3-1-1998

A framework for World Wide Web client-authentication protocols

Cem Paya
Dartmouth College

Dartmouth Computer Science Technical Report PCS-TR98-328

A FRAMEWORK FOR WWW CLIENT AUTHENTICATION PROTOCOLS

Cem Paya
payac@acm.org

UNDERGRADUATE HONORS THESIS
March, 1998

Advisor: Prof. Robert H. Morris, Sr.
rmorris@dockmaster.ncsc.mil

ABSTRACT

Existing client-authentication protocols deployed on the World Wide Web today are based on conventional distributed systems and fail to address the problems specific to the application domain. Some of the protocols restrict the mobility of the client by equating user identity to a machine or network address, others depend on sound password management strategies, and yet others compromise the privacy of the user by transmitting personal information for authentication. We introduce a new framework for client-authentication by separating two goals that current protocols achieve simultaneously:

1. Maintain persistent sense of identity across different sessions.
2. Prove facts about the user to the site.

These problems are independent, in the sense that any protocol for solving the first problem can be combined with any protocol for solving the second. Separation of the two purposes opens up the possibility of designing systems which balance two conflicting goals, authentication and anonymity. We propose a solution to the first problem, based on the Digital Signature Standard. The implications of this framework from the point of view of user privacy are examined. The paper is concluded with suggestions for integrating the proposed scheme into the existing WWW architecture.

1.1 Introduction

This paper distinguishes between two different goals in authentication protocols:

A: Maintain persistent sense of identity across different session instances
B: Prove facts about the user to another party

Existing authentication protocols perform these functions at the same time. We argue that at least for client authentication on the World Wide Web, it is beneficial to separate them. In particular, separating these two goals allows one to build systems which balance two conflicting goals, *authentication* and *anonymity*. We propose strictly independent protocols for solving these two problems. The first protocol addresses the question of maintaining persistent identity across sessions, while the second one deals with proving user credentials to the site. An informal review of the problem is given by considering the important characteristics of the WWW and arguing that conventional authentication schemes designed for distributed systems are inadequate for the web. Following this is a review and critique of the authentication protocols in use today. The next two sections introduce the new protocols, one for each of the problems mentioned above, examining strengths and weakness. By far the emphasis will be on the first protocol, since there are already existing solutions to the second problem. After a generic description, an implementation based on the Digital Signature Standard is proposed. Attacks against the generic protocol and the particular implementation are considered. A protocol for solving second problem is introduced in section §3, without undertaking a similar analysis of weaknesses or considering implementations. The paper is concluded with suggestions for integrating the first protocol into the existing WWW architecture, paying attention to issues specific to this domain.

1.2 Definitions relating to the World Wide Web

In this paper we will be concerned with communication channels operating over a network. For simplicity assume that all connections are taking place using the Internet Protocol (IP) which is based on the idea of routing small chunks of data called *packets*. The task of arranging the data in the packets into a continuous stream is handled by the Transmission Control Protocol, TCP. A connection using these two protocols will be referred to as a TCP/IP connection. Without going into details about the architecture of the network, we restrict our attention to TCP/IP connections taking place between two parties. All nodes on the network have definite addresses. To initiate a communication channel with any other party on the network, it is sufficient to know their address. The simplest representation of a network address is a string called the *hostname*. Another representation more suitable for the network layer is a number called the *IP address*. Since hostnames are convenient for humans and numerical addresses are convenient for software, it is frequently necessary to translate between the two. In the early days of the Internet where the number of hosts was small, each site maintained a local file of hostnames and IP addresses. In response to the increasing number of sites— rendering the original conception impractical— the Domain Name System (DNS) was developed. The DNS system is described in [12, 15, 16] and can be used to execute two types of queries: frequently the IP address is looked up given the hostname; this process is called hostname *resolution*. Occasionally the hostname is looked up given the IP address. The mapping between hostnames and IP addresses is not unique: a given hostname can have several IP addresses and multiple hostnames can resolve to the same IP address. There are legitimate reasons for such arrangements, such as creating aliases for some hostname.

The standard for specifying the location of resources on the network is the Uniform Resource Locator (URL) format. An example of a URL is `ftp://www.site.com:19/example.txt`. The structure of URLs is formally defined using extended BNF notation in [5]. A URL encodes several pieces of information about the object in question. The first part is an acronym for the protocol to be used for retrieving the object, which in this case happens to be the File Transfer Protocol. This is followed by the hostname and an optional port number, which is 19 in this case. When port number is not specified, the default implicit in the protocol definition is assumed. The last part of a URL is the location of the object on the file system.

The standard type of content on web pages is Hypertext Markup Language, HTML. HTML is an instance of the Standard Generalized Markup Language and the current version is defined in [17]. All HTML documents are composed of text. An important element of HTML is the notion of *hypertext links*, which are pointers to other objects. HTML documents can reference arbitrary types of content such as images, sounds and executable code. This is achieved by embedding links; the HTML document describes the object and has a link referencing the URL where the object is located. An example is shown below, a snippet from an HTML document:

```
<IMG  SRC="http://www.site.com/image.jpg">
```

The expression in angle-brackets is called a *tag* and conveys formatting information. In this case the tag describes an object of type “image” and the source is located at the indicated URL. After the page has been retrieved, the web-browser software will proceed to download the images and other objects referenced on the page. . The possibility of linking to objects at arbitrary locations on the network in an HTML document has important consequences in terms of user privacy, examined in the next section.

The standard application-level protocol for requesting web pages is the Hypertext Transfer Protocol, HTTP. This protocol assumes a client-server model where the objects are located on the server and the client makes requests to retrieve them. In the context of navigating the web, the client is referred to as the *user-agent*. This role is commonly played by the web-browser software. A web-server is a piece of software running on the server side which accepts connections and gives out documents in response to client requests. The protocol is described comprehensively in [2]. The two important methods from the point of view of this paper are the *GET* and *POST* directives. These are both issued by the user-agent in the request header. GET issues a request for a document on the server, while POST allows the client to send information to the server in response to an HTML form, as described in [1, 2]. HTTP is stateless and there is no concept of a session extending beyond the sense of a single TCP/IP connection.

1.2 Review of cryptographic primitives

Definition:
A one-way hash function $H : D \rightarrow R$ is a function with the following properties:

1. One-way: Given $r \in R$, it is difficult to find $d \in D$ with $H(d) = r$
2. Weakly collision-free: Given $d \in D$, it is difficult to find $d' \in D$ with $d' \neq d$ and $H(d') = H(d)$
3. Strongly collision-free: It is difficult to find $d, d' \in D$ with $d' \neq d$ and $H(d') = H(d)$

These conditions are not independent and it can be proved that 3⇒2 and 3⇒1. [11]
The word "difficult" is used in a computational sense. If $|D| > |R|$ it must be the case that $\exists d_1, d_2 \in D$ with $d_1 \neq d_2$ and $H(d_1) = H(d_2)$, because there is no way to map the domain into the range in a one-to-one fashion. Hash functions are based on the premise that collisions may exist, as long as it is difficult to find them. Let $B = \{0,1\}$ denote the binary alphabet. We will restrict our attention to one-way hash functions of the form $h : B^* \rightarrow B^n$ for some finite value of n, where B^* is the Kleene closure of the alphabet; informally it is the set of all finite-length strings over that alphabet. The significance of this class of functions in cryptography is that they generate relatively small fingerprints for arbitrarily long messages. The probability that two messages chosen at random produce the same hash is extremely small and the task of crafting two different messages with the same hash is computationally difficult.

1.3 Characteristics of the World Wide Web

- *Different levels of trust and privacy demands*

Because the content on the web is highly diverse, users will have different patterns of interaction with different sites. A user may be willing to disclose his identity to a bank in order to execute financial transactions, and even go to the trouble to obtain a digital certificate from a trusted third-party to present to the site. For another site the user may demand that no other party—including the site owner— be able to find out that he has been viewing pages on the site.

- *Conflicting interests: more information to sites vs. more privacy to users*

Sites would like to collect information about the users visiting them. There are important commercial motivations behind this since advertisements posted on web pages generate the bulk of revenue for popular sites. An important variable which determines whether it is cost-justifiable to place an ad on a given site is the profile of the users most likely to visit the site. Another motivation for sites to know more about the user is the notion of customized-content where the same collection of pages looks different to every user depending on their personal profile. Users on the other hand, want to protect their privacy by submitting information only as necessary and making sure that the information disclosed is not used for other purposes. A 1997 survey by Harris-Westin described in [28] found 53% of users are concerned that "information about which sites they visit will be linked to their email addresses and disclosed to some other person or organization without knowledge or consent."

- *WWW is extremely large and inherently decentralized*

The Internet is a heterogeneous collection of systems and there is no central entity. A basic design principle behind the IP protocol has been to provide alternative routes for all packets. A TCP/IP connection can be initiated provided that there is some path from the source to the destination; there is no intermediate node on which the connection ultimately depends. This basic philosophy has been employed consistently in the design of Internet protocols. A corollary is that authentication systems requiring all users to coordinate their actions with one distinguished party are infeasible beyond a small scale implementation. Users have become accustomed to the ability to exercise choices and will continue to demand systems that provide alternatives. This is one reason why standard authentication protocols will not scale up to the WWW: they assume the existence of an omnipotent central authority supervising all transactions and users willing to place unlimited confidence in that authority.

- *Network architecture is already strained and congestion is a frequent problem*

The number of hosts capable of taking part in the HTTP protocol as clients is so large that on-line checks for authentication incur an unacceptable performance penalty. This is another reason why the standard protocols for distributed systems will not scale up to the requirements of the WWW. These protocols require interactions with a trusted third-party. When download speeds can be measured in seconds and users typically browse as many as three pages per minute, a comparable delay to provide for authentication on each request is unacceptable. Contrast this with the credit card system: A purchase is relatively rare and the user is willing to wait an additional few seconds for the transaction to be authorized by the clearing system.

- *Users need to be mobile*

Access control based on static attributes such as IP address or host name are not useful, even though web servers can be configured to implement this. Such policies tend to equate user identity with machine identity, which is problematic for two reasons. First it places too much emphasis on the physical security of the machine, making it an attractive target for intruders. More importantly it restricts the mobility of the user, rendering them incapable of accessing the same resources from another network address. Even though this has not become an important concern for contemporary users, the trend in the direction of increasing mobility is clear. [15] There is an emerging need to develop systems enabling authorized users to access resources over HTTP regardless of their location on the network.

- *HTTP is a stateless protocol*

The Hypertext Transfer Protocol (HTTP) is a stateless protocol which does not store any information about the client to correlate different requests. A proposal in [4] has introduced an extension to store persistent state information on the client machine, but the proposal has not been formally incorporated into the most recent version of the HTTP protocol. This is particularly problematic because typical navigation patterns involve requesting several logically related objects in a sequence. For example, after downloading an HTML page the browser will proceed to download the images on the page. The first version of the HTTP protocol required a new TCP/IP connection to be initiated for each request. Since this was extremely inefficient, the next version introduced persistent connections and the "*keep-alive*" directive that allowed the user-agent to request multiple objects in one connection. [2] This solves the problem of efficiently downloading a page and all the associated resources at once. It does not address the problem of requesting pages which are logically connected in the navigation context, instead of connected directly by content. It is possible that before viewing page B, the user has to view page A. Even with HTTP/1.1 the site has no way of matching a request for page B with the corresponding request for page A.

- *Patent issues and export restrictions*

Another series of problems with the WWW security architecture are purely legal in nature. Public-key cryptosystems are covered by patents and require licensing fees for commercial use in North America. Garfinkel and Schneier have both pointed out that this has slowed down the integration of strong public-key cryptography into software products developed in the US. [10, 12] There is still some debate over whether an algorithm can be patented, and by implication whether patents granted to date will hold up in court. There are no legal precedents to answer this question. Large software companies by and large have preferred to avoid litigation and license the necessary algorithms from Public Key Partners. Adding to this complication is the fact that software implementing strong encryption is classified as munitions and covered by the International Traffic in Arms Regulations (ITAR) agreement which imposes strict restrictions on export. Often software vendors release two different versions of the same software, one for use within the United States, and another one with reduced cryptographic capabilities for international users.

- *Compatibility with existing protocols and software is important.*

Even though the growth of the WWW is relatively recent, there is already a wealth of software which has been written and widely deployed on the Internet. Any changes have to be compatible with the current state of the web. The utility in adopting a new protocol should be weighed against the cost of modifying existing systems. In terms of security, there are already different protocols designed for the WWW which attempt to provide authentication, secrecy and message integrity. By far the prevailing system in use today is the Secure Sockets Layer (SSL) protocol which will be examined in the next section.

1.4 Review of existing client authentication protocols for the WWW

1.4.1 Hostname and IP-Based Authentication

The simplest form of access control is to configure the server such that parts of the document tree are only accessible to HTTP requests from trusted hosts. These trusted entities can be identified either by name or by IP address, leading to the hostname and IP-based authentication schemes respectively. The major advantage of this idea is that existing web server software can be easily configured to enforce these restrictions. Despite this convenience and ubiquitous availability, there are some serious short-comings with this approach.

- DNS supports two types of queries. Either the hostname can be looked up given the IP address, or more commonly the IP address is looked up given the hostname. By design the first function is easy to tamper with and hostname based authentication is vulnerable to DNS-spoofing attacks where the adversary temporarily subverts the name-lookup system to return a different hostname for a given IP address. This is described in [12] and [14]. IP spoofing is harder and requires exploiting the source-routing feature in version 4 of the IP protocol. [14]
- Both of the schemes authenticate network addresses or machines instead of users or agents. It is conceivable that the physical security of the network infrastructure has been compromised; either a different machine has been plugged into the connection previously associated with the trusted host or the machine with that address has been compromised.
- In terms of usability, there is the added inconvenience to the user of always having to use a small number of distinguished machines, restricting mobility.
- IP-based restrictions are difficult to apply when there is no definite one-to-one correspondence between IP addresses and machines. This could happen when IP addresses are assigned dynamically so that the same address could be used by different machines over time. It could also happen because a single IP address is used by several machines; for instance, when a proxy is used to handle outgoing HTTP requests, all users going through the proxy will appear to have the same IP address.

In general hostname and IP-address based restrictions are only applicable for internal networks where the security policy explicitly dictates that resources are to be made available over HTTP only to a designated group of hosts on the network. It is not adequate for authenticating mobile clients from arbitrary locations or clients using untrusted machines.

1.4.2 HTTP/1.0 Basic Authentication

Version 1.0 of the HTTP protocol implements a primitive form of password based authentication called basic authentication. As stated in the protocol definition, [1]

> The "basic" authentication scheme is based on the model that the user agent must authenticate itself with a user-ID and a password for each realm. The realm value should be considered an opaque string which can only be compared for equality with other realms on that server. The server will service the request only if it can validate the user-ID and password for the protection space of the Request-URI.[1]

A *protection realm* is a subset of the document tree with access restrictions such that documents in the realm are transmitted only to authorized users. When the server receives a request for a document in a protected realm, it will respond with the name of the realm and request authentication. The client then sends the user name and password in cleartext. Following the example in the protocol specification, a typical exchange may proceed as follows:

«***server*** ⇒ ***user-agent***»	`WWW-Authenticate: Basic realm="WallyWorld"`
«***user-agent*** ⇒ ***server***»	`Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==`

[1] URI: Universal Resource Identifier. This is a generalization of the notion of URL, Uniform Resource Locator.

The problem with this approach is that user names and passwords are transmitted in the clear. This is not apparent from the transcript above, because the two strings are encoded after concatenation. The particular scheme used is base-64 encoding, a straightforward process associated with the Multipurpose Internet Mail Extensions (MIME) standard as defined in [6]. This encoding is used for embedding binary data inside ASCII text, using six-bits of each printable ASCII character, roughly giving a 33% expansion of the original data. Since base-64 encoding and decoding is very simple, basic authentication is vulnerable to eavesdropping.

1.4.3 HTTP/1.1 Digest Authentication

To overcome the shortcomings of basic authentication, RFC 2069 defines digest authentication in conjunction with HTTP/1.1. This document states: [3]

> Like Basic Access Authentication, the Digest scheme is based on a simple challenge-response paradigm. The Digest scheme challenges using a nonce value. A valid response contains a checksum (by default the MD5 checksum) of the username, the password, the given noncevalue, the HTTP method, and the requested URI. In this way, the password is never sent in the clear. [2]

When the user-agent requests a document without prior authorization, the server responds with the name of the protection realm and the challenge, which is referred to as a *nonce*. The user agent obtains the username and password from the user, prompting if necessary. An intermediate value is obtained by concatenating the protection realm, username and password, and applying a secure-hash function to the resulting string. This value is then concatenated to the nonce, and hashed once again to produce the response to the challenge. The default hash algorithm used in this procedure is MD5, described in [7]. A sample exchange from the same RFC, where hypothetical user named Mufasa requests the document http://www.host.com/dir/index.html:

«server ⇒ user-agent»

```
WWW-Authenticate: Digest   realm="testrealm@host.com",
                           nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                           opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

«user-agent ⇒ server»

```
Authorization: Digest      username="Mufasa",
                           realm="testrealm@host.com",
                           nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
                           uri="/dir/index.html",
                           response="e966c932a9242554e42c8ee200cec7f6",
                           opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

(The opaque string is meant to be passed back to the server exactly as is. The name is derived from the fact that the user-agent does not attempt to interpret or operate on the string in any way.)

Digest authentication is an improvement over basic authentication, although it paradoxically introduces a problem of secure storage. With basic authentication the server does not need to keep a copy of the passwords in the clear. Needham and Guy suggested that a one-way hash of the passwords should be stored instead. [10] The password transmitted by the client is verified by computing its hash and comparing against the stored entry. The advantage of this scheme is that the file containing the one-way hash values is useless to a potential adversary. This list can even be made publicly accessible, as in the case of the UNIX password system. [10, 12] In the case of digest authentication, the server does not store usernames and passwords in the clear either. Instead a one-way hash of the protection realm, username and password is stored. But this is precisely the same

[2] MD5 stands for "Message Digest 5," a secure hash-algorithm developed by Ronald Rivest. MD5 is a strengthened version of MD4, which in turn is based on MD2. See Schneier [6] for a decription of these algorithms.

combination used by the client to respond to the challenge, and as such these values have to be stored securely. The authors of RFC 2069 state: [3]

> The security implications of this are that if this password file is compromised, then an attacker gains immediate access to documents on the server using this realm. Unlike, say a standard UNIX password file, this information need not be decrypted in order to access documents in the server realm associated with this file. … There are two important security consequences of this. First the password file must be protected as if it contained unencrypted passwords, because for the purpose of accessing documents in its realm, it effectively does.

There are a number of generic criticisms which apply to both basic and digest authentication, and in fact to any protocol based on passwords. The criticism can be summed up by saying that these schemes are not scaleable beyond a dozen hosts because of their dependence on sound password management strategies on the part of the end-user. Historically it has been the case that:

1. Users choose bad passwords vulnerable to intelligent guessing and the problem is compounded as the number of sites increases. This is supported by extensive literature on cracking passwords on UNIX systems by copying the password file (usually stored in `/etc/passwd`) and mounting a dictionary attack off-line. Donald Klein was able to crack up to 40% of passwords in one experiment described in [21]. It is possible to impose some restrictions on the choice of passwords; for example by requiring a minimum length or mixed-case characters, or even issuing random strings as passwords. Good passwords have high information content and, *a priori* difficult to remember. In contrast, a natural language such as English has small entropy per symbol which is why users often find it convenient to choose words from a dictionary. In order to get around the difficulty of remembering several good passwords, sometimes users decide to choose the same password on different systems or reuse another password with minor modifications, both of which are unsound practices. Digest authentication protects against password sniffing but not against dictionary attacks targeting poor choice of passwords.
2. Users forget their passwords and are denied service until they contact the site administrator to request that the password be reset to some agreed value.
3. In order to remember a large collection of passwords, users are forced to write down passwords on paper or otherwise store them in an unsecure location, including cleartext files on the computer.
4. Having to type username and password every time a site is visited makes for an extremely tedious browsing experience. The current generation of popular web browsers remember the last user name and password typed and will resubmit these if necessary, without prompting the user. Care is taken to avoid writing this information to permanent storage, although there is no way to enforce such a policy on operating systems with virtual memory where the page could be swapped to disk anytime. For this reason, trying to hold more than a single username/password pair in memory is not a good idea.
5. It is conceivable that all username/password pairs will be stored encrypted on persistent storage with a master key required to decrypt the whole collection. This approach suffers from primarily restricting the mobility of the user. In addition, keeping the passwords on the local machine is dangerous because it makes the machine an attractive target for vandalism. If the integrity of data on the persistent storage is compromised through a deliberate attack, software failure or administrative error, the passwords are permanently lost.
6. Users need to register with the site before either form of authentication can be employed. Typically there is a registration page where a form is submitted through the HTTP protocol. The site processes the request and possibly contacts the user to confirm the registration, a potentially time-consuming exchange. The two parties have to agree on the password— typically determined unilaterally by one of the parties— which requires an encrypted connection.

1.4.4 Cookies

A *cookie* is a piece of data stored by a web-site on the client's machine. Cookies were proposed in [4] for storing persistent state information as an extension to version 1.1 of the HTTP protocol. Since HTTP is a stateless protocol, they are the only way of tracking users across multiple visits. Cookies were not designed to provide a form of authentication although they are currently used by web sites to solve a problem similar to the one examined in this paper, so we will give an overview of the

subject here. A cookie is set by the server in response to an HTTP GET request. RFC 2069 defines a new response header which the server uses to transmit cookies to the user-agent. An example is given below:

«server ⇒ user-agent»
```
Set-Cookie:  sessionID=24590812+page=document.html+visits=1;
```

This is a single cookie, even though the information encoded is logically segmented into several fields. The data contained in the cookie is a string composed of a subset of the printable ASCII characters. Spaces and special characters need to be escaped; the rules for encoding arbitrary strings into cookies are described in [17]. On subsequent requests for the same URL, the cookie will be returned to the server as part of the request. Access control on cookies is implemented by domain and pathname. When a cookie is set, the server may optionally specify a domain name and path such that the cookie will be transmitted to the server whenever the user-agents attempt to retrieve a page on a host in that domain and located under that path in the document tree.

This feature is commonly used to track users across different pages on a given site, providing an unreliable form of authentication. Upon encountering an unknown user, the server generates a new identifier and send this to the user in a cookie. The user-agent will successfully authenticate itself on future transactions by producing this identifier. One problem with this approach is that cookies are transmitted in the clear and easily captured by eavesdroppers. (This will not happen if the connection uses SSL, where all cookies set by the server or submitted by the client are encrypted.) Because of this problem, cookies are only used for low-risk authentication needs such as content customization. Another problem is that they tie down the user identity to the local machine where the cookies are stored. The underlying assumption in this model is that most people navigate the web from the same computer, and this argument could even be used to promote cookies since a machine at home presumably enjoys a degree of physical security. This assumption is true to a large extent today, although [15] argues that it is rapidly becoming necessary to accommodate mobile users. Writing cookies to permanent storage attached to the local machine suffers from all the shortcomings associated with storing passwords and keys. In the case of hardware failure or accidental corruption of the storage media, the cookies are lost along with the identification functionality on sites requiring them.

There are equally important privacy implications of cookies. Garfinkel and Spafford in [13] distinguish between privacy-protecting and privacy-compromising cookies. Because cookies are specific by domain and path, it is not possible for two unrelated sites to share cookies. This protects the privacy of the user to some extent, although it is still possible to subvert the restriction as follows. Suppose Mallory is a malicious site interested in deducing the navigation patterns of users. If two site administrators Carol and David agree to cooperate with Mallory, they will each place a resource on their pages which is located on Mallory's site. This is very easy to do with inline images, described in section §1.3. When the user visits such a page on Carol's machine, the user-agent software will deduce that an object on the page is located on Mallory's site and attempt to fetch it. Suppose Mallory sets a cookie containing a random value at this point, and records this random number. When the user visits David's page and is directed to Mallory's site to grab the remote object, the web browser will send the cookie as part of the HTTP request and Mallory can now determine that it is the same user who visited Carol's page. Stein [14] and Spafford [13] describe a well-known case where this was done by a company to rotate the advertisements that users saw on different web pages.

1.4.5 Secure Sockets Layer client authentication with certificates

SSL is a transport layer protocol that was first proposed in 1995. An official specification of the current version can be found at [29], and descriptions are given in [9] and [13]. SSL provides authentication, secrecy and message integrity. For our purposes, the relevant part of the protocol is the client-authentication mechanism. SSL makes extensive use of certificates. These are X509.v3 certificates signed by a trusted party known as *certification authority* and contain the owner's public key. The user-agent performs the authentication process on behalf of the client by proving knowledge of the private key corresponding to the public key embedded in the certificate.

The X509 certificate format is an ISO standard. Each certificate contains information about the owner, referred to as the subject, the subject's public keys, a serial number and information about the certifying authority. The latest version of the X509 standard is extensible, allowing arbitrary name/value pairs as attributes, in contrast to the standardized fields in the original specification. The structure of a certificate is shown below, adapted from [13].

Version
Serial Number
Algorithm Identifier: - Algorithm - Parameters
Issuer
Period of Validity: - Not Before Date - Not After Date
Subject
Subject's Public Key: - Algorithm - Parameters - Public Key
Signature of CA

The structure of an X.509v3 certificate

An X509 certificate effectively binds a public-key—and by implication the corresponding private key— to a set of user credentials. Parties honoring the certificate agree that an entity capable of demonstrating knowledge of private key is the subject whose distinguished name appears in the certificate, and that the information provided in the certificate about the subject is believed to be correct by the certifying authority. Both the server and the user can have certificates. The SSL protocol requires server certificates for verifying identity of the server, although there is an anonymous mode with Diffie-Hellman key exchange where neither party is authenticated and the communication channel provides secrecy but not authentication. Client certificates are optional. Currently server certificates are common and SSL is a *de facto* standard for sites dealing with confidential information, such as credit card numbers. By comparison, very few individual certificates have been issued although Stein argues in [14] that this trend is likely to change in the future.

The use of certificates to identify the client introduces serious privacy concerns. Disclosure of sufficient information to uniquely identify an individual is necessary in some circumstances. The question raised by the SSL protocol is whether there are alternatives to digital certificates for achieving that goal. There are important drawbacks to the X509 approach: Disclosure is an all-or-none operation, even if the server is interested in a small fraction of the information embedded in the certificate. There is no way to selectively disclose some of the fields and keep others hidden from the server. The amount of information required varies by site and to accommodate all possible modes of use, certificates have a bias in the direction of too-much-information, rather than too-little-information. As a result, the subject field in the certificate is crammed with personal information. SSL aggravates the situation because all certificates are sent in the clear, before the two parties start encrypting the communication channel. An eavesdropper will not succeed in reading the application-level HTTP data, but he will nevertheless learn the identity of the user involved. Finally, some information is too sensitive to be included in a certificate, because certificates are public documents which can be circulated beyond the control of the user. A good example would be credit card numbers and medical records. Certificates are useless when it is necessary to convince another party about the accuracy of this information.

1.5 The framework for authentication protocols

In view of the considerations above, we would like to propose an authentication protocol built around two independent goals:

- **A**: Maintain persistent client identity across sessions.
- **B**: Prove user credentials to the other parties as necessary.

The two problems are independent in the sense that any scheme which meets the requirements for problem A can be used in conjunction with any other scheme that is in accord with the requirements for problem B. The protocols in existence today solve both problems at once. Specifically for the World Wide Web we argue that making a distinction between the two and defining separate protocols to handle each one is beneficial. The envisioned mode of operation is that the client proves credentials only once by executing protocol B, and then protocol A ensures that the site will be able to relate the client to these credentials on subsequent transactions. By implication, protocol A is light-weight and can be carried out as many times as necessary. In contrast protocol B could involve significant amounts of computation or otherwise time-consuming on-line verifications with a certification authority.

Protocol A provides the minimal functionality required for sites that need to track users across different sessions without having to learn more about their identity. This protocol combines two apparently contradictory notions: authentication and anonymity. On the one hand users are completely anonymous because it is not even possible to determine whether two sessions on different sites involve the same user. On the other hand, the users are authenticated before accessing resources and impersonation by malicious parties is reasonably difficult. A client who does not participate in the authentication protocol is considered an unknown user. Since authentication is always with respect to some reference point, it is natural to raise the question of what one should take to be the defining feature of "Alice." At least with respect to this protocol, the answer is that Alice's identity is effectively constituted by her history of visits to the web site; authentication proceeds by correlating her identity with a history of previous exchanges. For lack of a better term, we will refer to this restricted sense of identity as *persona*. Ellison argues in [22] that this is a legitimate interpretation of identity in everyday affairs; individuals are viewed entirely in terms of a history of interactions. This is in contrast with the attribute-based notion of identity where entities are identified with a collection of attributes, more or less unchanging over time.

In most cases when disclosure of personal information is necessary, it is reasonable for the user to directly provide the information and for the site to assume that the given information is correct. To avoid manually typing in the same fields repeatedly, the user-agent maintains the personal information in a secure location and automatically gives out the requested fields when authorized by the user. There are already two proposals along these lines: one is the Open Profiling Standard defined in [26] and the other is the World Wide Web Consortium's Platform for Privacy Preferences initiative, summarized in [27]. The idea in both cases is that the site publishes a privacy policy. This policy lists the fields in the user profile that the site would like to learn and states how the site will use the data gathered. The user agent then determines if there is a match between the user privacy preferences and the site policy, submitting a subset of the requested information if appropriate. Presumably the site uses the information to selectively improve the content and since such content customization is going to benefit the end user, it is plausible to assume that users will have an incentive to supply correct information. In an increasingly large group of transactions however, this is not sufficient because there is more than content selection involved. Financial transactions or similar processes bearing legal implications require higher standards for authentication. In such cases it will be necessary to prove facts about the user's identity by enlisting the help of a trusted-third party. This is the goal of protocol B.

2.1 Requirements and rationale for Protocol A

Parties involved:
- *Alice,* user performing transactions with web sites.
- *Bob,* site owner handling requests for transactions.
- *Trent,* trusted-third party.

Requirements:
1. The protocol maintains a persistent identity for Alice across multiple sessions.
2. All persistent data storage is on the server side.
3. *Weak anonymity*: Bob does not gain any information about Alice.
4. *Strong anonymity*: Suppose that Alice executes the protocol with two different sites, say one operated by Carol and another one operated by David. Even if Carol and David collude by sharing transcripts, it is computationally difficult to determine that Alice is involved in both cases.

Bob will be able to determine when Alice visits his site again but he will not be able to learn anything else about Alice unless she provides more information herself. The term *persona* was introduced earlier for this sense of identity. We introduce a final requirement that:

5. Alice can assume an arbitrary number of persona such that Bob cannot correlate any two of these to the same user-agent.

The requirement labeled *strong anonymity* logically implies the weaker version, also making it redundant. This stronger version is introduced because it is not sufficient to ensure that one site by itself cannot learn personal information about the client. There are trusted sites where users may willingly disclose identifying information. Suppose that Alice trusts Carol and has already disclosed personal attributes on Carol's site. This information is later compromised, either through malicious intent on the behalf of Carol or an accidental breach of security on her site. If requirement #4 were not satisfied, then this would compromise Alice's privacy on every other site as well, because each site would be able to determine that Alice is the same person who visited Carol's site. From this point of view, the strong version ensures that a loss of anonymity remains strictly localized to one site.

Assumptions:

I. Trent's public key is known by all participants.

II. All sites wishing to take part in this protocol have certificates signed by Trent. A certificate will be denoted by C. It is assumed that logically distinct sites have different certificates.

Outline of the protocol:

Here we give a brief outline of the protocol. Details will be provided in the next section. We assume that there are two pieces of information available, one is a secret piece of data U which identifies the user-agent and the other is a public piece of data C which identifies the site. C is a *certificate* issued by Trent and U is the *user-identifier*. U is generated randomly by Alice and needs to be kept secret because all authentication is based on U; compromise of U implies the compromise of access to all sites. To authenticate herself, Alice combines U and C to generate a secret S and a description D of the secret which can be made public without compromising S. In other words, it will be computationally difficult to retrieve S from D. Alice sends D to Bob. Bob can identify Alice by D since she can reliably reproduce this value on subsequent visits. (Alternatively, Alice may directly generate a user-name by applying a different function on U and C.) Alice authenticates herself by proving that she knows S, which Bob can verify using D. This is similar to authentication based on user-name and password, except that the client never has to transmit a password to the server. Alice only needs to transmit D which can be done over an unsecure connection since it is computationally infeasible to derive S from D.

2.2 Protocol A: Generic description

In this section we give a generic description of the protocol. Specific implementations using existing public-key cryptosystems and secure hash-functions will be considered in the next section.

Let S be the set of secrets, C the set of certificates and U the set of user-identifiers.

0. Bob sends the certificate C and Alice verifies the signature on C.

1. Alice uses the parameter U and the site certificate C to generate a secret S for the signature scheme by computing S = F(C,U) where $F : \mathbb{C} \times \mathbb{U} \rightarrow \mathbb{S}$ is a publicly-known function with the following properties:
   - Given $S \in \mathbb{S}$ and $C \in \mathbb{C}$, it is difficult to find $U \in \mathbb{U}$ such that $F(C,U) = S$
   - Given $S_i \in \mathbb{S}$ and $C_i \in \mathbb{C}$ for $i = 1, 2, 3, \ldots, k$ it is difficult to determine whether $\exists U \forall i \ (1 \le i \le k) \ F(C_i, U) = S_i$
2. Alice computes D = V(S), the corresponding piece of information which could be used to verify that Alice knows the secret S.
3. Alice sends H(D) to Bob, where H is a one-way hash function. This is can be thought of as a nickname Alice generates specifically for use with Bob. All relevant information about Alice will be indexed by H(D). Bob can look up in the site database for an entry that matches H(D).
4. If he does not encounter such an entry, either Alice is visiting this site for the first time or the site has otherwise decided to discard Alice's identity. In this case the registration sequence is executed. Bob requests that Alice transmit the public piece D.
5. In either case, Bob is now in possession of D. Bob sends a challenge N to Alice to verify that Alice does indeed know the corresponding secret piece S. Alternatively, Bob and Alice jointly generate the challenge in such a way that neither one has complete control over the outcome.
6. Alice computes the answer R(S, N) to the challenge and sends this to Bob, who verifies that based on D, N and R(S, N) Alice does indeed know S.

In step #4 a hash of the secret *D* is transmitted to limit the exposure of the public piece. This forces a potential eavesdropper to be listening on the connection precisely when the public piece *D* is being transmitted, which will happen relatively infrequently. Without this public piece, it is even harder for an eavesdropper to derive the corresponding private key from subsequent exchanges. This is not an essential part of the protocol, since the security does not in any way depend on the secrecy of *D*. It is nevertheless more convenient to identify *H(D)* with the user-name rather than *D* itself, because *H(D)* is usually short and has fixed length, in contrast to *D* which is longer and has variable length determining the strength of the cryptosystem. It is possible to convert *H(D)* into human readable form if necessary, allowing people to communicate these numbers in a manner similar to credit-card numbers.

The site can change the certificate without necessarily losing the persistent identity for Alice. Here is how Alice can maintain her identity when the site represented by Bob switches from *C* to C'. Denoting the new parameters by primes, When Alice visit the site, she will first be asked to execute the protocol with certificate C'. She computes $S' = F(U, C')$ and D'. Since Bob does not encounter an identity matching $H(D')$ in this case, he informs Alice that a new certificate has been deployed recently and asks Alice for $H(D)$. If this entry is encountered in the database, Bob can now update the information by moving all the fields indexed by *H(D)* under $H(D')$

2.3 Analysis of the protocol and possible attacks

There are three different levels at which an adversary Eve could attack the protocol:

I. Eve discovers a way to impersonate Alice without knowing S. For simplicity, assume that Eve has found a way of responding to the challenge which has some non-negligible probability $0 < p \le 1$ of fooling Bob.

II. Eve recovers S; she can impersonate Alice every time on Bob's site.

III. Eve recovers U; she can impersonate Alice successfully on all sites.

To give an example where an adversary is able to accomplish **I** without knowing S, consider the RSA cryptosystem. Suppose Alice uses an RSA system with modulus N, signing exponent s and verifying exponent v satisfying $v \cdot s \equiv 1 \bmod \phi(N)$, where $\phi(N)$ is the number of units in the ring Z/NZ. Bob sends Alice a challenge c and Alice responds with $r = c^s \bmod N$. Bob verifies that $r^v \equiv r \bmod N$. Now suppose Eve knows the responses $\langle r_1, r_2, \ldots, r_k \rangle$ corresponding to *k* different challenges

$\langle c_1, c_2, \ldots, c_k \rangle$. Here is how she can impersonate Alice without knowing the signing exponent s. When Bob sends the challenge C, Eve looks for a set of numbers $\{e_1, e_2, \ldots, e_k\}$ such that

$$C \equiv \prod_{i=1}^{k} c_i^{e_i} \bmod N$$

If she can find such a set of numbers, then she can impersonate Alice because the response corresponding to the challenge C is given by

$$R \equiv \prod_{i=1}^{k} r_i^{e_i} \bmod N$$

From this example, it is clear that the difficulty of **I** and **II** depends on the specific algorithms used for implementation. Zero-knowledge proofs have the advantage that **I** and **II** does not become any easier when the protocol is used in interactive mode. Because of the zero-knowledge property, Bob does not learn anything new about the secret—and by implication, neither does an eavesdropper.

Even if Eve can derive S from D, she will be faced with the problem of recovering U from F(C,U) which is difficult, by the first requirement imposed on F. The strong anonymity requirement is satisfied in this case: Suppose Eve has $\{S_1, S_2, \ldots, S_k\}$ for k different sites and the corresponding site-certificates $\{C_1, C_2, \ldots, C_k\}$. By the second requirement imposed on the function F, it is difficult to determine whether there exists a user-identifier *U* such that $\forall i \ (1 \le i \le k) \ \ F(C_i, U) = S_i$. This guarantees that Eve will not learn whether the same person was involved in each case. Clearly if Eve cannot determine whether a given set of secrets $\{S_1, S_2, \ldots, S_k\}$ was generated by the same user, she cannot determine whether a set of public keys $\{D_1, D_2, \ldots, D_k\}$ was generated by the same user. This follows from the fact that an adversary in possession of the secret can compute the corresponding public key, implying that the first problem reduces to the second one.

It was pointed out in the last section that the Bob can get a new certificate from Trent and still maintain a persistent identity for users by verifying knowledge of the secret corresponding to the previous certificate. Private keys derived from old site certificates are not useful provided that the user associated with the key has already visited the site and replaced the old public key with the newer one, forcing the site to discard the former. (Recall that a public key can only be replaced by a party in possession of the corresponding secret.) A potential vulnerability in the system occurs when the site has changed to a new certificate but still remembers the public keys generated from a previous certificate. In this case, the adversary could find a user who has not visited since the new certificate has been installed and successfully impersonate that user by demonstrating knowledge of the private key derived from the previous certificate.

If the certificates were not signed by a trusted third-party, the protocol would be subject to a man-in-the-middle attack. Here is how Mallory could proceed: When Alice visits Mallory's site, Mallory will send Bob's certificate C_B instead of her own certificate C_M. Then Alice computes a public key D based on C_B, which is Alice's public key for Bob's site. Mallory at this point initiates a network connection to Bob and requests pages from his site, presenting the public key *D*. When Bob sends Mallory a challenge to confirm that she knows the secret corresponding to *D*, Mallory simply forwards the same challenge to Alice. Alice does not suspect anything unusual and responds to the challenge with *R*. Mallory can now present *R* to Bob, succesfully completing the authentication process. This situation will be avoided if the certificates are signed. When Mallory substitutes C_B for her own certificate, Alice will detect this because C_B has Trent's signature to the effect that it belongs to Bob. This is based on the assumption that Alice already knows *something* about the person that she intends to communicate with. For example, if certificates contained the name of the subject, Alice would have to know the name of the person she intends to communicate with. Garfinkel [13] points out—specifically in the case of X509 certificates, although the criticism applies to digital certificates in general— that there is a potential problem with certificates when there is insufficient information to identify the other party.

Since the user-identifiers are generated by Alice herself, she can assume any number of persona, limited only by the size of the set of secrets and the set of user-identifiers. Each persona corresponds to a choice of U. For example, Alice can use U_1 for business purposes and U_2 for personal navigation. She can maintain multiple presence on the same site with different persona. Since a single user appearing under two different persona is indistinguishable from two different users, Bob will not be able to determine that U_1 and U_2 are the same user.

There is one other problem in the protocol which needs to be considered separately for each implementation. It is conceivable that two different users generate the same secret on the same site. The protocol will not detect this situation but effectively treat them as the same user. Suppose Alice has already visited Bob's site and sent her public key D_{Alice}. When Carol visits the site, she *might* generate the same secret $S_{Carol} = S_{Alice}$ and consequently the same public key $D_{Carol} = D_{Alice}$, leading Bob to believe that Alice has returned. Since Carol successfully completes the authentication process, she will be given access to the site as it would look to Alice. This situation will be referred to as a *collision*, in analogy with collisions generated by hash functions. Clearly this is an undesirable situation and we would like to argue that the chances of encountering a collision are extremely small. Note that this situation may arise even if Carol has no intention of impersonating Alice. It is still a problem because this coincidence will prevent Alice and Carol from having different identities on the site. Another type of collision is possible when $S_{Alice} \neq S_{Carol}$ but $H(D_{Alice}) = H(D_{Carol})$. This scenario is not taken into consideration because it is easily resolved by transmitting the public key *D* instead of the hash *H(D)*.

When calculating the likelihood of a collision, there are two scenarios to consider. Given the secret S corresponding to some U, it is highly unlikely that any other user on the same site will have the same secret. On the other hand, given a large collection of U values, the probability that *some* pair of users generate the same secret may be non-negligible. This situation is sometimes referred to as the birthday paradox. [10, 11] In the abstract, the birthday paradox is concerned with selecting elements uniformly at random from a set of size *N*. The expected number of draws until some element encountered earlier appears again is $O(\sqrt{N})$. We would like to point out that such problems are not unique to implementations of protocol A. For example, it is conceivable that when generating RSA keys, a user accidentally hits upon a prime dividing the modulus generated earlier by another user. This is possible in theory but not in practice because of the extremely small probability that two randomly generated primes with hundreds of digits are equal. In a similar vein, we would like to argue that the chances of coming across collisions are negligible. Since this probability depends on the specific algorithms used, the analysis has to be undertaken separately for each implementation.

2.4 Protocol A: Implementation based on the Digital Signature Standard

The description given above is referred to as the generic description since it leaves out the exact details of the implementation. This provides for flexibility in the choice of algorithms. An implementation is defined by fixing the value of three parameters:

- The function *F*, used by Alice to derive the secret *S* from *U* and *C*.
- The function *V*, used by Alice to derive the public-description of the secret *D* from *S*.
- The function *R*, used by Alice to respond to the challenge and a corresponding function *P* used by Bob to verify that the challenge has been answered correctly.

There is also some flexibility if the challenge is to be generated jointly by the two parties, instead of being determined by the server. This is desirable to prevent chosen-ciphertext attacks against the public-key cryptography system used to prove knowledge of the secret. For example, if the secret is a private key used for encryption, it is usually desirable to prevent Bob or a malicious party interfering with the connection from asking Alice to encrypt arbitrary strings.

The choice of cryptosystems for an implementation will depend on three factors:

I. The user does not have to exchange any secrets with the site. All communications can take place in the clear. The protocol requires that Alice compute S by herself, without any help from Bob and that D can be made public without affecting the security of the protocol.

II. It is advantageous to have an authentication protocol which can be carried out in non-interactive mode and converted into a signature scheme. This is useful if the user wants to communicate with the site asynchronously. That is to say, Alice should be able to send a message to Bob, without Bob being available at the time to provide challenge values. Bob should be able to read the message anytime in the future and be convinced that it has originated from Alice. For example, if the public piece of the secret could be used for verifying digital signatures, Alice can send signed anonymous email to Bob, with her username *H(D)* attached, linking the message to her history of transactions on the site.

III. There should be efficient means of generating the private and public pieces.

I requires that a public-key cryptosystem be used. **II** restricts the class of possible systems to digital signature protocols, and zero-knowledge proofs which can be converted to signature schemes. **III** eliminates a subset of the remaining candidates. In particular RSA-based systems are ruled out, since prime generation takes much longer than signing/verifying and it is not possible for users on the same site to share the modulus. Suppose all users on the site shared the modulus generated earlier by Bob, and only each user only gets to choose the signing key. There would be two problems with this arrangement: First Alice would need help from Bob to compute the corresponding verifying key. Second, users would be able to impersonate each other, since knowledge of one pair of encryption/decryption keys allows an adversary to factor the modulus or derive other private keys without factoring the modulus. [10] Similarly, Alice can not use the same modulus on different sites because this would violate the strong anonymity requirement. There are two well-known cryptosystems remaining which satisfy all the requirements. One is the Digital Signature Standard (DSS) introduced in 1994 by the National Institute of Standards and Technology. The other is Schnorr's zero-knowledge proof-of-identity protocol, first introduced in [24]. These are very similar in terms of the mathematical constructs used and the secret is a discrete logarithm in each case. We propose an implementation of Protocol A based on DSS.

Review of DSS:

DSS is a public-key cryptosystem which can only be used for signatures. The mathematical algorithm employed by DSS is referred to as the Digital Signature Algorithm, (DSA). A brief review of DSA is given here. For details, including a proof that the signature can be verified if and only if the message hash is the same as the one used in signing, refer to [11] or [18]. DSA is a based on the El-Gamal cryptosystem, first proposed by Taher El-Gamal in [25]. The ElGamal system can be used to provide both secrecy and authentication, and relies on the difficulty of computing discrete logarithms in large finite fields. DSA is based on the assumption that working in a relatively small subgroup embedded inside the large prime field is also secure. In addition to the modulus p which defines the field, another prime q is needed to define the subgroup. In contrast to p whose size is variable and determines the overall strength of the system, the size of q is always 160 bits. The exact value of q determines the size of the subgroup under consideration.

Digital Signature Algorithm
Public key: p, a prime with length between 512 and 1024 bits q, a 160 - bit prime dividing $p-1$ $g = h^{(p-1)/q} \bmod p$, where h is a generator of $(Z/pZ)^{\times}$ $y = g^{x} \bmod p$, where x is the private key
Private key: x, a number less than q.
Signing:

Choose random k such that $0 < k < q-1$
$r = \left(g^k \bmod p\right) \bmod q$
$s = k^{-1}\left[SHA(M) + xr\right] \bmod q,$ where M is the message.
The signature is the pair of numbers *r* and *s*.
Verifying:
$w = s^{-1} \bmod q$
$u_1 = w \cdot SHA(M) \bmod q$
$u_2 = r \cdot w \bmod q$
$v = \left(g^{u_1} y^{u_2} \bmod p\right) \bmod q$
The signature is valid if $v = r$.

Summary of DSA

Note that $g \bmod p > 1$, since *h* is a generator of the group and has order *p-1*; no smaller power of *h* can be equal to the identity. In fact the order of *g* is exactly *q*. This is a corollary of the following basic result about groups:

Lemma:

Let *G* be a finite group and let $|x|$ denote the order of an element $x \in G,$ which is equal to the size of the subgroup generated by x: $|x| = \#\left\{x^i : i \in Z\right\}$. Then, $\forall m \in Z \ \left|x^m\right| = \dfrac{|x|}{\gcd(m,|x|)}$

The private key *x* is the discrete logarithm of *y* to base *b* in the field. This is the motivation for the conjecture that the security of ElGamal type systems rests on the difficulty of taking discrete logarithms, although there is no proof of this fact. It is clear that an oracle capable of computing discrete logarithms in the field Z/pZ can be used to recover the private key and forge signatures. The converse to this statement has not been proved— namely, that an oracle capable of forging signatures can be used to efficiently compute discrete logarithms. [10, 11] DSA is based on an additional assumption, namely that taking discrete logarithms in a subgroup of $(Z/pZ)^{\times}$ of size *q* is as difficult as taking discrete logarithms in the field Z/pZ. Currently there are no known algorithms in the literature for taking discrete logarithms in the smaller group more efficiently than solving the same problem in the whole field. [10]

One major difference between RSA and ElGamal-based systems is that the latter class of algorithms require reliable pseudo-random number generators. This follows from the fact that the value of k used for signing has to be unpredictable and different for each signature. [3] If an adversary discovers the value of k used to sign the message, the private key will be compromised. This is easily seen by solving for the private key in the expression for the signature:

$$x \equiv r^{-1}\left[sk - SHA(M)\right] \bmod q$$

An adversary who discovers the value of k used for signing the message will have all the values for the variables appearing on the right hand side and will deduce the private key with a straightforward calculation. Likewise if two messages are signed with the same *k*, the adversary can recover the private key without knowing *k*. Suppose two different messages with SHA hashes H and $H' \neq H$ are signed with the same value of *k*. Then the adversary has:

$$r' = r = \left(g^k \bmod p\right) \bmod q$$

$$s = k^{-1}(H + xr) \bmod q$$

$$s' = k^{-1}(H' + xr) \bmod q$$

The private key can be recovered by noting that:

$$k \equiv (s - s')^{-1}(H - H') \bmod q$$

Since $H \neq H' \Rightarrow s \neq s'$, the required inverse of $s - s'$ exists and *k* can be recovered.

Assume that each site has generated the values *p*, *q* and *g* in advance, and these are made available to the client at the time of the connection. (According to [18] these three parameters can be shared among a group of users.) With this assumption a specific implementation of the protocol is defined by fixing the parameters as follows:

- Set $F(U,C) = SHA(U + C) \bmod q$, where plus denotes the concatenation of two strings and SHA is a function implementing the revised Secure Hash Algorithm, SHA-I as defined in [19]. This corresponds to the private key x in the DSS specification.
- Set $V(S) = g^S \bmod q$. This corresponds to the public key y in the DSA specification. Only this piece of the public-key is different for each user. The other parameters are shared by all users on the same site.
- Set $R(S,N)$ to be the DSS signature of *N* using the private key *S*. The fact that *R(S, N)* is the signature of *N* can be verified by any party in possession of *D*, which is the public key corresponding to the secret *S*.

There are specific considerations relating to the use of DSS to implement protocol A. It is conceivable that the site will choose *p* and *q* such that the discrete logarithm problem is very easy to solve in comparison to the same problem in a field of the same size as *p*. For example, Pohlig and Hellman proposed an algorithm in [23] which efficiently computes discrete logarithms in the field Z/pZ provided that $p-1$ has only small factors; these moduli are said to be "smooth." In this case $q|p-1$ and *q* is a large prime, so the Pohlig-Hellman algorithm is not going to be efficient in the field Z/pZ. There are other properties of a prime modulus which could make it easy to take discrete logarithms. We would like to argue that this is not very dangerous. In the first place, the private key by itself is of no use to the site owner because it is not computationally feasible to derive *U* from *S*. The key for each site is different, so it is not going to help Bob impersonate Alice on any other site. Bob could presumably give this key to another user who could then go on to impersonate Alice. This is not a major concern because if Bob is malicious, he could have granted the same person access to Alice's account without any authorization. There is nothing Bob can do with the private key that he could not have accomplished before. Secondly, these moduli are unlikely to be generated randomly, since they are extremely rare and easy to recognize. More important, NIST recommended a specific method for generating *p* and *q* by starting out with two values called *seed* and *counter*. If the site publishes these two values, anybody can verify that *p* and *q* have been generated by following the same procedure outlined in [19]. (Since this process is time-consuming, it is easier if the trusted-party performs this check prior to signing the certificate.) The important point about this proposal is that *p* and *q* are obtained by applying a secure one-way hash function to the seed and counter. A malicious site could choose cooked moduli but will not be able to produce the corresponding seed and counter.

Since SHA produces a 160-bit hash, each user-identifier should be at least this long and preferably much longer: $|\mathrm{U}| > H$, where $|\mathrm{U}|$ is the size of the set of user-identifiers and *H* is the size of the hash-space.[3] If the user-identifiers are *k* bits long, an adversary will have to be in possession of at least

[3] The important parameter is the entropy of the distribution used for generating the identifiers, rather than the actual length of an identifier. For instance, when a 50-bit random seed is used to initialize a pseudo-random number generator which outputs a 1000 bit user-identifier, the effective length is still 50 bits. We will assume that user-identifiers are chosen from a uniform probability distribution. In this case the entropy of the distribution is a maximum and equal to the length of a user-

$\lceil k/160 \rceil$ different secrets to launch a successful attack to recover the user-identifier *U*. This is an information-theoretical argument and completely independent of the question of whether SHA is a reliable hash function. Under this assumption about $|U|$ we can compute the probability of collisions. A collision in the sense of section §2.3 directly corresponds to a collision in the hash function. We can derive a rough estimate of the probability that two users on the same site have the same private DSA key. Assume that SHA distributes the hash values uniformly: when a large number of random strings are hashed and equivalence classes are defined according to the hash value, there will be approximately equal numbers of strings in each class. For a site with *N* different persona, a lower bound on the probability that there are no collisions is:

$$P(\text{No collisions}) = \prod_{k=0}^{N-1}\left(1-\frac{k}{H}\right) \geq 1-\sum_{k=0}^{N-1}\frac{k}{H} \geq 1-\frac{N^2}{2H}$$

Using the following estimates,

$H = 2^{160}$, since SHA produces a 160-bit hash

$N = 10^9 < 2^{30}$, assuming a billion persona on the site

$P > 1 - 2^{-100}$

This is the probability that no collisions occur for a given site. If the protocol is implemented by a billion sites each with a billion different persona, the probability that *any* site encounters a collision is less than 2^{-70}.

This is an efficient scheme because key generation is reasonably fast, requiring only one application of SHA and one modular exponentiation. Responding to the challenge is also fast, and involves one application of SHA to hash the challenge, one exponentiation, one addition, two multiplications and one modular inversion. The random value *k* can be generated in advance, even before any connection is made. In fact a common optimization in DSA implementations described in [11] and [19] is to generate a sequence of *k* values, along with the corresponding inverses and *r* values. This optimization is not applicable here in full, since the values of *p*, *q* and *g* depend on the site.

As of this writing in early 1998, the most efficient way published in the literature for attacking a DSS signature system is to derive the private key by solving the discrete-logarithm problem. We argued above that the site will not gain anything by doing this. It follows that the major threat is from eavesdroppers. The fastest algorithms for the discrete-logarithm all have a very long precomputation stage, after which taking individual discrete logarithms *in the same field* is relatively easy. [11] Since all users on the site share the same modulus *p*, if an adversary succeeds in completing the precomputation for the field Z/pZ he will be able to impersonate any user at will.

2.5 Integrating Protocol A into the existing WWW security architecture

In this section we would like to consider the issues likely to emerge when attempting to integrate the protocol into the existing WWW architecture. The assumptions listed in section #2.3 are valid to a large extent: there are already certification authorities issuing signed certificates and the standard distribution for popular web-browsers include the public-keys of the CAs. This infrastructure was developed mainly to support the SSL protocol, which has become the standard for electronic commerce applications. The number of sites carrying signed X509v3 is rapidly increasing. For sites which do not have certificates, it is possible to substitute an implicit piece of information— such as the hostname or the complete URL of the object being retrieved— in place of the certificate. The problem with this is that it opens up an opportunity for man-in-the-middle-attacks if the DNS system can be subverted, because it is equating the identity of the site with a network address.

Care has to be taken to ensure that the goals envisioned in the protocol are not compromised by other means. One of the headers transmitted in an HTTP request is the *referrer*, which is the page the user had last visited. This feature was designed for the purpose of discovering broken links. Suppose that a link on Carol's page to an object on Bob's site is invalid; perhaps Bob moved the object somewhere

---

identifiers in bits. Because of this assumption, we can continue to refer to "the length of a user-identifier" to mean the entropy of the distribution.

else or deleted it entirely. If Alice tried to follow this link and ended up requesting a non-existent object from Bob, she would get an error message and Bob would learn that Carol has a broken link on one of her pages. Unfortunately this violates the privacy of the user by allowing one site to learn what document the user had been viewing on a different site. This is especially worrisome because URLs can encode additional information such as search queries. [13, 14] Another way of compiling information about a user's browsing history using cookies was summarized earlier. It is not too dangerous because it requires cooperation between the sites and web browsers today can be configured to drop all cookies.

The important point here is that protocol A can be exploited in the same way as cookies. This is a consequence of the structure of HTML documents. In particular, retrieving a URL does not always mean dowloading the contents of an existing file in the document tree. For example, it is possible to indicate a script to be executed on the server side and even specify arguments to be passed to the script. This means that for all practical purposes, a hyperlink can encode arbitrary information. When a request for such a URL is received, the script is run with the indicated parameters and the output is sent back to the user as an ordinary web page. A corollary is that it is possible to generate web pages dynamically and send a different page to every user. Suppose that Bob and Carol are site owners. Each one places a different hyperlink link on their page to an object on Mallory's site; say Bob has the link L_B and Carol has the link L_C. Each link encodes information about Alice's site-specific public key. That is to say, L_B contains D_{Alice}^{Bob} and L_C contains D_{Alice}^{Carol}, where superscripts on the public-key indicate the site and subcripts indicate the user involved. When Alice visits Bob's page, the user-agent will retrieve the object at location L_B, which happens to be on Mallory's site; Mallory makes a note of D_{Alice}^{Bob} at this point. Similarly, Alice will try to fetch the object at location L_C when she views Carol's page, unintentionally giving away D_{Alice}^{Carol} to Mallory. Since protocol A allows Mallory to determine that these requests are from the same user, she can inform Bob and Carol that the keys D_{Alice}^{Bob} and D_{Alice}^{Carol} belong to the same user. The lesson here is that there are issues specific to the WWW which could lead to a compromise of the protocol. A general rule for avoiding these situations is to avoid executing the protocol when downloading an object which is located on a different host than the page containing the link to the object. (This will not solve the problem completely, because the link L_C could be an ordinary link on Carol's page and Alice herself may decide to visit the page associated with the link.)

A user only needs to generate an identifier *U* to take part in the protocol. Since this parameter is meant to be kept secret and there is no correspoding public piece to share, there are no distribution problems involved. The user-identifier does not have to registered with a trusted party or escrowed. Generating a user-identifier requires a reliable source of randomness, but this functionality is already bundled with the current generation of web browsers; they are capable of generating keys for public-key cryptosystems. Generating user identifiers is easier than generating public-keys because there is no mathematical structure to the set U; any binary string is an acceptable user-identifier. When Alice needs a new persona, all she has to do is to generate a new user-identifier. The whole problem is one of secure storage and management of the identifiers. At any given time Alice will have a collection of different persona represented by the user-identifiers $\langle U_1, U_2, \ldots, U_k \rangle$. The challenge is to protect the collection against unauthorized access, at the same time allowing Alice to use any persona when necessary, add new identifiers or discard old ones. The problem is identical to managing private keys, and once again existing software is capable of performing these functions.

In view of these considerations, we conclude that minor modifications to the user-agent software will make it possible to leverage the existing WWW architecture to implement protocol A.

3.1 Protocol B: Motivation and assumptions

We first provide an outline of the protocol. Before participating in the protocol, users register with the trusted-party and disclose as much information as necessary. The structure of the information accepted by the trusted-party is a set of ordered pairs $\langle A, V \rangle$. The interpretation of each ordered pair is that the attribute A of the user has value V. Examples of attributes commonly used would include biographical data such as name, date of birth, address, occupational data such as profession, company name, position, yearly salary and even personal information such as hobbies. The envisioned mode of operation is that the site requests permission to execute queries with the trusted-party. Since some of these attributes are considered confidential information, the user gets to choose what type of queries can be executed by each site. Let Ξ denote the set of all possible queries. We assume that there exists a language for encoding queries such that the user, the server and the trusted party all agree on the semantics of a syntactically valid query string. The site sends a set of queries to the client. The user-agent consults with the user to determine to determine a subset of the queries that the user consents to answer. For this subset, the user-agent sends an authorization to the server who forwards it to the trusted-party. The trusted party verifies that the authorization is valid and for each authorized query, sends the results back to the site.

Neither the user nor the trusted-party has to do any additional work off-line, since they would have to go through the same process of verifying credentials before the certificate is digitally signed. The same arrangements are necessary to convince the trusted party that the user attributes are accurate. All of this takes place off-line and before any transactions are initiated across the network. In both cases a database of user attributes is necessary. The major difference in the mode of operation is that the trusted party processes queries from sites, providing a real-time interface between the database and the outside world. On the other hand, when certificates are used the information in the database is statically encoded once in the certificate and sites refer to this digital document from there on.

Any information that the site infers from a certificate can be retrieved as the result of an authorized query, assuming that the query-description language admits queries of the form *"What is the value of attribute A?"* Similarly, if all user attributes known by the trusted party were disclosed to the site at once, the site would be able to derive the answers to its own queries via logical deductions. From this point of view, the information from a certificate is equivalent to the information obtained by executing queries with a trusted party who is in possession of the same attribute-value pairs contained in the certificate. Despite this equivalence, there are three important differences.

- In contrast to certificates, it is possible to ask specific questions. For example, it is possible for a site to verify that the user is not a minor without learning the exact birth-date. This protects the privacy of the user by revealing information on a "need-to-know" basis. Such considerations are important since it is estimated that zip code, date of birth and profession can be used to uniquely identify an individual.
- The information embedded in a certificate is static. If the value of a single attribute changes, a new certificate has to be issued by the CA and installed by the client. To complicate the situation further, the previous certificate has to be revoked. Under certain revocation rules, this means that the user will lose all access to sites where the previous certificate had been used for authentication.
- In general, it is true that an on-line verification with a trusted party introduces an additional delay and increases the network load. This is a minor performance penalty since the proof of credentials has to be done only once in theory. Under the assumption that protocol B is being used in conjunction with protocol A and that the server maintains a database of user attributes indexed by the site-specific public key, it will be easy for the server to determine that some user has already responded to the queries in subsequent visits.

3.2 Protocol B: Requirements and generic description

Participants:

- *Alice* user-agent, performing transactions with web sites.
- *Bob* web-server accepting requests for transactions from clients.
- *Trent* trusted-third party.

Requirements:

1. Bob is convinced at the end of the protocol that the answers received from Trent to the queries reflect accurate information about Alice.
2. Bob cannot find out more facts about Alice other than those implied by the authorized queries.
3. An eavesdropper listening on the connection between Alice and Bob cannot find out any information about Alice, nor can he use the information to impersonate Alice.
4. *Restricted strong anonymity:* If Alice performs this protocol with two different sites, say Bob and Carol, it is not possible for Bob and Carol to determine by colluding whether the same user is involved, *unless* this is implied by the answers to the queries.

To clarify the fourth requirement with an example: if Bob and Carol both learn a unique fact about the user, for example that her name is "Alice J. Random" and her social security number is 738522914, this would imply that they have been dealing with the same person. On the other hand, if the queries only established whether Alice had a legal driver's license, then nothing else in the protocol should allow Carol and David to deduce whether the same user is involved. The problem can be phrased in terms of the uncertainty about Alice's identity. When the site has no information about Alice, the uncertainty about her identity is a maximum; she could be anyone with Internet access. The uncertainty can be quantified as an entropy. After a query has been answered, the uncertainty is reduced because the set of all possible users is partitioned into disjoint sets depending on the outcome. It is helpful to think of the user identity as an independent variable *x* and the answer to the query as a function *R(x)* of the independent variable. Unless *R(x)* takes on the same value for all choices of the independent variable—in which case the query would be unnecessary— there is some information conveyed about *x* by *R(x)*. The information content of the query is the difference between the uncertainty before and after the answer is disclosed. If the uncertainty becomes zero, the individual is uniquely identified and it is at least theoretically possible for two sites to conclude that they are dealing with the same person.

Assumptions:

1. Trent's public key is known by all participants.
2. Alice has a private key reserved for use in this protocol. Only Alice has the private key and only Trent has the corresponding public key. (From this point of view, the public-key is a secret shared by Alice and Trent, rather than being "public" in the usual sense.)
3. There exists a protocol for creating a secure channel between Alice and Trent and a protocol for creating a secure channel between Bob and Trent. For the purposes of this section, a secure channel is defined to be one providing authentication with defense against man-in-the-middle attacks, secrecy and message integrity. The fact that Trent's public key is known to all participants is sufficient to ensure this. No assumptions are made about the connection between Alice and Bob.
4. Bob has a certificate signed by Trent.

Generic description:

The protocols uses *tickets* T_i and *ticket series* $\langle T_0, T_1, T_2, \ldots \rangle$. Each ticket T_i is an integer and the series is defined by the ordered pair $\langle T_0, F \rangle$ where F is a function used for generating the next ticket in the series from a recurrence relation: $\forall i > 0 \; T_i = F(T_{i-1})$. We impose the following requirements on the ticket series, and by implication, on the function used to generate the tickets:

- Given a sequence $\langle T_1, T_2, \ldots, T_k \rangle$ from the same series, it is computationally difficult to compute the next ticket T_{k+1} without knowing the function *F*.
- Given a set $A = \{T_1, T_2, \ldots, T_k\}$, it is difficult to determine whether these tickets belong to the same series. In other words, it is difficult to determine whether there exists a function *F* and an initial ticket *s* such that $\forall t \in A \; \exists i \in N \;\; t = F^i(s)$, where *N* is the set of natural numbers and F^k is the function *F* composed with itself *k* times.

1. Before visiting any sites, Alice establishes a secure connection with Trent and authenticates herself. Trent issues Alice a ticket series by sending her $\langle T_0, F \rangle$ and inserts the same pair into a

database, along with the fact that it has been issued to Alice. When Alice needs to prove some facts about her identity by answering queries from the site, she uses one of these tickets. Each ticket is used exactly once. A ticket series has a lifetime measured either from the time that it is issued to Alice, or from the time that Alice was last seen performing a transaction with the ticket.

2. Bob sends Alice his certificate and a set of queries, $Q = \{q_1, q_2, \ldots, q_a\}$, where $Q \subseteq \mathbb{P}$
3. Alice determines a set S of queries for which Bob will be granted authorization. It is not necessary to have $S \subseteq Q$; Alice can determine a policy for answering queries in advance, applied independently of the queries received from the site.
4. Alice signs the packet $\langle T_i, C_B, S,$ optional time-stamp$\rangle$, and sends it to Bob.
5. Bob establishes a secure communication channel with Trent and forwards this packet.
6. Trent looks up the ticket in the database. If the ticket does not exist in the database Trent sends an error message to Bob. If the ticket has expired, Trent sends a message asking Alice obtain a new ticket series.
7. Otherwise the ticket is in the database and Trent determines that it belongs to the series associated with Alice. At this point T_i is removed from the database. Since Trent has access to Alice's public key, he can verify the signature on the packet forwarded. Trent also check that C_B belongs to Bob.
8. If the signature is valid (and the time-stamp in the bundle, if supplied, is within a tolerance Δ of the current time) Trent determines the answers to the queries in the set $S \cap Q$. He sends the results back to Bob and inserts $T_{i+1} = F(T_i)$ into the database. Otherwise an error message is sent to Bob, and the ticket series is now revoked because T_i had been removed from the database in step #8.
9. Bob informs Alice of the result. If the exchange has been successful, Alice computes $T_{i+1} = F(T_i)$ to use for the next time that she needs to authorize queries.

No information identifying Alice is ever exchanged in the connection between Alice and Bob. For this reason, the connection does not have to be encrypted. The query-authorization that Alice sends to Bob is useless to other sites for two reasons. First Bob's certificate is included in the signed packet; if Mallory were to attempt to use it by replacing C_B by C_M, Trent would detect this in step #9 because the signature would not verify. Secondly, Bob can only use this authorization once: this is because the ticket T_i is only valid for use by Alice once. In case Alice ever happens to hold the same ticket in the future, an optional time-stamp can be added to the bundle, although this is unnecessary if the tickets are selected randomly from a large set.

3.3 Protocol B: Modifications and variants

The protocol as stated has four short-comings:

I. It is not fault-tolerant. In case Bob fails to secure a connection with Trent or communicate the results to Alice, Alice may end up with using T_{i+1} for the next instance of the protocol when Trent expect T_i. (The reverse is also possible: if Alice assumes that Bob failed to forward the packet to Trent, she may believe that T_i is still valid.) Also note that if the signature on the packet does not hold for some reason, perhaps because of tampering by a malicious party, the ticket series is invalidated and Alice has to start from scratch to obtain a new one. Both of these problems are caused by the fact that at any given instant, Alice has exactly one valid ticket.

II. A corollary to I is that the protocol cannot be executed with different sites in parallel; each operation must wait for the preceding one to finish so that the next ticket in the series becomes available.

III. Trent learns about all the web sites that Alice has been visiting, i.e. Alice's *browsing history.* Moreover Trent can deduce additional information about Alice's relationship to each site because he has access to the list of queries authorized in each case. From the point of view of Alice's privacy, this is worse than simply knowing the browsing history. (After all, given the current architecture of the Internet a lot of people who could theoretically determine a user's browsing history, most prominently the service providers.) For instance, if Alice authorizes a query about her credit-card number, this might suggest that she is involved in purchasing goods from the site.

IV. The protocol allows only one trusted-party and requires all users to be registered with that person. This is not satisfactory in practice, given the number of people involved. Since there is already an existing hierarchy of certification authorities, it is likewise necessary to admit multiple Trents, such that Alice can be registered with any subset of the trusted-parties.

It is easy to modify the protocol to solve I and II. Instead of having a single ticket T_i, Trent will accept a set $V = \{T_{i_1}, T_{i_2}, \ldots, T_{i_r}\}$ of tickets such that $|V| \le r$ for some fixed value of r. When the ticket series is issued for the first time, this set is initialized to $V = \{T_0, T_1, \ldots, T_{r-1}\}$. When a ticket $t \in V$ is used and the protocol fails— the signature on the packet forwarded by Bob does not verify— set $V \leftarrow V - \{t\}$. If the protocol succeeds, set $V \leftarrow V - \{t\} \cup \{F(t)\}$, where we now assume that F is mostly collision-free: Given t_1, t_2 with $t_1 \neq t_2$, the probability is very high that $F(t_1) \neq F(t_2)$ This ensures that for each ticket used successfully and removed from the set, there is a unique ticket taking its place. This ensures that the size of the set of valid tickets remains constant provided that the tickets are being used by the authorized party. Each incorrect use of a ticket results in a reduction of the set by one element. After r failures the ticket series is invalid because $V = \varnothing$.

Likewise it is address problem IV. Instead of Trent, consider a hierarchy of trusted third-parties, with a distinguished one labeled the *root*. Alice still receives her ticket series from Trent and does not have to know about the structure of the CA hierarchy. When Trent issues a ticket series to Alice, he needs to propagate the set of valid tickets $V = \{T_{i_1}, T_{i_2}, \ldots, T_{i_r}\}$ up to the root. The root notes that the tickets in V are associated with Trent. The structure of the hierarchy is also transparent to Bob who always contacts the root to submit the bundle $\langle T_i, C_B, S \rangle$. The root CA determines that T_i was issued by Trent and forwards the packet down Trent. Trent performs steps #7 through #9 of the protocol and communicates the results back to the root. The root will then forward the results to Bob. Assuming that secure communication channels exist between the root and Trent, this will ensure that the information is accessible to Bob only.

It is possible to further modify the protocol to solve III. Suppose that the signature scheme that Alice uses is *generative*. That is to say, given a signature Trent can determine the message (or the hash of the message, if a one-way hash function is used before signing) that corresponds to the signature. Schematically we have

$$Message \xrightarrow{\text{Secure Hash Function}} Hash \xrightarrow{\text{Signing function}} Signature$$

Anybody can determine the hash from the message. Signature algorithms are based on the assumption that without the private key, it is not possible to derive the signature from the hash. A proper subset of public-key signature systems are generative, meaning that anybody can derive the hash from the signature, without knowing the private key. For example, RSA is generative whereas DSA is not. Given a generative scheme, here is the modified protocol after step #5:

Let K_T denote Trent's public key, K_R the root's public key and K_B Bob's public key.
Let $E_k(a_1, a_2, \ldots, a_n)$ denote the result of encrypting the strings $a_1, a_2, \ldots, a_n$ using the key *k*. This key could be a public-key or it could be the session-key for a symmetric cipher. We will assume that the encryption is done in a such a way that each of the strings can be recovered individually.

1. Bob sends to the root: $E_{K_R}(\langle T_i, C_B, S \rangle, Q)$ and $E_{K_T}(r)$, where *r* is a random session-key.
2. Root decrypts $E_{K_R}(\langle T_i, C_B, S \rangle, Q)$ to extract $\langle T_i, C_B, S \rangle$. This packet was signed by Alice in step #5. Root removes C_B from the packet, invalidating the signature and forwards $\langle T_i, S \rangle$ and $E_{K_T}(r)$ to Trent, along with the signature on the original packet.
3. Trent proceeds as before, except that he does not attempt to verify the signature. Instead he computes the one-way hash value h which gave rise to the signature.
4. Trent determines the answer A to the queries $S \cap Q$, sending h and $E_r(A)$ to the root.

5. If $h = H(\langle T_i, C_B, S \rangle)$ then the root sends $E_{K_B}(E_r(A))$ to Bob. Otherwise the authorization is invalid, and an error message is sent.

This arrangement ensures that Trent does not know the site requesting query authorization but knows the user involved, while the root knows the site but not the user. The answer to the queries are encrypted in a session-key generated by Bob, so that only Bob can read the results.

4.1 Conclusion

In this paper we proposed a new framework for WWW client authentication protocols. A distinction was made between the problem of associating the user with a persistent identity across sessions and the problem of providing correct information about the user to the site. The baseline for authentication is to create the sense of persistent for the user. This allows the user to function in a completely anonymous mode, not even allowing for a correlation of identity between two sites. Unlike using digital certificates, this protocol can be used freely everywhere without compromising the privacy of the user. It is up to the user to go beyond anonymity and disclose personal information as necessary. The persistent identity allows the site to collect and index all the information under a single entry. This paradigm forces the site to revise its definition of *sameness* and *identity*: the attribute-based conception—identity defined by personal characeteristics which are unique to the user— is abandoned in favor of associating the user with a history of interactions on the site. The protocol only proves that the user is the *same* person who visited at some point in the past, combining two apparently incompatible goals: authentication and anonymity. The only information revealed is a site-specific public key, which effectively becomes an alias for the user. The requirements of this framework are logical consequences of the requirements in the application domain, the fundamental asymmetry between the user and the server—mobile vs. static, privacy concerns vs. information needs. Finally we argued that it is possible to integrate the first protocol into the existing WWW security architecture. Since the two problems are independent, any conventional authentication system can be executed on top of this layer to prove user credentials.

Acknowledgements:

The author would like to thank Prof. Robert H. Morris, Sr. for supervising this thesis.

References:

[1] T. Berners-Lee, R. Fielding, H. Frystyk. RFC 1945: “Hypertext Transfer Protocol HTTP/1.0,” May 1996.
[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee. RFC 2068 “Hypertext Transfer Protocol HTTP/1.1,” January 1997.
[3] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, L. Stewart. RFC 2069 “An Extension to HTTP : Digest Access Authentication” January 1997
[4] D. Kristol, L. Montulli. RFC 2109: “HTTP State Management Mechanism,” February 1997.
[5] T. Berners-Lee, L. Masinter, M. McCahill. RFC 1738: “Uniform Resource Locators,” December 1994.
[6] N. Borenstein, N. Freed. RFC 1521: “MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies,” September 1993.
[7] R. Rivest, RFC 1321: “The MD5 Message-Digest Algorithm,” April 1992.
[8] G. Bossert, S. Cooper, W. Drummond. RFC 2084: “Considerations for Web Transaction Security,” January 1997.
[9] L. Stein, “World Wide Web Security FAQ,” January 1997. http://www.w3.org/Security/faq/www-security-faq.html
[10] B. Schneier, “Applied Cryptography” 2nd edition, Wiley 1994
[11] D. Stinson, “Cryptography: Theory and Practice,” CRC Press, March 1995.
[12] S. Garfinkel, G. Spafford “Practical UNIX and Internet Security,” 2nd edition. O’Reilly & Associates, April 1996.
[13] S. Garfinkel, G. Spafford “Web Security and Commerce,” O’Reilly & Associates, June 1997.
[14] L. Stein, “Web Security : A Step-By-Step Reference Guide,” Addison-Wesley, January 1998.

[15] A. Tanenbaum, "Computer Networks" Prentice-Hall, August 1988
[16] D. Comer, "Internetworking with TCP/IP: Principles, Protocols and Architecture" Volume 1, 3rd edition. Prentice Hall, April 1995.
[17] D. Raggett, A. Le Horse, I. Jacobs, editors. "HTML 4.0 Specification," W3C Recommendation, December 1997. http://www.w3.org/TR/REC-html40/
[18] National Institute of Standards and Technology, *NIST FIPS PUB 186* "Digital Signature Standard", U.S. Department of Commerce, May 1994.
[19] National Institute of Standards and Technology, *NIST FIPS PUB 180* "Secure Hash Standard", U.S. Department of Commerce, May 1993.
[20] National Computer Security Center, "Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria," NCSC-TG-005 Version 1, July 1987.
[21] D.V Klein "Foiling the Cracker: A Survey of, and Implications to, Password Security", *Proceedings of the USENIX UNIX Security Workship*, Aug 1990.
[22] C. Ellison, "Establishing Identity Without Certification Authorities," 6th USENIX Security Symposium, July 1996.
[23] S.C. Pohlig and M.E. Hellman, "An Improved Algorithm for Computing Logarithms in GF(p) and Its Cryptographic Significance," *IEEE Transactions on Information Theory*. v.24 n1 Jan 1978.
[24] C. P. Schnorr, "Efficient Signature Generation for Smart Cards", *Advances in Cryptology—CRYPTO 89 Proceedings*,Springer-Verlag 1990.
[25] T. ElGamal, "A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms," *Advances In Cryptology: Proceedings of CRYPTO 84*, Springer-Verlag. 1985.
[26] P. Hensley, M. Metral, U. Shardanand, D. Converse, M. Myers. "Proposal for an Open Profiling Standard" http://www.w3.org/TR/NOTE-OPS-FrameWork.html, June 1997
[27] J. Reagle, "P3P and Privacy on the Web FAQ." http://www.w3.org/P3P/P3FAQ.html, October 1997.
[28] L. F. Cranor, J. Reagle. "Designing a Social Protocol: Lessons Learned from the Platform for Privacy Preferences Project," http://www.w3.org/P3P, September 1997.
[29] A. Freier, P. Karlton, P. C. Kocher "The SSL Protocol: Version 3.0," Transport Layer Security Working Group. http://www.netscape.com/eng/ssl3/draft302.txt, November 1996.

Glossary of Acronyms:

BNF: Backus-Naur Form
CA: Certification Authority
DNS: Domain Name System
DSA: Digital Signature Algorithm
DSS: Digital Signature Standard
HTML: Hypertext Markup Language
HTTP: Hypertext Transfer Protocol
IP: Internet Protocol
ITAR: International Trafficking and Arms Regulations
MD5: Message Digest 5 algorithm
MIME: Multipurpose Internet Mail Extensions
RFC: Request For Comments
RSA: Rivest-Shamir-Adleman algorithm
SHA, SHA-I: Secure Hash Algorithm
SSL: Secure Sockets Layer
TCP: Transmission Control Protocol
URI: Universal Resource Identifier
URL: Uniform Resource Locator
W3C: World Wide Web Consortium
WWW: World Wide Web