User Privacy Issues Regarding Certificates and the TLS Protocol: The Design and Implementation of the SPSL Protocol

Pino Persiano Ivan Visconti

Dipartimento di Informatica ed Applicazioni
Università di Salerno
via S. Allende, 84081 Baronissi (SA), Italy

Abstract

The aim of this paper is two-fold.

1) We raise concerns regarding possible violations of user privacy relative to the use of X509 Certificates and the Transport Layer Security protocol. We stress that this approach to secure network transactions, while preserving the interests of service providers, neglects to consider the right to privacy of the users.

2) We propose the concept of a *crypto certificate* and the Secure and *Private* Socket Layer protocol (SPSL protocol, in short) and show their effectiveness in preserving user privacy and, at the same time, protecting the interests of service providers.

Focusing on the particular case of web transactions, we describe a system based on SPSL for secure and private web navigation. Our implementation includes an SPSL-proxy for an SSL-enabled web client and a module for the Apache web server along with administrative tools for the server side. The system has been developed starting from the implementation of an API for the SPSL protocol that we describe in the paper. Experimental results show that SPSL is an effective and efficient solution to the problem of privacy in web transaction.

The protocol we propose and, consequently, the implementation we describe are *fully dynamic* and provide an *adjustable* level of privacy.

1 Introduction

Our work deals with possible violations of user privacy relative to the use of X509 Certificates and the Transport Layer Security protocol [8, 10]. We show that these standards in security technology, while preserving the rights of service providers to screen users, neglect to consider the right to privacy of the users.

As a possible countermeasure we propose the concept of a *crypto certificate* and the *Secure and Private Socket Layer* protocol. Our aim is to provide a framework in which users can hide their identity and thus achieve anonymity while at the same service providers can be assured that only qualified users gain access to their services.

1.1 Background

A fair amount of research has traditionally centered around anonymity starting with early works by Chaum [7]. The need for anonymity has been made more pressing with the widespread of the Internet. It is well-known that Internet provides very minimal security and privacy and it is a trivial task to monitor and record many of the electronic actions taken by a user. This information can be used for targeted advertising or more intrusive purposes. The solutions to the problem of anonymity provided so far have mainly concentrated on the problem of preserving the privacy in the context of electronic mail and web navigation. For electronic mail, this is achieved either through mix networks [7, 11] or by means on anonymous remailers. Several solutions have also been announced for web navigation. CROWDS [13] for example is a system that makes it very difficult for a web server to traceback the requester of a resource. The basic idea is that a user should blend into a crowd before accessing the web. The web server upon receiving the request knows only that the request originated from a member of the crowd. The solution proposed by CROWDS is not appropriate for a setting in which users access a restricted service. For example, it might be well the case that certain services offered by a web server are restricted only to subscribers in which case the server wants to make sure that the request comes from a subscriber.

This is a paradigmatical example of two conflicting interests: the provider of the service wants to restrict access only to "qualified" users while users seek to protect their identity.

The Secure Socket Layer (SSL in short) proposed by Netscape Communication [10] and its evolution, the Transport Layer Security (TSL in short [8]), used in conjunction with X509 certificates [12] promise to become the standard tool for implementing such an access control. SSL and TLS allow a service manager to securely identify users requesting access to the service and grant or deny access according to a pre-existing access control list. Typically such an access control list consists of a list of qualified users that will be granted access to the service. The certificate carries the name of the user and after verifying that the certificate is valid and has not been revoked, the manager can check the name on the certificate against a list of qualified users. If the certificate corresponds to a qualified user then the client gives a proof that he/she knows the private key corresponding to the public key of the certificate presented (this is usually achieved by exhibiting a digital signature of the hash of the transcript of the conversation up to this point). After that, a secure (i.e., encrypted and authenticated) channel is established between client and server. This approach guarantees that no unqualified user can ever gain access to the service unless he has access to the private key of a certificate corresponding to an authorized user.

2 Contributions of the paper

Let us now discuss the contributions of our research.

2.1 Identifying Risks

In this paper we identify three potential threats to user privacy.

1. The SSL/TLS protocol requires that all certificates travel in clear over the network. This follows from the consideration that the certificate by itself (i.e., without the corresponding

private key) cannot be used for impersonating the user. However, we point out that an eavesdropper can intercept all communication reaching a given service and build a list of users that access the service and compute for each of them the frequency of access.

2. A second and more serious threat comes from the server himself. This is best explained by considering the following simple example. Users pay a monthly fee to access data from a medical publication database. The database contains links to information regarding various diseases. The database manager implements access control using X509 certificates and SSL/TLS and the list of qualified users consists of users that have paid the fee for the current month. Each user is required to present his/her certificate in order to gain access to the database, and thus the manager can discriminate between users that have paid the monthly fee and users that have not. However, presenting the certificate allows the service manager to collect information on who has read articles about which disease. This information is not essential to the role of the service manager which consists in making sure that only qualified users access the database.

3. Roughly speaking, an X509 certificate consists of a public key, a list of attributes (called *extensions*) regarding the owner of the public key (for example, his Common Name, his Organization, his Organization Unit, et cetera) and the public key itself (for example, the Key Usage extension defines the purpose of the key) along with a signature of the whole computed by the issuing authority.

   It is expected that new extensions will be introduced as the use of certificates spreads. Netscape Communication has already proposed some extensions to offer specific services to users of Netscape Navigator (for example, access to the policy regulating the issuing of the certificate at hand). Another example comes from the health care domain. In some situations people afflicted by a chronic disease benefit of exemption for some specific prescriptions related to their condition. So it is conceivable to have an extension in the certificate of the person carrying this information.

   However, we believe that, although convenient, having too much information on the certificate is dangerous for the user's privacy. A person afflicted by diabetes uses his certificate to buy drugs from the web (in which case the extension specifying his condition is relevant) and to access the company web server (in which case the extension is not only irrelevant but giving this information might delay or endanger a promotion). In general, each time the certificate is exhibited the whole set of information about the individual is given away, even though, some of them is irrelevant, and it is a very simple task to log this information into a database. This is exactly the case for personal id that carry, besides first and last name, also the address, date and place of birth and current job. Showing the id reveals all the information that, in principle, could be logged.

   Notice that, as it is now, it is not possible to reveal just some of the extensions of the certificate (i.e., only the extensions relevant to the transaction being performed) as the certificate is signed as a whole by the issuing authority and so the signature can only be verified if the whole content of the certificate is disclosed.

2.2 Proposing Countermeasures

In this paper we present the *Secure and Private Socket Layer* protocol (SPSL protocol, in short). The SPSL protocol is an extension of the SSL/TLS protocol that allows the server to present a list of certificates corresponding to qualified users and the client to prove that he *"knows"* the private key associated with at least one of those certificates without revealing which one. This protects the right of the manager to give access to his service only to those who have paid for it and at the same time protects the right of the client to preserve his/her privacy. The manager can still collect statistics about which piece of information has been accessed but cannot link this to a name.

The mechanism we propose for anonymous identification is *fully dynamic* in the sense that users can be easily added and deleted from the qualified set of users without affecting other users.

2.3 Implementing a private navigation system

The features of the new protocol can be added to existing SSL/TLS applications with very little effort. The resulting applications can transparently interoperate with SSL/TLS and SPSL applications. The new features offered by SPSL interoperate with SSL/TLS client and server as in our implementation we have used cipher suites codes reserved for experimental algorithms (those with the first byte equal to 0xFF, see Appendix A.6 of [10]). Building on this we provide a browser-server framework for SPSL. More precisely, we provide:

- Server side. We provide a module for the HTTP server Apache [1] for the SPSL protocol. The module is based on the widely used ModSSL [2] module. The SPSL Apache module allows a web site administrator to specify access restrictions to the resources. Access to a protected resource is granted to a qualified user if he/she proves that he/she owns one of the qualified certificates (SPSL Protocol). The web site administrator builds the access lists associated with the restricted resources of the site and he publicizes them as a resource with mime type `application/x-x509-mask`. Apache servers supporting the SPSL protocol are fully compatible with SSL/TLS compliant browsers. Details are presented in Section 4.7 and in Section 5.1.
- Client side. An SPSL proxy application for Netscape Navigator has been designed and implemented. This allows to use the SPSL protocol from a standard browser. The communication between the proxy and the browser uses SSL/TLS. In order to let the proxy gain access to the user certificate, we have developed a library called MOZ2I [3] that reads the browser's local key database and extracts the appropriate private key. The SPSL proxy supports the mime type `application/x-x509-mask` to exchange the set of certificates corresponding to the users qualified to access a resource. This allows to cache the certificates thus dispensing with the need of exchanging the set of qualified certificates each time the resource is requested. Details are presented in Section 5.2.

Our proposal for private web navigation is *adjustable* in the sense that it is possible for the client to choose the degree of anonymity sought and decrease accordingly the computational effort required. In very large scale settings (qualified sets consisting of a few thousands users) the cost of performing the protocol might be prohibitive for small clients. On the other hand, in some situations, hiding the identity in a set of a few hundreds guarantees an adequate

level of privacy. For these cases, it is possible, even in presence of very large sets of qualified users, to scale down the protocol to a few hundreds. This ability has a dramatic effect on the performance of the system as shown by the experiments conducted (see Section 5.3).

All the software developed and the documentation regarding configuration options are publicly available at [5]. In Section 5, we give the skeleton of the client and server of an application that employs SPSL.

In Section 5.1 we present the new directives for the SPSL-compliant web server Apache. In Section 5.2 we present the features of the SPSL proxy application.

2.4 Selective disclosure of certificate extensions

We propose a modification of the structure of the certificates so that partial disclosure of information is possible. Our proposed modification still allows to verify the integrity of the certificate but gives the user the possibility of disclosing only some of the extensions. At the same time, it is not possible for the user to modify the value of the extensions shown.

For example, going back to the example above, when buying drugs the user discloses its health-care related extension whereas when accessing the company web he discloses the extensions related to his job. In both cases, the other party is sure that the extensions disclosed have not been forged and gets no information about the undisclosed extensions.

2.5 Related work

Our proposed protocol is based on the concept of anonymous group identification. A full and general solution to the problem of anonymous group identification has been given in [14], as an application of some structural results on the class of languages having perfect zero-knowledge proofs.

Specifically, the results in [14] give perfect zero-knowledge proofs of knowledge for any set of witnesses associates to a satisfying assignment of any monotone formula over random self-reducible languages (i.e., quadratic residuosity modulo composite integers, graph isomorphism, membership to a subgroup modulo a prime, decision Diffie-Hellman problem).

Group Signature [6] is another concept related to anonymity. Any member of a group can sign on behalf of the group and the group manager is the only entity that can reveal the identity of a signer. Unfortunately, no efficient group signature scheme supporting deletion and addition of new member is known: removing one user involves redistributing keys to all the users. In contrast, changing the set of qualified users in SPSL is trivial and does not require re-distribution of pairs of public and private keys. Moreover using Group Signature requires to place trust into the service manager which is exactly what we wish to avoid.

Recently, Schecter *et al.* in [15] have proposed a scheme for anonymous authentication in dynamics groups. Here the server sends the same message m encrypted using each of a set of l keys. The client proves possession of the private key corresponding to one of the l keys by returning the original message m to the server. We observe though that, even if not specified by the protocol, once the message m has been obtained using the private key in his possession and before sending m back to the server, the client has to verify that the same message m has been encrypted by using the l keys. If this verification step is skipped, the server might

pick a different message for each of the l keys and infer the identity of the client from the message received. As it will be clear in the following, the protocol in [15] imposes the same computation burden on the client as our proposed protocols. However, the protocol is not adjustable: the client needs to perform l encryptions and there is no immediate way to reduce the effort required according to the level of privacy sought.

Moreover, the protocol of [15] suffers of one major drawback that might seriously limit its applicability. Consider a malicious adversary that initiates a huge number of fake connections. The server then has to produce l encryptions of the same message for each of the connections initiated thus affecting its ability to serve real connections. In other words, the protocol proposed in [15] is subject to denial of service attacks as without performing any computation the client forces the server to compute l encryptions. As we will discuss in Section 3, our protocol does not suffer of the same problem.

3 The basic anonymous identification protocol

In this section we present the anonymous identification protocol which is at the base of SPSL. For sake of concreteness, we consider the public keys to be RSA keys. We point out that the same protocol can be used if some other public-key cryptosystem is considered.

Consider the following two-party game: a manager M has a list of RSA public keys: $(N_1, e_1), \cdots, (N_l, e_l)$ and a user U wants to convince M that he knows the secret key corresponding to one of the public keys without revealing which one he knows. The manager sends a random message m to the user asking for the signature of l messages (one for each public key) such that the XOR of the l messages is equal to the original message m. In the SPSL, the message m is not picked by the manager but is instead a hashed version of the transcript of the messages exchanged so far. A detailed description of the protocol is found in Figure 3.

Next we discuss the properties of the protocol. By inspection it is easy to see that if the user is given the private key corresponding to one of the l public keys then verification step M.2 is always successful. On the other hand we observe that if a user which does not know any of the private keys has negligible probability of succeeding in computing the signature of l messages whose XOR gives the message m is extremely slow. Indeed if the user tries to sign random messages by choosing signatures s_j at random the probability that the XOR of the l messages is about 2^{-n} (n is the key length). We also remark that using standard techniques it is possible to modify the above protocol so that any user U that succeeds in passing the manager's verification step can be used to forge RSA signatures.

Let us argue that the manager does not learn which key is known to the user. Consider two users U_1 and U_2 and suppose U_1 knows key i_1 and U_2 knows key i_2 and for each message m let us look at the distributions $D_1(m)$ and $D_2(m)$ induced by U_1 and U_2 on the message sent to the manager. It is easy to prove that distributions $D_1(m)$ and $D_2(m)$ coincide and thus the protocol is witness indistinguishable [9]. Using a standard technique and one extra round of communication the protocol can be turned into a perfect zero-knowledge protocol.

Finally, consider an attack in which a malicious user engages in the protocol only to make the manager waste computing cycles. Since the manager stops as he receives the first incorrect signature, the potential attacker has to perform essentially the same amount of computation

The basic anonymous identification protocol

Common Input: l RSA public keys: $(N_1, e_1), \cdots, (N_l, e_l)$ of length n.
U's Private Input: (i, d_i) such that $d_i \cdot e_i \equiv 1 \pmod{\phi(N_i)}$.

Instructions for Manager and User:

M.1 Pick a random message $m \in \{0,1\}^n$.
Send m to U.
U.1 For each $1 \leq j \leq l$ and $j \neq i$
U.1.1 Pick a random *signature* $s_j \in \{0,1\}^n$.
U.1.2 Compute $m_j \equiv s_j^{e_j} \pmod{N_j}$.
U.2 Compute $m_i = m \oplus m_1 \oplus \cdots m_{i-1} \oplus m_{i+1} \oplus \cdots m_l$.
Compute $s_i \equiv m_i^{d_i} \pmod{N_i}$.
Send $(m_1, \cdots, m_l, s_1, \cdots, s_l)$ to M.
M.2 Verify that $m_1 \oplus m_2 \oplus \cdots \oplus m_{l-1} \oplus m_l = m$.
For $i = 1, \cdots, l$ if $s_i^{e_i} \not\equiv m_i$ then ABORT

Figure 1: The basic anonymous identification protocol

that will be required to the manager. This makes the attack ineffective.

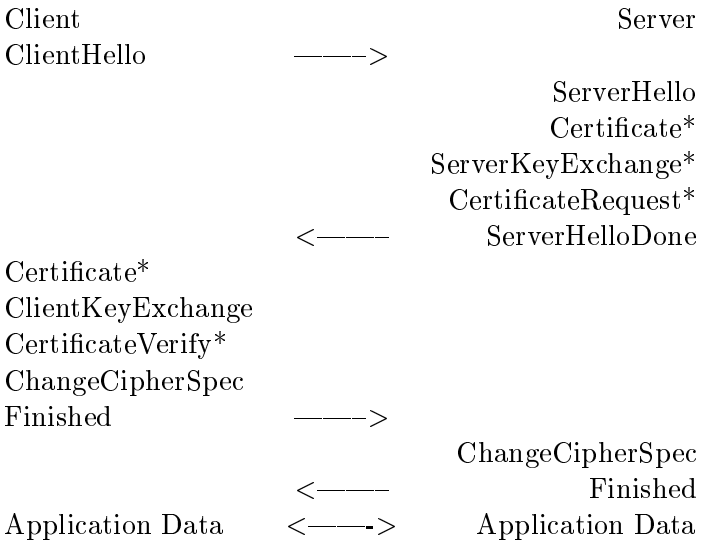
4 The SPSL protocol

In this section we describe the SPSL protocol. We use the same syntax used in specifying the TLS protocol. SPSL contains TLS as a special case (corresponding to the NULL level of anonymity) and in the description we only focus on the newly added features.

4.1 Background on TLS

The TLS protocol (derived from the SSL protocol proposed by Netscape corporation, see [10, 8]) is the de-facto standard for secure communication over the Internet. It guarantees secure message exchange, authentication and secure identification. The SSL/TLS protocol consists of different protocols. Our work is concerned with the Handshake Protocol which we now briefly review. During the Handshake the two parties negotiate the *cipher suite* to be used for the connection. The cipher suite is a 4-tuple specifying the algorithms to be used to exchange the master key, to perform identification, to encrypt application data and to compute the MAC of the data. The negotiation is performed by having the client send a *ClientHello* message to which the server must respond with a *ServerHello* message. The *ClientHello* and *ServerHello* establish the following attributes: Protocol Version, Session ID, Cipher Suite, and Compression Method. Additionally, two random values are generated and exchanged: ClientHello.random and ServerHello.random. Following the hello messages, the server sends his certificate. If required by the server by means of a *CertificateRequest* message, the client exhibits an X509 certificate. We assume that the client certificate has signing ability (e.g., it includes an RSA public key). The server checks the validity of the certificate (i.e.,

verifies that the signature of the issuing authority is correct) and then asks the client to "prove knowledge" of the private key associated with the public key of the certificate. This step is necessary as certificates are public and just holding a certificate does not prove your identity. At this point, the client and server could engage in a zero-knowledge proof of knowledge, but for efficiency reasons, the "proof of knowledge" is replaced by a signature of a hashed version of the transcript of the conversation so far. The rationale behind this choice is that it is very unlikely that a client not possessing the private key could succeed in exhibiting such a signature and that, since the message being signed is somewhat random, the server does not get any information about the private key of the client. At this point, a *ChangeCipherSpec* message is sent by the client followed by the *Finished* message under the newly established cipher suite. In response, the server sends his own *ChangeCipherSpec* message followed by the *Finished* message under the new cipher suite. At this point, the handshake is complete and the client and server may begin to exchange application layer data. (See Figure 2).



(*) Optional message.

Figure 2: The SSL/TLS Handshake Protocol

4.2 The SPSL protocol overview

SPSL differs from SSL/TLS exactly in the phase in which the client proves knowledge of the private key associated with the certificate he has exhibited. Instead we have one of the two parties (in the simplest case the server) exhibit a list of l certificates (we call such a list the *mask* as it masks the actual certificate held by client), and then the client proves knowledge of at least one private key that corresponds to one of the l public keys. This is done using the protocol presented in Section 3. The mask thus consists of certificate belonging to qualified users and is usually provided by the server and different services or resources might be associated with different masks. As an example, consider an HTTP server. The various resources (be it a

simple HTML file or a service in the form of a CGI application) managed by the server could be associated with different sets of qualified users. The client issues the actual command to get the resource as part of the application-level data only after the handshake protocol has been completed. To avoid having to repeat the handshake protocol, it is possible for the server to publicize the mask associated with the various resources he manages (see Section 4.7). Thus the client provide the server with the mask associated with the resource he is interested in during the handshake protocol. Once the request is actually performed by the client, the server verifies whether the client has used an appropriate mask (i.e., a mask consisting of a subset of the set of certificates associated with the resource) for otherwise another proof of identity is requested.

4.3 The SPSL cipher suite

The SPSL cipher suite is a 5-tuple consisting of the following five components:

1. an algorithm to establish the ciphering key;
2. an algorithm for the authentication;
3. an algorithm to cipher the application data;
4. an algorithm to perform the message digest;
5. the anonymity level.

For example, the cipher suite RSA-RSA-RC4-SHA-STRONG uses RSA for authentication and key exchange, RC4 to encrypt application data, the SHA algorithm as message digest, and specifies the STRONG (see below) level of security.

The following anonymity levels have been introduced:

NULL This is the anonymity level of SSL/TLS; the cipher suites with anonymity level NULL can be used for compatibility with the SSL/TLS protocol.

WEAK When this anonymity level has been selected, the server may ask for a client certificate in which case the client sends his certificate encrypted with the server's public key. The server's certificate must have an encrypting key. WEAK anonymity addresses the first privacy concern discussed in Section 2 that considers an eavesdropper intercepting the traffic toward the server.

MEDIUM A set of certificates each with an encrypting key (or an identifier of a set of certificates, see later for more details) is proposed by either the client or server. The client then proves knowledge of the private key associated to one of a set of certificates using the protocol of Section 3. MEDIUM anonymity addresses the second privacy concern discussed in Section 2 and protects the user from the service provider.

STRONG This is similar to the MEDIUM level with the exception that if the set of certificates (or its id) is proposed by the client then it is sent encrypted using the server's public key.

4.4 The structure of SPSL messages

In this section we specify the structure of the messages introduced by SPSL and how the SSL/TLS message types are to be modified. The CertificateRequest is used by the server to request message and it has the following structure:

```
struct {
  select (PrivacyLevel)
    case NULL:
      ClientCertificateType certificate_type⟨1..2^8-1⟩
      DistinguishedName certificate_authorities⟨3..2^16 − 1⟩
    case WEAK:
      ClientCertType cert_type⟨1..2^8 − 1⟩
      DistinguishedName cert_auth⟨3..2^16 − 1⟩
    case MEDIUM:
      opaque ID[4];
      ASN.1Cert ca_list ⟨0..2^24 − 1⟩
      ASN.1Cert mask_list ⟨0..2^24 − 1⟩
    case STRONG:
      opaque ID[4];
      ASN.1Cert ca_list ⟨0..2^24 − 1⟩
      ASN.1Cert mask_list ⟨0..2^24 − 1⟩
} CertificateRequest;
```

Here, *ID* is an identifier for a list of certificates, *mask_list* is a list of Certificate and ASN.1Cert is a sequence of bytes (see Section 5.6.2 of [10, 8]). The CertificateList message is used in the MEDIUM case and it has the following structure:

```
struct {
  opaque ID[4];
  uint16 subset⟨0..2^16 − 1⟩
  ASN.1Cert ca_list⟨0..2^24 − 1⟩
  ASN.1Cert mask_list⟨0..2^24 − 1⟩
} CertificateList
```

Here, *subset* is an ordered sequence of 2-byte integers, each specifying the position of a certificate in the list *ID*. When *subset* is not an empty sequence the proof of identity is based only on the certificates specified in this sequence. This feature of SPSL allows to obtain better performance without comprimising the privacy (see Section 5.3). The EncryptedCertificateList is used when STRONG anonymity is sought and it has the following structure:
public-key-encrypted CertificateList *EncryptedCertificateList*. The CertificateVerify message has the following structure:

```
struct {
  select (PrivacyLevel)
    case NULL:
```

```
    Signature: signature;
  case WEAK:
    Signature: signature;
  case MEDIUM:
    opaque proof_of_identity⟨0..2^24 − 1⟩;
  case STRONG:
    opaque proof_of_identity⟨0..2^24 − 1⟩;
} CertificateVerify;
```

In the description above *proof_of_identity* is the output of the procedure described in Section 3 on the same string that is signed in SSL/TLS (see Section 5.6.8 of [10, 8]).

4.5 The SPSL handshake protocol

The ClientHello and the ServerHello messages specify the anonymity level that parties have negotiated. The anonymity level guides the next messages of the handshake. There are 4 anonymity level so we examine a scenario for each anonymity level. For the NULL anonymity level the handshake messages are the same as those exchanged in the SSL/TLS protocol (see Figure 2). let us consider the WEAK anonymity level.

WEAK

Client		Server
ClientHello	—->	
	<—–	ServerHello
		Certificate
		ServerKeyExchange(*)
		CertificateRequest(*)
	<—–	ServerHelloDone
EncryptedCertificate(*)		
ClientKeyExchange		
CertificateVerify(*)		
ChangeCipherSpec		
Finished	—–>	
		ChangeCipherSpec
	<—–	Finished
Application Data	<—>	Application Data

(*) Optional message.

The client must send the EncryptedCertificate message and the CertificateVerify message only if the server has sent a CertificateRequest message.

Let us consider the MEDIUM anonymity level.

	Client		Server
	ClientHello	—–>	
		<—–	ServerHello
			Certificate
			ServerKeyExchange(*)
			CertificateRequest(*)
		<—–	ServerHelloDone
MEDIUM	CertificateList(*)		
	ClientKeyExchange		
	CertificateVerify(*)		
	ChangeCipherSpec		
	Finished	—–>	
			ChangeCipherSpec
		<—–	Finished
	Application Data	<—>	Application Data

(*) Optional message.

The CertificateRequest message contains a list of certificates with an encrypting public key or an id for such a list. The list can be empty. The client must send the CertificateList message only if the server has sent a CertificateRequest message with an empty list. The CertificateList message contains a list of certificates with an encrypting public key or an identifier of such a list. The certificates are presented in a random order so that no information is given about which private key is known by the client. The message can specifies a subset of the list. The message is sent in plain format. The client must send the CertificateVerify message only if it has received a CertificateRequest message.

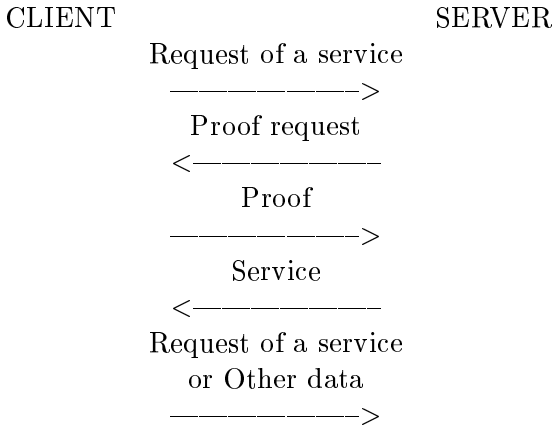
Let us consider the STRONG anonymity level.

	STRONG	
Client		Server
ClientHello	—–>	
	<—–	ServerHello
		Certificate
		ServerKeyExchange(*)
		CertificateRequest(*)
	<—–	ServerHelloDone
EncryptedCertList(*)		
ClientKeyExchange		
CertificateVerify(*)		
ChangeCipherSpec		
Finished	—–>	
		ChangeCipherSpec
	<—–	Finished
Application Data	<—>	Application Data

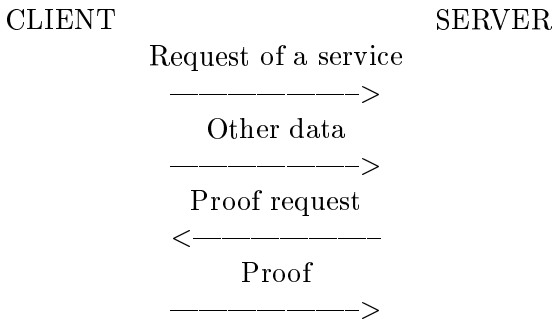
(*) Optional message.
The CertRequest message contains a list of certificates with an encrypting public key or an identifier for such a list. The list can be empty. The EncryptedCertList message must be sent only if the server has sent a CertificateRequest message with an empty list; the message contains a list of certificates or an identifier for such a list. The client knows the private key of at least one of them. The certificates are presented in a random order so that no information is given about which private key is known by the client. This message can specify only a subset of the list of certificates. The client must encrypt the message with the server's public key. The client send the CertificateVerify message only if it has received a CertificateRequest message.

4.6 Overflowing the server

When the handshake is finished parties exchanges application data; the typical scenario is the following:



Look at the following case:



If the server sends a proof request it waits for a Proof sent by the client but on the link there can be other data. In some protocols (see HTTP/0.9) this is not a problem because for each request the client waits for a reply. A server could store in a buffer any message different from the Proof message; this trival solution is dangerous because if a malicious client sends only application data the server wastes many resources. A possible solution is the following: if the server choose a Medium or a Strong cipher suite than it can send another two fields in the

SERVER-HELLO message:
opaque *buffer*[32]
opaque *iter*[2]

When the handshake is finished, the client must send a LIVE message every *buffer* byte of application data that it sends to the server. When the server receive a LIVE message, it must send a LIVE message. The client can send application data to the server only if (num(LIVE sent)-num(LIVE received))<*iter*.

4.7 Mask ID

One of the major communication bottlenecks of the SPSL protocol is the exchange of masks. It would be possible for the client to request from the server an access list of the site. This is simple a list of sets of certificates each with its own identifier and the list of resources each with the id of the relative qualified set. This list can be made public by the server and read by the client just once (we propose to exchange such list using the mime type `application/x-x509-mask`). Each time a resource is requested there is no need to exchange the mask if an identifier for it has been sent. We observe that in this context it is highly advisable to use crypto certificates (see Section 6) instead of plain certificates. In fact, since the access-list of a site is public, the use of plain certificates would be give away information on who is interested (e.g., has paid the fee) for which resources. An HTTP message with content-type `application/x-x509-mask` has the following format:

```
ID: AAAAAAAA
DIR: namehost11:port11/path11
DIR: namehost12:port12/path12

....
....
DIR: namehost1n:port1n/path1n
-----BEGIN CERTIFICATE-----
...
(a certificate encoded in PEM format)
...
-----END CERTIFICATE-----
.....
.....
-----BEGIN CERTIFICATE-----
...
(a certificate encoded in PEM format)

...
-----END CERTIFICATE-----
```

The value of ID is the identifier of the list of certificates (in te example it is AAAAAAAA), the sequence of the values of DIR are the rotected resources and the sequence of certificates

in PEM format are relative to qualified users. A SPSL client application stores in its cache this information and when a protected resource is requested it can efficiently authenticate the user sending the corresponding identifier of the list in the CertificateList (or EncryptedCertificateList) message and sending a proof based on the correspondig set of certificaes in the ClientVerify message. A qualifed user could download all the access lists with the WEAK anonymity level of the SPSL protocol and then it could use the STRONG anonymity level of the SPSL protocol to request reserved resources. If the access list of the site changes, the server could publicize only a patch for the previous list.

4.8 Reducing the computation time

Every time that the client must send a proof to the server the value of the message m may differ. By pre-computing off-line different values for s_j and m_j for each $i \neq j$ the client can dramatically reducing the time needed to perform the protocol as it has only to compute one encryption.

5 Implementation and Experimental Validation

In this section, we present the API for SPSL giving the skeleton of the client and server of an application that employs SPSL. Every data structure is defined in the OpenSSL [4] library.

In Section 5.1 we present the new directives for the SPSL-compliant web server Apache. In Section 5.2 we present the features of the SPSL proxy application.

5.1 SPSL web server

Building Apache with the patched ModSSL module the HTTP web server Apache supports the SPSL protocol and the web server administrator can use the new features adding some directives to the httpd.conf configuration file. The following new directives have been implemented:

- SPSLAccessFile Syntax: SPSLAccessFile *path-file:ID*
  This directive specifies the path (*path-file*) of the file that holds the certificates corresponding to qualified users; the client must give a proof of identity based on this set of certificates. The file that holds the certificates must be in PEM format. The *:ID* part of this directive is optional and specifies eventually an identifier (*ID*) of eight hexadecimal characters for the set.
- SPSLAccessPath Syntax: SPSLAccessPath path-dir
  This directive specifies the path to a directory that holds groups of certificates corresponding to different access groups.

5.2 SPSL proxy

The SPSL proxy application allows to use the SPSL protocol from the Netscape Navigator browser. The proxy always requests a certificate in the SSL/TLS handshake with the browser and then it extracts the corresponding private key from the Netscape Navigator database. The proxy uses the private key to authenticate the client in an SPSL connection. In the

```
SSLeay_add_ssl_algorithms();
// initialize the supported algorithms
ctx=SSL_CTX_new(SSLv3_client_method());
// create a secure context
ssl=SSL_new(ctx);
// create a free and secure connetion
SSL_use_certificate_file(ssl, filename, type);
// certificate to be used is in file filename
SSL_use_PrivateKey_file(ssl, filename, type);
// private key of the certificate in use is in filename
st=SSL_load_mask_file(filename);
// load the mask certificates
SSL_set_mask_list(ssl, st, "03BB8457", "ms_pth");
// set the mask for the connection
SSL_set_cipher_list(ssl, "sPRVC");
// set the acceptable cipher suites
sock=socket(....);
// open a socket
SSL_set_fd(ssl,sock);
// assign a file descriptor
connect(sock,....);
// try to establish the connection
SSL_connect(ssl);
// do a secure and private connect
SSL_write(ssl, buf_out, len_out);
// do a secure write
SSL_read(ssl, buf_in, len_in);
// do a secure read
SSL_shutdown(ssl);
// close a secure connection
SSL_free(ssl);
// free memory
SSL_CTX_free(ctx);
```

Figure 3: The skeleton of an SPSL client application.

command line the directory with the Navigator database can be specified. The proxy requires the Netscape password used to cipher the database.

5.2.1 How it works

When the proxy receives from the web server a message with mime-type `application/x-x509-mask` it stores the body of the message in its cache indexing the identifier and the locations. Then it notifies the browser that the cache has been updated. When the proxy receives from the browser an HTTP-header, it searches its cache for an entry corresponding to the location (or a prefix of it). If a match occurs the proxy starts an SPSL connection using the qualified certificates to mask the client identity. In the other cases, the communication between the proxy and the web server is done following the SSL protocol.

5.3 Performances

The SPSL web navigation system performances have been measured by running several times the web server Apache with our SPSL module, the SPSL proxy and the Netscape Navigator browser. The experimental results gathered show that SPSL is an effective and efficient solution to the problem of privacy in web transaction. In Table 1 the results of the experiments are presented; the different cases are based on the number of the certificates used to mask the client identity (100, 200, 1.000, 10.000, 100 and 500 on a set of 10.000) and whether an identifier has been used to specify the set (the YES row) or the actual set of certificates has been sent (the NO row). Each cell reports the average over several runs of the number of seconds elapsed between the request of a page by the browser and the reply of the web server. The experiments have been performed on Pentium III-based PCs with 128 Mb of RAM, running Linux 2.2.10. The server and the client are connected through Fast 100Mb Ethernet LAN. Only the more interesting case with the STRONG level anonymity has been considered. The server sends a CERTIFICATE-REQUEST message with an empty list of certificates and a NULL id, and the client sends the qualified certificate list or the corresponding identifier in the EncryptedCertificateList. In a previous session the SPSL proxy has stored in its database the access lists for the resources of the site. As expected, the overhead of the protocol grows with the size of the qualified set (see the first 4 columns in Table 1). However, we observe that it might be sufficient for a user to hide his identity in a set of a few hundreds of users. Therefore, in case in which the set of qualified users is large (in the order of several thousands), the user can choose a random subset of the set of qualified users and give a proof only relatively to the selected subset (see field *subset* of message CertificateList in Section 4.4). Column 5 and 6 report the slowdown incurred into in the case in which the user picks a random set of 100 or 500 certificates out of 10000. The slowdown reduces dramatically and shows that SPSL is suitable even for large scale settings.

	100	200	1000	10000	100(10000)	500(10000)
YES	1.1	2.4	8.6	51.4	2.9	5.4
NO	2.3	3.1	12.4	62.8	4.1	7.3

Table 1: Experimental results

6 Crypto certificates for the selective disclosure of certificate extensions

In this section we briefly discuss the concept of a *crypto certificate* that makes possible to disclose only some of the extensions of a certificate. In [12] it is specified that the issuing authority signs the entire certificate using its own private key. The values of the extensions are provided by the user himself and there might be different policies as to how the veridicity of the data provided by the user is ensured. Instead we propose that the issuing authority upon receiving a signing request, first encrypts the value of each extension with its own encrypting key, and then signs the *crypto request* obtaining the *crypto certificate*.

We notice that the crypto requests can also be computed by the requesting user (the key of

the issuing authority is publicly available) In this case the user must also provide the issuing authority with the values of the extensions for which the authority has to verify the veridicity. When requested, the user will exhibit the crypto certificate along with the value of the relevant extensions. The verification of the crypto certificate consists then of two steps:

- verify that the signature of the issuing authority is a valid signature of the crypto request.
- verify that the values of the extensions provided correspond to the encrypted extensions contained in the crypto certificate.

7 Conclusion

The use of SPSL does not protect the user against an attack that tries to get information from the IP address from which the request originated. For this reason, we propose to use SPSL in conjunction with CROWDS. In this way we would get the benefit of untraceability and anonymity provided by CROWDS also in the context of services in which user identification is crucial. It is obvious that SPSL needs to be adopted by the server, as it cannot operate without the server's collaboration. This is the greatest obstacle to a widespread use of SPSL as it is conceivable that servers will be very reluctant to allow anonymous identification or even the use of crypto certificates. It is thus important to make users aware of the threats to their privacy posed by current methods and that practical solutions are available so that the user base will start to request this type of service from servers. Identifying threats so to increase user awareness on this very sensitive aspects is one of the main motivations of this research.

8 Acknowledgements

We thank Enzo Auletta, Claudio Chimera, Alfredo De Santis, Mimmo Parente, Mike Reiter and Moti Yung for several discussions related to the SPSL protocol.

References

[1] The Apache web server project, http://www.apache.org.

[2] The Modssl home page, http://www.modssl.org.

[3] The Moz2i home page, http://spsl.security.unisa.it/moz2i.html.

[4] The OpenSSL home page, http://www.openssl.org.

[5] The SPSL home page, http://spsl.security.unisa.it.

[6] G. Ateniese and G. Tsudik. Some open issues and new directions in group signatures. In *Proc. of SODA 98.*

[7] D. Chaum. Untraceable electronic mail, return addresses and digital pseudonyms. *Communication of the ACM*, 2(24), February 1981.

[8] T. Dierks and C. Allen. The TLS protocol version 1.0. *Network Working Group RFC 2246 January 1999.*

[9] U. Feige, and A. Shamir. Witness indistinguishable and witness hiding protocols. In *Proc. of the 22 Annual ACM Symposium on Theory of Computing (STOC90)*, pages 416–426.

[10] A. O. Freier, P. Karlton, and P. C. Kocher. The SSL protocol version 3.0.

[11] C. Gulcu and G. Tsudik. Mixing e-mail with babel. In *Symposium on Network and Distributed System Security, Feb. 96, pp. 2-1.*

[12] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X509 public key infrastructure: Certificate and crl profile. *PKIX Working Group, Internet Draft, September 23, 1998.*

[13] M. K. Reiter and A. D. Rubin. Crowds: Anonymity for web transactions. *ACM Trans. Inf. Syst. Secur.*, 1(1):66–92.

[14] A. D. Santis, G. D. Crescenzo, P. Persiano, and M. Yung. On monotone formula closure of SZK. In *Proc. of the 35th IEEE Symposium on Foundations of Computer Science (FOCS94)*, pages 454–465.

[15] S. Schecter, T. Parnell, and A. Hartemink. Anonymous authentication of membership in dynamic groups. In *Proc. of Financial Cryptography 99, Lecture Notes in Computer Science, Springer Verlag.*

```
SSLeay_add_ssl_algorithms();
// initialize the supported algorithms
ctx=SSL_CTX_new(SSLv3_server_method());
// create a secure context
ssl=SSL_new(ctx);
// create a free and secure connetion
SSL_use_certificate_file(ssl, filename, type);
// certificate to be used is in the file filename
SSL_use_PrivateKey_file(ssl, filename, type);
// private key of the certificate in use is in filename
st=SSL_load_mask_file(filename);
// load the mask certificates
SSL_set_mask_list(ssl, st, "A34B5599", "ms_pth");
// set the mask for the connection
SSL_set_access_control(ssl, SSL_AC_YES, NULL);
// set the type of access control for the handshake
SSL_set_cipher_list(ssl,
SSL_DEFAULT_CIPHER_LIST_STRONG_MEDIUM);
// set the cipher suite
SSL_set_fd(ssl,sock);
// assign a file descriptor
SSL_accept(ssl);
// do a secure and private accept
SSL_read(ssl,buf,len);
// do a secure read
st=SSL_load_mask_file(file_rigth);
// load the mask certificates
ret=SSL_access_control_verify(ssl, st);
// verify the access for the client
if (ret<0)
error(...);
// an error was occured
else
if (ret==0)
// the access must not be granted
SSL_write(ssl, "Forbidden ...", ...);
else
// the access is granted
SSL_write(ssl, "OK ...", ...);
SSL_shutdown(ssl);
// close a secure connection
SSL_free(ssl);
// free memory
SSL_CTX_free(ctx);
// free memory
```

Figure 4: The skeleton of an SPSL server application.