On Secure and Pseudonymous Client-Relationships with Multiple Servers

Eran Gabber, Phillip B. Gibbons,
David M. Kristol, Yossi Matias, and Alain Mayer
Bell Labs, Lucent Technologies

This paper introduces a cryptographic engine, *Janus*, that assists clients in establishing and maintaining secure and pseudonymous relationships with multiple servers. The setting is such that clients reside on a particular subnet (e.g., corporate intranet, ISP) and the servers reside anywhere on the Internet. The Janus engine allows for each client-server relationship to use either weak or strong authentication on each interaction. At the same time, each interaction preserves privacy by neither revealing a client's true identity ("modulo" the subnet) nor the set of servers with which a particular client interacts. Furthermore, clients do not need any secure long-term memory, enabling scalability and mobility. The interaction model extends to allow servers to send data back to clients via e-mail at a later date. Hence, our results complement the functionality of current network anonymity tools and remailers.

The paper also describes the design and implementation of the Lucent Personalized Web Assistant (LPWA), which is a practical system that provides secure and pseudonymous relations with multiple servers on the Internet. LPWA employs the *Janus* function to generate site-specific personæ, which consist of alias usernames, passwords and e-mail addresses.

Categories and Subject Descriptors: K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*cryptographic controls and authentication*

General Terms: algorithms, experimentation, security

Additional Key Words and Phrases: anonymity, persistent relationship, pseudonym, mailbox, Janus function

---

Affiliation: Bell Labs, Lucent Technologies
Address: 600 Mountain Ave., Murray Hill, NJ 07974
E-mail: {eran, gibbons, dmk, matias, alain}@research.bell-labs.com
Name: Yossi Matias
Affiliation: Computer Science Dept., Tel-Aviv University
Address: Tel-Aviv 69978 Israel
E-mail: matias@math.tau.ac.il.

1. INTRODUCTION

We consider the following problem: there is a set of clients located on a particular subnet and a set of servers on the Internet. For example, the set of clients could be employees on a company's intranet or subscribers of an ISP and the servers could be Web-sites. See Figure 1, where the c_i are clients and the s_j are servers. A client wishes to establish a persistent relationship with some (or all) of these servers, such that in all subsequent interactions (1) the client can be recognized and (2) either weak or strong authentication can be used. At the same time, clients may not want to reveal their true identity nor enable these servers to determine the set of servers each client has interacted with so far (establishing a dossier). This last property is often called *pseudonymity* to denote persistent anonymity. Equivalently, a client does not want a server to infer through a relationship more than the subnet on which the client is located, nor to connect different relationships to the same client. This paper introduces a client-based cryptographic engine, which allows a client to efficiently and transparently establish and maintain such relationships using a single secret passphrase. Finally, we extend our setting to include the possibility of a server sending data via e-mail to a client.

We consider the specification and construction of a cryptographic function that is designed to assist in obtaining the above goal. Such a function needs to provide a client, given a *single passphrase*, with either a *password* (weak authentication) or a *secret key* (strong authentication) for each relationship. Furthermore, a *username* might be needed as well, by which a client is (publicly) known at a server. Such passwords, secret keys, and usernames should neither reveal the client's true identity nor enable servers to establish a dossier on the client. We name such a cryptographic function (engine) the *Janus* function (engine). We will briefly review arguments why simple choices for the Janus function, such as a collision-resistant hash function, are not quite satisfactory for our purposes and consequently, we will show a Janus function that is more robust. We will also show how to implement a mailbox system on the client side, such that a server can send e-mail to a client without requiring any more information than for client authentication.

We implemented a practical system called the Lucent Personalized Web Assistant (LPWA) that provides pseudonymous persistent relations between users and web-sites on the Internet. LPWA employs the *Janus* function to generate site-specific personae for a given user. The personae consist of alias usernames, passwords and e-mail addresses.

1.1 Related Work and Positioning of Our Work

Network anonymity is being extensively studied (see, e.g., [Pfitzmann and Waidner 1986; Goldberg et al. 1997]). For example, Simon in [Simon 1996] gives a precise definition for an *Anonymous Exchange Protocol* allowing parties to send individual messages to each other anonymously and to reply to a received message. Implementation efforts for approximating anonymous networks are being carried out by several research groups (e.g., anonymous routing [Syverson et al. 1997] and anonymous Web traffic [Syverson et al. 1997; Reiter and Rubin 1998]). Besides that, there are several anonymous remailers available for either e-mail communication (see, e.g., [Goldberg et al. 1997; Gulcu and Tsudik 1996; Bacard ; Engelfriet]) or

Web browsing (see, e.g., [Ano]). We will discuss some of these in more detail later.

We view our goal as *complementary*: All of the above work tries to find methods and systems to make the Internet an (approximately) anonymous network. This is a hard task and consequently the resulting tools are rather difficult to use and carry some performance penalties. We focus on a method for assisting a client to interact with multiple servers easily and efficiently, such that the server cannot infer the identity of the clients among all clients in a given subnet, but at the same time the client can be recognized and authenticated on repeat visits. We do not address communication between a subnet and a server. Consequently, a server can easily obtain the particular subnet in which a client is located. In many cases, this degree of anonymity is sufficient, for example, if the client is a subscriber of a large ISP, or an employee of a large company. In the language of Reiter and Rubin [Reiter and Rubin 1998], the anonymity of such a client is somewhere between *probable innocence* and *beyond suspicion*. Alternatively, our method can be used in conjunction with existing remailers to enable a client to interact with a server without revealing the particular subnet. We elaborate on this point in Section 2 for client-initiated traffic and in Section 4.3 for server-initiated traffic. The work closest in spirit to the Janus engine are the visionary papers of Chaum [Chaum 1981; Chaum 1985] on *digital pseudonyms*.

In [Gabber et al. 1997], we described the design and implementation of a Web-proxy which assists clients with registering at multiple Web-servers. In this paper, we focus on a new, simpler, and *correct* construction of the Janus engine, a new and different method of conveying anonymous e-mail that greatly reduces the required trust in the intermediary, and a discussion of moving features to shift trust from a proxy to the client's machine. The latter allows, for example, a Janus engine to be integrated with the P3P proposal, giving clients the power to use pseudonymous P3P personæ. (See Section 5.) Thus, our methods and design are applicable to a variety of client-server interactions, well beyond the proxied Web browsing for server registration of [Gabber et al. 1997].

Outline: In Section 2 we describe our interaction model and our function requirements. Section 3 contains a detailed description of the Janus function. Section 4 extends the model of interaction to allow servers to send data to clients' anonymous mailboxes. Section 5 presents various applications and configurations and discusses some of the trade-offs involved. Finally, Section 6 contains an overview of the Lucent Personalized Web Assistant (LPWA), an application of the Janus function. The Appendix provides more details on the design and implementation of LPWA.

2. MODEL AND SPECIFICATIONS

In this section, we present the framework for interaction between clients and servers, and the way in which the Janus engine is incorporated within such interaction. There is a set of clients $\mathcal{C} = \{c_1, c_2, \ldots, c_N\}$ and a set of servers $\mathcal{S} = \{s_1, s_2, \ldots, s_M\}$. Each client can interact with any server. Interaction can take place in one of the following two ways:

—*Client-initiated*: A client c_i decides to contact a server s_j. The server s_j requires c_i to present a username and a password (secret shared key) at the beginning of this interaction to be used for identification and weak (strong) authentication on

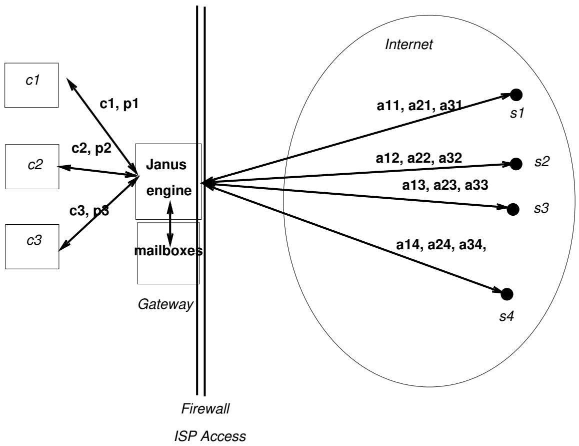

Fig. 1. Client Server Configuration with Janus Engine on the Gateway

repeat visits.

—*Server-initiated*: A server s_j decides to send some data to a client c_i which has contacted s_j at some earlier time (using the client's username).

Individual clients may wish to remain anonymous in the above interaction; i.e., a client does not want to reveal her real identity c_i to a server (beyond the particular subnet on which c_i is located).

Client-initiated interaction: A client c_i, on a first visit, presents to a server s_j an *alias* $a_{i,j}$, which includes a username and either a password or a key. On repeat visits a client simply presents the password again for weak authentication or uses the key with a message authentication code (MAC) for strong authentication (see [Matias et al. 1997]). We would like the alias $a_{i,j}$ to depend on the client c_i, the server s_j, and a secret client passphrase p_i. Since we want this translation of names to be computable, we define a function which takes c_i, p_i and s_j, and returns an alias $a_{i,j}$. This function is called the *Janus function*, and is denoted $\mathcal{J}$. In order to be useful in this context, the *Janus function* has to fulfill a number of properties:

(1) *Form properties:* For each server, $\mathcal{J}$ provides each client with a *consistent* alias, so that a client, by giving her unique identity and passphrase, can be recognized and authenticated on repeat visits. $\mathcal{J}$ should be *efficiently* computable given c_i, p_i, and s_j. The alias $a_{i,j}$ needs to be accepted by the server, e.g., each of its components must have appropriate length and range.

(2) *Secrecy of passwords/keys:* Alias passwords/keys remain secret at all times. In particular, an alias username does not reveal information on any alias password/key.

(3) *Uniqueness of aliases among clients & Impersonation resistance:* Given a client's identity and/or her alias username on a server s_j a third party can guess the corresponding password only with negligible probability. Moreover, the distribution of the alias usernames should be such that only with negligible probability do two different users have the same alias username on the same server.

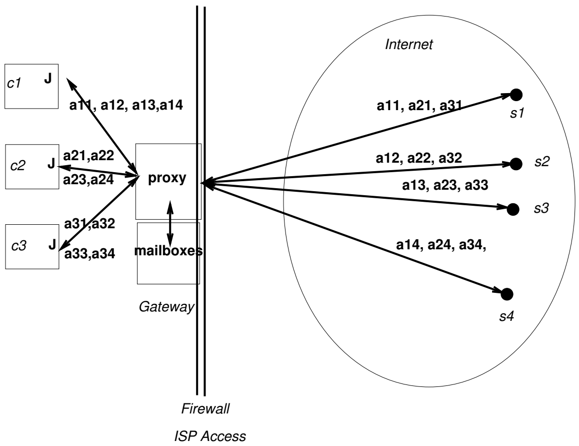

Fig. 2. Client Server Configuration with Local Janus Engine

(4) *Anonymity / Uncheckability of clients:* The identity of the client is kept secret; that is, a server, or a coalition of servers, cannot determine the true identity of the client from her alias(es). Furthermore, it is not checkable whether a particular client is registered at a given server.

(5) *Modular security & Protection from creation of dossiers:* An alias of a client for one server does not reveal any information about an alias of the same client for another server. This also implies that a coalition of servers is unable to build a client's profile (dossier) based on the set of servers with which he/she interacted by simply observing and collecting aliases.

One possible physical location to implement the Janus function is on the gateway. See Figure 1, where we refer to the implementation as the *Janus engine*. Clients provide their identity c_i and secret passphrase p_i to the gateway, where the translation takes place. An alternative location for the Janus engine is on each client's machine, as depicted in Figure 2, where the locally generated aliases are sent to the server via the gateway. See Section 5 for a discussion of trade-offs. The following property is of practical significance, as it provides robustness against the possibility to recover privacy-sensitive information "after the fact":

(6) *No storage of sensitive data:* When a client is not interacting with a server, the Janus engine does not maintain in memory any information that may compromise the above properties of the Janus function. This excludes the simple approach of implementing a Janus function by a look-up table.

Consequently, an entity tapping into a (gateway) machine on the subnet cannot infer any useful information, unless it captures a client's passphrase (which is never transmitted in Figure 2). Additionally, a client can use different Janus engines within her subnet, given that she remembers her passphrase (*mobility*).

If a client desires to hide her subnet from a server, she can easily combine our method with other anonymity tools. For example, if she contacts a server via the Web (HTTP), she can use either Onion Routing [Syverson et al. 1997] or

Crowds [Reiter and Rubin 1998]. In the first case, the connection from the gateway to the server is routed and encrypted similar to the methods used by type I/II remailers (see also Section 4.3) and in the second case her connection is "randomly" routed among members (on different subnets) of a crowd.

Server-initiated interaction: A server knows a client only by the alias presented in a previous, client-initiated interaction. We allow a server s_j wishing to send data to client c_i, known to it as $a_{i,j}$, to send an e-mail message to the corresponding subnet, addressed to the username component u of $a_{i,j}$. The message is received by the Janus engine, see Figure 1, which will make sure that the message is delivered to the appropriate client, or is stored by the gateway, until a local Janus engine retrieves the messages, as in Figure 2. Our scheme of storing mailboxes maintains forward secrecy. More details are in Section 4, where we also show how server-initiated interaction can be combined with pseudonymous remailers.

3. THE JANUS FUNCTION

In this section we present the Janus function in detail. We first develop our requirements, then discuss some possible constructions.

The Setting of the Janus-function: A client inputs her identity c_i, her secret passphrase p_i, the identity of the server s_j, and a tag t indicating the purpose of the resulting value. Depending on this tag, the Janus function returns either an alias-username $a^u_{i,j}$ for the user c_i on the server s_j or the corresponding password $a^p_{i,j}$. In this section we use the two tags u and p, but we can easily extend the function, by adding additional tags, to generate secret values for other purposes (see also [Matias et al. 1997]). For example, in Section 4 we extend the Janus function to a third tag, m, for the purpose of anonymous mailboxes.

Adversarial Model: We assume that a client c_i does not reveal her passphrase p_i to anyone (other than the Janus engine). However, we allow that an adversary E can collect pairs $(a^u_{i,j}, a^p_{i,j})$ and the corresponding server names s_j. Note that registered alias usernames may be publicly available on some servers and that we cannot assume that all servers can be trusted or that they store the passwords securely. In some cases it might even be possible to deduce a client name c_i (e.g., from the data exchanged during a session, or simply because the client wishes to disclose her identity), and we also have to assume that a chosen message attack is possible (e.g., by suggesting to a client c_i to register on a specific server). Roughly speaking, we will require that an adversary does not learn more useful information from the Janus function than he would learn if the client chose all her passphrases and aliases randomly.

3.1 Janus function specifications

Definition 1. *We say that a client c_i is* **corrupted** *if the adversary E has been able to find p_i. We say that c_i is* **opened** *with respect to a server s_j if the pair $(a^u_{i,j}, a^p_{i,j})$ has been computed and used. (Note that if c_i has been opened with respect to a server s_j then an adversary E may know only $(a^u_{i,j}, a^p_{i,j})$ but not necessarily c_i.) We say that c_i has been* **identifiably opened** *with respect to a server s_j if an adversary knows $(a^u_{i,j}, a^p_{i,j})$ together with the corresponding c_i.*

Let $\mathcal{C}$ be the set of clients, $\mathcal{S}$ be the set of servers, $\mathcal{P}$ be the set of allowable client secret passwords, $\mathcal{A}_{\mathcal{U}}$ be the set of allowable alias usernames, and $\mathcal{A}_{\mathcal{P}}$ be the set of allowable alias passwords. Let k be the security parameter of our Janus function meaning that a successful attack requires about 2^k operations on average. Let the Janus function be $\mathcal{J} : (\mathcal{C} \times \mathcal{S} \times \mathcal{P} \times \{u, p, m\}) \mapsto \{0,1\}^k$.

Since usernames and passwords normally consist of a restricted set of printable characters, we also need two functions that simply convert general k-bit strings into an appropriate set of ASCII strings. Thus let $\pi_U : \{0,1\}^k \mapsto \mathcal{A}_{\mathcal{U}}$ and $\pi_P : \{0,1\}^k \mapsto \mathcal{A}_{\mathcal{P}}$ be two injective functions that map k-bit strings into the set of allowable usernames and passwords.

Let $c_i \in \mathcal{C}$ and $p_i \in \mathcal{P}$. The client's identity $a^u_{i,j}$ and password $a^p_{i,j}$ for the server s_j are then computed by

$$\begin{aligned} a^u_{i,j} &:= \pi_U(\mathcal{J}(c_i, s_j, p_i, u)) \\ a^p_{i,j} &:= \pi_P(\mathcal{J}(c_i, s_j, p_i, p)). \end{aligned}$$

The two functions π_U and π_P are publicly known, easy to compute, and, we may assume, easy to invert. Thus knowing $\pi_U(x)$ of some x is as good as knowing x. In particular if an adversary can guess $\pi_U(x)$ then he can guess x with the same probability.

Following our adversarial model, the Janus function has to satisfy the following requirement:

(1) *Secrecy:* Given a server s_j, an uncorrupted and not identifiably opened client c_i and $t \in \{p, u, m\}$, the adversary E cannot find $\mathcal{J}(c_i, s_j, p_i, t)$ with nonnegligible probability even under a chosen message attack, that is under the assumption that the adversary can get $\mathcal{J}(c_i, s_{j'}, p_i, t')$ for any $s_{j'} \neq s_j$ or $t \neq t'$.

(2) *Anonymity:* Given a server s_j, two uncorrupted clients $c_i, c_{i'}$ that are not opened with respect to s_j and $t \in \{p, u\}$. Then an adversary cannot distinguish $\mathcal{J}(c_i, s_j, p_i, t)$ from $\mathcal{J}(c_{i'}, s_j, p_{i'}, t)$ with nonnegligible probability even under a chosen message attack, that is under the assumption that the adversary can get $\mathcal{J}(c_{i''}, s_{j'}, p_{i''}, t')$ for any list of arguments not used above.

Note that the two requirements are indeed different. For example if we were to implement the function $\mathcal{J}$ using a digital signature scheme, i.e., $\mathcal{J}(c_i, s_j, p_i, t) = \mathrm{sig}_{p_i}(c_i||s_j||t)$, then the first requirement would be satisfied, but not the second one, since the client's identity could be found by checking signatures. On the other hand a constant function satisfies the second requirement, but not the first one.

Our requirements are stated in a rather general form. In particular, the first requirement states that no result of the Janus function can be derived from other results. This implies the secrecy of passwords, impersonation resistance and modular security.

3.2 Possible Constructions for $\mathcal{J}$

Assume that ℓ_c is the maximal bit length of c_i, ℓ_s the maximal length of s_j, ℓ_p the maximal length of p_i and ℓ_t the number of bits required to encode the tag t. Throughout this section we will assume that all inputs c_i, s_j, p_i are padded to their maximal length. This will assure that the string $c_i||s_j||p_i||t$ is not ambiguous.

Given the function specification, an ideal construction would be via a cryptographic pseudorandom function $f : \{0,1\}^{\ell_c+\ell_s+\ell_p+\ell_t} \mapsto \{0,1\}^k$. Unfortunately, there are no known implementations of pseudorandom functions. Typically, they are approximated via either strong hash functions or message authentication codes (MAC), even though, strictly speaking, the definitions of these primitives do not require them to be pseudorandom. In the following sections, we are going to examine both options and give some justifications for preferring a MAC-based solution over other tempting constructions.

3.2.1 *Using hash functions.* One possible attempt might be to use the hash of the inputs $h(c_i||s_j||p_i||t)$ as our function. However, hash functions are not designed to keep their inputs secret. Even if it is hard to invert the hash function for a given input, it might still be possible to derive p_i given $h(c_i||s_j||p_i||t)$ for many different servers s_j. A hash function that is weak in that respect can for example be found in [Anderson 1993]. Some apparently better constructions for keyed functions based on hash functions have been proposed (e.g., MDx-MAC [Preneel and van Oorschot 1995]). But our requirements are quite different from the goals of these constructions. Therefore, we decided not to use hash functions for our Janus function.

3.2.2 *MACs.* A much more promising approach is the use of message authentication codes (MACs). In particular if $\mathrm{MAC}_K(x)$ denotes the MAC of the message x under the key K then we can define a potential Janus function as

$$\mathcal{J}(c_i, s_j, p_i, t) = \mathrm{MAC}_{p_i}(c_i||s_j||t).$$

This approach has the advantage that some of our requirements are already met. In particular if the MAC is secure then the secrecy of passwords and impersonation resistance for the Janus function are implied. Other requirements, like consistency, efficient computation of the function, single secret and acceptability, are just consequences of the actual implementation of the Janus function and the mappings π_U and π_P. The only additional requirement is the anonymity of clients.

To this end, we consider the following result of Bellare, Kilian, and Rogaway ([Bellare et al. 1994]): Let $x = x_1, \ldots, x_m$ be a message consisting of m blocks x_i of size ℓ bits. Given a block cipher $f_K : \{0,1\}^\ell \mapsto \{0,1\}^\ell$ where K denotes the key, define the CBC-MAC by

$$\mathrm{MAC}_K(x) = f_K(\cdots f_K(f_K(x_1) \oplus x_2) \cdots \oplus x_m).$$

Assume that an adversary can distinguish a MAC_K from a random function with an advantage ϵ by running an algorithm in time t and making q queries to an oracle that evaluates either $\mathrm{MAC_K}$ or the random function. Then the adversary can distinguish f_K from a random function running an algorithm of about the same size and time complexity having an advantage of $\epsilon - q^2m^2 2^{-\ell-1}$. Hence, if we use CBC-MACs, then anonymity is just a consequence of [Bellare et al. 1994].

If the underlying block cipher f_K behaves like a pseudorandom function then the above result shows that a birthday attack is almost the best possible attack. In particular an attacker can not do much better than collecting outputs of the function and hoping for an internal collision, i.e. two messages x, y such that $f_K(f_K(\cdots f_K(f_K(x_1) \oplus x_2) \cdots \oplus x_{i-1}) \oplus x_i) = f_K(f_K(\cdots f_K(f_K(y_1) \oplus y_2) \cdots \oplus$

$y_{i-1}) \oplus y_i)$ for some $i < m$. In that case the attacker would know that replacing the first i blocks in any message starting with $x_1, \ldots, x_i$ by $y_1, \ldots, y_i$ would result in another message having the same hash value.

We thus caution that a block cipher with ℓ-bit block size should not be used if an attacker can collect about $2^{\ell/2} m^{-1/2}$ MACs. Concretely, block ciphers having 64-bit blocks, such as DES, triple-DES, or IDEA [Lai and Massey 1991] should not be used if it is feasible for an attacker to collect about 2^{32} samples, thus giving only marginal security to the overall scheme. However, newer block ciphers, such as SQUARE [Daemen et al. 1997] and one variant of RC5 [Rivest 1994] have 128-bit block sizes and are therefore more suitable in this case.

4. AN ANONYMOUS MAILBOX SYSTEM

We will first summarize the history of anonymous remailers, then describe our anonymous mailbox system, and finally discuss how enhanced privacy can be achieved by using our mailbox system in conjunction with remailers.

4.1 Brief History of Anonymous E-mail

Tools for anonymous e-mail communication have been around for a few years now. (See, e.g, [Goldberg et al. 1997; Bacard ; Gulcu and Tsudik 1996; Engelfriet].) Early anonymous remailers (Type 0, e.g., `Anon.penet.fi`) accepted e-mail messages by a user, translated them to a unique ID and forwarded them to the intended recipient. The recipient could use the ID to reply to the sender of the message. The level of security of this type of remailer was rather low, since it did not use encryption and kept a plain text (translation) database. A next (and still current) generation of remailers (Type I, Cypherpunk remailers) simply takes a user's e-mail message, strips off all headers and sends it to the intended recipient. The user can furthermore encrypt the message before sending it and the remailer will decrypt the message before processing it. For enhanced security, a user can *chain* such remailers. In order to use a chain $r_1 - r_2$ of remailers, a user first encrypts the message for r_2 and then for r_1. (See also the efforts on Onion Routing, [Syverson et al. 1997].) Still, even such a scheme is susceptible to traffic analysis, spam and replay attacks. Mixmaster remailers (Type II) are designed to withstand even these elaborate attacks. This kind of remailer yields a more untraceable way of sending messages, but it gives no way to reply to a message. This gives rise to "pseudonymous / nym" remailers, which, in a nutshell, work as follows: A user chooses a pseudonym (nym), which has to be unused (at that remailer). Then the user creates a public/private key pair for that nym. When sending a message, the user encrypts with the server's public key and signs a message with her private key. The recipient can reply to the message using the nym. Some remailers store the message, and the original sender can retrieve this mail by sending a signed command to the remailer. Other remailers directly forward the message by using a "reply block," an encrypted file with the user's real e-mail.

The ultimate goal of all these remailers is to enable e-mail communication as if the Internet were an anonymous network. This is a very hard task and consequently these tools induce a performance penalty and are rather difficult to use.

4.2 Anonymous Mailboxes

In this section, we show how to construct an anonymous mailbox system within our model. As before, we assume that the users are in a particular subnet. Our goal is to provide these users (clients) with a transparent way to give e-mail addresses to outside parties (servers) that maintains the properties of the aliases (anonymity, protection from dossiers, etc.). For example, a client might want to register at a (Web-site) server for mailing-lists, personalized news, etc. Such an e-mail address provides a server with the means to initiate interaction with a client by sending an e-mail message to the client.

We first consider a setting with the Janus engine on the gateway (Figure 1). We propose that the Janus engine computes "$a^u_{i,j}$@subnet-domain" as c_i's e-mail address to be used with s_j. We further suggest storing a mailbox for each such active (c_i, s_j) pair on the subnet's gateway, such that an owner of a mailbox is only identified by the respective alias. Messages are stored in these mailboxes, passively awaiting clients to access them for retrieval. We require that (1) given a previous, client-initiated interaction, a server can send data to the mailbox created for the (client, server) pair, (2) the Janus engine (upon being presented with (c_i, p_i)) lets a client c_i retrieve the messages in *all* of her mailboxes without remembering a corresponding list of servers, (3) neither the Janus engine nor the mailboxes compromise the property that the server must not store sensitive data (see Section 2). In particular, the knowledge of e-mail headers of messages (which contain $a^u_{i,j}$ and s_j) does not reveal client identity c_i. We show that the Janus function can be used to overcome the apparent contradiction of requirements (2) and (3). Note that the secrecy of the actual data stored *within* a mailbox is an orthogonal issue and can be solved, for example, by using PGP. For the setting of a Janus engine on each client (Figure 2), most of the scheme above remains unchanged with one important exception: When a client wants to retrieve her messages, the local Janus engine tells the gateway which mailboxes to access and hence p_i is never revealed to the gateway.

Data Structures for (c_i, s_j)-mailbox: Let $a^m_{i,n_i} = \pi_M(\mathcal{J}(c_i, n_i, p_i, m))$, where we use the tag m for the "mail index", n_i an integer indexing c_i's mailboxes, and π_M a corresponding injective function to map the output of $\mathcal{J}$ into a suitable range. We explain the extensions in turn below. The following record R is stored with the (c_i, s_j)-mailbox. R has three fields: (1) $R_{alias} = a^u_{i,j}$, (2) $R_{index} = a^m_{i,n_i}$, (3) $R_s = s_j$. The argument n_i in (2) indicates the index of the mailbox created for client (c_i, p_i) and server s_j. The record R (and consequently the mailbox) can be accessed both via R_{alias} or R_{index}. The R_{alias} field contains the name of the mailbox that is used for messages sent from s_j to the client c_i. A second data structure, stored together with the mailboxes, holds a counter C_i for each of the clients (c_i, p_i). C_i is the number of mailboxes the client (c_i, p_i) has established so far. These counters are initialized to 0. Note that $0 < n_i \leq C_i$. The counter itself is indexed by $a^m_{i,0}$, so that the Janus engine, upon being presented with (c_i, p_i), can easily find it.

Creating a Mailbox: Whenever the client c_i instructs the Janus engine to give out an e-mail address for s_j, the engine checks if a record R with $R_{alias} = a^u_{i,j}$

already exists in the first data structure. If it does not exist, then the engine retrieves the counter C_i by accessing the second data structure with the key $a^m_{i,0}$. If no C_i is found, it is initialized to zero. The counter C_i is incremented and a new record R is created, with: $R_{alias} = a^u_{i,j}$, $R_{index} = a^m_{i,C_i}$, $R_s = s_j$. Afterwards, the engine stores the updated value of C_i in the second data structure with key $a^m_{i,0}$. Finally, the Janus engine creates a new mailbox under the name of R_{alias}.

Retrieving Mail: Whenever client c_i connects to the Janus engine, it will retrieve all of c_i's accumulated e-mail messages. The engine first retrieves the counter C_i by accessing the second data structure with the key $a^m_{i,0}$. Then it retrieves all records R with $R_{Index} = a^m_{i,\gamma}$ for $0 < \gamma \leq C_i$. For each such record R, Janus retrieves the corresponding mailbox and presents it, together with R_s, to the client c_i.

The above scheme constitutes a service to store mail for any client and allows a client c_i to retrieve all her mail upon presenting (c_i, p_i). If c_i is uncorrupted and not identifiably opened with respect to server s_j, then adversary E cannot do better than guessing the identity of the corresponding mailbox. Furthermore, given any two such mailboxes, E cannot do better than guessing whether they have the same owner. This is a simple consequence of the properties of the Janus function $\mathcal{J}$.

The above system can easily be extended to allow a client to actively send e-mail to servers using the Janus engine to generate a different address depending on the server.

4.3 Combining our Solution with Pseudonymous Remailers

When we allow the adversary to execute more elaborate attacks (than we introduced in our model of Section 3), such as eavesdropping or traffic analysis, a client visiting several servers within a short period of time, might become vulnerable to correlation and building of dossiers (albeit not enough to compromise anonymity). Also, if a client happens to reside on a small subnet, the subnet's population might not be large enough to protect her identity. In these cases, it makes sense to combine our method with anonymous remailers or routing (for Web traffic) for enhanced protection: We can view the Janus engine as a client's "front end" to a pseudonymous remailer. It computes the different nyms on a client's behalf and presents them to the remailer. It manages all the client's mailboxes and presents incoming messages to the client. It also manages a client's public/private keys for each nym. Furthermore, even the remailer closest to the client (of a possible chain) can neither infer the client's identity nor correlate different aliases. All this remailer sees (when decrypting a reply block) is the client's alias e-mail address.

5. TRADE-OFFS AND APPLICATIONS

In this section we examine the trade-off between the configurations corresponding to Figure 1, which we refer to as the *gateway approach* and to Figure 2, which we refer to as the *local approach*. We then present a few concrete applications.

5.1 Local vs. Gateway

The basic advantage of the *local approach* is that the Janus functionality is pulled all the way to the client's machine, minimizing outside trust. Thus, the client does not have to reveal her secret passphrase to another machine (the gateway). A client also has the flexibility to choose a mailbox location outside her own subnet, minimizing the trust in the subnet (e.g., the client's ISP). There are also a number of scenarios where the Janus functionality is required to be on the client's machine: For example, in the realm of Web browsing, the Janus engine can be integrated with the *Personal Privacy Preferences* (P3P) standard proposal to make a P3P *persona* (see [Ackerman et al. 1997]) pseudonymous: P3P enables Web sites to express privacy practices and clients to express their preferences about those practices. A P3P interaction will result in an agreement between the service and the client regarding the practices associated with a client's implicit (i.e., click stream) or explicit (i.e., client answered) data. The latter is taken from data stored in a *repository* on the client's machine, so that the client need not repeatedly enter frequently solicited information. A *persona* is the combination of a set of client preferences and P3P data. Currently, P3P does not have any mechanisms to assist clients to create pseudonymous personæ. For example, a client can choose whether to reveal his/her real e-mail address, stored in the repository. If the e-mail address is not revealed, the Web-site cannot communicate with the client and if the e-mail address is indeed revealed, the Web-site has a very good indication on the identity of the visitor. Using a Janus engine provides a new and useful middle ground: The data in repository corresponding to usernames, passwords, e-mail addresses, and possibly other fields can be replaced by macros which, by calling the Janus engine, expand to different values for different Web-sites and thus create pseudonymous personæfor the client.

For the case of the *gateway approach*, we note that the Janus engine does not have to be distributed throughout the subnet. Thus, the clients do not have to download or install any software and no maintenance is required, also giving *scalability*: when the population in the subnet grows, it is easy to add gateway machines (helped by the *Forward Secrecy* property). The proxy might also provide *alias management capabilities* in the case where the gateway is for a corporate intranet: Such capabilities might allow two clients to share their aliases for all the servers, a client to transfer one or more of his/her aliases to another client, or even two clients to *selectively* share some of their aliases. For example, when going on vacation, a manager might use such functionality to have an assistant take over some of his daily correspondence. Such alias management functions have the potential to considerably simplify login account and e-mail management in big intranets. We note that to achieve this potential, state has to be added to the proxy design, which goes beyond the scope of this paper.

5.2 Applications

Web browsing. There is a growing number of web-sites that allow, or require, users to establish an account (via a username and password) before accessing the information stored on that site. This allows the web-site to maintain a user's personal preferences and profiles and to offer personalized service. The *Lucent Personalized Web Assistant* is an intermediary Web proxy that uses a Janus engine to translate a user's information (user's e-mail and passphrase) into an alias (username, password, email) for each web-site. Moreover, this alias is also used by the web-site to send e-mail back to a user. More details of this work can be found in Section 6, as well as [Gabber et al. 1997] and at `http://lpwa.com:8000/`. The intended configuration for this project is the gateway approach of Figure 1. We note that such concrete applications typically execute in conjunction with many other mechanisms. For instance, Web browsing based on the HTTP protocol interfaces, among others, with SSL for encrypting the communication and with Java and JavaScript for downloadable executables. Each such interface can potentially undermine the pseudonymity of the client-server interaction. In the case of SSL, the proxy can spoof SSL on behalf of the internal client (see [SSL]). The proxy can initiate SSL between itself and other servers and thus maintain the client's pseudonymity. Both Java applets and JavaScript scripts, when downloaded from a server by a client, can potentially obtain compromising client information. Research is being conducted which might lead to including *customizable security policies* into these languages (see [Gong et al. 1997; Anupam and Mayer 1998]). A client can then choose a policy strict enough to preserve his/her pseudonymity. Another approach is to bundle an LPWA proxy with an applet/script blocking proxy, as described, e.g., in [Martin et al. 1997]. In summary, it is necessary to consider all possible interfaces, and offer encompassing solutions to clients.

Authenticated Web-traffic. Consider a Web site which offers repeated *authenticated* personalized stock quotes to each of its subscribers. The value of a single transaction (e.g., delivery of a web-page with a customized set of quotes) does not warrant the cost of executing a handshake and key distribution protocol. A lightweight security framework for extended relationships between clients and servers was recently proposed [Matias et al. 1997]. The Janus engine provides a persistent client-side generated shared key for each server, used in application-layer primitives. Hence, no long-term secure memory is needed on the client side, enabling scalability and mobility.

6. THE JANUS FUNCTION IN OPERATION: LPWA

The Lucent Personalized Web Assistant (LPWA) is a particular instantiation of the pseudonymous client-server scheme we describe above, customized for the Web. LPWA provides consistent alias personae for people accessing personalized web sites, which require registration. Our goal was to build a system that could be readily deployed for public use. In the Appendix we will briefly present the design and implementation of the public trial version of LPWA.

LPWA has the following three functional components:

—*Persona Generator*: Generates a unique, consistent site-specific persona on demand by a user. The generator requires two pieces of identity information from a user: a *User ID*, which is a valid Internet e-mail address for the user; and a *Secret*, which serves as a universal password. Using these two pieces of information, plus the destination web-site address, the generator computes a persona for this web-site on the user's behalf. The persona consists of an alias username, alias password, and an alias e-mail address for the user at the particular site.

—*Browsing Proxy*: Increases a user's privacy by providing indirection on the TCP level and filtering on the HTTP level.

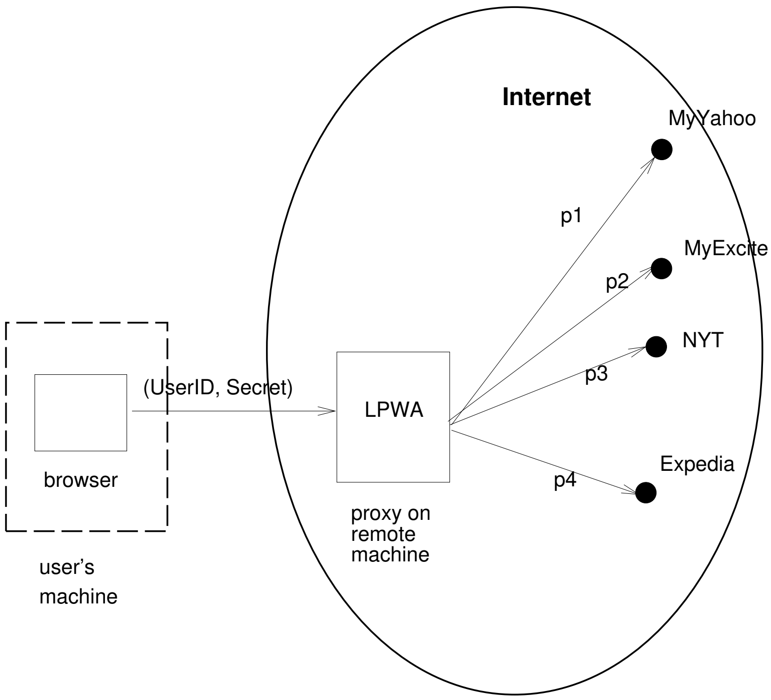

Fig. 3. LPWA HTTP proxy configuration

—*E-mail Forwarder*: Forwards mail, addressed to a persona e-mail address, to the corresponding user.

Our intention to quickly deploy a trial version prevented us from considering browser changes (no source code available at the time). Furthermore, distributing software that contains cryptographic modules posed difficulties that at the very least would delay our trial considerably. As a result, we decided to implement LPWA for a public trial using the following two components:

—An HTTP proxy server, located on our premises in Murray Hill, New Jersey, that implements both the Browsing Proxy and the Persona Generator. This configuration is depicted in Figure 3.

—A remailer, located on the same machine as the proxy server, that implements the E-mail Forwarder.

Again, in the interests of creating a demonstration site quickly, we chose to forward e-mail, rather than to implement the e-mail handling scheme described in Section 4. In e-mail forwarding, the alias e-mail is an encoded, encrypted version of the user's real email address. The forwarding engine decodes and decrypts the alias e-mail address to derive the user's real e-mail address and to forward the message to that address. Obviously this approach entails a trust relationship between the user and LPWA that is significantly different than the scheme we described earlier.

The Appendix contains a more detailed description of the design and implementation of LPWA. It also summarizes the experience from the public trial of LPWA.

7. SUMMARY

We have described a client-based cryptographic engine that allows clients to efficiently establish and maintain persistent anonymous (or *pseudonymous*) relationships with multiple servers. The relationships allow the clients to be recognized by the servers, and may employ weak or strong authentication. The cryptographic engine ensures that the clients do not reveal their true identity, and the servers cannot infer other relationships that the client may have established with other servers. We have presented the specification and construction of the *Janus* function, which can be used to establish such persistent anonymous relationships.

The *Janus* function is used in the Lucent Personalized Web Assistant (LPWA), which is a practical system that provides persistent anonymous relations between users and web sites on the Internet. LPWA generates a different persona for each web site for a given user. The persona comprises an alias username, an alias password and an alias e-mail address.

ACKNOWLEDGMENTS

Daniel Bleichenbacher provided many valuable contributions to the design of the Janus function. We are also grateful to Russell Brand for thought-provoking discussions.

APPENDIX

A. THE DESIGN AND IMPLEMENTATION OF LPWA

A.1 LPWA Usage

This section summarizes a user's interaction with LPWA. Further details will be provided in subsequent sections.

The user configures her browser's HTTP Proxy setting to use the LPWA HTTP proxy. (The current trial LPWA proxy is located at `lpwa.com`.) Subsequently, at the beginning of a browsing session, the user is presented with the LPWA start-up page. This page asks the user to supply her User ID (real e-mail address) and Secret (universal password). From that point on, LPWA is transparent while the user is browsing the Web. Whenever a web-site asks the user to supply any of a username, password, or e-mail address, the user may invoke LPWA by supplying a corresponding LPWA escape sequence. As it passes along the request to the destination web-site, LPWA recognizes these sequences, computes a persona username, password, or e-mail address specific to that web-site, and inserts them into the user's request. On repeat visits, LPWA will produce those same personæ, so when the user returns to a web-site, she is recognized as a repeat visitor. When a web-site sends a message to a persona e-mail address, the message arrives at LPWA, which then forwards the message to the corresponding user.

A.2 A Little HTTP Review

One of the design requirements of LPWA is that it should follow the HTTP standard in order to make it more widely usable. This section contains a short review of the relevant parts of the HTTP protocol that are important for the operation of LPWA.

The Hypertext Transfer Protocol (HTTP) is the controlling protocol for the Web. HTTP is a stateless protocol between a client and a server. A client (typically a web browser) connects to a server, sends a *request line* and zero or more *request headers*, and, possibly, a *message body* (such as the contents of a form), and awaits a response. The server responds with a *status line*, zero or more *response headers*, and, usually, a message body (such as a web page). After this transaction, both sides will close the connection. (More recent versions of HTTP allow for both sides to keep the connection open, but any subsequent requests are treated as logically independent.) An *HTTP proxy server* acts as a go-between, sitting between the user's web browser and the intended server (denoted the *origin server*). To the user's browser, the proxy behaves like a server; to the origin server it behaves like a client. When a browser connects to the proxy, the proxy must interpret the request and make its own request to the origin server. It must then interpret the response from the origin server and pass the response along to the browser.

A.3 Design of the LPWA HTTP Proxy

In this section, we first present our design requirements for the proxy. Then we show how the user supplied identification (User ID and Secret, as described in Section 6) is managed. Finally, we describe LPWA's filtering of privacy-sensitive HTTP header fields.

A.3.1 *Requirements.* We had several design requirements for the LPWA HTTP proxy:

(1) It should work with most already-available web browsers.
(2) It should be easy to use and should work transparently.
(3) It should be relatively easy to implement.
(4) It should be stateless.
(5) It should follow the HTTP standard, RFC 2068 [Fielding et al. 1997].

A.3.1.1 *Browsers.* We wanted to be able to build, test, and deploy LPWA quickly. These factors precluded any kind of custom web browser. Thus although the LPWA technology could be incorporated into a web browser, we deliberately chose a mechanism that would work with existing browsers.

A.3.1.2 *Easy to Use and Transparent.* If users were going to find LPWA convenient, it had to be easy to use and non-intrusive. So we made it simple to set up a browser to use LPWA, and, after the initial identification, LPWA is invisible.

A.3.1.3 *Easy to Implement.* We decided to base the LPWA proxy on the Apache Server, a public domain server produced by the Apache Group. This server is widely used, and the source code is freely available and actively supported. We found that our changes could be inserted "surgically" with modest changes to the existing code base.

A.3.1.4 *Stateless.* For both operational and privacy reasons we decided that the proxy server should retain no information about user identities. From an operational standpoint, making the server stateless meant that we could easily stop, restart, or replace the server. The system could easily recover from server or machine crashes. Furthermore, if there were a wide selection of LPWA proxies available worldwide, a user could use any one of them equally well. From a security standpoint, not keeping state information on the server reduces the threat to privacy. If identities necessarily had to be kept on the server, an intruder could possibly obtain the identity information and learn who is using the server.

Statelessness would be less important if the proxy server were to reside on an intranet's firewall, as described earlier. In that case, the server is within a trusted environment, and keeping state allows for some interesting extensions.

A.3.1.5 *Follow the HTTP Standard.* Adhering to published standards renders LPWA more widely usable, which was our goal.

A.3.2 *Management of User ID and Secret.* Given the above requirements regarding statelessness, we needed to find a way to coax a web browser to remember the user-supplied information, and to forward it to the LPWA proxy with each HTTP request. An obvious choice is to use the HTTP's `Proxy-Authorization` header. Before further discussions, we need a little review of HTTP.

A.3.3 *How to Keep State in a Stateless Proxy.* One of our design requirements was that the LPWA HTTP proxy should be stateless, which meant that the proxy could not retain the User ID and Secret for active browsing sessions from one request to the next. However, for LPWA to be as easy to use as possible, the user should have to enter her User ID and Secret at most once per session. We resolved this contradiction by inducing the browser to tag each user request with the User ID and Secret information. The LPWA proxy uses this information whenever it has to compute an alias. In all other cases this information is discarded. The next section describes the mechanism we used to tag HTTP requests.

A.3.3.1 *Using Proxy Authentication.* In HTTP, a proxy may require *user authentication*. Typically authentication is required to verify that a user is authorized to use the proxy. LPWA uses the mechanism for another purpose. A proxy demands authentication by answering an HTTP request with a response that contains an appropriate (error) status code and a response header. Upon seeing the particular status code and header, a browser presents a dialog box to the user that asks for a Username and Password for the proxy. After the user fills in the information, the browser repeats the original request, this time adding a `Proxy-Authorization` request header with the request; it contains the Username and Password. Thereafter, every request that the browser sends to the proxy includes the same `Proxy-Authorization` request header. A normal proxy would verify that the information in `Proxy-Authorization` matched some table of authorized users, but LPWA uses it differently. The fact that the `Proxy-Authorization` request header accompanies every request was exactly the kind of mechanism we needed for LPWA. (The proxy authentication Username and Password serve as the LPWA User ID and Secret.) LPWA removes the `Proxy-Authorization` request header before it forwards the request.

A.3.3.2 *The LPWA Login Process.* We modified the Apache proxy code so it would only forward requests that included a well-formed `Proxy-Authorization` header. Otherwise the proxy rejected the request, as outlined above, which induced the browser to ask the user for authentication information (User ID and Secret).

Our design of the LPWA login sequence went through four iterations. Responding to users' comments, we tried to reduce the amount of typing a user must do. However, we wanted to preserve some level of safety, because if the user enters the wrong User ID or Secret, the generated personæwill be incorrect, and they will be unable to access their personalized information. In the final version, the LPWA proxy gives the user the choice of entering the User ID and Secret twice (the "safer" method) or once (the "quicker" method).

To detect errors, we stored a cryptographic hash value of each UserID/Secret pair encountered by the proxy, so the proxy could distinguish between first time and repeat users (without inferring their identities), and to provide distinct greetings. Thus when a repeat user mistypes her UserID and Secret, the proxy greets her as a first time user, which alerts her to the mistake. The greeting page provides the user a second chance to login, so that any user so alerted can correct her mistake. This addition is a slight departure from the statelessness requirement, but was well received by LPWA users.

A.3.4 *The LPWA Proxy in Use.* After successfully logging into LPWA, a user surfs the Web transparently with respect to LPWA. But note that each HTTP request by the user and each answer by the web-site is routed through LPWA, thus providing the required *indirection*. A user explicitly invokes LPWA, whenever a persona is needed, by typing one of the following LPWA escapes:

Escape	Used for...
`\U`	(alias) username, nickname, etc.
`\P`	(alias) password
`\@`	(alias) e-mail address

Users can supply these escapes in two contexts: in HTML forms, and as identity information for HTTP *basic authentication.* Basic authentication is similar to the previously described proxy authentication, except that the origin server, instead of the LPWA proxy, demands the authentication information.

A.3.5 *Other Proxy Processing.* In the interests of enhanced privacy and security, LPWA filters HTTP request headers. Specifically,

—The `From` header, which is seldom used, but which could contain the user's real e-mail address, is removed.

—The `User-Agent` header, which can disclose information about what type of machine the user has, is trimmed to remove the platform-specific information. The latter is a potential hint for a hacker trying to break into the user's machine.

—The `Referer` [*sic*] header is removed. `Referer` contains the URL of the web page in which the URL of the current request appeared. Thus it permits a server to learn the previous page the user visited, which may contain personal information, especially if it is a user's home page, "personal favorites" page, or information about the user's organization. The problem with removing this header is that there are sites which restrict access based on the value of the `Referer` field. For example, one web-site of syndicated comic-strips restricts access to requests where the `Referer` has the value of a newspaper site. We accommodate such cases via a configuration file. This is discussed further in Section A.5.

A.4 Design of LPWA E-mail Forwarding

As described earlier, the LPWA proxy creates an alias e-mail address for users in response to the `\@` escape in forms. Earlier we described an e-mail scheme in which the alias e-mail address generated is the alias username at an appropriate domain; `lpwa.com` in our case. The e-mail system then stores incoming messages, and a user agent retrieves messages for all aliases that belong to a particular user. This scheme has the advantage that the alias e-mail address generation is trivial and that no privacy-compromising information has to be stored on the e-mail system. However, such a scheme is better suited for environments in which the proxy resides on a firewall or an ISP access point.

In our trial configuration as an external proxy, a user typically expects e-mail to be forwarded to her real mailbox. In [Gabber et al. 1997], we describe such a scheme and show that the resulting alias e-mail address has the same desirable properties as the alias username and password. Actively forwarding without maintaining state implies that the alias e-mail address is an *encryption* of the user's real e-mail address (User ID). The drawback of such a scheme is that the proxy and the forwarder must store the secret encryption/decryption key. Possession of this key compromises user privacy, and hence security of this key is paramount. Note that storing the encryption/decryption key does not contradict the statelessness of the proxy, since the key is fixed and may be considered as a part of the proxy code.

While implementing LPWA, we quickly noticed that many web-sites limit e-mail addresses in registration forms to some arbitrary and rather small length. The method of [Gabber et al. 1997] produced alias e-mail addresses that were too long. Hence, we had to resort to a more heuristic approach: We first compress the user's real e-mail address, which is the LPWA User ID, and then encrypt it to generate the *mailbox* part of the alias address. The *domain name* part of the alias address is the address of the machine that runs the LPWA e-mail forwarding software. The forwarding software is derived from a Simple Mail Transport Protocol (SMTP) gateway daemon that was written at Bell Labs. It was modified so that the incoming mailbox name is decrypted to reverse the previous encryption. If the decryption fails to result in a valid e-mail address (according to RFC 822 [Crocker 1982]), the forwarder rejects the e-mail, and it writes a log entry. Otherwise the forwarder uses the host system's e-mail subsystem (*qmail*, in our case) to forward the e-mail on to the true recipient.

A.4.1 *Anti-Spam Tool.* As part of the Persona Generator, a user obtains a different and seemingly unrelated alias e-mail address for each web-site for which she registered. For example, a user might be known as `hwfyh8yocY8XUKm9t5OKvnNW@lpwa.com` to `my.yahoo.com` and as `lN8illidPtFk5OSthNoXzGuS@lpwa.com` to `www.expedia.com`. This feature enables effective filtering of junk e-mail (spam), as follows.

Whenever the LPWA E-mail Forwarder decrypts an alias e-mail address in order to forward a message to the user's real e-mail address, it includes the alias e-mail address in the `CC` e-mail header of the forwarded message. We decided to use the `CC` field, since many commercial e-mail readers already support filtering of incoming e-mail messages based on this field.

Assume that a user registers at `www.crook.com` and LPWA creates `bd1YnEWOmot3CX-_UxonbznP@lpwa.com` as the alias e-mail address. Now the address database at `crook.com` gets sold to spammers. As soon as the user gets the first piece of junk e-mail, she can install a local mail filter for the string `bd1YnEWOmot3CX-_UxonbznP`. This will eliminate all e-mail caused by the selling of the crook's database to spammers, while at the same time e-mail from all other sites is unaffected. Most current anti-spam tools filter according to sender addresses or keywords, both of which are easily changed by spammers (e.g., address spoofing). Our method is the first to filter according to the *recipient* address. A spammer who bought the address database from crook knows the user only as `bd1YnEWOmot3CX-_UxonbznP@lpwa.com` and hence cannot change (spoof) this string!

Furthermore, the user can easily keep a small local database, mapping alias e-mail addresses to the web-site for which the address was created. Then, when receiving junk e-mail, the user can determine which web-site is responsible, even when the junk e-mail was sent by a third party. The user can complain to the web-site or take other action, as needed.

A.5 Problems and Their Solutions

After extensive internal testing, we announced a public LPWA trial that became operational in June, 1997. In this section, we describe the problems we encountered and their solutions during both the testing and the (ongoing) public trial.

As described in the previous sections, the LPWA design is conceptually simple. But the devil (or God) is, as always, in the details. Initially we solved the problems we encountered by building the changes directly into the LPWA proxy source. Eventually it became evident that what we needed was to add *directives* to our modified Apache proxy server, so we could change things easily through the server's configuration file.

These were some of the problems we encountered:

—Some sites use one server to handle registrations (*e.g.*, `verify.nytimes.com`) and another to handle returning logins (*e.g.*, `www.nytimes.com`). Because the personæthat LPWA generates for these two sites are different, a return login fails. We created a way for LPWA to treat the sites as equivalent, after which the personægenerated for them were the same.

—Some sites would not function correctly without a `Referer` header. Usually when this was the case, the site insisted on seeing a `Referer` header to gain access to an interior web page. We did not consider it to be a privacy risk to provide such a header when the `Referer` was from the same site, and we programmed LPWA to provide such a header, but only for the sites that require it.

—We had many problems with e-mail forwarding. Some sites' registration forms had too little space for the alias e-mail address. Worse were the sites that would not accept valid e-mail addresses (according to RFC 822) because they contained some non-alphanumeric characters. Worst of all were those that accepted the e-mail address but did not treat the mailbox part as being case-sensitive (again violating RFC 822). We went through a series of adjustments to our encoding algorithm as we coped with these problems, leading finally to a mono-case alphanumeric encoding.

A.6 Performance

Introducing a proxy between the user's browser and the origin server will always produce a performance penalty because of the extra-hop TCP/IP connection. We wanted to verify that LPWA's processing was otherwise inconsequential, and indeed it was. The actual proxy processing delay, that is, the time between when the proxy read an HTTP request from the browser and started to make a new request to the origin server (and not counting the extra TCP connection), was about 4 ms. (on a 166MHz Pentium, running Linux). The total CPU time that the proxy required to process a request was about 10 ms. The time to process requests that contained LPWA escape sequences was unmeasurably different from requests without them. By contrast, the time to set up the connection from the client to the proxy (the client and proxy were "close") was about 35 ms.

Clearly the best way to minimize the performance impact of an LPWA HTTP proxy is to place it as close to users as possible. In a dial-up ISP setting, that would mean putting the proxy close to the dial-up access servers. In a corporate setting, that would mean putting the proxy near the corporate firewall.

A.7 Trial Experience

The LPWA trial has run since June, 1997, and has thus far attracted over 90,000 unique users (by May, 1999). About 35% of those users have logged in more than once. For the last few months, an average of 700 to 800 distinct returning users log into LPWA every day. In order to count the users without compromising their anonymity, the LPWA proxy logs the one-way hash value of the User ID and Secret.

The trial version of LPWA is currently available at `lpwa.com`. Apart from network problems early on and some later hardware failures, the LPWA proxy has run smoothly. Likewise, the e-mail forwarding software has run well (for correctly supplied To addresses), forwarding over a thousand messages per day.

Based on the number of users logging in, and on the network traffic, we believe that the ongoing trial has been a success. The LPWA user base has grown steadily, despite performance degradation for those users whose location is "inconvenient" with respect to the proxy's location in Murray Hill, New Jersey.

REFERENCES

The anonymizer. `http://www.anonymizer.com`.

SSL FAQ. Available at `http://www.consensus.com/security/ssl-talk-sec03.html`.

Ackerman, M., Cranor, L., DesAutels, P., Dunn, M., Presler-Martin, M., and Reagle, J. 1997. General overview of the p3p architecture. `http://www.w3.org/TR/WD-P3P-arch`.

Anderson, R. 1993. The classification of hash functions. *Cryptography and Coding IV*, 83–94.

Anupam, V. and Mayer, A. 1998. Security of web browser scripting languages: Vulnerabilities, attacks, and remedies. In *Proceedings of 7th USENIX Security Symposium* (1998).

Bacard, A. Anonymous remailer faq. `http://www.well.com/user/abacard/remail.html`.

Bellare, M., Kilian, J., and Rogaway, P. 1994. The security of cipher block chaining. In *Advances in cryptology – CRYPTO'94* (1994), pp. 341–358. Springer Verlag LNCS 839.

Chaum, D. 1981. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM 24*, 2 (February), 84–88.

Chaum, D. 1985. Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM 28*, 10 (October), 1030–1044.

Crocker, D. 1982. Rfc822: Standard for the format of arpa internet text messages. Available at ftp://ftp.isi.edu/in-notes/rfc822.txt.

Daemen, J., Knudsen, L., and Rijmen, V. 1997. The block cipher square. In *4th International Workshop on Fast Software Encryption* (January 1997). Springer-Verlag LNCS 1267.

Engelfriet, A. Anonymity and privacy on the internet. `http://www.stack.nl/~galactus/remailers/`.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., and Berners-Lee, T. 1997. Rfc2068: Hypertext transfer protocol — http/1.1. Available at ftp://ftp.isi.edu/in-notes/rfc2068.txt.

Gabber, E., Gibbons, P., Matias, Y., and Mayer, A. 1997. How to make personalized web browsing simple, secure, and anonymous. In *Financial Cryptography'97* (1997), pp. 17–31. Springer-Verlag LNCS 1318.

Goldberg, I., Wagner, D., and Brewer, E. 1997. Privacy-enhancing technologies for the internet. In *Proceedings of Compcon* (1997).

Gong, L., Mueller, M., Prafullchandra, H., and Schemers, R. 1997. Going beyond the sandbox: An overview of the new security architecture in the java development kit 1.2. In *Proceedings of USENIX Symposium on Internet Technologies and Systems* (1997).

Gulcu, C. and Tsudik, G. 1996. Mixing email with babel. In *Proceedings of ISOC Symposium on Network and Distributed System Security* (1996).

LAI, X. AND MASSEY, J. 1991. Markov ciphers and differential cryptanalysis. In *Proceedings of EUROCRYPT'91* (1991), pp. 17–38. Springer Verlag LNCS 437.

MARTIN, D., RAJAGOPALAN, S., AND RUBIN, A. 1997. Blocking java applets at the firewall. In *Proceedings of ISOC Symposium on Network and Distributed System Security* (1997).

MATIAS, Y., MAYER, A., AND SILBERSCHATZ, A. 1997. Lightweight security primitives for e-commerce. In *Proceedings of USENIX Symposium on Internet Technologies and Systems* (1997).

PFITZMANN, A. AND WAIDNER, M. 1986. Networks without user observability - design options. In *Proceedings of Eurocrypt'85* (1986), pp. 245–253. Springer-Verlag LNCS 219.

PRENEEL, B. AND VAN OORSCHOT, P. 1995. MDx-MAC and building fast MACs from hash functions. In *Proceedings of Crypto'95* (August 1995), pp. 1–14. Springer-Verlag LNCS 963.

REITER, M. AND RUBIN, A. 1998. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security 1*, 1 (November), 66–92.

RIVEST, R. 1994. The rc5 encryption algorithm. In *2nd International Workshop on Fast Software Encryption* (December 1994), pp. 86–96. Springer Verlag LNCS 1008.

SIMON, D. 1996. Anonymous communication and anonymous cash. In *Proceedings of Crypto'96* (August 1996), pp. 61–73. Springer Verlag LNCS 1109.

SYVERSON, P., GOLDSCHLAG, D., AND REED, M. 1997. Anonymous connections and onion routing. In *Proceedings of IEEE Symposium on Security and Privacy* (1997).