Secure Internet Access to Gateway Using Secure Socket Layer

Deep Vardhan Bhatt, *Member, IEEE*, Stefan Schulze, and Gerhard P. Hancke, *Senior Member, IEEE*

***Abstract*—The Internet is the most widely used medium to access remote sites. Data sent and received using transmission control protocol/Internet Protocol (TCP/IP) is in plain text format and can be accessed and tampered with quite easily and, hence, provides no data security. This is the case especially if the data are confidential and access to the gateway server has to be strictly controlled, although there are several protocols and mechanisms that have been thoroughly scrutinized to tackle these problems. This paper also intends to provide a model that uses secure socket layer (SSL) to provide a secure channel between client and gateway server. A smart card will be used for client authentication and encryption/decryption of the data.**

***Index Terms*—Cryptography, Internet, network, secure socket layer (SSL), security, smart card, transmission control protocol/Internet Protocol (TCP/IP).**

I. Introduction

THE PROBLEM of Internet security is nothing new; every year, the Computer Emergency Response Team (CERT) receives well over 2000 reports of security incidents, and almost ten times as many e-mails regarding security problems. Since the Internet can be viewed as a large network of interconnected computers, almost every user on the Internet is at risk against a security attack. More at risk are companies that transmit critical data across the Internet to their corporate sites all over the world. The need for reliable secure communication over the Internet is as prevalent now as it ever was. This fact is very much endorsed by several studies every year; Molva [2], [4] reiterates these views in great depth and presents several scenarios to have better Internet security architecture.

This paper addresses the problem of providing a secure means for a client to communicate with a server over an insecure channel, the Internet. The project aims to develop a proprietary Internet Protocol (IP) using the transport layer security (TLS)/secure socket layer (SSL) protocols that use a smart card as a client-authentication mechanism. More specifically, the paper provides a cost-effective solution for field-bus-related access where the data rate or high bandwidth is not that critical. Some of the achievements of field bus and their security concerns presented by Sauter and co-workers in [25] and [26] are covered as well. In the paper, we provide a practical solution to overcome the authenticity, integrity, and confidentiality issues when two clients need to communicate securely.

This is how the rest of the paper is organized: Next, we present a brief definition of smart card and SSL, Section II outlines the paper contribution, Section III presents the functional view of the system, Section IV presents design and implementation choices made, Section V presents the results, and last, the summary is presented in Section VI.

A. SSL

The SSL security protocol provides data encryption, server authentication, message integrity check, and optional client authentication for a transmission control protocol (TCP)/IP connection. Simply installing a digital certificate turns on their SSL capabilities. SSL supports the use of 40- and 128-bit symmetric cipher keys. Implementation of public-key systems is slower than symmetric ciphers. It is this fact that has led to the combination of the two techniques to achieve security and speed. SSL uses X.509 certificates for authentication, Rivest, Shamir, Adelman (RSA), or Digital Signature Algorithm (DSA) as its public-key cipher and one of ARCFOUR or Rivest Cipher 128bit (RC4-128), Data Encryption Standard (DES), Triple-DES, or International Data Encryption Algorithm (IDEA) as its bulk symmetric cipher.

B. Smart Card

A smart card contains a small computer, usually an 8-bit microprocessor, RAM, ROM, and either erasable programmable ROM (EPROM) or electrically EPROM (EEPROM). Smart cards can also have different cryptographic algorithms and protocols programmed into them and can be used to sign documents or unlock resources [5].

For smart cards to be used as an authentication mechanism, they first need to carry some secret information that could uniquely identify the user. This could either be a digital certificate, username–password pair, or a private key public key pair. Early problems in the development of smart cards included the fact that public-key cryptography required a lot more processing and computations than symmetric cryptography. A low-cost smart card would have difficulty in computing a 512-bit digital signature on the card and sending the result to the host application.

Authentication is a major issue for any online communication. Leach [1] proposed the low-cost dynamic authentication schemes using smart card over RSA-based ones. Since then,

Manuscript received November 29, 2004; revised October 13, 2005.

The authors are with the Department of Electrical and Computer Engineering, University of Pretoria, Pretoria 0002, South Africa (e-mail: dbhatt@postino.up.ac.za).

Digital Object Identifier 10.1109/TIM.2006.862009

several advanced and sophisticated proposals have been presented on numerous occasions; some of these schemes using smart cards for authentication are available [8]–[15], especially Kumar's [8] proposed scheme that caters to attacks via the registration phase and authentication phase of Hwang and Li's [9]. The design is based on digital certificates that are issued on the smart card by an in-house certificate authority using public key infrastructure (PKI); several important benefits using PKI are presented in [16] and [17].

II. Contribution

The project implementation aims to deliver the maximum amount of security to the communications link of the Internet gateway for unified automation network access (IGUANA) system, while still being practical and effective in its use and implementation. There is currently a lot of work going on in the area of smart cards and using smart cards as user-authentication mechanisms [8]–[15] and in the research area of a secure TLS protocol as well.

Using a combination of certificates, for server authentication, and smart cards, for client authentication, the project will bridge the gap between the two technologies of smart cards and a secure transport layer protocol.

This project satisfies one of the requirements of the IGUANA project; a brief description is presented in Section III [25]–[27].

The paper gives our findings, experiences, and observations during the implementation of the IGUANA project. In an application where the number of clients that have to be connected via the Internet to a remote server is substantially low, the use of the smart card for authentication, storage, easy handling, and lower cost can be significant. The paper presents our observations and results that are relevant to instrumentation and measurement from the point of view of data capturing, monitoring, and control of data nodes in the field area network (FAN). Although our focus is not on data capturing and control, other members of the IGUANA team have accomplished those tasks. Specifically, the paper presents the performance of the gateway server under various load conditions with PKI and smart-card-based authentication; the performance issues are discussed and presented in detail in Section V. In the summary in Section VI, some suggestions are presented to enhance the server performance.

III. System and Functional Overview

The IGUANA system provides access to a field-bus network where data are collected from different nodes by the gateway. Clients that connect to the gateway, through the TLS server program, can then access these nodes over the Internet. Fig. 1 provides an overview of the IGUANA system.

The access-control (AC) server (Fig. 1) receives data from the TLS server and sends these data to the Extended Service Daemon (ESD) server program, which then passes these data on to a specified node. The function of the AC server is to provide authorization for each connected client. The function of the ESD program is to interface with specific nodes and the

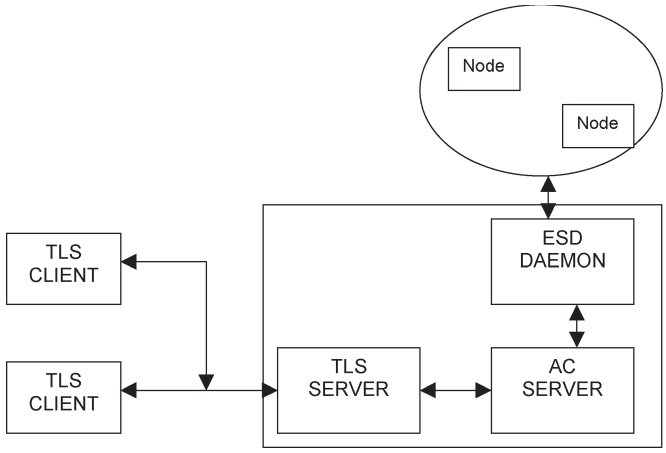

Fig. 1. System overview of the IGUANA system.

FAN. The client program has the same TLS interface as the server. The client also has a Graphical User Interface (GUI) for the user to interface with the client program. The client authentication is provided by a smart card attached to the user computer. Nodes are several data points that are attached to the field bus for various types of data collection and control of these nodes.

The functional diagram for the TLS server program can be seen in Fig. 2, and the functional diagram for the TLS client program can be seen in Fig. 3.

A socket is created through which data will be sent and received between the client and the TLS server as well as between the TLS server and the AC server. The socket created will encapsulate the data packet within a TCP/IP packet. The TCP/IP protocol will provide error detection, error correction, and retransmission to the communications link between the different entities. The TCP/IP Berkely stream sockets are used. The connection-control unit is responsible for managing, creating, and closing connections between the TLS client and the TLS server on the gateway as well as for managing, creating, and closing connections from the TLS server to the AC server. The handshake and authentication unit is to set up the security parameters for the connection, as well as authenticate the connecting client. The handshake and authentication unit for the client TLS program has the added responsibility of also authenticating the user to the smart card by using a personal identification number (PIN) in addition to performing server authentication. Public-key cryptography is used for client authentication and X.509v3 certificates for server authentication. The communication unit is to provide asynchronous input/output (I/O) to the server and client programs. The communication unit encrypts outgoing data and decrypts and verifies incoming data. A symmetric-key algorithm is used for chipper block chaining. The error control unit handles errors that occur during the execution of the server and client TLS programs and/or display error messages to the user, and in the case of the server, writes the errors in an error log file. The GUI unit is to provide a graphical interface to the user, which will display the received data and connection settings as well as provide interface controls to open a connection, close a connection, send data, and set any other security-related options for the

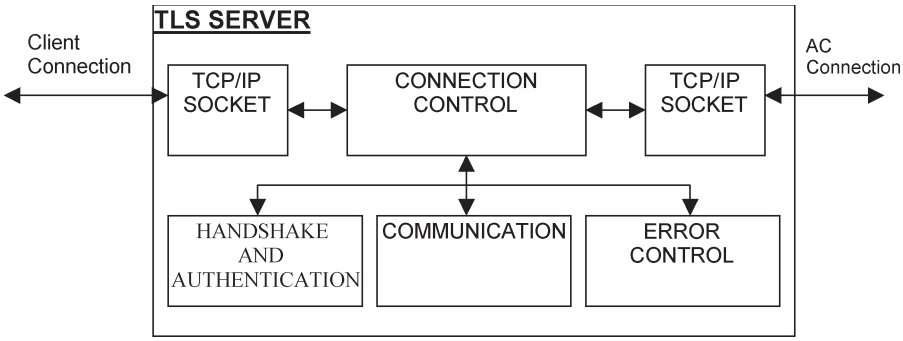

Fig. 2. Functional diagram of the TLS server program.

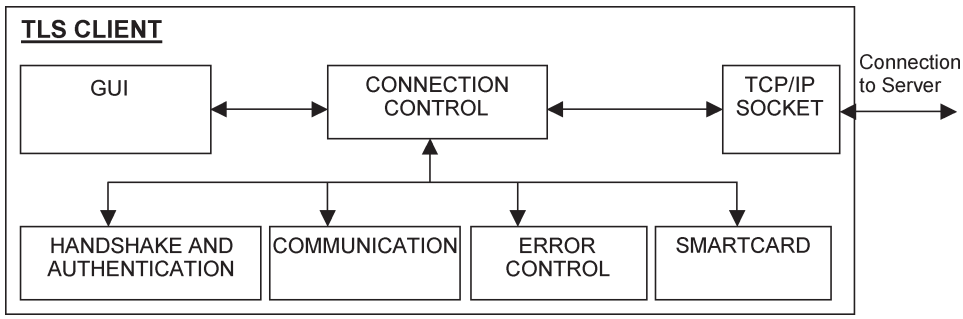

Fig. 3. Functional diagram of the client TLS program.

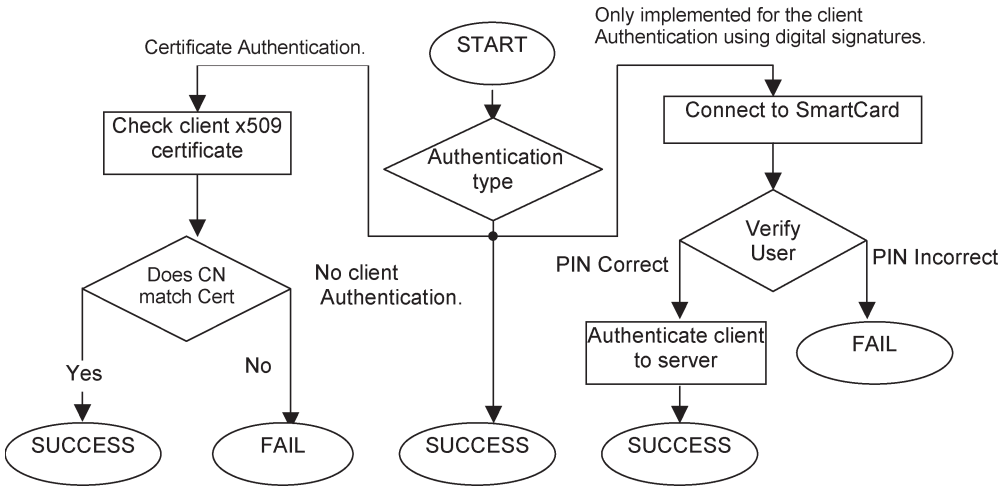

Fig. 4. Software flow diagram for the handshake and authentication unit.

connection to the server. The smart-card unit is to keep secret information secure and to perform cryptographic computations when requested to do so by the client program.

IV. Design and Implementation

OpenSSL appears to be most versatile SSL application programming interface (API) software library and is used to provide a secure SSL communication channel between the client and the server. The server program is programmed in American National Standard Institute (ANSI) C using the gcc compiler on a Linux system. The client is programmed in C on a Windows system using the Visual C++ 6 development environment. The smart-card system is simulated and tested using a software simulator (debugger) of the smart card. The Certification Authority (CA) is implemented using the OpenSSL command-line tools to issue certificates, and the BasicCard basic compiler is used to implement a terminal issuer program to initialize and issue smart cards.

A. Handshake and Authentication Unit

The flow diagram for the handshake and authentication unit is given in Fig. 4. The unit will implement three different types of authentication methods. These three methods are

1) no authentication;
2) client authentication with x509 certificates;
3) or client authentication using digital signatures.

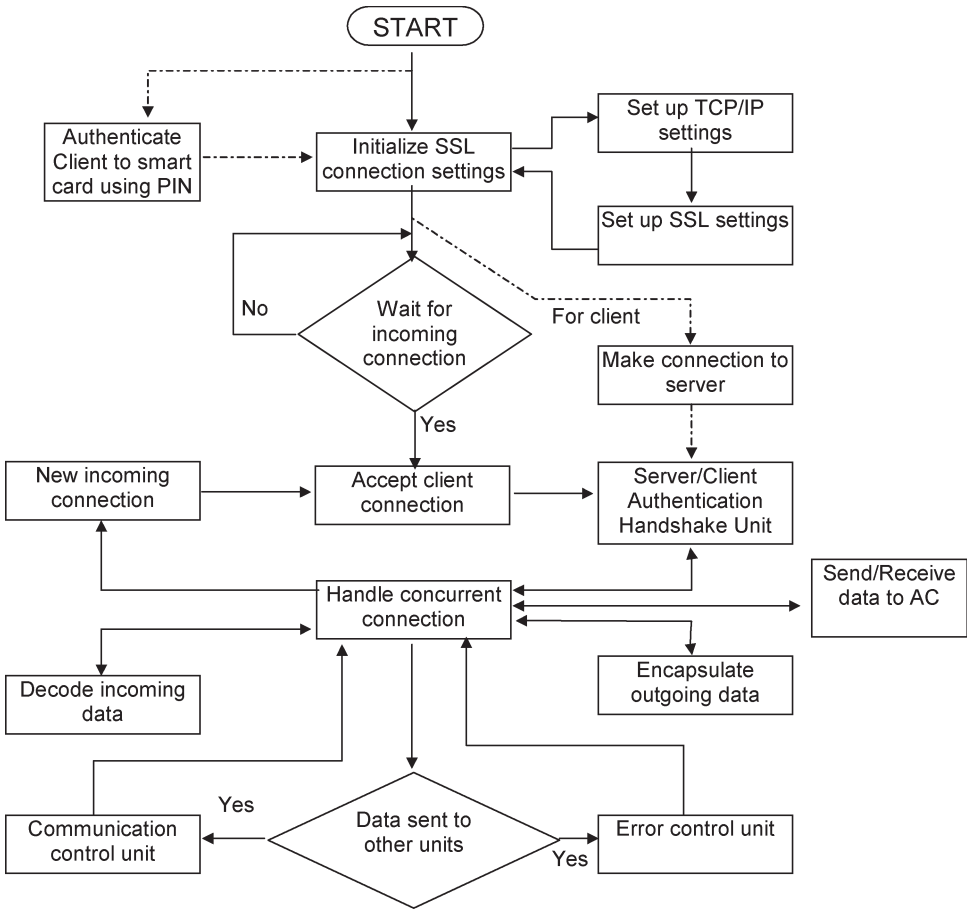

Fig. 5. Flow diagram of the connection-control unit.

For the client program, the handshake and authentication unit will also need to implement user authentication to the smart card by using a four-digit PIN that the user will need to know in order to activate certain functions on the card.

B. Connection Control and TCP/IP Socket

The conceptual flow diagram for the connection-control unit is shown in Fig. 5.

C. Client-Authentication Protocol

Public-key cryptography was chosen to implement the client authentication, and its implementation policies are presented and discussed in [17]–[20]. Symmetric-key cryptography relies on both the client and server having to share some sort of secret information before authentication can take place. It also relies on the server to securely store the client secret information in some sort of database. Here, the process of registering new clients means that new client information will need to be entered for each new client in the database itself. If IGUANA has a number of distributed IGUANA gateways, this will be a considerable implementation burden, especially when a new administrator's information will need to be entered at each IGUANA gateway server. Certain secret symmetric keys could possibly be given to a specific group of users. The downside is that if even one of those users were compromised, all users would be compromised. The problem of users that are compromised and have to be removed has not been solved, however. This is because the system still needs a secure means for specifying which user smart cards are no longer valid. The implementation does not consider compromised users and has effectively shifted the responsibility for this to the AC server. Username–password authentication is not secure enough for the system.

D. Communications Unit

The conceptual flow diagram for the communications unit is given in Fig. 6.

E. Error Control Unit

The conceptual flow diagram for the error control unit is given in Fig. 7.

V. Results and Discussion

A. Cost of Handshaking

A very common complaint from a lot of network administrators is that SSL is very slow. The primary reason for this is that the cryptography, particularly public-key cryptography, is extremely CPU intensive. There are two phases in the SSL connection: the handshake phase and the data-transfer phase.

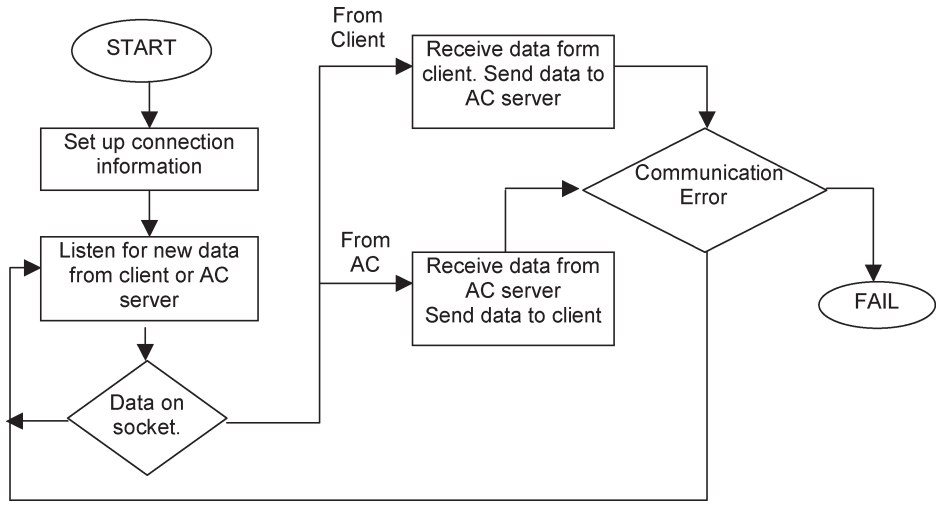

Fig. 6. Conceptual flow diagram for the communications unit.

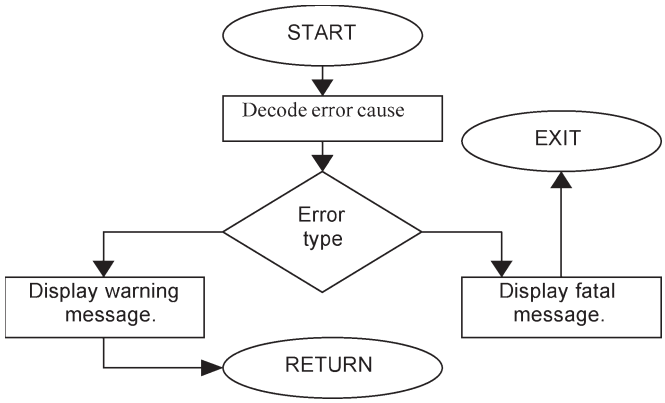

Fig. 7. Conceptual flow diagram for the error control unit.

The handshake phase only takes place once during a connection and is, compared with the data-transfer stage, expensive. The cipher suite used for encryption, authentication, and message integrity for each of the connections is 168-bit 3DES, with 1024-bit RSA authentication and Secure Hash Algorithm 1 (SHA-1) Message Authentication Code (MAC).

For the server, more than half of the CPU time is spent on the RSA decryption operation with the private key. The RSA decryption operation performed by the server is between the time when the client has sent the Finished message and the server replies with ChangeCipherSpec message. The client spends most of its time waiting for the server. The most expensive operation that the client needs to perform is verifying the server certificate and encrypting the pre-master-secret [3].

If client authentication were used, the client would perform an RSA private-key operation equivalent to the operation performed by the server. This RSA operation would thus become the dominant performance cost operation that the client would need to perform. Using client authentication with certificates adds almost 0.0071 s to the authentication process. The performance of public-key algorithms declines with key size. The RSA operation with a 1024-bit key is almost four times slower than RSA with a 512-bit key [3]. Thus, using RSA with a key size of 512 bits will decrease the time needed for handshaking.

The results of client authentication with a smart card provide an indication of the minimum time that would be required for handshaking and authentication when using the client-authentication protocol. The result does not take network latency or the smart-card interface to the client computer into account. The client-authentication protocol used with a smart card almost doubles the time needed for handshaking. The server program spends almost 0.0402 s to set the connection information, create a thread to handle the new client, and decrypt the authentication string. The server spends another 0.0020 s to generate a random value. The client spends 0.0102 s to RSA private encrypt the random value.

The handshake time is doubled because the server now has two extra RSA operations to perform. One operation is to decrypt the authentication string from the client with its CA public key, and the other is to decrypt the encrypted response from the client with the public key from the authentication string. The client performance is now dominated by the RSA private-key encryption operation that is performed on the smart card.

Nagle's algorithm [28] is intended to reduce small TCP packets. Nagle's algorithm requires that the network interface does not immediately send out TCP data but buffers it and only sends the data when the algorithm times out or receives an Acknowledgment (ACK). The Berkley socket option TCP_NODELAY is used to disable the algorithm. Results with a disabled Nagle algorithm have reduced the time needed for handshaking by an average of 0.0101 s for ten simulated clients.

Too much time is needed to perform client authentication using a smart card. If a server were ever bombarded with a large number of clients, the handshaking and authentication would create a huge bottleneck and limit the server performance in terms of throughput and response time. There are a number of ways to improve performance. The first method would be to use smaller keys, but this would limit the security of the authentication protocol. The system should always disable Nagle's algorithm to increase throughput and prevent temporary deadlock

TABLE I
PERFORMANCE RESULTS OF THE SERVER UNDER LOAD CONDITIONS

No. of concurrent client connections	Total connections	No. of successful connections	No. of failed connections	% of connections that failed
1	1169	1169	0	0%
10	1274	1272	2	0.16%
20	918	896	22	2.46%
30	908	856	52	5.73%
40	918	828	90	9.80%
50	932	812	120	12.88%
60	1299	1118	181	13.93%
70	1173	996	177	15.09%
80	1174	1002	172	14.65%
90	1097	929	168	15.31%
100	1202	1022	180	14.97%

between the server and the client. The third method to increase performance is to decrease the time needed to perform an RSA operation. This can be accomplished by using dedicated RSA processing hardware or cryptographic accelerators.

B. Performance of Server Under Load

SSL/TLS can very easily become a huge part of the processing overhead of an application, especially server applications. The cryptographic primitives used in the protocol (most noticeably the session handshaking) can consume a lot of processing time and resources, and the result is that an application running at its SSL/TLS processing capacity can drag a system to its knees. Servers running SSL/TLS are therefore at the mercy of the demands placed. Listed below are some questions that are asked by this experiment.

1) How much traffic can a given server sustain if it is saturated to capacity?
2) If a given simulation of "client use" is thrown at the server, how does it behave?

Thorpe created a benchmarking tool named "sslswamp" [6], which simulates the SSL/TLS client(s) so that the speed, capacity, throughput, and behavior of an SSL/TLS server can be tested. Each run of the swamp program lasted for 60 s. The results of the load generator can be seen in Table I. The cipher suite used for generating the results was 3DES in Cipher Block Chaining (CBC) mode, with SHA MAC and RSA as the authentication algorithm.

Fig. 8 shows the comparison of the client connections accepted by the server and those clients that were queued due to the fact that the server was not able to process all the incoming clients. This was done by first increasing the number

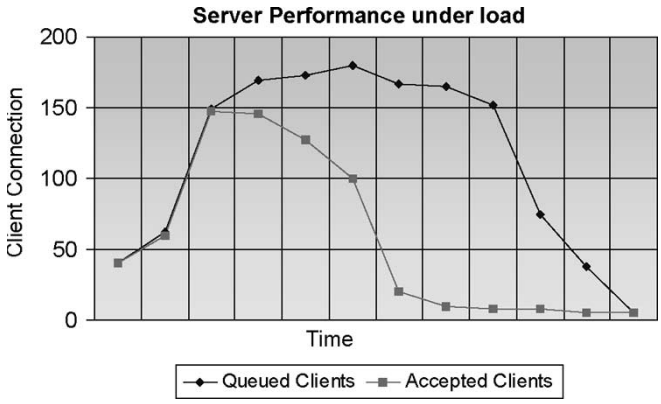

Fig. 8. TLS server performance under load.

of concurrent incoming clients and then reducing the number of incoming clients. The results were generated by saving the values of the number of connecting clients from the load generator and the number of accepted clients in a file. A connecting client is a client that has not yet been authenticated. As the number of concurrent client connections increases, the percentage of failed connections also increases. The connection failures are caused by handshaking errors during the client and server SSL handshaking phase.

Under heavy-load conditions, the cryptographic computations necessary for SSL become a bottleneck to server performance. When the number of client connections is relatively low, the server will serve each request as it comes in. As the number of concurrent client connections increases, the server will have to queue connecting clients while it is still busy handshaking with others (Fig. 8). Thus, the server is being saturated by the SSL handshakes that it is busy performing. The result is that the server is keeping a number of clients in queue. As the number of new connections decreases, the server is able to serve the clients in the queue. The average connection time for a client is between 1 and 2 s. From the results, the optimal number of clients that can connect to the server without being queued is between 90 and 120 clients.

C. Throughput and Response Time of Server

Latency is the time it takes between making a request and seeing a result. Throughput is the number of total transactions that can be maintained over a period of time. The result of the experiment to determine the throughput and response time for the TLS server using the siege load-generator program by Fulmer [7] is given in Table II. The simulation was repeated ten times for a certain number of concurrent client connections. The average HTML page size sent to the client by the web server was 370 B. The average over the ten runs was taken as the final result. Network latency between the TLS server on computer 1 and the web server on computer 2 will have an effect on the results. Running the traceroute command between computer 1 and computer 2 gives the following latency result:

b121pc142.up.AC.za (137.215.121.142)

0.962 ms 0.617 ms 0.156 ms.

TABLE II
THROUGHPUT AND RESPONSE-TIME RESULTS FOR THE SSL SERVER UNDER CERTAIN LOAD CONDITIONS (TCP MAXIMUM SEGMENT SIZE = 4096 B)

Number of concurrent clients	HTTP Throughput (Bytes/sec.)	Availability of server	SSLv3 Throughput (Bytes/sec.)	SSLv3 Response Time (sec.)	Availability of server
1	3765.41	100%	4637.50	0.08	100%
10	4457.44	100%	5523.32	0.08	100%
20	6451.85	100%	6745.45	0.11	100%
30	8975.61	100%	8772.23	0.12	100%
40	12517.01	100%	9222.21	0.12	100%
50	13757.01	100%	9919.79	0.14	100%
60	13886.79	100%	10357.14	0.15	100%
70	14553.67	100%	9988.46	0.17	100%
80	14360.98	100%	10393.26	0.19	100%
90	13301.20	100%	10701.92	0.21	100%
100	12390.57	100%	9003.48	0.20	100%

In the data-transfer phase of an SSL connection, there are two relevant cryptographic operations. These two operations are the record encryption and decryption with a symmetric cipher and the MAC calculations and verification operation. The choice of the MAC and encryption algorithm affects the performance of the data-transfer phase.

Choosing a proper MAC chipper is also very useful, as can be seen from Table III, where RC4 with MD5 is almost eight times faster than 3DES with SHA. The results in Table II are obtained using 3DES–SHA. Table III presents only the record generation cost and not network overhead. Significant server response time can be improved using RC4–MD5.

The throughput of the normal HyperText Transfer Protocol (HTTP) transaction between the client and the web server increases as the number of concurrent connected clients increases. The throughput reaches a maximum when there are between 60 and 80 connected clients. The response time for all the number of concurrent connected clients is thus lower than 0.01 s.

The throughput for the SSL v3 connections reaches a maximum value at around 60 concurrent clients. The throughput is initially higher than for the normal HTTP connections, but as soon as more than 20 concurrent clients connect to the TLS implementation server, the throughput remains lower than the HTTP throughput. The SSLv3 throughput is almost one and a half times the HTTP throughput.

The response time of the SSLv3 connections is much higher than the HTTP connections and increases as the number of concurrent client connections increases. The SSLv3 response times are much higher than the HTTP connections because of the cryptographic computations that the TLS server needs to perform for each client connection.

TABLE III
SSL RECORD PROCESSING SPEED FOR A CERTAIN SYMMETRIC CIPHER AND A CERTAIN MAC ALGORITHM [3]

Cipher-MAC algorithm	Kilo Bytes/second
RC4-MD5	15034
RC4-SHA	10831
3DES-CBC-SHA	2068

Using the RC4 symmetric algorithm along with the MD5, as seen in Table III, the MAC algorithm can decrease the response times. Another possibility is by using dedicated cryptographic accelerators to perform all cryptographic calculations, which will also decrease response time.

VI. SUMMARY

The project achieved the goal of securing the Internet communication for the IGUANA gateway by developing a server TLS program capable of implementing the SSL v3 and TLS v1 protocols. Securing the communication means that the server is capable of implementing the security services of confidentiality, integrity, and authentication for a connection between a client and the server. The confidentiality and integrity were implemented using the OpenSSL toolkit, which provides an interface to the SSL protocol. Authentication was implemented using certificates, on the server side, and smart card, on the client side; a PKI-based procedure that is mentioned in [17] can also be investigated to improve the server performance. A client-authentication protocol was developed to run on top of SSL, taking advantage of all the security features of SSL, including authentication, confidentiality, and integrity. The client-authentication protocol is a challenge–response-type protocol, where a private key stored on a smart card is used to encrypt a random value sent by the server.

The client-authentication protocol effectively doubles the handshaking time needed between a client and a server before both client and server are authenticated to each other, and a secure connection is set up. This has a dramatic impact on performance, as handshaking is the most costly operation of an SSL or TLS connection. The server is capable of handling a moderate client load, but performance will decrease dramatically with an increasing number of client connections. The server-performance issues presented can be dramatically improved with the hardware-based server module, e.g., field-programmable gate array (FPGA), application-specified integrated circuit (ASIC), or DSP, instead of the smart-card-based one, but this will add the additional cost for these hardware-accelerator modules; some of these modules are discussed in [18]–[24].

REFERENCES

[1] J. Leach, "Dynamic authentication for smart cards," *Comput. Secur.*, vol. 14, no. 5, pp. 385–389, 1995.

[2] R. Molva, "Internet security architecture," *Comput. Netw.*, vol. 31, no. 8, pp. 787–804, Apr. 23, 1999.

[3] E. Rescorla, *SSL and TLS: Designing and Building Secure Networks*. Indianapolis, IN: Addison-Wesley, 2001, pp. 175–217.

[4] W. Stallings, *Network Security Essentials: Applications and Standards*. New York: Prentice-Hall, 1999, ch. 2.

[5] P. Urien, "Internet card, a smart card as a true internet node," *Comput. Commun.*, vol. 23, no. 17, pp. 1655–1666, 2000.

[6] G. Thorpe, "Distcache: Distributed session caching tools and APIs, primarily for SSL/TLS servers, and the sslswamp SSL/TLS benchmark/test utility," 2006, [Online]. Available: http://distcache.sourceforge.net/

[7] J. Fulmer, "Siege: a http regression testing and benchmarking utility, that can stress a single or many URL into memory with a user defined number of simulated users simultaneously," 2006, [Online]. Available: www.joedog.org/siege/

[8] M. Kumar, "New remote user authentication scheme using smart cards," *IEEE Trans. Consum. Electron.*, vol. 50, no. 2, pp. 597–600, May 2004.

[9] M.-S. Hwang and L.-H. Li, "A new remote user authentication scheme using smart cards," *IEEE Trans. Consum. Electron.*, vol. 46, no. 1, pp. 28–30, Feb. 2000.

[10] W.-S. Jaung, "Efficient three-party key exchange using smart cards," *IEEE Trans. Consum. Electron.*, vol. 50, no. 2, pp. 619–624, May 2004.

[11] N.-Y. Lee and Y.-C. Chiu, "Improved remote authentication scheme with smart card," *Comput. Stand. Interfaces*, vol. 27, no. 2, pp. 177–180, Jan. 2005.

[12] W.-J. Tsaur, C.-C. Wu, and W.-B. Lee, "A smart card-based remote scheme for password authentication in multi-server Internet service," *Comput. Stand. Interfaces*, vol. 27, no. 1, pp. 39–51, Nov. 2004.

[13] C.-C. Yang and R.-C. Wang, "Cryptanalysis of a user friendly remote authentication scheme with smart cards," *Comput. Secur.*, vol. 23, no. 5, pp. 425–427, Jul. 2004.

[14] W.-S. Juang, "Efficient password authenticated key agreement using smart cards," *Comput. Secur.*, vol. 23, no. 2, pp. 167–173, Mar. 2004.

[15] S.-T. Wu and B.-C. Chieu, "A user friendly remote authentication scheme with smart cards," *Comput. Secur.*, vol. 22, no. 6, pp. 547–550, Sep. 2003.

[16] S. Lancaster, D. C. Yen, and S.-M. Huang, "Public key infrastructure: A micro and macro analysis," *Comput. Stand. Interfaces*, vol. 25, no. 5, pp. 437–446, Sep. 2003.

[17] A. F. Gómez, G. Martínez, and Ó. Cánovas, "New security services based on PKI," *Future Gener. Comput. Syst.*, vol. 19, no. 2, pp. 251–262, Feb. 2003.

[18] G. P. Saggese, L. Romano, N. Mazzocca, and A. Mazzeo, "A tamper resistant hardware accelerator for RSA cryptographic applications," *J. Syst. Archit.*, vol. 50, no. 12, pp. 711–727, Dec. 2004.

[19] L. Batina, S. B. Örs, B. Preneel, and J. Vandewalle, "Hardware architectures for public key cryptography," *Integr. VLSI J.*, vol. 34, no. 1–2, pp. 1–64, May 2003.

[20] H.-Y. Chien and J.-K. Jan, "An integrated user authentication and access control scheme without public key cryptography," in *Proc. IEEE 37th Annu. Int. Carnahan Conf. Security Technology*, Taipei, Taiwan, R.O.C., Oct. 14–16, 2003, pp. 137–143.

[21] J. Goodman and A. P. Chandrakasan, "An energy-efficient reconfigurable public-key cryptography processor," *IEEE J. Solid-State Circuits*, vol. 36, no. 11, pp. 1808–1820, Nov. 2001.

[22] X. Zeng, C. Chen, and Q. Zhang, "A reconfigurable public-key cryptography coprocessor," in *Proc IEEE Asia-Pacific Conf. Advanced System Integrated Circuits*, Fukuoka, Japan, Aug. 4–5, 2004, pp. 172–175.

[23] S. S. Raghuram and C. Chakrabarti, "A programmable processor for cryptography," in *Proc. IEEE Int. Symp. Circuits and Systems (ISCAS)*, Geneva, Switzerland, May 28–31, 2000, vol. 5, pp. 685–688.

[24] A. Royo, J. Moran, and J. C. Lopez, "Design and implementation of a coprocessor for cryptography applications," in *Proc. ED&TC*, Paris, France, Mar. 17–20, 1997, pp. 213–217.

[25] T. Sauter and C. Schwaiger, "Achievement of secure internet access to fieldbus systems," *Microprocess. Microsyst.*, vol. 26, no. 7, pp. 331–339, Sep. 10, 2002.

[26] P. Palensky and T. Sauter, "Security Considerations for FAN-Internet Connections," in *Proc. IEEE International Workshop on Factory Communication Syst.*, Porto, Portugal, Sep. 6–8, 2000, pp. 27–35.

[27] T. Sauter and C. Schwaiger, "A secure architecture for Fieldbus/Internet gateways," in *Proc. 8th IEEE Int. Conf. ETFA*, Oct. 15–18, 2001, vol. 1, pp. 279–286.

[28] J. Nagel, "Congestion control in IP/TCP internetworks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 1, pp. 61–65, Jan. 1995.

Deep Vardhan Bhatt (M'01–A'02–M'04) received the B.Eng. degree in electrical and electronic engineering from Abubakar Tafawa Belewa University (ATBU), Nigeria, in 1992. He is currently working towards the M.Eng. degree in computer engineering at the University of Pretoria, Pretoria, South Africa.

He has been a Lecturer at the Department of Electrical Electronic and Computer Engineering, University of Pretoria, since 2001. In 1995, he joined C. N. Mahlangu Technical College, South Africa, as a Lecturer and Network Administrator. In 1997, he developed and introduced a computer-literacy program for the disadvantaged community. His current research interests are in data security, wireless-network security, and new protocols and algorithms for security.

Stefan Schulze received the B.Eng. degree with honors in computer engineering from the University of Pretoria, Pretoria, South Africa, in 2002 and 2003, respectively.

Currently, he is working in a private firm as Security Consultant.

Gerhard P. Hancke (M'88–SM'00) received the B.Sc., B.Eng., and M.Eng. degrees from the University of Stellenbosch, Stellenbosch, South Africa, and the D.Eng. degree from the University of Pretoria, Pretoria, South Africa, in 1983.

He is currently the Chair of the Computer Engineering Program in the Department of Electrical, Electronic, and Computer Engineering, University of Pretoria. He is responsible for curriculum development and research activities within this program. He partakes extensively in collaborative research programs with research institutions internationally. He is a Professional Engineer and held offices in various national and international scientific and professional bodies.