8

Security Concepts for the WWW

Peter Lipp, Vesna Hassler
Graz University of Technology
Institute for Applied Information Processing and Communications
Klosterwiesgasse 32/I, 8010 Graz, Austria
e-mail: plipp@iaik.tu-graz.ac.at

Abstract

This paper gives an overview of the existing World Wide Web security concepts. For each proposal we give a short description of the most important properties and discuss them briefly. The concepts are divided into three groups according to their relative position to the HTTP layer. A comparison of some of the properties of various concepts is given as well.

Keywords

World Wide Web, Security

1 INTRODUCTION

The most rapid growth of the Internet is coming from the use of the World Wide Web (WWW). The WWW was originally designed for complete openness rather than security. There is, for example, no need for any log-in. Nevertheless, it has become clear that several forms of security are required. It is sometimes necessary to restrict access to specific WWW pages, or to protect confidentiality of the data in transfer between two Web parties. It can also be desirable to ensure non-repudiability of Web pages. Security requirements of Web commercial transactions can include customer/supplier authentication, transaction confidentiality and transaction non-repudiability.

The protocol used for transferring data between Web-Servers and Web-Clients is HTTP. HTTP has a dual nature: on the one hand, it has a defined syntax on how to transfer messages, and, on the other hand, it is used also as a pure data transfer protocol, like telnet or SMTP. It would therefore appear to need different types of protection. For the message exchange a *message oriented* protection is needed, and for the data transfer a *session oriented* channel protection.

It is possible to add security at different layers: above HTTP, below HTTP or at the HTTP level. In the following sections we are going to analyze the briefly different approaches, and present existing solutions or proposals for adding security to the WWW.

P. Horster (ed.), *Communications and Multimedia Security II*

2 INTEGRATING SECURITY ABOVE HTTP

Above HTTP we can protect documents or other data by employing the Common Gateway Interface (CGI, [9]), in which case HTTP is an oblivious transport protocol that sends the protected objects in "clear" messages. One example of this is NCSA's CCI-based PGP solution [10]. The protected messages are handled by helper applications, which perform all the operations relevant to security. The main *advantage* is that this solution is immediately deployable by providing the necessary helper applications. The main *disadvantage* is that no real authentication of the server process is possible. The solution cannot replace application layer security and cannot make use of HTTP session properties like session keys or nonces.

2.1 CCI

The Common Client Interface (CCI) is a client-side API designed to expand the functions of browsers without having to alter the code of the browsers themselves. The CCI is a two-way communication protocol with functions that drive the browser.

In [10] the authors describe several ways of using the CCI for Web message oriented security. In their approach the Web browser communicates with an external CCI application which handles the processing of data that is digitally signed or encrypted (Fig.1). They define a PGP-CCI protocol to be used to protect HTTP message exchanges, and describe a scheme for PGP-protected form submissions which requires only a slight modification of Web servers.

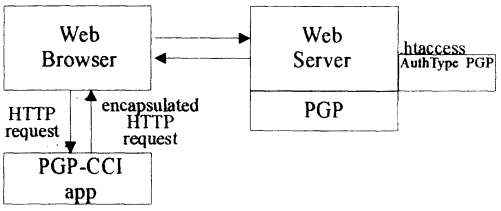

Figure 1 PGP- CCI app

The "PGP-CCI app" is a GUI-based front end to PGP [12] which communicates with the Web browser via CCI. PGP-CCI is a protocol which can be used to allow the signing and encryption of HTTP messages. As with SHTTP, PGP-CCI uses HTTP extensions, new HTML anchor attributes and advanced CCI features. The "PGP-CCI app" also functions as a viewer for documents that have been encrypted or signed by PGP.

Hyperlinks pointing to documents to be retrieved using PGP-CCI contain special anchor attributes giving the server's public Key ID, as well as other optional information (e.g. whether the HTTP request should be signed, encrypted or both). When the users clicks on such a link, the browser passes the complete HTTP request to the CCI app together with the anchor attributes for the desired hyperlink. The CCI app uses PGP to apply the proper enhancements to the request, which may involve encrypting the message using the Web server's public key. Before signing or encrypting the request, the CCI app adds a Date header and a header with a randomly generated session key. The client's HTTP request must be encrypted in order to

securely transfer the session key. After being processed by PGP, the request is encapsulated within a generic HTTP request which is then given to the browser.

The server uses PGP to decrypt or verify the signature of the encapsulated request. If the client is not authorized to access the requested document, the server responds with "401:Unauthorized". If everything goes well, the server prepares a HTTP response to be sent to the client, previously processed by PGP. On the client side, the CCI app uses PGP to decrypt or verify the signature of the server's response. It passes the plaintext to the browser as a temporary local file. The PGP-CCI app displays the relevant information to the user, including the server's PGP username and Key ID.

To use the PGP-CCI scheme, the support must be compiled in the server. The server's PGP related information is specified in configuration files. By using, for example, ".htaccess" the server may be configured to require PGP-CCI authentication/encryption for accessing protected Web documents. The security issues for the PGP-CCI app fall into two categories:

User interface of the PGP-CCI app itself It is important that the PGP-CCI app displays to the user information relevant to security, and that the processes of signing or decrypting data with the users private key should be performed explicitly by the user. Otherwise an adversary could try to force the PGP-CCI app to inadvertently sign or decrypt attacker-chosen data using the client's private key.

Interactions between the PGP-CCI app, the browser and the server The main weakness of this scheme is the handling of "401:Unauthorized" server responses, because an attacker could intercept a PGP-enhanced request and return a 401 message with the attacker's Key ID in the re-submission anchor.

3 INTEGRATING SECURITY BELOW HTTP

Below HTTP we can set up a secure channel between the Web server and client and thus protect the network connections. In this case, HTTP is like an application which is not aware of the underlying security mechanisms. The server and/or the client are authenticated, and the connection between them is then protected by computing the MAC, or by establishing a session key that is used to encrypt the data in transfer. This is done transparently to the applications. The most prominent examples are the SSL, PCT and GSS-API-approach, as well as IPv6, the next generation Internet Protocol which will provide this functionality for all applications. *Advantages* of this approach are that the solutions are easily deployable and widely applicable, also for other widely used protocols like ftp or telnet. The methods involved are well known and deploy an easily verifiable set of cryptographic algorithms. The main *disadvantage* is that these protocols know nothing about messages in the data stream, and thus cannot handle them individually. For example, it is not possible to authenticate a particular transmitted message, and the security policy chosen cannot be switched between messages. This also makes caching of encrypted messages impossible.

3.1 The Secure Socket Layer Protocol

SSL was developed at the Netscape Corporation and implemented in their products. There is also a public domain implementation by Eric A. Young [11]. The last version of the SSL protocol is the Version 3, currently specified in an Internet Draft [3].

SSL's main design goal is to prevent eavesdropping, tampering and message forgery. It therefore provides privacy, authenticity and integrity of an association between two communicating applications. Privacy is achieved using symmetric cryptography. For authentication, both symmetric and asymmetric cryptographic algorithms may be used. Integrity (originally called "reliability", which in our opinion may be confusing) means message integrity ensured by applying a keyed Message Authentication Check (MAC). Three authentication modes are supported: that of both parties, of the server only, and total anonymity. Anonymous sessions are vulnerable to a man-in-the-middle attack.

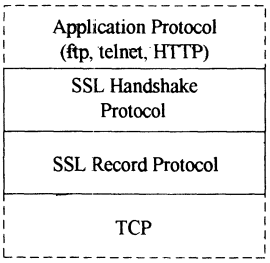

Figure 2 SSL protocol structure

SSL consists of two main protocol layers (Fig.2). SSL Record Protocol lies on top of a reliable transport protocol (e.g. TCP). Its task is to encapsulate higher level protocols. SSL Handshake protocol is the part that provides security, and is responsible for server/client authentication and negotiation of the key exchange algorithm, encryption algorithm, cryptographic keys' parameters and MAC algorithm. Higher level protocols can be layered transparently on top of SSL protocol.

SSL takes messages to be transmitted, fragments the data into manageable blocks, optionally compresses the data, applies a MAC, encrypts, and transmits the result. Received messages are decrypted, verified, decompressed, reassembled and then delivered to higher level clients (to e.g. SSL Handshake).

SSL Handshake protocol consists of messages that enable negotiation of cryptographic parameters, peer authentication and generation of shared secrets between a client and server at the beginning of their communication (Fig.3). ClientHello and ServerHello messages serve the negotiation purposes. In ClientHello the client first proposes a set of combinations called CipherSuites containing a key exchange algorithm (RSA, Fortezza or Diffie-Hellman), a data encryption algorithm (RC2, RC4, IDEA, DES, 3DES) with certain parameters and a MAC algorithm (MD5, SHA). In other words, it is only possible to negotiate the combinations of these parameters, and not the individual parameters. Certificates are generally an X.509 type certificates. The compression method is also negotiable, and is applied optionally prior to

encryption. The messages CertificateVerify and Finished have a similar purpose. They both ensure integrity of the previously exchanged messages. ServerKeyExchange is used only in cases when the server has no appropriate certificate.

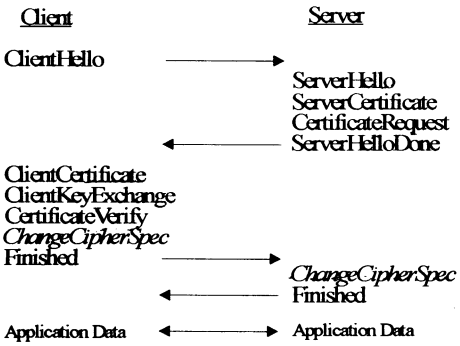

Figure 3 SSL Handshake

From the cryptographic point of view, the client and server exchange two independent random numbers in ClientHello and ServerHello messages that are sent in the clear. The security of the session key depends on the secrecy of the so-called pre_master_secret randomly generated by the client and sent to the server encrypted with the server's public key in ClientExchangeKey. Both random values together with the pre_master_secret are used to compute the master_secret that is in turn needed for the computation of encryption keys and MAC secrets.

ChangeCipherSpec is not a part of SSL Handshake and presents a message which is used for notifying the receiving party that the sending party will begin to apply the just-negotiated security parameters starting with the next data record.

When resuming a previously created session or duplicating an existing session to avoid negotiation, a session_id field in ClientHello contains the number of the session, so that its security parameters can be applied immediately. This could be considered a security hole, because an adversary only needs to obtain the session key whose cryptographic strength is in fact independent of the authentication performed at the session creation time. For the adversary it is a significant advantage if the session key is weaker than the authentication.

3.2 The Private Communications Technology Protocol

The PCT protocol was developed at the Microsoft Corporation [2]. The main design goal of the protocol was to prevent eavesdropping on communication lines in client/server applications. The authentication of the server is mandatory, while clients are authenticated at the server's option.

The PCT Protocol is very similar to SSL, but improves and corrects some of SSLv2's weaknesses. The main differences between the two protocols are in the handshake phase. According to PCT authors, the most important improvement is the elimination of a security hole in SSL's client authentication. As SSL's client authentication is independent of the cipher strength used in the session, and of whether the authentication is being performed for a

modifying the standard HTTP/HTML constructs. For the integration, a new secure HTTP URL type is defined, namely "gss-http". This URL type indicates that the associated Web data must be accessed/submitted using GSS-API security services. In Fig.4 a typical GSS-API secured Web client/server interaction is illustrated.

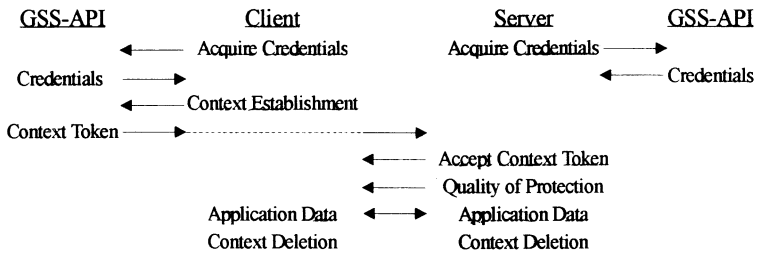

Figure 4 GSS-API secured Web communication

Acquiring credentials
Before a security context between a Web client and server can be established, each of them must acquire their respective credentials (e.g. Kerberos tickets, private/public key pairs and public key certificates).

Security context establishment. Accepting a security context
The security context establishment process authenticates the client and server identities and establishes a secret key for protecting the subsequent Web transaction. The Web client initiates the security context, and the Web server accepts the security context. The mechanism negotiation process is used to determine whether mutual authentication is required. If mechanism negotiation is not supported, mutual authentication is used by default.

A simple GSS-API mechanism negotiation based on a two-way negotiation model is proposed in [1]. However, it does not provide any security features to protect the initially exchanged values for security context parameters, which makes a denial-of-service attack, as well as a man-in-the-middle-attack possible. In SSL the integrity of negotiation is protected by computing a keyed hash in the *Finished* message (see Section 3.1), and in PCT by the VERIFY-PRELUDE mechanism (see Section 3.2).

Message protection
After a security context is established, the Web client and server can exchange information in a secure manner using various GSS-API functions. Transactions can be secured by a Message Integrity Check (MIC) alone, or by a MIC with data encryption. Immediately after a security context is established, the server sends the client a secured message in which the level and type of transaction security required by the server is determined (Quality of Protection). That is, the server indicates to the client whether integrity alone or integrity with encryption is required, and which integrity/encryption algorithm must be used, as well as key size. In SSL and PCT these parameters are negotiable.

reconnection of an old session or for a new one, a man-in-the-middle attack is possible. An attacker who has obtained the key for a cryptographically weak session can authenticate himself in a cryptographically strong session. This is presumably done for performance reasons in SSLv3, because, on the one hand, a new session with newly negotiated security parameters can always be created, and, on the other hand, it is sometimes convenient to quickly resume an old session if the security of the old session key is not considered to be threatened. An important security improvement in PCT compared to SSLv2 is the protection of the security negotiation's integrity. In SSLv3 it is improved even further, so that it provides for the integrity of all handshake messages. The number of handshake messages to be exchanged was greater in SSLv2 than in PCT, but in SSLv3 it has been improved to equal the current version of PCT. SSLv3 also introduces optional message compression prior to encryption which may decrease the size of packets to be transmitted over the network.

PCT is asymmetric between client and server. The client authenticates the server because only the server can decrypt the MASTER_KEY which is encrypted under the server's public key. Additionally, the server's challenge response depends on knowing the MASTER_KEY. The server authenticates the client because the client signs its challenge response with its public key. PCT can be distinguished from SSL, as in the PCT protocol there is a challenge response mechanism in the handshake protocol even when resuming an old session. In other words, to resume an old session, the client and server must know the MASTER_KEY to be able to compute new session keys based on the new client and server challenge. All other cryptographic keys are derived from the MASTER_KEY, i.e. the client's and server's encryption and MAC keys.

PCT uses a mechanism called VERIFY-PRELUDE to provide for the integrity of the CLIENT_HELLO and SERVER_HELLO messages that are sent in the clear. SSLv3 provides in addition for the integrity of *all* handshake messages.

PCT permits RSA (PKCS#1), Diffie-Hellman (PKCS#3) and Fortezza as key exchange algorithms. If the key exchange is token-based, the cipher is specified together with the key exchange algorithm. Otherwise, the encryption algorithm may be DES with several key lengths, IDEA, RC2 and RC4. Possible hash functions are MD5, SHA and DES-based Davies-Meyer hash algorithm. Allowed certificate types are X.509 and PCKS#7. Signature types may be RSA with MD5 or SHA, and DSA with SHA.

3.3 GSS-API

The Generic Security Service Application Program Interface has been developed by the IETF Common Authentication Technology WG for securing client/server applications. GSS-API [8] provides authentication, integrity, and/or confidentiality security services to callers, typically communication protocols. [1] defines GSS-API services and primitives at a level independent of underlying mechanisms and the programming language environment. Parameter bindings for particular language environments as well as token formats, protocols, and procedures are to be specified by other related documents.

HTTP/GSS-API integration occurs at the network communication level. The initial HTTP connection is strongly authenticated, and the entire transaction is protected using GSS-API functions. All types of HTTP transactions are secured, including document retrievals, form submissions and CGI results. Web transactions are protected (i.e. encapsulated) without

Context deletion
After a Web client/server transaction is completed, or an error is detected, the security context is deleted by both the client and server.

The GSS-API functions do not actually perform any network communication between the Web client and server. A GSS-API function may return a token, sent by the client to the server, or vice-versa, which then passes the token into another GSS-API function for processing.

4 INTEGRATING SECURITY AT THE LEVEL OF HTTP

At the HTTP level we can protect HTTP message units and thus change HTTP into a security-aware protocol. Each message or element of the protocol can then be protected individually, with any combination of encryption, authentication and signature. Proposals include SHTTP and SEA. The *advantage* of such approaches is that all HTTP properties can be protected, including resource names or return status codes. The methods also use cryptographically well known algorithms. The protected messages are separable and separately archivable. SHTTP's *disadvantage* ist that it requires major changes in the HTTP agent code. SEA is still in the development phase.

4.1 SHTTP

The goal behind designing SHTTP was to *provide a flexible protocol that supports multiple orthogonal operation modes, key management mechanisms, trust models, cryptographic algorithms and encapsulation formats through option negotiation between parties for each transaction* [7].

Thus, SHTTP does not fix the users to certain protocols or mechanisms, but provides a security framework for servers and clients, leaving the details of the cryptographic options to the users. Option negotiation is used to allow clients and servers to agree on transaction modes, cryptographic algorithms and certificate selection. There is no minimal set of algorithms or specific message formats defined which every SHTTP-compliant software has to implement. This might result in incompatible sets of servers and clients, although this seems rather improbable. It is recommended though, that every implementation understands the PEM-message format.

SHTTP supports end-to-end secured transactions which can be initiated symmetrically, so that servers and clients are treated equally with respect to their preferences. Message protection may be provided by signing, authenticating or encrypting the message, or by applying any combination of these (including no protection).

Signature
For signatures, RSA and NIST-DSS are defined as possible algorithms. To be able to verify a signature, a certificate can be appended to the message. If this is not done, the recipient is expected to retrieve the necessary certificate(s) independently.

Key Exchange and Encryption
For bulk encryption, symmetric algorithms are used. The necessary keying material can be exchanged in different ways:

- RSA: the symmetric key is passed encrypted using the receiver's public key.
- Outband: a prearranged session key is used. The information necessary to identify the key is transmitted in the headers.
- Inband: by assigning names to keys, new keys can be transferred encrypted in SHTTP messages.
- Kerberos: keys are extracted from Kerberos-tickets.

The algorithms DES, IDEA, RC2 and IBM's CDMF in various modes and with various key lengths may be used for encryption.

Message Integrity and Sender Authentication
To ensure message integrity and sender authenticity for an HTTP message, a Message Authentication Code (MAC) is computed, using a keyed hash over the document and a shared secret. The arrangement of the shared secret can be performed manually, using Kerberos or by other means. No cryptography is needed.

Of course, this does not provide non-repudiability, but this service is not always needed. It can, for example, be used for mutual identification in a transaction. This is optional, because signatures are only to be used if the user explicitly and consciously decides to do so.

Freshness
SHTTP provides a simple challenge-response mechanism, allowing both parties to ensure the freshness of transmissions. If a S-HTTP message contains a nonce-value, the other party has to include that nonce-value in the answer.

Preenhanced Documents
To achieve better performance and security, documents can be pre-signed and pre-encrypted.

4.2 PEP and SEA

The World-Wide-Web-Consortium is currently working on a draft for Web security which is called *SEA: A Security Extension Architecture for HTTP/1.x* [5]. and is based on the Protocol Extension Protocol (PEP [4]).

PEP
HTTP messages can be extended with additional header fields. However, different extensions of HTTP need to coexist, so that a guideline for HTTP agents is needed, helping them to decide how to react to a certain header (e.g. whether to ignore it, in what order to act on headers, etc.).

Protocol extensions can be used to specify any associated header lines and also to provide guidance on each of the above decisions. PEP is an extension protocol for HTTP that captures such information about protocol extensions (hence 'PEP': Protocol Extension Protocol).

PEP can be used by HTTP-agents to negotiate a set of common protocol extensions to be used, and also to find out about the details of unknown protocol extensions, which can principally be connected without changing the original protocol. An PEP-compliant agent should thus be able to bootstrap itself, starting with no knowledge of protocol extensions and

learning about the necessary extension while communicating with other agents. This will not be possible for security extensions like SEA.

SEA

The starting point in the design of SEA were the ideas behind S-HTTP which lead to a modular design based on the principles of PEP. SEA defines three different protocol extension classes: Signature, Encryption and Key-Exchange.

The details of SEA are based on the specifications from January 1996, which will no doubt be changed drastically, so we will describe only the basic principles.

Using PEP, an agent specifies in an HTTP-message the order in which the protocol extensions used have to be applied. Simplified, this can be seen as an ordering of black box modules, with the original message as input and the message to transmit as output, or the received message as input and the message to display as output. In the case of SEA, this would look as follows:

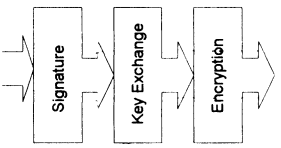

Figure 5 SEA modules

SEA Signature

The SEA signature is an operation on the entire body of the HTTP-message, ensuring its authenticity and integrity, but not implying any authorship or agreement on the contents. RSA-MD5 or DSS-SHA are possible algorithms. Signatures are transmitted in the header part. Therefore non-SEA-compliant agents are also able to read the message, ignoring unknown headers.

SEA Key-Exchange

In the first design of SEA, Key Exchange was placed after encryption only, and the task of this stage was to transport the session key from the sender to the recipient. This is not always a possible scenario. E.g. if Diffie-Hellman type key exchange is to be applied, the Key Exchange phase has to generate a session key, which can the be used at the encryption stage.

Implementations might need to combine the Key-Exchange module and the encryption module into one binary, as the information flow between these stages is critical for the overall security.

SEA Encryption

Encryption is a process that protects the message body. Because of the header collisions, a message cannot be encrypted twice. Taking the initial set of encryption algorithms from SHTTP's selection is being considered.

SEA Recipes

PEP/SEA leaves the implementers and users with a wide range of choices. However, it would be possible for them not to choose the best alternatives or combinations possible. E.g. they could sign an encrypted document, which is not recommended. To avoid these problems,

recipes will be made available, that can be seen as recommendations, for example saying which algorithms and algorithm combinations should be used. Examples of recipes could be:

- It is hardly ever appropriate to sign an encrypted text.
- Only cleartext can be encrypted, and this only once

5 CONCLUSIONS

Security issues in the Web are rapidly gaining in importance. The reasons for this are the need to protect Web client/server hosts, to authenticate Web clients, servers and users, to restrict access to various Web resources, and to protect integrity, authenticity and confidentiality of Web data. Security concepts can generally be divided into three main groups, depending on their placement relative to HTTP. The concepts above HTTP (e.g. PGP-CCI app [10]) are aimed at securing a particular content (document or database record) and use HTTP as an oblivious transport only. It is therefore difficult to authenticate communications parties or to secure HTTP operations. The concepts at the HTTP layer (SHTTP [7], SEA [5]) can protect each HTTP protocol message using encapsulation, but require rather monolithic implementation. Below HTTP are solutions that secure communications channels and that can be used by any higher level application (e.g. SSL[3], PCT[2], GSS-API[8]). Their main disadvantage is that they cannot distinguish messages, and therefore can set a security policy per-channel only. Which of these concepts to deploy depends mainly on a minimal set of security requirements. The most flexible concept from the security point of view is certainly SHTTP, but at the same time this requires the most complex implementation. On the commercial side, only SSL and PCT are currently in widespread use, as they have been launched by two of the major players in the field, i.e. Netscape and Microsoft. Which, if any, of the possible alternatives is successful, will depend to a large extent on market forces.

6 REFERENCES

[1]Baize, E. and D. Pinkas, Simple GSS-API Negotiation Mechanism ,<draft-ietf-cat-snego-01.txt>, February 1996

[2]Benaloh, J. et. al., Private Communication Technology Protocol , URL:<http://www.microsoft.com/windows/ie/pct.htm>, September 1995

[3]Freier, A.O., P. Karlton and P.C. Kocher, SSL Version 3.0, <draft-freier-ssl-version3-00.txt>, December 1995

[4]Khare, R., PEP: An Extension Mechanism for HTTP/1.1, W3C Working Draft, February 1996, URL:<http://www.w3.org/pub/WWW/TR/WD-http-pep.html>

[5]Khare, R., SEA: A Security Extension Architecture for HTTP/1.x, World-Wide-Web-Consortium Working Draft, , URL:<http://www.w3.org/pub/WWW/TR/WD-http-sea>, January 1996

[6]Kolletzki, S., Privacy Enhanced Mail for WWW, Proceedings of the Third International World Wide Web Conference, April 1995.

[7]Rescorla, E. and A. Schiffman, The Secure HyperText Transfer Protocol , <draft-ietf-wts-shttp-01.txt>, February 1996

[8]Rosenthal, D., Use of the GSS-API for the Web Security , <draft-ietf-wts-gssapi-00.txt>, November 1995

[9]Thompson, D., Common Client Interface Protocol Specification, URL:<http://yahoo.ncsa.uiuc.edu/mosaic/cci.spec.html>, April 1995

[10]Weeks, J., A. Cain and B. Sanderson, CCI-Based Web Security: A Design Using PGP, URL:<http://sdg.ncsa.uiuc.edu/~jweeks/www4/paper/current_rev.html>

[11]Young, E.A. and T.J. Hudson, SSLeay and SSLapps FAQ, URL:<http://130.102.32.1/~ftp/Crypto>

[12]Zimmerman, P., The Official PGP User's Guide, MIT Press, Cambridge Massachussets, 1995

7 BIOGRAPHY

Peter LIPP received his M.Sc. and Ph.D. in Computer Science (*Technische Mathematik*) from Graz University of Technology (TU-Graz), Austria, in 1981 and 1989, respectively. Since 1982 he has been assistant researcher at TU-Graz. His main interest is network security, especially WWW Security. He is currently responsible for the WWW Security Work Package of the ICE-TEL project whose aim is to set up a public-key certification infrastrucure for Europe.

Vesna HASSLER (née Ristic) received a B.Sc. and M.Sc. in Electrical Engineering from Zagreb University, Croatia, in 1988 and 1991, respectively. From 1989-1992 she was assistant researcher at the Faculty of Electrical Engineering in Zagreb. Since 1992 she has been assistant researcher at Graz University of Technology. In December 1995 she received a Ph.D. in Computer Engineering and Communications (*Telematik*) from TU-Graz. Her research interests include network security, WWW security, cryptography and secure applications.