Performance Improvements on the Network Security Protocols

Tarek S. Sobh[1], Ashraf Elgohary[1], and M. Zaki[2]
(Corresponding author: Tarek S. Sobh)

Information System Department, Egyptian Armed Forces, Cairo, Egypt[1]
(Email: tarekbox2000@yahoo.com)
Computer and System Engineering Department, AL-Azhar University, Nasr City, Cairo, Egypt[2]

(Received Mar. 3, 2006; revised and accepted May 31, 2006 & Nov. 8, 2006)

Abstract

In a subscription-based remote service, a user is charged a flat fee for a period of time independent of the actual number of times the service is requested. The main concern of the service manager is to make sure that only customers that have paid the fee for the current period are granted access to the service. To do this, the service manager might give each user a username and a password to be used for accessing the service. An SSL/TLS (Security Sockets Layer/ Transport Layer Security) session is started each time a user requests the service. As a part of the handshake protocol of SSL/TLS, the user hands a certificate to the server and proves to be the legitimate owner of the certificate. Then, the server application matches the certificate against a list of qualified certificates and decides whether to grant access. The most time-consuming phase of the SSL/TLS security protocol is the handshaking process between the client and the server, since many messages should be sent until successful negotiation is done and a secure session is created. In this paper we introduce a security management system in order to: 1) improve the handshaking process by making use of SSL/TLS client-side session caching, and 2) allowing trusted users to share sessions with others. According to our experimental setup, the proposed enhancement of SSL/TLS has improved its performance relative to the corresponding traditional handshaking of SSL/TLS protocol.

Keywords: Certificate authority, client-side caching, security protocols, session management, session sharing, SSL/TLS

1 Introduction

There is a significant and growing set of distributed computing environments where the resources, resource stake holders, and users are geographically and organizationally distributed [27]. Security of data in transit over the Internet has become increasingly important because of the steadily growing data volume. Nowadays, every user of a public network sends various types of data that range from email to credit card details daily and he would like such data to be protected when being in transit over a public network. To this end, a practical SSL (Secure Sockets Layer) protocol has been adopted for protection of data in transit that encompasses all network services that use TCP/IP to support typical application tasks of communication between servers and clients. Security is also critical to a wide range of current and future wireless data applications and services. Anand Raghunathan highlights the challenges posed by the need for security during system architecture design for wireless handsets, and provides an overview of emerging techniques to address them. The Internet holds unlimited promise for changing the way we do business [19, 20], but not without first addressing the security risks. A secure network design starts with a strong security policy that defines the freedom of access to information and dictates the deployment of security in the network. Privacy, integrity, and authenticity should be realized to protect information transfer across network links.

There are a number of security protocols available for securing communications on the Internet. The most relevant security protocols to this work are IPSec (IP Security Protocol), S/MIME (Secure/Multipurpose Internet Mail Extension [15], Kerberos [4, 18, 26] and SSL/TLS (Secure Sockets Layer /Transport Layer Security).

SSL/TLS runs above TCP/IP and below higher-level protocols such as HTTP or IMAP as shown in Figure 1. It uses TCP/IP on behalf of the higher-level protocols, and in the process allows an SSL-enabled server to authenticate itself to an SSL-enabled client, allows the client to authenticate itself to the server, and allows both machines to establish an encrypted connection.

SSL/TLS has a concept of a session. Once a session negotiation is completed, the participants share the same session secret and thus, they can authenticate the follow-

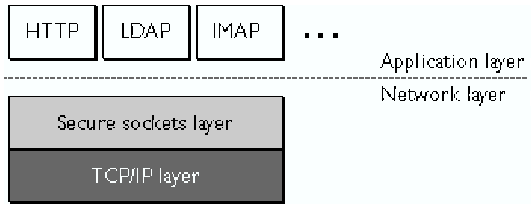

Figure 1: SSL runs above TCP/IP and below high-level application protocols

ing messages using the session key. If the dynamically created session secret is given to each service instance, it can be used to authenticate the session participants [21, 30]. However, SSL/TLS is a two-party protocol, thus it is not designed to share the same communication channel among multiple parties. Another disadvantage of the TLS is its dependence upon digital certificates, from certificate authorities, where some times sensitive organizations need to build and use there own digital certificates.

Another difficulty when using TLS is that the amount of memory consumed by the session cache increases roughly with the number of sessions cached on the TLS server. So a hardware load balancer [6], or the client-side session caching is used to overcome this problem. The goal of this work is to overcome the TLS protocol's problem by designing a security management system that uses the traditional TLS protocol, and can use its own digital certificates. Moreover the system uses the client-side session caching and session sharing techniques to improve the TLS secure system performance. It should be obvious that the proposed enhancement could improve the protocol speed while its security behavior is still unchanged.

This paper is organized as follows: Section 2 presents the security protocols overview and SSL/TLS objectives and architecture. Section 3 defines what is a session, the session's parameters and explains client side caching for TLS approach in distributed implementations. Section 4 presents the proposed system model, goals, and components. Section 5 presents formal analysis of performance improvements on the proposed network security protocol. Section 6 includes the implementation details of the proposed security system. Section 7 explains the performance analysis of the proposed system. Section 7 refers to some related works to the targets of this paper. Section 9 notifies the conclusion and recommendations for future work.

2 SSL/TLS Protocols Overview

It is convenient to explain here the SSL/TLS structure from the network point of view, and to explain the objectives behind using it, as such.

2.1 The SSL Handshaking

The SSL protocol uses a combination of public-key and symmetric key encryption. Symmetric key encryption is much faster than public-key encryption, but public-key encryption provides better authentication techniques. An SSL session always begins with an exchange of messages called the SSL handshake. The handshake allows the server to authenticate itself to the client using public-key techniques, then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows. Optionally, the handshake also allows the client to authenticate itself to the server. The exact programmatic details of the messages exchanged during the SSL handshake are beyond the scope of this paper. However, the steps involved can be summarized as follows (assuming the use of the cipher suites listed in Cipher Suites with RSA Key Exchange):

1) The client sends the server the client's SSL version number, cipher settings, randomly generated data, and other information the server needs to communicate with the client using SSL.

2) The server sends the client the server's SSL version number, cipher settings, randomly generated data, and other information the client needs to communicate with the server over SSL. The server also sends its own certificate and, if the client is requesting a server resource that requires client authentication, requests the client's certificate.

3) The client uses some of the information sent by the server to authenticate the server. If the server cannot be authenticated, the user is warned of the problem and informed that an encrypted and authenticated connection cannot be established. If the server can be successfully authenticated, the client goes on to Step 4.

4) Using all data generated in the handshake so far, the client (with the cooperation of the server, depending on the cipher being used) creates the premaster secret for the session, encrypts it with the server's public key (obtained from the server's certificate, sent in Step 2), and sends the encrypted premaster secret to the server.

5) If the server has requested client authentication (an optional step in the handshake), the client also signs another piece of data that is unique to this handshake and known by both the client and server. In this case the client sends both the signed data and the client's own certificate to the server along with the encrypted premaster secret.

6) If the server has requested client authentication, the server attempts to authenticate the client (see Client Authentication for details). If the client cannot be authenticated, the session is terminated. If the client

can be successfully authenticated, the server uses its private key to decrypt the premaster secret, then performs a series of steps (which the client also performs, starting from the same premaster secret) to generate the master secret.

7) Both the client and the server use the master secret to generate the session keys, which are symmetric keys used to encrypt and decrypt information exchanged during the SSL session and to verify its integrity–that is, to detect any changes in the data between the time it was sent and the time it is received over the SSL connection.

8) The client sends a message to the server informing it that future messages from the client will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the client portion of the handshake is finished.

9) The server sends a message to the client informing it that future messages from the server will be encrypted with the session key. It then sends a separate (encrypted) message indicating that the server portion of the handshake is finished.

10) The SSL handshake is now complete, and the SSL session has begun. The client and the server use the session keys to encrypt and decrypt the data they send to each other and to validate its integrity.

2.2 TLS Goals

The goals of TLS protocol, in order of their priority [3] are:

- Cryptographic security: TLS should be used to establish a secure connection between two parties.
- Interoperability: Independent programmers should be able to develop applications utilizing TLS that will then be able to successfully exchange cryptographic parameters without knowledge of one another's code.
- Extensibility: TLS seeks to provide a framework into which new public key and bulk encryption methods can be incorporated as necessary. This will also accomplish two sub-goals: to prevent the need to create a new protocol (and risking the introduction of possible new weaknesses) and to avoid the need to implement an entire new security library.
- Relative efficiency: Cryptographic operations tend to be highly CPU intensive, particularly public key operations. For this reason, the TLS protocol has incorporated an optional session caching scheme to reduce the number of connections that need to be established from scratch. Additionally, care has been taken to reduce network activity.

2.3 TLS Protocol

The primary goal of the TLS Protocol is to provide privacy and data integrity between two communicating applications. The protocol is composed of two layers [21].

These two layers are the *TLS Record Protocol* and the *TLS Handshake Protocol*. The TLS Record Protocol provides connection security that has two basic properties:

1) The connection is private: Symmetric cryptography is used for data encryption (e.g., DES, RC4, etc.) The keys for this symmetric encryption are generated uniquely for each connection and are based on a secret negotiated by another protocol (such as the TLS Handshake Protocol). The Record Protocol can also be used without encryption.

2) The connection is reliable: Message transport includes a message integrity check using a keyed MAC. Secure hash functions (e.g. SHA, MD5, etc) are used for MAC computations. The Record Protocol can operate without a MAC, but is generally only used in this mode while another protocol is using the Record Protocol as a transport for negotiating security parameters.

The TLS Record Protocol is used for encapsulation of various higher level protocols. One such encapsulated protocol, the TLS Handshake Protocol [8], allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives its first byte of data. The TLS Handshake Protocol provides connection security that has three basic properties:

1) The peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA, DSS, etc.). This authentication can be made optional, but is generally required for at least one of the peers.

2) The negotiation of a shared secret is secure: the negotiated secret is unavailable to eavesdroppers, and for any authenticated connection the secret cannot be obtained, even by an attacker who can place himself in the middle of the connection.

3) The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the parties to the communication.

One advantage of TLS is that its application protocol is independent.Higher level protocols can work on top of the TLS Protocol transparently. The TLS standard, however, does not specify how protocols can add security. The decision on how to initiate TLS handshaking and how to interpret the authentication certificates exchanged is left up to the judgment of the designers and implementors of protocols that run on top of TLS.

When we study the TLS architecture, we find that the most consuming phase is the handshake phase, which requires a lot of messages to be sent between the client

and the server, until both ends agree each other. This affects the overall communication, performance and network traffic [21].

3 Concepts of Session Management and Client Side Caching for TLS

In TLS, when the handshake process between a client and a server is successfully completed, a session is created to link the client with the server. In practical, The server while working will receive a lot of connection requests which leads to a large number of connections will be created and simultaneously running, so the server will need a way to manage and monitoring these connections.

3.1 Session Definition

A TLS session is an association between a client and a server. Sessions are created by the handshake protocol. Sessions define a set of cryptographic security parameters, which can be shared among multiple connections. Sessions are used to avoid the expensive negotiation of new security parameters for each connection.

3.2 Session Parameters

A session state is defined by the following parameters:

- Session identifier: this is an identifier generated by the server to identify a session with a chosen client;
- Peer certificate: X.509 certificate of the peer;
- Compression method: a method used to compress data prior to encryption;
- Algorithm specification termed CipherSpec: specifies the bulk data encryption algorithm and the hash algorithm used during the session;
- Master secret: 48-byte data being a secret shared between the client and server;
- "is resumable": this is a flag indicating whether the session can be used to initiate new connections.

There are several models for authorization in session management systems. One is the pull model where the user presents only his authenticated identity to the resource gateway (the policy enforcement point-PEP). PEP finds (pulls) the policy information for the resource and evaluates the user's access [27]. A classic example of this is local file system access where the user id of the process that is attempting to reference the file is compared to the access control list of the file. The other general model is the push model, where the user presents one or more tokens or assertions that grant the holder specific rights to the resource. In this model, the gatekeeper has to verify that the user has the rights to use the tokens and then to interpret the rights that have been presented. The original examples of this model were capability-based operating systems where access to files and other objects was granted on the basis of unforgettable tokens, called capabilities, associated with a process [14] With the growing use of digitally signed certificates that can be verified for data integrity and authenticity, the push model is gaining wider usage. There are also hybrids of the two models, such as when a user presents identity information that includes restrictions on his full set of rights, or presents a handle to an authentication/authorization server from which the gatekeeper may pull information about the user and his rights. Here we recommend using hybrids of both pull model and pushing model in order to handle an authentication/authorization server.

3.3 Client Side Caching for TLS

TLS is a widely deployed protocol for securing network traffic. It is commonly used for protecting web traffic and some e-mail protocols such as IMAP and POP. In this paper we consider a modification to the TLS handshake protocol that extends TLS' session resumption mechanism to reduce the load on the server. The amount of memory consumed by the session cache scales roughly with the number of sessions cached on the server. The exact amount of memory consumed by each session varies, but is at minimum 48 bytes for the master secret. Since session IDs are themselves 32 bytes, 100 bytes is a reasonable approximation. Assuming a server operating at 1000 handshakes/second these rates are equally shared between client and server, which are easily achievable with modern hardware. Such session cache may occupy a considerable amount of memory.

Shacham [24] indicates that when a cluster of SSL servers behind a load balancer serves a web site, the problem of sharing the session cache becomes a distributed systems problem. In general, server implementers choose to ignore this problem. Instead, each server has its own session cache and the load balancer is expected to direct returning clients to their original server. This increases load balancer complexity. Moreover, the need to maintain connection locality to make use of the session cache can interfere with the load balancer's ability to distribute load evenly.

4 The Proposed Model

This model aims informally at proposing a reliable security scheme that can provide secure communication using TLS and to improve the handshaking process by using a client session caching mechanism to reach the targets stated in what follows.

4.1 Objectives

The main model objectives are to:

- Create a security system on the basis of the traditional SSL/TLS technology.
- Overcome the main disadvantage of SSL/TLS, which is the dependence upon using digital certificates to work properly. Actually, it is not designed to share the same communication channel among multiple parties by creating secure sessions between participants and allows super users permitted by the security managers to share sessions with other connected users.
- Use the client side session caching to improve the server performance by caching the session parameters needed for authentication on the client and using them directly to connect to the server rather than repeating the whole process again.

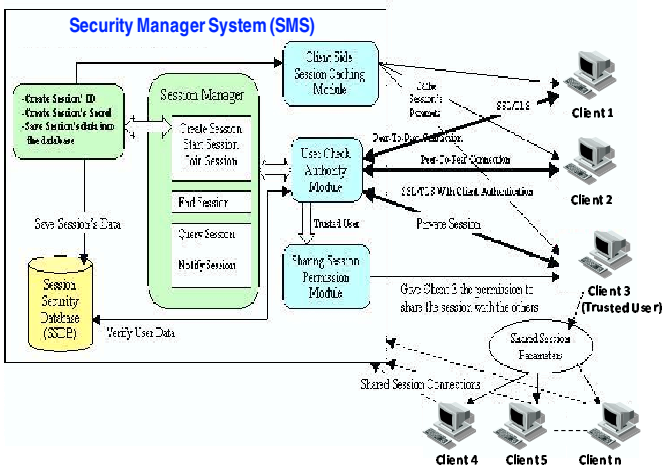

Figure 2: System model

4.2 System Model Components

Here many clients connect to the server for different business purposes and all clients need to transmit data over the network. The proposed system model contains network resources, and different clients geographically and organizationally distributed. Clients send various types of data such as email and credit card details and they protect their data using secure connections over SSL/TLS between servers and clients. You can notice that the handshake protocol associate with each connection over SSL/TLS will affect the overall performance and postpone the end user services especially in large business applications. In order to achieve this secure connection with better performance the proposed model uses client side session caching for collecting register users information and current connections and trusted users such as Client 3 in Figure 2 can share this connection to other users based on business security policy.

The proposed Security Manager System (SMS) consists of the User Check Authority Module (UCAM), the Client Side Session Caching Module (CSSCM), and the Sharing Session Permission Module (SSPM). All these modules cooperate with both the Session Manager Module (SMM) and Session Security Database (SSDB) as shown in Figure 2.

4.2.1 The User Check Authority Module

It is responsible for checking the client entry verification, specifying the user type (Trusted or Normal), and connection type (TLS or Private Secure Session). When a client try to connect to the server, this module checks the user name and password and gets the data of the user account and decides if this user is a Trusted User (i.e. he can share session data with another user). Next, this module decides which Security option will take to apply secure authentication with the requesting client. A Client may connect to the server by one of the following: 1) use SSL/TLS, 2) use SSL/TLS With Client Authentication, and 3) Use Secure Private Sessions.

4.2.2 The Session Manager Module

It is responsible for all the session's operations (create, join, end, query, notify), and monitoring the current sessions.

Next, in web services the Session Manager Creates the TLS Session, saves the session parameters, which is successfully created, and the user connection data in the security database, now a secure connection is established and data can be send in both sides.

The Session Manager's Functions:
The session management functions [12] are shown in Figure 3, where a Security Management System (SMS) manages sessions using the session manager. An SMS is responsible for:

- Assigning session IDs;
- Creating session secrets;
- Maintaining the status information for each session and keeping the participants informed of the status, and
- Shutting down sessions.

A session is created after a successful handshake process or by resuming it using the cached parameters on the client (by sending a StartSession request to the SMS).

There are two ways for a client to join the created session. One is to ask the SMS by sending a JoinSession request. If it is permitted according to the admission policy set by the initiator and the SMS, the client receives the session secret. The alternative way is for a Trusted User to forward the session secret to the receiving client.

The Session Manager can do the following operations:

- CreateSession: when a client requests for a new connection, a new session is created, then its data is stored in the security database (SSDB).

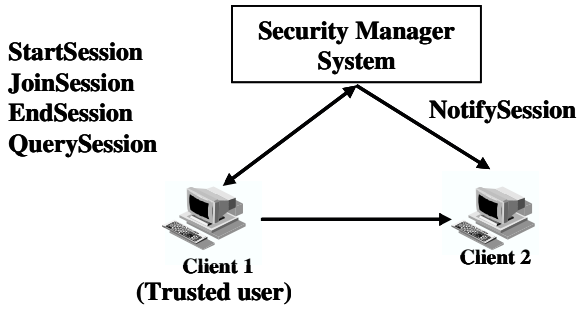

Figure 3: The security manager operations

- StartSession: when a session is ended, it can be resumed using the parameters that were cached on the client.
- JoinSession: when a client (a registered user) asks to share a created session, the Session Manager can send the session secret to that client and allows him to share the specified session.
- EndSession: this is done by the Session Manager to destroy a certain session.
- QuerySession: this is done by the Session Manager to retrieve information about certain session.
- NotifySession: An Session Manager should be capable of notifying its participants of changes in the session status. In contrast to the previous messages, which are all request/response, this message is one-way. The most important change to be notified of is the shutdown of the session. This message can also be used for renewing a session secret before it expires.

If a trusted user would like to share the session he will use:

- ForwardSession: When a trusted user wishes to invite another party to participate in the session, he can do so by sending a ¡ForwardSession¿ message to the new participant.
- The forwarder must have the permission when forwarding a session. The forwarder also needs to trust the other party. This message needs to be protected from eavesdroppers by an appropriate means because this message contains the session secret.

The details of the admission policy are not presented here but the policy must be digitally signed by the SMS if the policy needs to be protected from unauthorized modifications. Also if the forwarder needs to know the forwarding path, the forwarding path from the original requester to the forwarder must be included in the message.

4.2.3 The Client Side Session Caching Module

As explained before, the client side session caching technique is used to decrease the load upon the server to manage the huge number of sessions usually being opened. This module is responsible for allowing the client to cache his session parameters, so that he can resume his connection without need to do full handshake process with the server. After the session is created the SMS sends the session's parameters to the client, including the session id, the cipher suite used by session.

4.2.4 The Session Security Database

Contains data for the registered users and their authorities, all data of the connected sessions.

- Registered user table: stores information of all users registered by the system like: user name, user password, security option, and connection type. This information is used by the User Check Authority Module to check new user, and by the Session Manager to create sessions.
- Current sessions data: stores information of all current opened sessions like: session's id, user name who opened the session, cipher suite used. This information is used by the Session Manager to perform any session's management operation (join, end, notify, etc.).

4.2.5 The Sharing Session Permission Module

After the client is connected to the server and a secure session is created, if the client is a trusted user, this module sends to the client a permission message, to give the Client the ability to share his session with the others (the Share Session button in the client interface will be changed from disabled status to enabled status).

5 Formal Analysis

As we have seen in Section 2 that the most time-consuming phase of the SSL/TLS security protocol is the handshaking process between client and server. Figure 4 presents server state diagram [1] with normal handshaking process.

So we introduce the new management system in order to improve the performance measure of the SSL/TLS protocol using session sharing and client side session caching.

A formal analysis to the proposed SSL/TLS enhancements is pointed out in what follows:

- UID: Requesting guest user with its identity.
- PID: Service Provider with its identity.
- AS: Administrator who can share session with other trusted users with its identity.

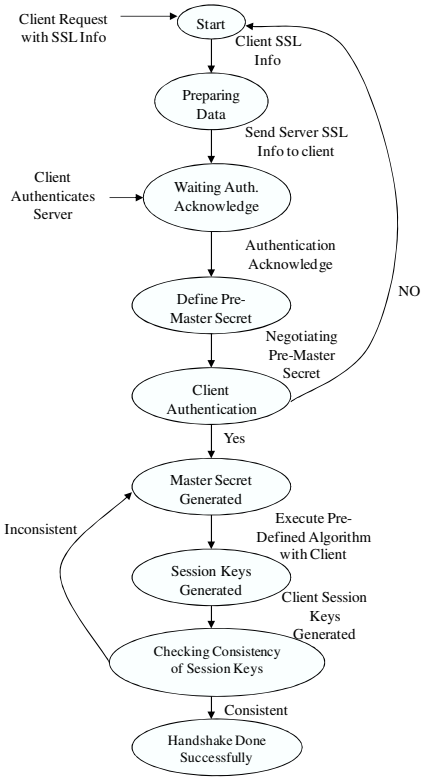

Figure 4: Server state diagram with normal handshaking process

- RU: The user who uses shared session in order to communicate with the server.
- SA: The session assigned by AS.
- CS: The session caching contains the necessary information parameters (session ID, issuer, owner ID, public key and issuer's signature, etc.)
- KDC: Key Distribution Center.
- WS: Web server connected with the client by a secure connection.
- DC: Disconnect secure connection session.

5.1 Initialize/Resume SSL/TLS Connection

The user tries to get a secure connection and is granted the connection parameters (session keys) for any reason it disconnects (lost the secured connection). Now the user tries to gain the secure connection again it does not have to do the same steps again like normal SSL/TLS protocol but it just retrieves the connection parameters from the cache to regain the connection. Figure 5 introduces the corresponding Finite State Machine of these steps.

$$\begin{aligned} U_{ID} &\rightarrow P_{ID} : (U_{ID}, A_S, R_U) \\ P_{ID} &\rightarrow KDC : (U_{ID}, A_S, R_U, P_{ID}) \\ KDC &\rightarrow P_{ID} : (U_{ID}, A_S, R_U) \\ KDC &\rightarrow C_S : (U_{ID}, A_S, R_U, P_{ID}) \\ P_{ID} &\rightarrow U_{ID} : (P_{ID}, A_S)A_S \\ U_{ID} &\rightarrow C_S(U_{ID}) \\ C_s &\rightarrow U_{ID} : (U_{ID}, A_S, R_U, P_{ID}) \end{aligned}$$

Step 1. The user UID initiates the authentication process by sending his identity to the service provider PID.

Step 2. Service provider just forwards the received data and his identity to the KDC.

Step 3. KDC realizes that both the user and the service provider are its registered principal's Then the KDC assigns a session key to this request, encrypts the session key and related data.

Step 4. KDC notifies the cache CS of new users that are approved connection. Then it sends CS the connection parameters.

Step 5. Authenticated and administrator user receive acceptance to open his secure session SA.

Step 6. User lost the secured connection (Disconnected).

Step 7. User requests the connection parameters to reconnect the secured connection.

Step 8. Cache CS sends the connection parameters that are saved in (Step 4)

5.2 Share Secure SSL/TLS Session

The administrator user tries to authenticate through the service provider to the web server and gets connected. The administrator user receives a request for secure connection from a trusted user, so instead of bypassing the request to the service provider and leave it take the normal sequence it just verifies the user and share the secure connection with the trusted user so as to minimize the time of creating a new secure connection.

$$\begin{aligned} U_{ID} &\rightarrow P_{ID} : (A_S) \\ W_S &\rightarrow P_{ID} : (A_S, W_S) \\ U_{ID} &\rightarrow U_{ID} : (U_{ID}, A_S, R_U) \\ A_S &\rightarrow A_S : (W_S, A_S) \\ R_U &\rightarrow R_U : (S_A, A_S, W_S) \\ R_U &\rightarrow C_S : C_S \end{aligned}$$

Step 1. The administrator user AS initiates the authentication process by sending his identity to the web server via service provider PID.

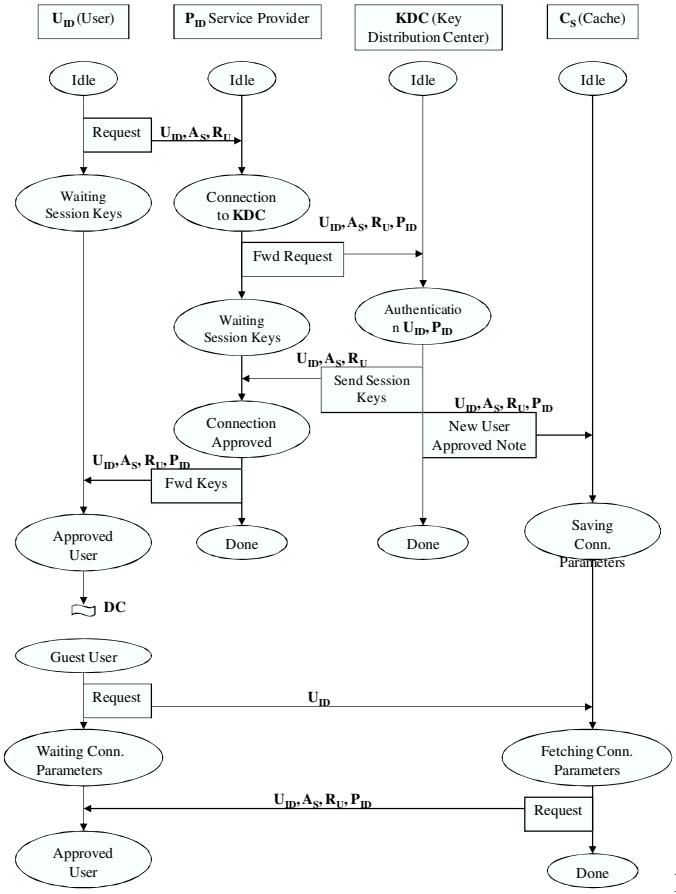

Figure 5: Finite state machine of the proposed protocol enhancements with client side caching

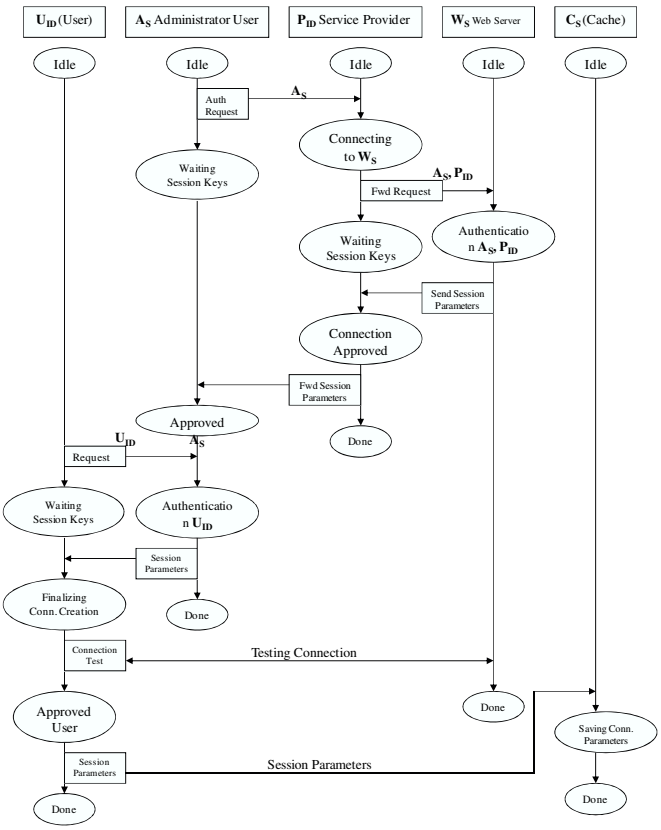

Figure 6: Finite state machine of the proposed protocol enhancements with session sharing

Step 2. The web server opens secure session with administrator user AS receives acceptance to open his secure session SA.

Step 3. The trusted guest user to the administrator authenticates his identity with administrator user.

Step 4. The authenticated user RU receives connection acceptance from the administrator user AS.

Step 5. Trusted user RU connects to the server using shared session with administrator user AS.

Step 6. Trusted user RU caches shared session parameters such as session ID into client side caching CS for fast future connection with server. A finite state machine of the proposed protocol enhancements with session sharing is shown in Figure 6.

6 System Developments and Implementation

Java language is mainly used in the implementation of the model. It has been chosen to give the implementation the ability to run either as a web application or a stand-alone java application.

6.1 Secure Socket Communication

Secure Socket Communication is created either by an SSLSocketFactory in client side or by an SSLServerSocket in a server side for accepting an in-bound connection (In turn, an SSLServerSocket is created by an SSLServerSocketFactory) [29].

Server Code for Secure Socket Communication: When writing a Java program that acts as a server and communicates with a client using secure sockets, the socket communication is set up with code similar to the following:

```
import java.io.*;
import javax.net.ssl.*;
. . .
int port = availablePortNumber;
SSLServerSocket s;
try {
      SSLServerSocketFactory sslSrvFact =
        (SSLServerSocketFactory)
        SSLServerSocketFactory.getDefault();
```

```
s =(SSLServerSocket)sslSrvFact.createServerSocket(port);
SSLSocket c = (SSLSocket)s.accept();
OutputStream out = c.getOutputStream();
InputStream in = c.getInputStream();
// Send messages to the client through the Output-
Stream
// Receive messages from the client through the Input-
Stream
}
catch(IOException e) {
}
```

Client code for secure socket communication:
The client code to set up communication with a server using secure sockets is similar to the following:

```
import java.io.*;
import javax.net.ssl.*;
. . .
int port = availablePortNumber;
String host = "hostname";
try {
    SLSocketFactory sslFact =
      (SSLSocketFactory)SSLSocketFactory.getDefault();
    SSLSocket s =
      (SSLSocket)sslFact.createSocket(host, port);
     OutputStream out = s.getOutputStream();
    InputStream in = s.getInputStream();
    // Send messages to the server through the Output-
Stream
    // Receive messages from the server through the
InputStream
}
catch(IOException e) {
}
```

6.2 The System Interface

When the Security Manager System (SMS) starts, it creates server sockets for secure communications and guest connections and the server is waiting for new connections from the clients. The system logs all events occurred in the first tab page "Log Data", so that the SMS can monitor all operations. In the "Registered Users" tab page, The SMS retrieves the registered users data from its security database (SSDB), including the user name and password, the security option, and the user type. When a user is successfully connected to the server, and a session is created, the data of this session is stored in the security database.

The trusted user allows sharing of his session with any other user request by choosing the user name from right list, then press the share session button, this will send the session ID to the selected user. In the normal client window the share session button is disabled because it is allowed for trusted users only.

The guest client can ask for sharing a trusted user by typing his nickname and select the trusted user he wants from the left list then press the guest button; this will send a request message to the trusted user, who can send back the session ID to the requester. Now, the clients can chat with the trusted client, and they can exchange messages or files to each other.

7 The System Performance

In this section we introduce the performance analysis for this work according to our experimental setup. The experimental work has been performed on a real network, Figure 7. Such network includes coaxial and UTP cables, multi-port repeater, switches, and a router. It contains neither fiber optic cables nor wireless communications, e.g. IEEE 802.11b through n.

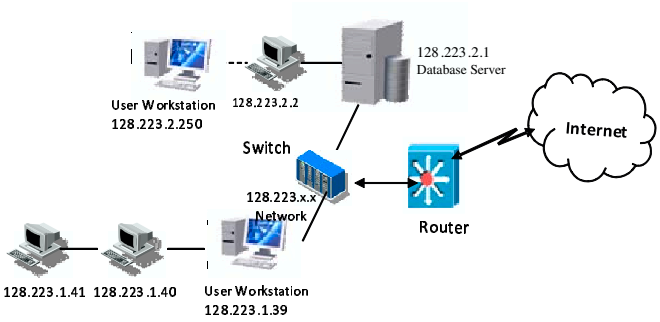

Figure 7: The experimental setup

7.1 Performance of Session Creation

While experimenting our system we noticed that when a trusted user try to establish a full handshake connection it costs in average 7 seconds, but when he tried to share his session with other guests or try to resume a session it costs in average 2 seconds. This is shown in Figure 8, where the proposed design has improved the performance of the underlying system by 3.5 times in average compared with the handshaking of typical TLS.

From the above figure, we conclude that the performance is improved by 350% (from 7 sec. for full handshake, to 2 sec. for session sharing) when using the session sharing technique rather than the full handshake to create a new session.

7.2 Performance of File Transfer

We tried to measure difference in the performance of a file transfer using the plain file transfer method and the secure file transfer method which we present in our model, our results were as shown in Figure 9.

From the above figure we conclude that using the secure file transfer which was proposed in our system increases the time for a file transfer, but this is normal because in our design the file is converted to an encrypted

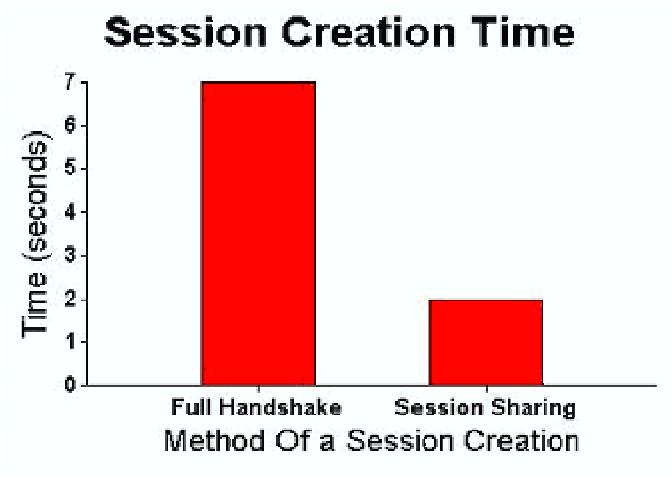

Figure 8: Analysis of session creation performance

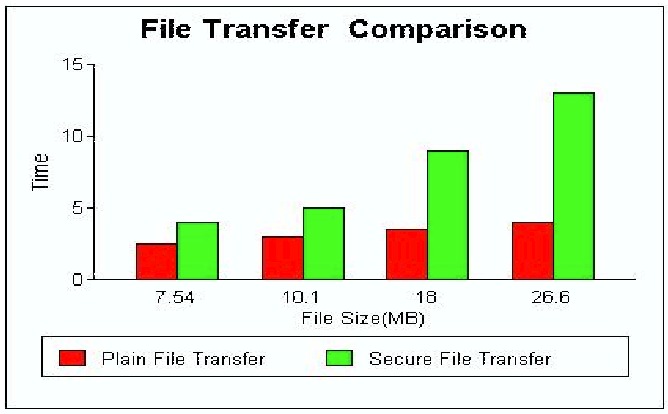

Figure 9: Comparison between methods of a file transfer

stream of bytes, this encryption from sender side and the decryption at the receiver side takes this delay, but finally we are ensured of the safe file transfer.

7.3 The Effect of Number of Users on File Transfer Performance

We tried to test the secure file transfer of the system using the same previous file sizes, but in case of different number of connected users we got the results represented in Table 1.

From that table it is concluded that, when the number of concurrent users increased, the time of file transfer for the same file size is also increased. The relation is nonlinear, however, this is normal due to the increase in network traffic.

8 Related Work

Because each new SSL connection incurs a handshake process, to be established high-traffic will potentially take

Table 1: File transfer time

Size (MB)	Number of connected users	Transfer (Seconds)
7.54	5	3.50
	15	3.90
	25	4.00
	35	4.2
10.10	5	4.00
	15	3.9
	25	5.00
	35	5.30
18.00	5	850
	15	8.80
	25	9.00
	35	9.50
26.60	5	12.50
	15	12.80
	25	13.00
	35	13.70

place. Thus, such sites may need a means of minimizing the performance degradation by bearing the security burden. Many existing systems and both hardware and software products had been introduced to supports the SSL/TLS tasks.

8.1 Existing Systems

The Community Authorization Service (CAS) [17] is a new authorization service being developed by the Globus project [10] for Grid environments. Their authorization model allows a resource site to grant a community access to resources and the authorization server for that community to grant access to the community members. This is implemented by having the user contact the CAS server to get a delegated proxy certificate [28], which includes a rights restriction extension that limits what resources can be accessed. The resource gatekeeper must interpret the restricted rights extension and verify that the community has such rights to the resource. Since the delegated proxy is a short-lived X.509 certificate it can be passed between the user and the resource gateway as part of the GSI/SSL connection. There is no additional information that needs to be conveyed, as is the case when a user needs to pass attribute certificates to the gatekeeper. CAS examination of policy and granting of rights is done before the gatekeeper is contacted. This means the user must ask for all the rights she will need in advance of referencing the resource.

Policy about resources is stored and managed by the CAS servers and so far mainly consists of lists of objects and allowed rights. This information is included in the rights restriction extension of the delegated proxy. The intent of the CAS project is to extend the policy language as the need arises. The CAS administrator is responsible

for adding each community member to the appropriate groups. The CAS administrator may also delegate administration of subsets of the objects to additional people. VOMS is another ad hoc solution to authentication in a GSI-enabled Grid [2]. It is similar to the CAS model, but the VOMS server is run by a virtual organization and supplies authorization information about its own members. This service is implanted and used within the European Data Grid.

8.2 Existing Products

Many hardware and software solutions had been introduced to improve the SSL/TLS performance.

8.2.1 Software Products

Beside the Client Side Session Caching technique used in this paper, Hovav Shacham and Dan Boneh had introduced two software approaches to improve the SSL/TLS performance [23, 24]. First was fast-track mechanism, second was improving the SSL handshake performance via batching.

- Fast-track mechanism: The "fast-track" mechanism provides a client side cache of a server's permanent public parameters and negotiated parameters in the course of an initial, enabling handshake. These parameters (like the server's certificate chain, the preferred client-server cipher suite, and the preferred client-server compression method) need not be resent on subsequent handshakes. Fast-track reduces both network traffic and the number of round trips, and requires no additional server state. These savings are most useful in high latency environments such as wireless networks [24].

- Batching the handshake requests: It is an algorithmic approach for speeding up SSL's performance on a web server. The batching improves the performance of SSL's handshake protocol by up to a factor of 2.5 for 1024-bit RSA keys. It is designed for heavily loaded web servers handling many concurrent SSL sessions [23]. The basic idea is as follows: the web server waits until it receives b handshake requests from b different clients. It then treats these b handshakes as a batch and performs the necessary computations for all b handshakes at once. The experiments show that, for b = 4, batching the SSL handshakes in this way results in a factor of 2.5 speedup over doing the b handshakes sequentially, without requiring any additional hardware.

8.2.2 Hardware Products

Many hardware solutions were introduced to improve the performance when using the TLS protocol, the result was the appearing of the SSL Accelerators and the SSL Offloaders.

- The SSL accelerator: It was introduced in 1998, the hardware SSL Accelerator was a dedicated coprocessor used to sped the RSA operation (handshake process), but it had a number of drawbacks: it required special software and drivers in order to work, it was only able to accelerate one server at a time, and it did nothing for the other components of SSL. While the first two drawbacks affected interoperability, maintainability, and scalability, the third proved to be the greatest limiting factor of the accelerator. Because the accelerator only acted upon the RSA key exchange, and not on the bulk (symmetric) cryptography or the message digest, it left the decryption and the hashing to the server's host CPU. Although this was relatively not very burdensome for the servers, it nonetheless left the data obfuscated and unreadable by content networking gear, caching systems, and intrusion detection systems.

- The SSL offloader: It was the second step to improve SSL performance; it processed not only the asymmetric component of SSL, but also all components of SSL. This means that with an offloader, the host CPU on the web-server is not responsible for processing any portion of the SSL traffic. Also, offloaders provide universal compatibility with web and application server platforms, avoiding the need to load special software or drivers on the servers. Moreover, because offloaders are network attached, they can offload multiple servers rather than just one, and they can also scale easily to accommodate any size site.

  Many SSL accelerator products appeared like the Alteon SSL Accelerator (from Nortel Networks) [16], the BIG-IP SSL Accelerator 400/800 (from f5 networks), also many SSL offloader products appeared like SonicWALL SSL-R and SSL-RX offloaders. [25]

9 Conclusion

In this paper a security scheme has been proposed to extend the use of SSL/TLS protocol and to provide a high performance session management system. It allows building secure communications without need to get a digital certificate from a Certificate Authority (CA). This feature is adequate for organizations that need to build their own SSL/TLS environments. According to the proposed model the security management system (SMS) can give the trusted users a facility to share sessions among themselves and other users. Thus a secure community can be built up under the protection of SSL/TLS.

Also, here an enhancement to the TLS handshake protocol that makes it easier to use is presented. Such enhancement will be useful, specially, in environments where fast connections are needed while a large number of clients are already connected to the server. This is achieved by making use of the client-side session caching approach. In fact, such session caching relocates the session cache from

the client to the server allowing the server to maintain a much larger set of records. Accordingly, the connections that have required a full handshake can be resumed rather than activated. It is noticed that using client side session caching and session sharing techniques have reduced the session creation time by 350%. Such performance improvement has been achieved whilst the security measurements are kept unchanged.

References

[1] A. V. Aho, R. Sethi, and T. D. Ulman, *Compilers: Principals Techniqes and Tools*, A disown Wesley, 1986.

[2] R. Alfieri, R. Cecchini, V. Ciaschini, L. Dell'agnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro, VOMS, "An authorization system for virtual organizations," in *1st European Across Grids Conference*, Feb. 2003.

[3] C. Allen and T. Dierks, *The TLS Protocol Version 1.0*, IETF RFC 2246, 1999.

[4] C. Baliello, A. Basso, and C. D. Giusto, *Kerberos Protocol: an Overview*, Distributed Systems, Italy, Fall 2002.

[5] M. Blaze, J. Ioannidis, and A. D. Keromytis, "Trust management for IPsec," *ACM Transactions on Information and System Security (TISSEC)*, vol. 5, no. 2, pp. 95-118, 2002.

[6] W. Chou, "Inside SSL: Accelerating secure transactions," *IEEE, IT Professional*, vol. 4, no. 5, pp. 37-41, Sep./Oct. 2002.

[7] D. Clark, *Vulnerability's of IPSEC*, SANS Institute, as part of the Information Security Reading Room, Mar. 14, 2002.

[8] G. Diaz, F. Cuartero, V. Valero, and F. Pelayo, "Automatic verification of the TLS handshake protocol," in *Proceedings of the 2004 ACM symposium on Applied computing*, pp. 789-794, 2004.

[9] T. Dierks and C. Allen, *The TLS Protocol*, version 1, IETF RFC 2246, 1999. (http://www.ietf.org/rfc/rfc2246.txt)

[10] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Mateo, CA, 1999.

[11] A. Godber and P. Dasgupta, "Secure wireless gateway," in *Proceedings of the ACM workshop on Wireless security*, pp. 41-46, 2002.

[12] S. Hada and H. Maruyama, *Session Authentication Protocol for Web Services*, IBM Research, Tokyo Research Laboratory, 2002.

[13] R. Housley, W. Polk, W. Ford, and D. Solo, *Internet X.509 Public Key Infrastructure Certificate and CRL Profile*, 2001. (http://www.ietf.org/internetdrafts/draft-ietf-pkix-new-part1-12.txt)

[14] H. M. Levy, *Capability-Based Computer Systems*, Digital Press, Bedford, MA, 1984. (http://www.cs.washington.edu/homes/levy/capabook/)

[15] H. Moser, *S/MIME*, Dec. 2001-Jan. 2002, mail@heinz.at

[16] Nortel Networks, *Alteon 8661 SSL Acceleration Module for the Passport 8600*, 2003. (http://www.nortelnetworks.com)

[17] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke, "A community authorization service for group collaboration," in *Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, 2002. (http://www.globus.org/research/papers.html#CAS-2002)

[18] A. A. Pirzada and C. McDonald, "Kerberos assisted Authentication in Mobile Ad-hoc Networks," in *Proceedings of the 27th conference on Australasian computer science*, vol. 26, pp. 41-46, 2004.

[19] A. Raghunathan, N. Potlapally, and S. Ravi, "Securing wireless Data: System architecture challenges," in *Proceedings of the 15th international symposium on System Synthesis (ISSS'02)*, pp. 195-200, Oct. 2002.

[20] S. Ravi, A. Raghunathan, N. Potlapally, and M. Sankaradass, "System design methodologies for a wireless security processing platform," in *39th Design Automation Conference (DAC'02)*, pp. 777, Jun. 2002.

[21] E. Rescorla and Addison-Wesley, *SSL and TLS Designing and Building Secure Systems*, IEEE Cipher, E40, Dec. 19, 2000.

[22] RSA, RSA BSAFE, RC2, RC4, RC5, and MD5 are trademarks or registered trademarks of RSA Data Security, Inc., 1999.

[23] H. Shacham and D. Boneh, "Improving SSL handshake performance via batching." in *proceedings of the RSA Conference 2001*, Lecture Notes in Computer Science, Vol. 2020, Springer-Verlag, pp. 28-43,Apr. 2001.

[24] H. Shacham, D. Boneh, and E. Rescorla, "Client Side Caching for TLS," in *Proceedings of the Symposium on Network and Distributed System Security (SNDSS'02)*, pp. 195-202, 2002.

[25] SonicWALL, *SSL-R and SonicWALL SSL-RX,High Performance Commercial Application Accelerators*, 2002. (http://www.sonicwall.com/products/trans.asp)

[26] P. Todaro, *An Overview of the Kerberos Authentication Protocol*, SANS Institute, as part of GIAC practical repository, Oct. 2003.

[27] M. R. Thompson, A. Essiari, and S. Mudumbai, "Certificate-based authorization policy in a PKI environment," *ACM Transactions on Information and System Security*, vol. 6, no. 4, pp. 566-588, Nov. 2003.

[28] S. Tuecke, D. Engert, I. Foster, V. Welch, M. Thompson, L. Pearlman, and C. Kesselman, *Internet X.509 Public Key Infrastructure Proxy Certificate Profile*, IETF draft, 2002. (http://ww.ietf.org/internet-drafts/draft-ietf-pkix-proxy-03.txt)

[29] B. R. Wetmore, *Java Secure Socket Extension (JSSE) API*, JavaOne, Sun'2000 worldwide java developer conference, 2000.

[30] A. Yasinsac and J. Childs, "Analyzing internet security protocol" in *Proceedings of the 6th IEEE International Symposium on High Assurance Systems Engineering (HASE'01)*, pp. 149-159, 2001.

Tarek S. Sobh received his B.Sc. degree in computer engineering from Military Technical College, Cairo, Egypt in 1987. He received his M.Sc. and Ph.D. degrees from Computer and System Engineering Department, Faculty of Engineering, Al-Azhar University, Cairo, Egypt. He has designed and developed several package for business applications and security systems. His research of interest includes network management and security, distributed systems, knowledge discovery, and software engineering.

Ashraf Elgohary received his B.S. in computer engineering from Military Technical College, Cairo in 1993. He received his M.Sc. degree from Computer and System Engineering Department, Faculty of Engineering, AL-Azhar University, Cairo, Egypt in 2005. He works in network security and database systems fields.

M. Zaki is the professor of software engineering, Computer and System Engineering Department, Faculty of Engineering, Al-Azhar University at Cairo. He received his B.Sc. and M.Sc. degrees in electrical engineering from Cairo University in 1968 and 1973 respectively. He received his Ph. D. degrees in computer engineering from Warsaw Technical University, Poland in 1977. His fields of interest include artificial intelligence, soft computing, and distributed systems.