such as WS-FTP. There are a number of ftp programs. You can use whatever you like. WS-FTP is perhaps the most popular, and most hosting services can help you with installation and configuration.

How to Use It

- (i) To use FTP, log in to the Internet and establish an online connection.
- (ii) Start the FTP program.
- (iii) Select your Profile. If you have only one, then it will appear. (If there are other profiles that you don't use, you can delete them.)
- (iv) Click OK.
- (v) FTP will connect to your web host. You will see the following dialog box.
- (vi) On the left side are the files on your computer (Local System.)
- (vii) On the right side are the files on your web hosting space.
- (viii) The first time you log in, you'll need to navigate to the web page folders. On the right side, there is a folder named http docs. This is the folder that holds your website files. Double-click it to enter it.
- (ix) On the left side, navigate to your website's folder on your computer.
- (x) When you're ready, you'll have your files on your computer on the left side, and the web host site on the right side.

- (xi) So you'll be in these folders in the future, you can save the current folders. Select Options, click the Session tab, and click *Save Current Folders as Connection Folders*. From now on, FTP will open in your folders, ready to go.
- (xii) At the bottom of the windows, there are two boxes marked Binary and Auto. Select both of these. This lets FTP use the right setting to transfer your files.
- (xiii) To transfer HTML documents and images, select the file on one side and click the arrows in the middle to transfer the file to the other side.
- (ivx) You can transfer files from your computer to your website. You can also transfer files from the website to your computer.
- (vx) You can also transfer a whole folder in one step. Click the folder to select it And then click the transfer arrow.
- (vxi) You can transfer multiple files simultaneously. Select all the files that you want to transfer and then click the transfer arrow.
- (vxii) And finally, you can open several FTP programs and transfer many files at the same time. In one, you can set it to transfer images, and in the other, you can transfer HTML files.
- (vxiii) To see if it worked, open a browser and visit your website. The page should open in your browser.

(b) Using protocol for digital product

(i) TLS and SSL Protocol

Transport Layer Security (TLS) is a protocol that ensures privacy between communicating applications (The term *application* is a shorter form of application program. An application program is a program designed to

perform a specific function directly for the user or, in some cases, for another application program. Examples of applications include word processors, database programs, Web browsers, development tools, drawing, paint, image editing programs, and communication programs. Applications use the services of the computer's operating system and other supporting applications. The formal requests and means of communicating with other programs that an application program uses is called the application program interface (API) and their users on the Internet. When a server and client communicate, TLS ensures that no third party may eavesdrop or tamper with any message. TLS is the successor to the Secure Sockets Layer (SSL).

TLS is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. The TLS Record Protocol provides connection security with some encryption method such as the Data Encryption Standard (DES). The TLS Record Protocol can also be used without encryption. The TLS Handshake Protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before data is exchanged.

(ii) SET Protocol

Cryptography has been used for centuries to protect sensitive Information as it is transmitted from one location to another. In a cryptographic system, a message is encrypted using a key. The resulting cipher text is then transmitted to the recipient where it is decrypted using a key to produce the original message.

There are two primary encryption methods in use today: secret-key cryptography and public-key cryptography. SET uses both methods in its

encryption process.

(c) Analysis

Digital product mean data or software can be downloading it by internet. After you've finished creating the HTML files on your computer, you'll need To transfer them onto your web server by ftp (FTP stands for *File Transfer Protocol*. It's a method of transferring files over the Internet. This is usually Done with an ftp program, such as WS-FTP. There are a number of ftp programs. You can use whatever you like. WS-FTP is perhaps the most popular and most hosting services can help you with installation and configuration.). So when who want to sell digital product by E-commerce must have security for protect data or software from thief or hacker. Such as seller should have chosen security protocol for protect.

In this subject have 3 protocols for choose TLS and SSL protocol have main property is a protocol that ensures privacy between communicating applications (The term *application* is a shorter form of application program. An application program is a program designed to perform a specific function directly for the user or, in some cases, for another application program. Examples of applications include word processors, database programs, Web browsers, development tools, drawing, paint, image editing programs, and communication programs. Applications use the services of the computer's operating system and other supporting applications. The formal requests and means of communicating with other programs that an application program uses is called the application program interface (API)) and their users on the Internet. When a server and client communicate, TLS ensures that no third party may eavesdrop or tamper with any message. And SET protocol is Cryptography has been used for centuries to protect

sensitive information as it is transmitted from one location to another.

In a cryptographic system, a message is encrypted using a key. The resulting cipher text is then transmitted to the recipient where it is decrypted using a key to produce the original message. There are two primary encryption methods in use today: secret-key cryptography and public-key cryptography. SET uses both methods in its encryption process. From content of 2 major methods will acknowledge about detail of security protocol per each type. Sending and receiving digital product more important for seller and buyer. So security protocol for use in this process must high security. Such as SET protocol more appropriate than SSL and TLS protocol.



4.4 Payment Online

(a) Meaning

On-line settlements have made possible instantaneous payments (especially instant payments at “virtual shops”) so that commercial transactions over the Internet have become a reality. Settlement through a membership system. Membership system settlements are offered by ISP's (Internet Service Providers) as one element of the service menu they provide to their network members. The settlements are performed using a) prepaid deposits or b) credit card numbers or bank account information which has been provided to the ISP ahead of time. Small charges can be paid little by little from the prepaid deposits, or the total amount which has been used each month can be invoiced to the credit card or other account as a lump sum, along with the ISP's service fees. Through such methods, transactions involving comparatively small amounts can be conducted efficiently. However, these systems have many restrictions. For example, because these are membership systems, one must become a member ahead of time.

Also, regarding security, since each service provider operates according to its own standards, the trustworthiness of the service provider company becomes very important.

(b) Using Security Protocol for Transferring

(i) SSL and TLS Protocol

With SSL settlements, safe communication channels (resistant to “wiretapping”) are assured by SSL encryption of transmissions between the shopping user and the virtual shop. In this method, credit card numbers, etc. flow through these secure channels and credit card settlements are done in the “backyard”

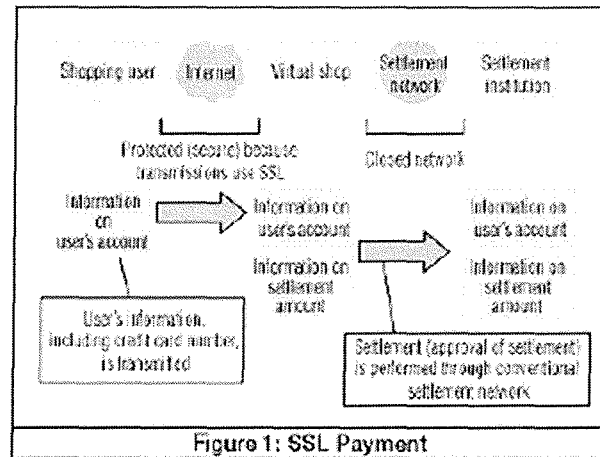


Figure 4.1. SSL Payment.

When the membership system is adopted at virtual shops, once one's credit card number has been registered, from the second time onward, product purchases can be done using only the member number (+ password.) If a shopper simply has a credit card, he/she using their browser (which in General are compatible with SSL communication) can make instantaneous purchases of products even at virtual shops they are visiting for the first time.

For this reason, in the US this is the most widely used settlement method. However, there are many problems of false (impersonated) virtual shops because, depending on the individual virtual shop, the standard for security may be quite different. (In general the majority of virtual shops do not have an adequate security level.) This is a particular problem for shopping users because they have no method for judging whether the security level is adequate or not, or whether the virtual shop they are dealing with has formally contracted with the credit card company to become a "member shop." There is a strong possibility that area could become a

hotbed of crime.

(ii) SET Protocol

SET settlement is the safest net settlement system, utilizing the four mechanisms of authentication agency, wallet, merchant POS, and settlement gateway.

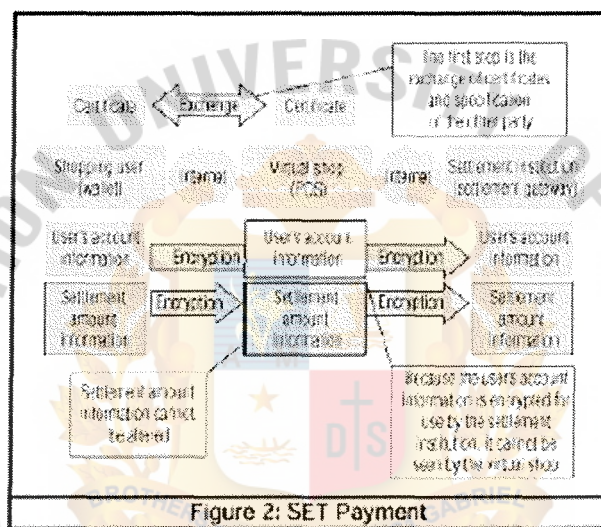


Figure 4.2. SET Payment.

It is designed to resist such attacks as impersonation of the shopper, impersonation of the virtual shop, falsification of settlement information, and hacking of the virtual shops, which are the element with the lowest level of security in this settlement channel.

At present, a scheme for standardization of settlements which supports both SET Debit (settlements linked to bank accounts) and SET Credit (settlement by means of credit cards) is being promoted. The group pushing

it is the Japan Conference on Promotion of the Internet Economy, in which over 300 companies participate, including the Postal Savings Bureau of the Ministry of Posts and Telecommunications, banks, and credit card companies.

(c) Analysis

Money or cash more important for every body. So when who want transfer money must have security for protect it. In this case mention 3 type of security are SSL, TLS, SET protocol. Each type has differentiated in detail of security.

For SSL and TLS When the membership system is adopted at virtual shops, once one's credit card number has been registered, from the second time onward, product purchases can be done using only the member number (+ password.) If a shopper simply has a credit card, he/she using their browser (which in general are compatible with SSL communication) can make instantaneous purchases of products even at virtual shops they are visiting for the first time. For this reason, in the US this is the most widely used settlement method. However, there are many problems of false (impersonated) virtual shops because, depending on the individual virtual shop, the standard for security may be quite different.

For SET protocol It is designed to resist such attacks as impersonation of the shopper, impersonation of the virtual shop, falsification of settlement information, and hacking of the virtual shops, which are the element with the lowest level of security in this settlement channel.

At present, a scheme for standardization of settlements which supports both SET Debit (settlements linked to bank accounts) and SET Credit (settlement by means of credit cards) is being promoted. So SET more security than SSL or

TLS because SET specification for shopping payment.



V. CONCLUSION

Internet Protocol Security (IPSec) provides application-transparent encryption services for IP network traffic as well as other network access protections for the operating system.

This guide focuses on the fastest way to use IPSec transport mode to secure application traffic between a client and a server. It demonstrates how to enable security using IPSec default policies between based systems that belong to a domain. Once the two computers have joined the domain, you should complete the first part of the walkthrough, which demonstrates default policies in 30 minutes or less. Notes are included on how to enable non-IPSec clients to communicate to the server. Steps are provided on how to use certificates, and how to build your own custom policy for further interoperability testing, or to demonstrate IPSec when a domain is not available.

Using Internet Protocol Security (IPSec), you can provide data privacy, integrity, authenticity, and anti-replay protection for network traffic in the following scenarios:

Provide for end-to-end security from client-to-server, server-to-server, and client-to-client using IPSec transport mode.

This report concerned about method of using security protocol for another application. The first of all mention sending password from one place to another place have property to concern is security between point to point in almost of thing may respect about security of password. So should have security protocol for transfer per each time and protocol for security in this case have 3 protocol are SSL, TLS, SET protocol for protocol have another property quite differentiate for SET protocol have a lot of security step per each time but SSL and TLS have security less than SET protocol. So SSL and TLS protocol are matched for sending password. Password same as sending E-mail because sender and receiver want comfortable for sending each time.

So it will choose SSL and TLS for transfer data. But sending digital product and payment online want high security more than E-mail and Password because product and money high value more than them and product and money can change to money easier than password and E-mail. So secure for they must be high security. Such as SET protocol will matched for digital product and payment online.



BIBLIOGRAPHY

1. "Transport Layer Security" Available at
http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci.557332,00.html
2. "TLS (Transport Layer Security)" Available at
<http://www.nwfusion.com/links/Encyclopedia/+789.html>
3. "The SSL Protocol Version 3.0" Available at
<http://wp.netscape.com/eng/ssl3/ssl-toc.html>
4. "The SSL Protocol" Available at
<http://wp.netscape.com/eng/security/ssl-2.html>
5. "The SSL Protocol" Available at
<http://byerley.es.waikato.ac.nz/~longm/articles/sslnode-3.html>
6. "Secure Electronic Transaction CSET protocol" Available at
<http://islab.oregonstate.edu/koc/ece478/project/2003RP/li-wang.pdf>
7. "How to Provide Security Transaction For Your Online Customer" Available at
<http://www.profiljump.com/articles/0725-secure-transaction.html>
8. "Description of The Password Fields" Available at
<http://www.instantweb.com/help/signup/password.html>
9. "Programming Internet Email" Available at
<http://www.oreilly.com/catalog/progintemail/close.html>
10. "File Transfer Protocol" Available at
<http://www.imeprint.com/ftp/description/asp>
11. "Online Payment" Available at
http://www.guideonce.be/dotNet Srv/Services_show.aspx?ServID=227

