On Secure and Pseudonymous Client-Relationships with Multiple Servers

Daniel Bleichenbacher Eran Gabber Phillip B. Gibbons
Yossi Matias* Alain Mayer

Information Sciences Research Center
Bell Laboratories, Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ 07974
{bleichen, eran, gibbons, matias, mayer}@research.bell-labs.com

June 11, 1998

Abstract

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 a particular client interacts with. 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.

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 c_i are clients and s_j are server. 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.

*Also with Computer Science Dept., Tel-Aviv University, Tel-Aviv 69978 Israel. Email: matias@math.tau.ac.il.

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 too, 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), after the Roman god with two faces. In this paper, we first briefly review arguments why simple choices for the Janus function, such as a collision-resistant hash function, are not secure for our purposes. Then we will show a Janus function that is more robust. Finally, we will 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.

1.1 Related Work and Positioning of Our Work

Network anonymity is being extensively studied (see, e.g., [PW85, GWB97]). For example, Simon in [S96] 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 [SGR97] and anonymous web traffic [SGR97, RR97]). Besides that, there are several anonymous remailers available for either e-mail communication (see, e.g., [GWB97, GT96, B96, E96]) or web browsing (see, e.g., [Anon]). 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 (approximate) 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, e.g., if the client is a subscriber of a large ISP, or an employee of a large company. In the language of Reiter and Rubin [RR97], 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 [C81, C85] on *digital pseudonyms*.

We realized the ideas presented in this paper by designing and implementing a web proxy, the *Lucent Personalized Web Assistant (LPWA)*, which assists clients with registering at multiple web-servers. A companion paper [KGGMM98] (see [GGMM97] for a preliminary version) provides an in-depth description of the concrete design and implementation of LPWA.

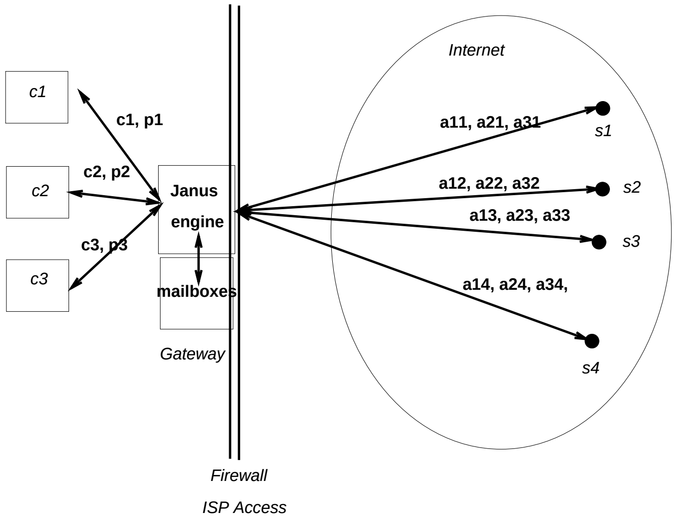

Figure 1: Client Server Configuration with a Janus Engine on the Gateway

In this paper, we focus on (1) the *provably correct* construction of the Janus engine, (2) a new method of conveying anonymous email that greatly reduces the required trust in the intermediary, and (3) a discussion of moving various Janus features from the intermediary 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 personae (see Section 5). Our methods and design are applicable to a number of client-server interactions, beyond their use in [GGMM97, KGGMM98] for a web proxy assisting clients with registering at multiple servers.

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. Finally, Section 5 presents various applications and configurations and discusses some of the trade-offs involved.

2 Model and Specifications

In this section, we present our 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 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 point in 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 [MMS97]). 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 two different users can have the same alias username on the same server only with negligible probability.

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 cannot be determined whether a particular client has an alias at a given server.

5. *Modular Security & Protection from Creation of Dossiers:* An alias of a client at one server does not reveal any information about an alias of the same client at 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":

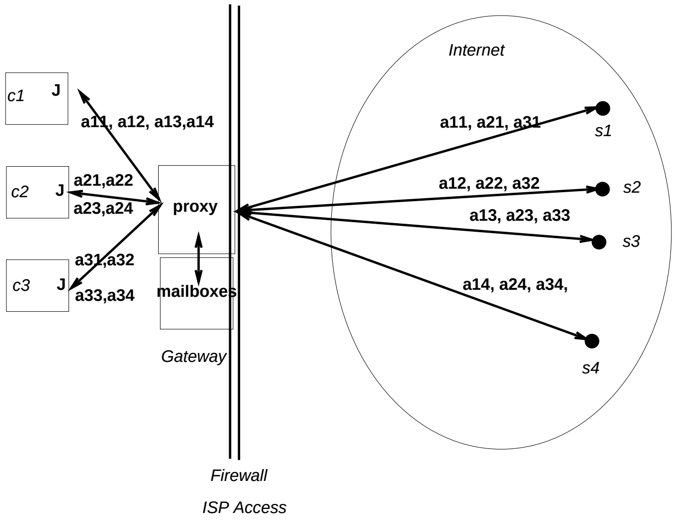

Figure 2: Client Server Configuration with a Local Janus Engine

6. *Forward Secrecy:* When a client is not interacting with a server, the Janus engine does not maintain in memory any information that may compromise on 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).

Mobility: The client may use different Janus engines within her subnet to create consistent aliases, given that she remembers her passphrase. This property is derived from *Forward Secrecy.*

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 [SGR97] or Crowds [RR97]. 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 ensure 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 it is also shown 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.

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, $a^u_{i,j}$, or an alias-password, $a^p_{i,j}$, for the user c_i on the server s_j. Note that although we limit our discussions to two tag values, the function can easily be extended to additional tags, for generating secret values for other purposes (see also [MMS97]).

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 lists of alias usernames may be publicly available on some servers and that we can not assume that all servers can be trusted or 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 having a client c_i visit a specific server s_j in order to learn her alias $a_{i,j}$). 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 would chose all her passphrases and aliases randomly.

3.1 Janus function specifications

Definition 3.1 *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, $\mathcal{A_U}$ be the set of allowable alias usernames, and $\mathcal{A_P}$ 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 $\mathcal{J} : (\mathcal{C} \times \mathcal{S} \times \mathcal{P} \times \{u, p\}) \mapsto \{0,1\}^k$.

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 $\pi_U : \{0,1\}^k \mapsto \mathcal{A_U}$ and $\pi_P : \{0,1\}^k \mapsto \mathcal{A_P}$ 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 $t \in \{p, u\}$, 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 and $c_{i'}$ that are not opened with respect to s_j, and $t \in \{p, u\}$, the adversary E 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 and t 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 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 [A93]. Some apparently better constructions for keyed functions based on hash functions have been proposed (e.g., MDx-MAC [PO95]). 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 ([BKR94]): 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

$$\mathrm{MAC}_K(x) = f_K(f_K(\cdots f_K(f_K(x_1) \oplus x_2) \cdots \oplus x_{m-1}) \oplus x_m).$$

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 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 $\epsilon - q^2m^22^{-\ell-1}$. Hence, if we use CBC-MACs, then anonymity is just a consequence of [BKR94].

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 do not 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 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 [LM91] 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 [DKR97] and one variant of RC5 [R95] have 128-bit block sizes and are therefore more suitable in this case.

4 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 A Brief History of Anonymous E-mail

Tools for anonymous e-mail communication have been around for a few years (see, e.g, [GWB97, B96, GT96, E96]). Goldberg, Wagner and Brewer [GWB97] classified anonymous remailers into three types, based on their level of privacy protection. Early anonymous remailers (Type 0, e.g., `Anon.penet.fi`) accepted an e-mail message by a user, stripped the message of user-identifying headers, and forwarded the message to the intended recipient. To enable reply e-mail, each user was assigned a unique, secret ID; e-mail addressed using this ID would be forwarded by the remailer back to the user. The level of privacy protection with this type of remailer was rather low, since it did not use encryption and kept a plain text database of e-mail addresses and corresponding IDs. A next (and still current) generation of remailers (Type I, Cypherpunk remailers) drop the use of IDs, and simply strip user-identifying headers before forwarding messages. Moreover, users can encrypt their messages before sending it and the remailer will decrypt the messages before processing it. For enhanced security, a user can *chain* such remailers. In order to use a chain, $r_1, \ldots, r_k$, of $k \geq 2$ remailers, a user first encrypts the message for r_k, then for r_{k-1}, and so on, until finally she encrypts it for r_1. (See also the efforts on Onion Routing [SGR97].) 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 development of remailer yields more and more untraceable ways of sending messages, but it provides no way to reply to a message. This gives rise to "pseudonymous" or "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 reply message, and the original sender can retrieve the message by sending a signed command to the remailer. Other remailers forward the message, along with an encrypted "reply block", to another remailer that then decrypts the reply block to obtain either the next remailer in the chain or the user's real e-mail address.[1]

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 both induce a performance penalty and are rather difficult to use.

[1] The remailer implemented for LPWA [KGGMM98] uses a simplified version of this approach: An LPWA-generated e-mail address is an encryption of the user's real e-mail address, appended by the domain name of the LPWA remailer. Upon receipt of a reply e-mail, the remailer simply decrypts the first part of the address to obtain the real address, and forwards the reply e-mail.

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), which maintain 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 client c_i's e-mail address to be used with server 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 on the *Forward Secrecy* property (defined in Section 2). In particular, Forward Secrecy means that 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 instructs 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 extend $\mathcal{J}$ to have m as a new tag for "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 via either R_{alias} or R_{index} fields. 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 is R 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 create 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 $0 \leq \gamma < C_i$. 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; the Janus engine is used 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 permitted in our adversarial 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 to compromise of 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-offs between configurations corresponding to Figure 1, which we refer to as the *gateway approach*, and those corresponding 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 [P3P]) 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 personae. For example, a client can choose whether to reveal 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 of the identity of the client. Using a Janus engine provides a new and useful middle ground: The data in the 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 personae for 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, new gateway machines can easily be added (helped by *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 enable two clients to share their aliases for all the servers, a client to transfer one or more of 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 her daily correspondence. Such alias management functions have the potential to considerably simplify login account and e-mail management in large intranets. We note that to achieve this potential, state has to be added to the proxy design; this is 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 (LPWA)* is an intermediary web proxy, which 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 the companion paper [KGGMM98] 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-FAQ]). 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 *customizable security policies* being included in these languages (see [GMPS97, AM98]). A client can then choose a policy strict enough to preserve her pseudonymity. Another approach is to bundle an LPWA proxy with an applet/script blocking proxy, as described, e.g., in [MRR97]. 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 [MMS97]. 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.

Acknowledgments

We thank David M. Kristol for his insights and for his many contributions to the design and implementation of LPWA, which uses the Janus engine. We are grateful to Russell Brand for thought-provoking discussions.

References

[A93] R. Anderson, The classification of hash functions. *Cryptography and Coding IV*, pp. 83–94, December 1993.

[AM98] V. Anupam, A. Mayer, Security of web browser scripting languages: Vulnerabilities, attacks, and remedies. In *Proc. 7th USENIX Security Symposium*, 1998.

[Anon] The Anonymizer. `http://www.anonymizer.com`.

[B96] A. Bacard, Anonymous Remailer FAQ. `http://wwww.well.com/user/abacard/remail.html`.

[BKR94] M. Bellare, J. Kilian, P. Rogaway, The security of cipher block chaining. *Advances in cryptology – CRYPTO'94*, Springer Verlag LNCS 839, pp. 341–358.

[C81] D. Chaum, Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, **24**(2), 1981, pp. 84–88.

[C85] D. Chaum, Security without identification: Transaction systems to make big brother obsolete. *Communications of the ACM*, **28**(10), October 1985, pp. 1030–1044.

[DKR97] J. Daemen, L.R. Knudsen, V. Rijmen, The block cipher SQUARE. *Fast Software Encryption'97*, Springer-Verlag LNCS, to appear.

[E96] A. Engelfriet, Anonymity and privacy on the internet. `http://www.stack.nl/galactus/remailers/index.html`.

[GGMM97] E. Gabber, P.B. Gibbons, Y. Matias, A. Mayer, How to make personalized web browsing simple, secure, and anonymous. *Financial Cryptography'97*, Springer-Verlag LNCS 1318.

[GMPS97] L. Gong, M. Mueller, H. Prafullchandra, R. Schemers, Going beyond the sandbox: An overview of the new security architecture in the Java Development Kit 1.2. In *Proc. USENIX Symposium on Internet Technologies and Systems*, 1997.

[GT96] C. Gulcu, G. Tsudik, Mixing email with babel. In *Proc. ISOC Symposium on Network and Distributed System Security*, 1996.

[GWB97] I. Goldberg, D. Wagner, E. Brewer, Privacy-enhancing technologies for the internet. In *Proc. Compcon*, 1997.

[KGGMM98] D.M. Kristol, E. Gabber, P.B. Gibbons, Y. Matias, A. Mayer, Design and implementation of the Lucent Personalized Web Assistant (LPWA). Manuscript, 1998.

[LM91] X. Lai, J. Massey, Markov ciphers and differential cryptanalysis. In *Proc. EUROCRYPT'91*, Springer Verlag LNCS 437, pp. 17–38.

[MMS97] Y. Matias, A. Mayer, A. Silberschatz, Lightweight security primitives for e-commerce. In *Proc. USENIX Symposium on Internet Technologies and Systems*, 1997.

[MRR97] D. Martin, S. Rajagopalan, A. Rubin, Blocking Java applets at the firewall. In *Proc. ISOC Symposium on Network and Distributed System Security*, 1997.

[PO95] B. Preneel, P.C. van Oorschot, MDx-MAC and building fast MACs from hash functions. *Crypto'95*, Springer-Verlag LNCS 963, pp. 1–14.

[PW85] A. Pfitzmann, M. Waidner, Networks without user observability – design options. *Eurocrypt'85*, Springer-Verlag LNCS 219, pp. 245–253.

[P3P] P3P Architecture Working Group, General Overview of the P3P Architecture. `http://www.w3.org/TR/WD-P3P-arch`.

[R95] R. Rivest, The RC5 encryption algorithm. *Fast Software Encryption*, Springer Verlag LNCS 1008, pp. 86–96, 1995.

[RR97] M. Reiter, A. Rubin, Crowds: Anonymous web transactions. *ACM Transactions on Information and System Security*, to appear. Manuscript at `http://www.research.att.com/projects/crowds/`.

[S96] D. Simon, Anonymous communication and anonymous cash. *Crypto'96*, Springer Verlag LNCS 1109, pp. 61–73.

[SGR97] P. Syverson, D. Goldschlag, M. Reed, Anonymous connections and onion routing. In *Proc. IEEE Symposium on Security and Privacy*, 1997.

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