13

Information Security on the Electronic Superhighway

S H. von Solms
Rand Afrikaans University and IBM South Africa
PO Box 524, Aucklandpark, 2006, Johannesburg, South Africa
Tel : +27 11 489 2843
Fax : +27 11 489 2138
Email : basie@rkw.rau.ac.za

Abstract

This paper discusses a number of security protocols for the Internet and World Wide Web. The first two, SSL and SHTTP, provide general authentication, confidentiality and integrity facilities, while SEPP is a payment protocol, specifically designed for use in electronic purchasing.

Keywords

Security protocols, Internet, World Wide Web, SSL, SHTTP, SEPP

1 INTRODUCTION

Whichever paper you read or person you speak to, there is little doubt that the overwhelming consensus is that any company not planning to utilise the Internet/WWW infrastructure on a corporate level, is going to lose out.

These same papers and people, however, explicitly warns about the information security risks of doing precisely what they suggest!

There are different routes a company can follow to expose itself to the Internet, but in general, there seems to be an identifiable pattern. Usually the following major phases can be identified, not necessarily in the order suggested below :

Phase 1 : Allow employees to retrieve information from the Net and WWW. Clients in the company therefore have access to WWW servers, but the company provides no servers accessible by clients outside the company.

Phase 2 : Start using the Net and WWW to provide information about company products and prices - including confidential information on a selective basis to specific customers. The

S. K. Katsikas et al. (eds.), *Information Systems Security*

company now provide a server facility which can be accessed by clients outside the company. This is often the beginning of Internet use for marketing

Phase 3 : Start using the Net and WWW for electronic purchasing in a 'closed' environment, where the customer must register before the time, and transactions are only performed with registered customers.

Phase 4 : 'Open' electronic purchasing, where transactions can be performed with any customer presenting a credit card number.

This paper is very much tutorial in nature, and intends to address some of the risks involved in these phases, providing some solutions, currently available. By no means can all possible risks and exposures be covered, and we will mainly concentrate on security solutions relevant to Phases 2 and 4, discussing a number of protocols which seem to be significant players in this area.

We will discuss each of the phases separately.

2 PHASE 1

This is basically a 'one-way' traffic, in that information is 'brought into' the company, but no information is provided to outsiders, so no information flows out of the company's systems to the outside world.

Usually at this stage, some type of firewall is installed. Firewalls come in many different flavours, and we will not discuss the concept of firewalls any further, apart from saying that a firewall is only as good as the way it is managed.

3 PHASE 2

At this stage, the environment starts opening up further, in that company data is now being moved from a company server to the outside world. The risks start to increase because outside contamination of the company's resources, as well as the possibility of providing confidential data to unauthorised outsiders, become possible.

In the next 4 sections, we will discuss solutions relevant to Phase 2, as defined above.

At this point in time, the following facilities are starting to become essential :

3.1 The facility to authenticate the parties involved in a transaction, i.e. the user (client) is sure he is talking to a legal system (server), and vica versa.
3.2 The facility to protect and secure data sent between the client and server, i.e. to maintain confidentiality.
3.3 The facility to prevent replay of transactions.
3.4 The facility to ensure that data is not changed during transmission, i.e.to maintain integrity.
3.5 The facility to digitally sign messages, i.e. to enforce non-repudiability.

Now identification and authentication of the parties involved in a transaction become essential, as well as the confidentiality and integrity of the data during transmission.

Solutions will therefore be more complex and difficult, but must always as transparent to the user as possible.

Two widely accepted technologies which are quite useful at this point in time, and provides some of the facilities mentioned above, are Secure Sockets Layer (SSL), (Reference 1) and Secure Hypertext Transport Protocol (SHTTP), (Reference 2).

Because both technologies make extensive use of digital signatures, a brief overview of such signatures are in order. A more detailed discussion can be found in many documents on encryption technology, for e.g. (Pfleeger, 1989).

4 DIGITAL SIGNATURES

In the traditional symmetric approach to encryption, two parties involved in exchanging secure messages, secure such messages by encrypting the content of the message using a shared algorithm and a shared key. This shared key is used for encrypting and decrypting the message. Figure 1 illustrates this.

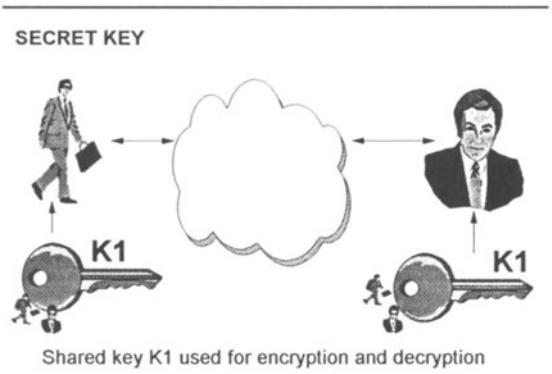

Figure 1 Symmetric encryption

In the asymmetric or public key approach to encryption, each party involved in exchanging secure messages, has a unique pair of related keys. One of the keys in the pair is called the party's public key, and the other his private key. The public key is publicised as wide as possible, for e.g. to everyone. The private key is kept private, and is only known to the specific owner (party). Anything encrypted under the public key of a specific owner, can only be decrypted by the related (corresponding) private key of the key pair, and vica versa. Figure 2 illustrates this.

The result of encrypting a message, say 'Hello', with the private key of a specific person, say P, is called the digital signature of P for message 'Hello'. Figure 3 illustrates this, where '&2%77(' is the digital signature of person P for the message 'Hello'.

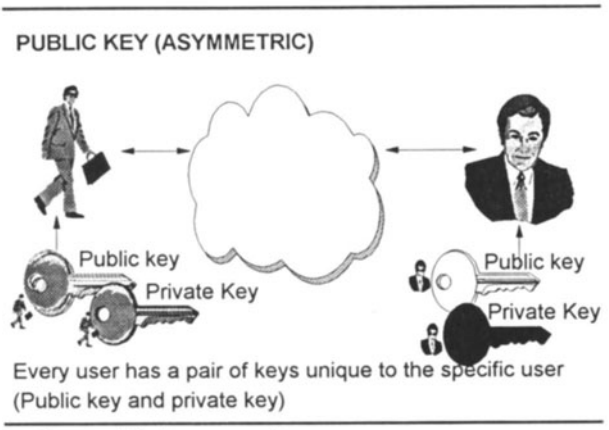

Figure 2 Asymmetric or public key encryption

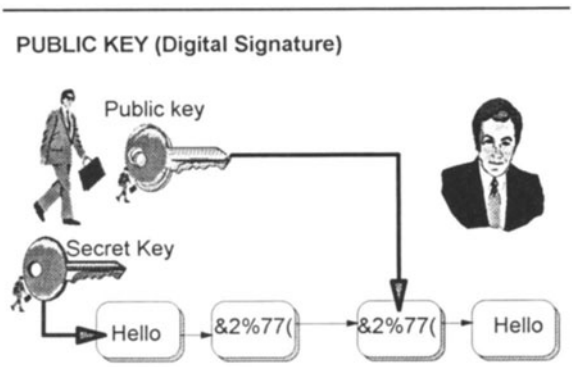

Figure 3 A digital signature

Public/private keys are issued by a trusted Certification Management Authority. Because it is essential to ensure that a specific public key really belongs to a specific person, say P, P also has a certificate, which is an electronic 'vouching statement', provided by the Certification Management Authority who issued the pair of keys to P, that the relevant public key is authentic, and really belongs to P.

5 SECURE SOCKETS LAYER (SSL)

The Secure Sockets Layer (SSL) secures the transmission channel between the client and server handling the transaction. Security is implemented 'below' the application level, and the

application has no control over what services to apply to a specific document. All document (transactions) are secured generically in precisely the same way.

This is referred to as channel-based security, and can provide the services 3.1 to 3.4 above. SSL therefore secures the channel of communication between 2 parties as private and authenticated, and do not 'see' individual documents.

The Secure Sockets Layer (SSL) Protocol is a secure protocol that provides privacy over the Internet. The protocol allows client/server applications to communicate in a way that cannot be eavesdropped. Each server always possesses a private/public key pair, and servers are always authenticated through their secret/public key pairs. Clients can (optionally) possess private/public key pairs, and if they do, they can be authenticated through such key pairs.

SSL requires a reliable transport protocol, like TCP/IP, for data transmission and reception. The SSL Protocol is application protocol independent, meaning that a higher level application protocol, like HTTP, FTP, TELNET, can layer on top of the SSL protocol transparently. This is illustrated in Figure 4 below.

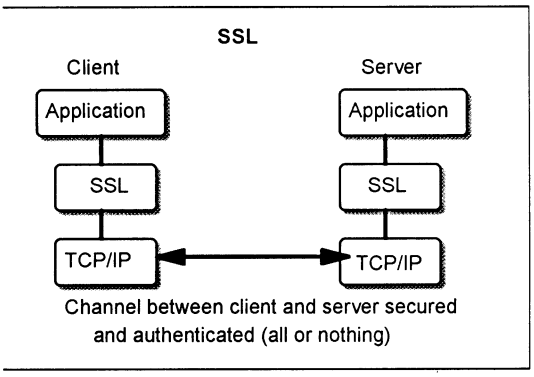

Figure 4 SSL in relation to other levels.

In an intended client/server WWW communication, the SSL Protocol will, through its Handshake (sub)Protocol, let the client authenticate the server, decide on an encryption algorithm supported by both the client and server, and establish a session key for the specific session, before the application protocol sends or receives its first byte of data.

Clients always initiates a connection with a server, by selecting (dereferencing) a Uniform Resource Locator (URL), which is an 'address' specifying the location of a Web page. The client must know whether the server supports SSL or not. This is indicated to the client by the form of the URL. If it starts with **http://**, the client knows the server does not require (or support) SSL, and issues an ordinary Connect() to establish the connection. If the URL starts with **https://**, the client know the server supports and requires the use of SSL, and therefore the client issues a SSL_Connect(). SSL_Connect() establishes an 'ordinary' connection between the 2 parties, and when that is successfully completed, performs the Handshake Protocol described below. When this handshaking is finished, the first application data starts to flow.

Suppose a client dereferences a https:// URL, indicating a secure channel between the client and server is required. The browser issues a SSL_Connect(), and the 2 parties now perform a handshake protocol, in which the server is always authenticated to the client, and optionally the client authenticated to the server. If the handshaking process is successful, a session key for encrypting data during the session, is established between the 2 parties, and the application data, secured by the established session key, can start flowing.

The server is always in possession of a public/secret key pair. The client may be in possession of a public/secret key pair, but not necessarily.

In the discussion below, we just indicate those data which are relevant for our explanation. All message do include other data, which is not directly relevant to our explanation.

The handshaking protocol discussed in the next section, creates a secure 'path' between the client and the server, and in the process ensures that the server is authentic, i.e. that the server is not masquerading as somebody else. Part of the process is also to establish a session key between the client and the server which will be used to encrypt any data exchanged between the two. We consider 3 cases :

- No session had yet between established between the client and the server
- A session had been established, but had been terminated, and are to be re-used. This is primarily done to eliminate unnecessary overhead
- The server wants to authenticate the client during an established session.

5.1 The SSL Handshake Protocol

5.1.1 Creating a session between the client and server:

5.1.1.1 The client send a ***client-hello*** message to the server. This message contains a random **challenge** generated by the client, as well as the **cipher-specs** (encryption facilities) supported by the client.

5.1.1.2 The server responds with a ***server-hello*** message. This message contains a **connection-id**, which is byte string randomly generated by the server, as well as the server's **public key certificate** and the server's **cipher-specs**.

5.1.1.3 The client now generates a masterkey, using a cipher chosen from the server's cipher-spec. The masterkey is generated in 2 parts. One part of the masterkey is encrypted using the server's public key, retrieved from the public key certificate sent by the server. The client sends this encrypted part as well as the clear part of the masterkey to the server, using a ***client-master-key*** message.

From the masterkey, the challenge sent to the server in step 5.1.1.1, as well as the connection-id received from the server in step 5.1.1.2, the client generates 2 private keys to be used for symmetric encryption during the to be established session with the server (the **session keys).** One of these keys is called the **client-write key**, and will be used to encrypt data to be sent to the server. The other key is called the **client-read key**, and will be used to decrypt data received from the server.

The master key as well as the session key are stored in the client's cache.

5.1.1.4 The server receives the 2 parts of the master key, decrypts the encrypted part using its secret key, and forms the master key. Using the same procedure as the client, the server now generates 2 keys, which will be the same as the 2 generated by the client, because the same parameters are used. The one key is called the **server-write key**, which will be the same as the client-read key, and the other the **server-read key**, which will be the same as the client-write key. These 2 keys as well as the master key are cached.

The server now waits for a client-finish message from the client.

5.1.1.5 The client encrypts the connection-id sent by the server in step 5.1.1.2, using its client-write key. It sends this to the server in a ***client-finish*** message.

5.1.1.6 The server now encrypts the challenge sent to it by the client in step 5.1.1.1, using the server-write key, and sends it to the client using a ***server-verify*** message.

5.1.1.7 The client decrypts the received data using the client-read key, and compares the content with the original challenge sent.

Note that if they match, the client can be satisfied that the server is authentic - so the client has authenticated the server.

5.1.1.8 The server now generates a **session-id** for the to be established session, place the session id in its cache, and sent this new session-id, encrypted under the server-write key, to the client using a server-finished message.

5.1.1.9 On receipt the client retrieves the session-id, and stores it in its cache.

Once both client and server had sent a ***finish*** message, the SSL handshake protocol is done, and the application protocol can begin to operate, using the established session key for bulk encryption.

5.1.2. Using an established session between a client and server

In this case, the server had already been verified by the client, but the server wants to re-authenticate the client. It may be that a session, with a session-id and session keys had been established earlier, used for some application, finished, and wants to be reused by the client.

5.1.2.1 The client sends a ***client-hello*** message to the server, containing a challenge, and the previously agreed upon session-id.

5.1.2.2 The server returns a ***server-hello*** message, containing a **connection-id** as described in 5.1.1.2 above, as well as the **session-id-hit** flag, if the session-id provided by the client, is known to the server. The contents is encrypted using the server-write key. If the session-id is not known to the server, we revert back to step 5.1.1.2 above.

5.1.2.3 The client encrypts the connection-id sent by the server in step 5.1.2.2, using its client-write key. It sends this to the server in a ***client-finish*** message.

5.1.2.4 The server now encrypts the challenge sent to it by the client in step 5.1.2.1, using the server-write key, and sends it to the client using a ***server-verify*** message.

5.1.2.5 The client decrypts the received data using the client-read key, and compares the content with the original challenge sent.

Note that if they match, the client can be satisfied that the server is authentic - so the client has (again) authenticated the server.

5.1.2.6 The server encrypts the existing session-id again, using the server-write key, and sends it to the client using a ***server-finish*** message.

Once both client and server had sent a ***finish*** message, the SSL handshake protocol is done, and the application protocol can begin to operate, using the established session key for bulk encryption.

5.2 Using an established session between a client and server and requiring the client to sign a challenge if in possession of a public/private key pair.

At any time during the re-establishment of a session, as described above, the server can request the client to further authenticate itself using the client's secret key, if it is in possession of such a key.

5.2.1 After 5.1.2.3 above, the server can issue a ***request-certficate*** message. This message contains some challenge data, and requests the client to provide its certificate to the server.

5.2.2 The client signs the challenge data received from the server with its private key, and sends the signed version, as well as a copy of its own certificate, to the server, using a ***client-certificate*** message.

5.2.3 The server authenticates the client by decrypting the signed challenge data by using the client's public key retrieved from the client's certificate.

5.3 The SSL Record Protocol

All SSL data, also those during the handshaking process, is sent in terms of SSL **records.** All data records are automatically provided with a sequence number, for synchronisation between the client and the server.

To protect the integrity of data, all records sent are automatically provided with a Message Authentication Code (MAC).

Many WWW browsers are now SSL aware, meaning that WWW traffic can be transparently secured by SSL.

6 SECURE HYPERTEXT TRANSPORT PROTOCOL (SHTTP)

If the WWW is now being introduced as a marketing vehicle, and more control over selective delivery of information is needed, SHTTP is a possible approach to take.

The Secure Hypertext Protocol (SHTTP) can secure individual documents (transactions) in different ways, because security is implemented on the application level. The application therefore has full control over what security services it wants to apply to a specific document. This is referred to as **document-based security**, and can provide the services 3.1 to 3.5 above. SHTTP therefore marks individual documents as private, signed etc.

As we can see from the discussion above, SSL in very much channel-oriented, and only protects the channel. Everything on the channel is protected, and there is no control on application level to decide what to encrypt and what not.

SHTTP addresses much more selective application control than SSL.

The Secure Hypertext Transport Protocol (SHTTP) is an extension of the Hypertext Transport Protocol (HTTP). HTTP is the protocol used for communicating via the WWW, and the SHTTP extension provides extensive security features to secure messages sent via the WWW.

SHTTP makes a very wide range of security mechanisms available to HTTP servers and clients, and even allows clients and servers to negotiate precisely which mechanisms they want to use for secure communication.

The security services provided by SHTTP are:

- confidentiality
- authenticity
- integrity
- non-repudiability
- replay-securing

These services are called **privacy enhancements** in SHTTP.

We will not discuss all these mechanisms in detail, but rather concentrate on a functional and highlevel description of the features of SHTTP.

SHTTP operates on the application level - that means that there is application control over what security features must be used, and what not. (This is opposed to SSL where the security features are implemented 'below' and transparent to the application level.)

Using SHTTP, every HTML page (document, hyperlink) on a SHTTP-aware server, can be handled security-wise, in a different way.

The application designer must therefore, as part of the design and implementation, specify the security options to be implemented for every document (hyperlink).

Figure 5 illustrates the ideas.

SHTTP makes uses of public key encryption. Every server will have at least one public/private key pair. If there are different application owners on the server, the server may have a public/private key pair for each (See DNs below). Although clients may have a public/private key pair, and SHTTP can leverage the possession of such a key pair, client key pairs are not mandatory.

SHTTP is considered to be rather complex - one reason is the many options available within the protocol. We will try to steer away from all these specific options, and rather try to give an overview.

The easiest will be to try to explain the operation of SHTTP is by using a simple example. Let us start with a HTML reference (hyperlink), 'enhanced' with some SHTTP features to secure the referred page .

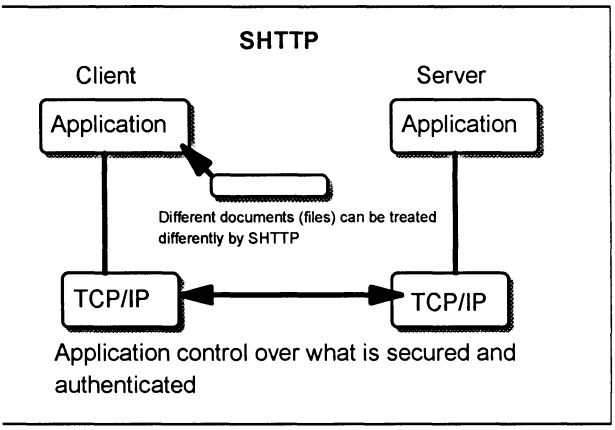

Figure 5 The Secure Hypertext Transport Protocol

6.1 An example of a SHTTP "transaction"

6.1.1 The Request

The client is confronted by a HTML page. On the HTML page several hyperlinks may appear, referencing documents belonging possibly to different owners.

Hyperlinks using SHTTP will use a protocol indication of **shttp://** instead of **http://,** indicating to the client that the server is SHTTP-aware for this specific reference. With every shttp:// hyperlink, two anchor properties are associated :

- The CRYPTOPTS block, containing information about algorithms used, as well as privacy-enhancement information. These privacy-enhancements specify the actions required by the server when the specific hyperlink is dereferenced, i.e. the page referred to by this hyperlink is requested. Suppose in our example, the server requires that the request must be **encrypted.**
- The distinguished name (DN) of the principal on the server requiring these privacy enhancements. There may be different 'document owners' on the server, and each will have its own DN.

The HTML page will also contain the public key certificates of all the DN's associated with shttp:// hyperlinks on that page.

The client now creates an HTTP request (GET) to retrieve the page, and, because of the privacy requirement of **encrypt** specified above, the client generates a DES key, encrypts his request with the DES key, and encrypts the DES key with the relevant DN's public key (found in the HTML page).

Several SHTTP headers are now added to the (encrypted) request, specifying information about the algorithms used by the client, the format of the request etc.

This SHTTP encapsulated message is now sent to the server.

Client actions as far as security enhancements are concerned, are therefore governed by the anchor security-enhancements specified in the anchor properties of the shttp:// hyperlink selected.

6.1.2 Server actions

The server uses the information in the SHTTP headers, and the relevant private key to retrieve the DES key supplied by the client, and using that key, retrieves the request itself. The server now retrieves the document specified in the request.

The security actions of the server, ie what must the server now do to the information to be delivered to the client, depend on whether the hyperlink chosen by the client, specifies a static file, or the execution of a Common Gateway Interface (CGI) program.

Every static file (page), has a local security configuration file ('dot file'), in which the server actions on delivering the results to the client, are specified.

Every CGI program contains a Privacy-Enhancements header in which the actions to be taken by the server on delivering the results to the client are defined.

Suppose the client chose a static page in our example above. The server checks the local configuration file of the page, and if required to encrypt, does so with the session key provided by the client, and again encapsulates the response with the necessary SHTTP headers.

The response is sent to the client.

6.1.3 Client actions

Using the information in the SHTTP headers, and the DES key generated above, the client decrypts the HTML document requested.

Suppose in the HTML page retrieved, there is a hyperlink to another HTML page on the server. The privacy enhancements for this request may now require the client to digitally sign the request before sending it to the server.

The client will now sign the request with its own private key, if it has one, otherwise it is stuck .

Suppose the client can sign it. Using the SHTTP different headers, information about the algorithms used for signature, and the client's public key, is added to the message, and sent off to the server.

In this way every page/request can be treated in a different and individual way, enforcing the security needed for that specific page.

7 PHASE 3

At this stage, the company starts using the Net and WWW for electronic purchasing in a 'closed' environment, where the customer must register before the time, and transactions are only performed with registered customers.

Because payment comes into play in this phase, banking institutions and credit card companies are essential parts of the environment.

Payment in this phase is usually handled by existing procedures.

Potential customers must register with the company, and electronic purchasing is only possible by such pre-registered customers. Such customers signs an agreement with the company for payment by debit order, or by debiting the customer's credit card, the number which is kept on the customer's record at the company.
Identification and authentication of the customer is done via a password shared between the customer and the company.

8 PHASE 4

This stage brings 'open' electronic purchasing, where transactions can be performed with any customer presenting a credit card number.

In this environment, no pre-knowledge about the customer exists at the company, and the company wants to allow this totally unknown customer to do electronic purchasing.
The company, however also wants to be sure that the potential cutomer is the real owner of the presented credit card number, and that the card issuing authority will authorize the transaction .

As the security risks of this environment is much greater than any other, much more sophisticated protocols are needed.

Several such secure payment protocols exist. Two major players in this arena are STT, the Secure Transaction Technology Protocol, (Reference 3), developed in collaboration between Microsoft and Visa. Another is SEPP, the Secure Electronic Payment Protocol, (Reference 4), jointly developed by IBM, Netscape, Mastercard and a number of other cooperators.

We will review SEPP in this section. To make the presentation as simple as possible, the protocol is simplified to a big extent, and only the really important aspects are discussed.

The following players are active in this scenario:

- The Card holder (potential customer)
- The Merchant
- The Acquirer bank
- The Issuer Bank and
- The Certificate Management System.

Figure 6 illustrates the relation between these parties.

We will now look at the flow of messages to complete a transaction, with specific reference to the security aspects.

Assume that the Card holder has completed the Negotiation phase, i.e. he has done his browsing, and has decided, possibly after negotiation with the Merchant, that he want to move to the Purchase/Payment phase.
SEPP address this Purchase/Payment phase.

Message 1
The first message, the ***initiate*** message, is now sent from the Card holder to the Merchant.
This message contains ia a customer-id and the brand-id of the to be presented credit card.

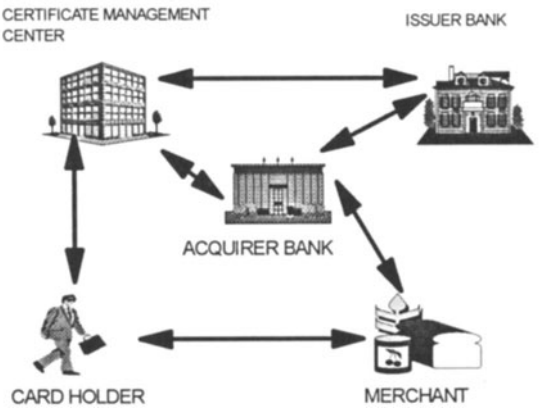

Figure 6 The SEPP Environment

Message 2
The Merchant now creates a message containing information about the goods and price decided on, and digitally signs it with his private key. This commits the Merchant to the agreed goods and price, which he cannot now deny at a later stage.
This is sent back to the Card holder in the ***invoice message.***

Message 3
The Card holder now checks the contents of the ***invoice*** message by retrieving the contents using the Merchant's public key.

If successful, this authenticates the Merchant, and commits the Merchant to the specific goods and price.

The Card holder now creates the ***purchase_order_request*** message, consisting of

1. information about the agreed goods and price, his credit card number and expiry date, all encrypted with the public key of the Acquirer. Let us call this M1. (This means that the Merchant cannot retrieve the credit card number and expiry date, and cannot change the information about the goods and price.)
2. the goods and price decided on, digitally signed by the Card holder with his private key. (This means that the Card holder cannot later deny placing the order, for the agreed goods at the agreed price.)

The ***purchase_order_request*** message is sent to the Merchant.

Message 4
The Merchant now creates the ***auth_request*** message, which is sent to the Acquirer to authorize the transaction.

This message contains :

1. M1 as described above

2. Information about the agreed goods and price, digitally signed by the Merchant. Let us call this M2.

The Acquirer now retrieves the contents of M1, and determines what the Card holder claims he is buying, and at what price.

The Acquirer also retrieves the contents of M2, and determines what the Merchant claims he is selling, and at what price.

If needed, the Acquirer now requests an authorisation from the Issuer.

The Acquirer receives an authorisation code back.

Message 5

The Acquirer now creates the ***auth_response*** message consisting of the authorisation code, digitally signed by the Acquirer.

This message is sent to the Merchant.

Message 6

The Merchant now sends the ***purchase_order_response*** message back to the Card holder. This message consists of the authorisation code, digitally signed by the Acquirer.

At this stage the transaction is basically completed.

Many other messages are specified to acquire certificates, check certificates etc., but the 6 messages above describe the situation when on-line authorisation is done.

9 SUMMARY

In this paper we tried to give a high level overview of some important security protocols for the Internet and WWW. By no means are these the only, or necessarily the most important. They are however all good protocols, enjoying wide support at the moment.

10 REFERENCES

Phleeger (1989) Security in Computing, Prentice Hall

Reference 1 http://www.netscape.com/info/SSL.html

Reference 2 http://www.commerce.net/information/standards/drafts/shttp.txt

Reference 3 http://www.visa.com/visa-stt/stt_home.html

Reference 4 http://mastercard.com/Sepp/sepptoc.htm

11 BIOGRAPHY

Prof Basie von Solms is Head of the Department of Computer Science at the Rand Afrikaans University in Johannesburg, South Africa. He is also the South African representative on Technical Committee 11 [Information Security] (TC 11) of the International Federation for Information Processing (IFIP), and is the present Chairman of TC 11.

Prof von Solms has published numerous research papers on Information Security, and had spent 1995 on a 12 month industry sabbatical at IBM Development Laboratory at Hursley in the UK. He is presently also a consultant on Information Security to IBM South Africa.

He is also a member of the Review Panel of the journal Computers and Security, as well as a member of the Editorial Board of the South African Computer Journal.