Lightweight Security Primitives for E-Commerce

Yossi Matias, Alain Mayer, Avi Silberschatz
Bell Laboratories, Lucent Technologies

Lightweight Security Primitives for E-Commerce

Yossi Matias Alain Mayer Avi Silberschatz
Bell Laboratories, Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ 07974
{matias,alain,avi}@bell-labs.com

Abstract

Emerging applications in electronic commerce often involve very low-cost transactions, which execute in the context of ongoing, extended client-server relationships. For example, consider a web-site (server) which offers repeated authenticated *personalized stock quotes to each of its subscribers (clients). 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. Also, a client might not always use the same machine during such an extended relationship (e.g., a PC at home, a laptop on a trip). Typical* transport/session-layer *security mechanisms such as SSL and S-HTTP either require handshake/key distribution for each transaction or do not support client mobility.*

We propose a new security framework for extended relationships between clients and servers, based on persistent shared keys. We argue that this is a preferred model for inexpensive transactions executing within extended relationships. Our main contribution is the design and implementation of a set of lightweight application-layer primitives, *for (1) generating and maintaining persistent shared keys* without *requiring a client to store any information between transactions and (2) securing a wide range of web-transactions (e.g., subscription, authenticated and/or private delivery of information, receipts) with adequate computational cost. Our protocols require public key infrastructure only for servers/vendors, and its usage only once per client (upon first interaction).*

1 Introduction

Considerable attention has been given recently to *transport/session-layer* security mechanisms. There are several proposals and implementations available, including SSL [SSL96], S-HTTP [SHTTP95], and SSH [SSH96]. Offering security mechanisms at the transport/session layer has the advantage of obtaining universal security primitives which have wide applicability (e.g., SSL or SSH can be used in conjunction with any TCP connection). However, universality in the proposed schemes comes at the expense of lacking flexibility with respect to complexity and cost of securing transactions, which vary in terms of their monetary value. In particular, web-transactions within the same client-server relationship but executing at different times, either appear unrelated to the transport layer or require the client to store data in secure memory, thus putting additional responsibility on the client and preventing mobility.

Emerging applications in electronic commerce often involve very low-cost transactions, which execute in the context of an ongoing, extended client-server relationship. Rivest predicts in [R97] the increase of low-cost transactions and the need for "low-cost crypto". For such transactions, general-purpose security mechanisms tend to be prohibitively expensive. In particular, both SSL and S-HTTP involve handshake/key-distribution that consist of a costly public key cryptography. We argue that a framework based on a *shared key* between a client and a server, *persistent* for the whole duration of a relationship, is an attractive choice. From a technical point of view, the main challenge is in obtaining low-cost establishment and maintenance of the persistent shared keys, in a transparent and mobility-enabling fashion for clients. We propose a novel mechanism for persistent, shared key generation and management on the client-side. We then leverage this approach to obtain basic security primitives well suited for securing low-cost transactions which repeatedly execute between a client and a server. Such transactions may span a variety of applications, from two party tasks to elaborated micro-payment schemes involving banks, arbiters, vendor, clients, and more. In particular, concrete applications may include (1) delivery of personalized information by a vendor (via web-pages) which ensures privacy, authenticity, and integrity for each client (e.g., authentication of personalized stock quotes which a vendor sends to a client on a daily/hourly basis); (2) support for secure subscription of such services; (3) delivery of receipts to a client which ensures authenticity and integrity, provable to a third party; and (4) support and integration for (micro)payments, such as SET [SET] and PayWord [RS96].

Our approach has the following noteworthy technical aspects:

1. *Client-side shared key computation:* We propose to use a client-proxy on the client side which transparently computes *modularly secure* shared keys on the client's behalf using the so-called *Janus function* (see Section 2). This computation is based on the server identity, the client identity, and a single secret provided by the client.
2. *Client-side shared key management:* We allow shared keys to persist between browsing sessions of a client. However, a client need not store any shared keys, or any other information. Rather, the (persistent) shared key is recomputed by the client-proxy transparently on demand.
3. *Server-side shared key management:* The server accepts and stores a client's shared key on their first interaction. This is easily integrated into client's records that are typically stored at a server (such records often include usernames, preferences, etc).
4. *Modular structure:* Modularity allows us to adjust the complexity and cost of securing a transaction to the importance and monetary value of the transaction.

The above properties imply that a client need not rely on data stored in memory, and is readily suitable for mobility. The client-proxy that operates on behalf of the client does not maintain any information about the client. Therefore, the client can use various instances of the client-proxy interchangeably. A client can have a copy of the client-proxy on her PC at the office and another copy on her laptop. She can then transparently continue interacting with a server when switching from her PC to her laptop. When on the road, the client may be able to use a client-proxy implemented on an Internet-kiosk placed at the airport, and later use one implemented at an Internet station placed in the hotel.
The client interface is simple. Upon first interaction with the client-proxy (e.g., when starting to run a browser), she provides her identity (e.g., e-mail address) and a secret; she can then reconnect transparently to any server with appropriate session information. Our scheme does not require a client to obtain and maintain her own public and private keys (certificates). The client information (identity, secret) can alternatively be stored on a smart-card, or in a secure file, and be submitted to the client-proxy automatically on the client's behalf. The proposed scheme further gains computational efficiency via minimizing the use of public-key cryptography.

Organization of the Paper: Section 2 presents the client-side generation and management of secret shared keys. In Section 3 we introduce our key establishments and data delivery protocols based on the shared key of Section 2. Section 4 extends our protocols to avoid exposure of the shared key. In Section 5 we show how receipts can be generated to help in potential conflicts between clients and vendors. Finally, we discuss implementation issues in Section 6.

2 Client-side key generation and management

In our scheme, it is up to the client to compute a different secure persistent shared key for each vendor (server) she interacts with. She submits to the vendor an "identity" (e.g., email address) and a shared key, which are to be used by both parties during their subsequent interactions. The shared key is private and should be protected during communication; hence, before submitting the shared key, the client uses the vendor's public key to encrypt it. Public-key encryption is used only during the first interaction with a vendor. After this first exchange, both the client and the vendor can use the secret shared key in their subsequent interactions to authenticate and encrypt data with low computational cost.
An important aspect of this scheme is the method by which a client computes her shared keys. A shared key is computed as a function of three arguments: the client's unique identity (e.g., e-mail address), a (single) secret provided by the client, and the identity of the vendor (expressed, e.g., as as the domain

name in the vendor's URL). The function computes a string with the following properties (with respect to an adversary with polynomially bounded computational power):

1. *Secrecy:* An adversary cannot do better than guessing the resulting shared key with negligible probability.
2. *Consistency:* the computed shared key for a given vendor is consistent.
3. *Efficiency:* the computation of the shared key is efficient.
4. *Modular security:* Knowing some of a client's shared keys cannot help an adversary in guessing the client's shared key for a different vendor.
5. *Impersonation resistance:* given a vendor and a client, the adversary cannot do better than guessing another client identity and a corresponding secret, such that the resulting shared keys are identical.

We propose that the shared key computation on the client's behalf is done transparently by a client-proxy. The proxy may be located on the client's machine. Alternatively, it can be located on a different machine with which the client has a trusted communication (e.g., a server within an intranet). Upon first interaction of the client with the client-proxy (e.g., when starting a browser) the client provides a single secret, which the proxy uses thereafter to compute a shared key for each vendor the client interacts with during that browsing session.
For the rest of the paper, we consider the *client* to be the combination of user, the user-interface (e.g., browser), possibly a user-assisting program (e.g., plug-ins to the browser), and the client-proxy. Whenever we say that a client computes or executes an operation, we mean that the computation or execution is done by the client-proxy. Whenever we say the client supplies input (e.g., id or secret), we mean that the user provides the input through the user interface, or that a user-assisting program does it on the user's behalf.

Design of the key generating function To meet the desired properties listed above, we propose to use the *Janus function* $\mathcal{J}$, as defined in [BGGMM97] in the context of personalized interaction. The design of the function $\mathcal{J}$ is based on pseudo-random functions and collision-resistant hash functions (see [GGM86] and [MOV97], respectively). Let h be a collision-resistant hash-function and let f_k be a pseudo-random function chosen from a pseudo-random function ensemble F_l by using k as a seed. Let $||$ denote concatenation and $\otimes$ denote exclusive or. Let id_C denote the identity of the client and let id_V denote the identity of the vendor. Finally, let s_C denote the secret of the client, for which we assume for simplicity $s_C = (s_C^1 || s_C^2)$. The Janus function $\mathcal{J}$ is defined as:

$$\boxed{\begin{array}{l}\mathcal{J}(id_C, id_V, s_C) = \\ \quad h(f_{s_C^1}(id_V))||(f_{s_C^2}(f_{s_C^1}(id_V)) \otimes id_C)\end{array}}$$

In [BGGMM97] it is shown that the function $\mathcal{J}$ as defined above satisfies the desired properties for a client password (weak authentication). The quality of a good (machine-generated, non-mnemonic) password and a secret shared key as required here are essentially the same. The length of a shared key is typically in the range of $56 - 128$ bits, which coincides with the output length of a typical hash-function.
A vendor stores each client's identity and the shared key (possibly along with some other data, such as a client's preferences) on the very first interaction with a client, so that it can retrieve the corresponding key upon being presented with a client's identity on a repeat visit.

3 Basic protocols

In this section, we describe the basic protocols used for establishing persistent shared keys, and for subsequent interaction between a client and a server. We also present a model and correctness arguments for our protocols. At the same time, we caution that a careful development of model and correctness proofs (as shown in [BR93] for a simpler, well-known protocol) is beyond the scope of this paper.
First, we present the *Simple Key Establishment Protocol (SKEP)* for establishing relationship between a client and a server, by having the client provide the server with the persistent shared key and some other identifying or payment related information. Then we present the *Simple Data Delivery Protocol (SDDP)* for the subsequent interactions involving data delivery, and the more robust *Extended Data Delivery Protocol (EDDP).*
In the following, let $E_K(x)$ denote the encryption of a plaintext x with a public-key K, and let $S_k(x)$ denote the signature of x with a private key k. We assume that a client can obtain

each vendor's certified public key, motivated by the emerging public-key infrastructure (see, e.g., "VeriSign.com"). Let $Enc_K(x)$ be a symmetric encryption of x with the shared key K, let MAC_K be a message authentication scheme with a shared key K. Consider two parties, Alice and Bob, that have a shared secret key $K = \langle K_1, K_2 \rangle$. Let $EMAC_K(x) = (Enc_{K_1}(x)||MAC_{K_2}(Enc_{K_1}(x)))$, which can be used in a basic secure communication step between Alice and Bob, that enables delivery of an encrypted, authenticated message x.

3.1 Model

In order to interact, a vendor V and a client C form a "session" s, during which a single shared key is first established and then used. Let the two threads $\Pi^s_{C,V}$ and $\Pi^s_{V,C}$ be the entities involved in session s on the client and, resp., vendor side.
We assume the presence of a polynomially bounded adversary E, which is in charge of the communication (e.g., sending, deleting, reordering of messages) and can execute the following actions:

- *get-private-key*(x): if $x = V$, then E obtains SK_V. If $x = C$, then E obtains s_C
- *get-shared-key*(s): E obtains K of s.
- *compute-*$EMAC_K(x)$: E gets the result of computing $EMAC_K(x)$.

Definition 1 *A vendor (client) x is* **corrupted***, if a successful* get-private-key(x) *was executed; a session s is* **opened***, if either a successful* get-shared-key(s) *was executed or either participant is corrupted.*

3.2 Simple Key Establishment Protocol (SKEP)

The SKEP protocol (illustrated in Fig. 1) is used when a client C requests to register (or subscribe) at a vendor V for the first time in order to subscribe. First, the client computes the appropriate persistent shared key $K = \langle K_1, K_2 \rangle$ as $K = \mathcal{J}(id_C, s_C, id_V)$. The component K_1 will be used for encryption, and a component K_2 will be used for authentication. The subsequent message of C to the vendor V contains the persistent shared key K, encrypted via the the vendor public key PK_V, and a random nonce R_C: $(E_{PK_V}(K)||R_C)$. The vendor V then decrypts the first part of the message to obtain K. V replies with $EMAC_K(R_C||R_V)$, where R_V is its own random nonce. C decrypts the message, verifies the MAC and its own random nonce; it then sends the message $EMAC_K(R_V||id_C||I_C)$, where I_C contains possible subscription data, such as start-date or expiration date, and possible payment information, such as credit-card data, SET [SET] payment, data or "commitments" used in electronic (micro-)payments (e.g., as in PayWord [RS96]). V decrypts the message, verifies the MAC, and compares R_V to what it sent earlier; it stores the data id_C and I_C in C's record.

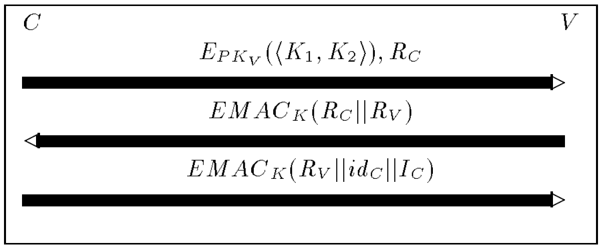

Figure 1: Simple Key Establishment Protocol (SKEP)

Desirable properties of a key establishment protocol include *Key Authentication*, *Entity Authentication*, and *Key Confirmation*, which we now define in turn:

Definition 2 Key Authentication*: For an unopened session s, E can only obtain non-negligible information on K of s.*
Matching Conversation*: A sequence of messages exchanged among $\Pi^s_{C,V}$ and $\Pi^s_{C,V}$, such that each message received by $\Pi^s_{C,V}$ corresponds to the last message sent by $\Pi^s_{V,C}$ and vice versa.*
Entity Authentication*: For an unopened session s, $\Pi^s_{C,V}$ and $\Pi^s_{C,V}$ accept the outcome of the protocol without s having a matching conversation only with negligible probability.*
Key Confirmation*: For an unopened session s, $\Pi^s_{C,V}$ and $\Pi^s_{C,V}$ accept the outcome of the protocol without K being known to other side only with negligible probability.*

We note that the above definition of Entity Authentication is essentially borrowed from [BR93], where a more in-depth discussion and model can be found.

Lemma 3 *SKEP provides Key Authentication, Entity Authentication, and Key Confirmation.*

Proof: For an unopened session s, adversary E can only obtain $E_{PK_V}(K)$. Assuming that the public-key encryption system employed by SKEP is sound, this implies Key Authentication. C

accepts the message of V, only if she can verify $MAC_{K2}(R_C \ldots)$. Given that SKEP assures Key Authentication and that R_C is a random value of "sufficient" length, E can neither compute $MAC_{K2}(R_C \ldots)$ nor guess R_C ahead of time (except with negligible probability) and execute *compute-EMAC*. Similarly, V accepts the message of C, only if he can verify $MAC_{K2}(R_V \ldots)$. Entity Authentication follows. If C successfully verifies $MAC_{K2}(R_C \ldots)$ and decrypts $Enc_{K_1}(R_C \ldots)$, and given Entity Authentication, Key Confirmation follows. □

Note that we define entity authentication, in the sense that a client can be assured that it consistently interacts with vendor, and vice versa. In the SKEP protocol, *identity* information is obtained via a client using a vendor's certified public key and a vendor using a client's payment data, such as credit card or SET data. This is consistent with today's business model of popular web-sites. For instance, on-line subscriptions to the *Wall Street Journal* and to *ESPN Sportzone*, and on-line book purchases at *amazon.com* require only this "weak authentication" from their clients.

3.2.1 Non-repudiation Key Exchange Protocol

SKEP is lacking the *non-repudiation* property, i.e., the possibility for a client to obtain *a receipt*, provable to a third party, for its subscription. SKEP can be extended by one more message exchange to obtain non-repudiation by having the vendor V sign the subscription data of the client; see Fig. 2.

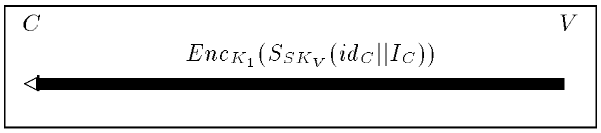

Figure 2: Extension for SKEP

3.3 Simple Data Delivery Protocol (SDDP)

The SDDP protocol (illustrated in Fig. 3) is used when a client C requests from a vendor V some information $D_{C,V}$ (e.g., a personalized web-page). The client sends the request, $R(D_{C,V})$, along with id_C and R_C, where R_C is a random nonce, where the request and the nonce are encrypted and MAC'd. If V finds id_C's key K and I_C assures validity then V replies with $EMAC_K(R_C||D_{C,V})$. The client then decrypts the message, checks that R_C is unchanged, and verifies the MAC.

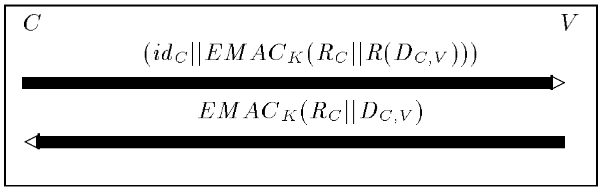

Figure 3: Simple Data Delivery Protocol (SDDP)

Desirable properties of a data delivery protocol include *Data Privacy*, *Entity Authentication*, and *Data Integrity*, which we now define in turn:

Definition 4 Data Privacy*: For an unopened session s, E cannot obtain non-negligible information on* $D_{C,V}$*.*
Entity Authentication*: see Definition 2.*
Data Integrity*: For an unopened session s, E cannot modify* $D_{C,V}$*, undetectable to C with a non-negligible probability. This also implies that C is assured that the received data is indeed the answer to her request.*

We mention without proof that SDDP fulfills Data Privacy, Entity Authentication for the client and Data Integrity. (The proofs can be derived from the proofs we give in the subsequent section for the EDDP protocol.) However, SDDP provides no Entity Authentication to the vendor. As a consequence E can prompt the vendor via impersonation attacks to send data (even though it might not be readable by E). This might be a problem in terms of chosen message and denial of service attacks. The variant in the next section is more robust in that sense.

3.4 Extended Data Delivery Protocol (EDDP)

The EDDP protocol (illustrated in Fig. 4) requires the client to demonstrate her possession of the appropriate shared key. A request of a client C from a vendor V for information $D_{C,V}$ (e.g., accessing a personalized web-page) is implemented as follows. The client first sends her identity id_C. If V accepts id_C as a client and I_C in the stored record assures validity then V replies with R_V, a random nonce. The client now issues her specific request by replying with $EMAC_K(R_C||R_V||R(D_{C,V}))$, where R_C is the client's random nonce. V checks that R_V is unchanged and verifies the MAC; it replies with

$EMAC_K(R_C||D_{C,V})$. The client checks that R_C is unchanged and verifies the MAC.

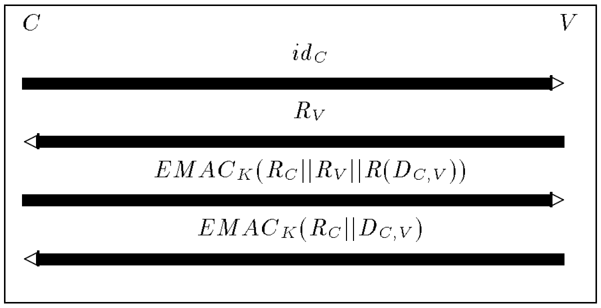

Figure 4: Extended Data Delivery Protocol (EDDP)

Lemma 5 *EDDP provides Data Privacy, mutual Entity Authentication, and Data Integrity.*

Proof: For an unopened session s, E can only obtain the corresponding encryption $Enc_{K_1}(.)$ of either $R(D_{C,V})$ or $D_{C,V}$. Furthermore, these values are prefixed with a random value (R_C), which (for suitable encryption algorithms) makes E executing *compute-EMAC* in advance to verify guesses (especially on $R(D_{C,V})$, a value in a small range) useful only with negligible probability. Thus, Data Privacy follows. Mutual Entity Authentication follows by similar arguments as used in proof of Lemma 3. If C successfully verifies $MAC_{K_2}(R_C||D)$, the Data Integrity is implied, since E cannot compute this MAC, nor guess R_C in advance to make use of *compute-EMAC* with a non-negligible probability. The same argument holds for V. □

EDDP provides *Entity Authentication* to the vendor, before he sends any data and hence gives better protection against chosen plaintext and denial of service attacks. Furthermore, the exposure of the shared key is better protected, since only the corresponding client can prompt the vendor to use it for encryption/authentication. We note that the state of art in modeling and proof techniques (e.g., see [BR93]) does not consider key exposure.

3.5 Discussion

The different applicability of the given protocol-variants can be illustrated by the following examples:

Example 1: Free Personalized Stock Quotes.

id_C is the client's email address and I_C is empty. V is a web-site which provides stock quotes. $D_{C,V}$ is a personalized web-page containing a set of stock quotes for client C. SKEP and SDDP are sufficient.

Example 2: Subscription to Personalized Stock Quotes.

id_C in Step 1 is the client's email address and I_C contains her credit card data and the duration of the subscription. Non-repudiation key exchange is preferable in this case, as it allows a client to use the vendor's signature as a receipt. EDDP is preferable, as the service is restricted to paying users and thus the vendor would like to provide adequate performance.

Computational cost and memory requirement of the protocols: The vendor has to sign a single message per client (none with SKEP). All subsequent communication is done via symmetric encryption/MAC. The client has to encrypt (public-key) a single message per vendor upon first interaction, and only MAC in subsequent interactions with that vendor. In addition, the client has to compute one Janus function per browsing-session and vendor. The client does not need any longterm memory. The vendor needs to store a persistent shared key K for each of its clients. Typically, a vendor stores some information about each of its client in any case, so this does not put an extra burden on the vendor.

4 Avoiding exposure of the persistent shared keys

In this section, we show how to extend the protocols presented in Section 3 so that (1) the exposure of the shared key K is minimized and hence cryptanalysis is made more difficult and (2) reuse of compromised keys is prevented.

We consider the notion of a *session-key* κ, which is used in lieu of the shared key K. The session key is only used for a single instance of SDDP (or EDDP). The session-key is updated by the following zero-message method, originally proposed for SKIP (see [AP95]): $\kappa_n = h(K, n)$, where h is a pseudo-random like function such as MD5 (see [R92]) and n is a strictly monotonically increasing counter. Should a session-key κ_n ever be compromised (for whatever reason) then it cannot be mis-used by an adversary to either decrypt past data transmissions or to forge data in future transmissions.

Note that the above method has the disadvantage of introducing "state" to be kept on both the client and the vendor, namely the counter n. In order to mitigate this drawback, we suggest that the counter be replaced by a strictly monotonic increasing function of the time (standard GMT), with a pre-specified granularity (e.g., day, hour, or minute) that should depend on the accuracy of the time of the client and server. One complication in this approach is that even with high accuracy, there may be discrepancy between two parties (e.g., around midnight when the unit is a day). To alleviate possible conflicts, we let the client determine the time function and notify the server during a first message. The server will accept the time function received from the client if it is the same as computed locally, or if it is within a (pre-determined) reasonable range from its computed time function (e.g., within an hour).

5 Extending the data delivery protocols to enable receipts

Consider the situation, where a client complains that she paid for a subscription of stock quotes, but never got information from the vendor. A third party cannot decide if indeed the vendor is at fault. We propose to adapt a technique used in the micropayment protocol "PayWord" (see [RS96]) to obtain the notion of *receipts*.

Assume for example that a subscription is for a month and that it entitles a client C to obtain stock quotes once a day. C chooses at random a value r and sets $w_{31} = r$ and $w_i = h(w_{i+1})$ for a suitable one-way hash function h and for $0 \leq i \leq 30$. C includes w_0 in I_C. Using non-repudiation SKEP gives the client a signature of V on w_0. C confirms the receipt of data (e.g., successful access of her personalized web-page) by sending back w_1, w_2, etc. V can test $w_{i-1} = h(i)$ and only if successful send data the next time. The ith time C acknowledges the receipt by sending w_i and V checks that $w_{i-1} = h(w_i)$. If at any time V's check is not successful, V will stop the session with C and possibly refund C for the rest of the subscription (via some verifiable payment scheme). This option is clearly not in the interest of a vendor, and thus it is unlikely that a vendor will wrongfully claim that he did not get a correct acknowledgment from the client.

The vendor can present w_i as proof that he delivered at least i data items to client C. Client C can present w_0 signed by the vendor as proof that a vendor agreed on a particular chain of receipts. Now if a client rightfully claims that she neither got the information she subscribed to nor any refund from the vendor, then the vendor cannot claim (1) that he did deliver the information, since he cannot compute w_i or (2) a different chain of receipts (since the client has his signature on the correct chain) or (3) that he paid a refund. If, on the other hand, a client wrongfully claimed that she was cheated, then either (1) she cannot show the vendor's signature on an altered chain, (2) the vendor can show w_i on the correct chain (3) or he can prove to have paid a refund.

6 Implementation

Almost all cryptographic methods used in our proposed framework are available in standard cryptographic libraries. The only exception is the function $\mathcal{J}$, which has been implemented within a web-proxy, as part of the Lucent Personalized Web Assistant (LPWA, see [GGMM97]). LPWA is being used by the general public since June 1997 and has been commended on its performance.

The LPWA software also forms the starting point for our first (and currently ongoing) implementation of the protocols presented in this paper. The protocols on the client-side are realized by a web-proxy, just as in LPWA. The client identifies herself to the proxy via proxy-authenticate. The web-proxy implements the Janus-function and embeds the client-side protocols into the client's HTTP-requests before forwarding them to the server. In line with our lightweight approach, HTTP headers are authenticated, but never encrypted. Care is taken that the sever can easily retrieve (and verify) the client's identity. The protocols on the server-side are realized via CGI (and FastCGI, see `http://www.fastcgi.com/`) scripts written in C. The goal of our first implementation is to obtain performance figures on the server-side to shed light on at least the following two issues: (1) Absolute measures of the cryptographic overhead and (2) the cryptographic overhead relative to minimal CGI-scripts (e.g., "Hello World" script) and scripts used in actual servers.

Another interesting direction is to explore ways to integrate our client-side key management scheme with SSL. This potentially yields a scheme which works works with current Web servers, but allows client mobility and requires no secure client-side long-term memory.

Acknowledgments

We thank Daniel Bleichenbacher for helpful discussions on secure protocol design and David M. Kristol and Eran Gabber for advice on software design.

References

[AP95] A. Aziz and M. Patterson, *Design and Implementation of SKIP (Simple Key Management for Internet Protocols)*, In *INET'95* conference.

[BGGMM97] D. Bleichenbacher, E. Gabber, P. Gibbons, Y. Matias, A. Mayer, *A Client-side Cryptographic Engine for Secure Relationships with Multiple Servers*, Bell Labs Technical Memorandum 1997, available at URL `www.bell-labs.com/projects/lpwa/papers.html`.

[BR93] M. Bellare, P. Rogaway, *Entity Authentication and Key Distribution*, Crypto'93 Proceedings, Springer-Verlag.

[GGM86] O. Goldreich, S. Goldwasser, S. Micali, *How to construct random functions*, J. of the ACM, **33**(4), 1986, pp 210 - 217.

[GGMM97] E. Gabber, P. Gibbons, Y. Matias, A. Mayer, *How to Make Personalized Web Browsing Simple, Secure, and Anonymous*, Proc. of Financial Cryptography'97, Springer-Verlag, LNCS 1318. Also available at URL `www.bell-labs.com/projects/lpwa/papers.html`.

[MOV97] A. Menezes, P. van Oorschot, S. Vanstone, *Handbook of Applied Cryptography*, CRC Press, 1997.

[R92] R. Rivest, *The MD5 Message Digest Algorithm*, Internet-RFC 1321, April 1992.

[R97] R. Rivest, *Perspectives on Financial Cryptography (Invited Lecture)*, Proc. of Financial Cryptography'97, Springer-Verlag, LNCS 1318.

[RS96] R. Rivest and A. Shamir, *PayWord and MicroMint: Two simple micropayment schemes*, 4th Cambridge Workshop on Security Protocols, 1996.

[SET] SET: SECURE ELECTRONIC TRANSACTION Specification.
See, e.g., at URL http://www.visa.com/cgi-bin/vee/sf/set/intro.html?2+0.

[SHTTP95] E. Rescorla and A. Schiffman *The Secure HyperText Transfer Protocol*, Internet-Draft (draft-ietf-wts-shttp-00.txt), July 1995.

[SSH96] T. Ylonen, *SSH – Secure Login Connections over the Internet*, USENIX Workshop on Security, 1996

[SSL96] P. Karlton, A. Freier, and P. Kocher, T*he SSL Protocol, 3.0*, Internet Draft, March 1996.