A Secure and Scalable Wide-Area *Upload* Service

William C. Cheng Cheng-Fu Chou Leana Golubchik Samir Khuller

Department of Computer Science, University of Maryland at College Park
{william,chengfu,leana,samir}@cs.umd.edu

[Proceedings of the 2nd International Conference on Internet Computing, Vol. 2, pps. 733-739, June 2001]

Abstract *In a recent paper, we introduced a framework termed* Bistro *to facilitate the building of scalable wide-area* upload *applications. An example of an important upload application is the submission of personal and business income tax forms over the Internet.*

Our intent for deploying the Bistro *platform is* not *to rely on adding resources (such as hosts) to the Internet. Rather, we envision that people will want to install* Bistro *services on their hosts on the Internet and contribute their resources to the overall* Bistro *infrastructure because it will improve their performance as well. Since anyone will be allowed to install a* Bistro *server, security becomes a great concern since sensitive information such as income tax forms will be uploaded through the* Bistro *system. In this paper, we describe the basic framework and present a security protocol for the* Bistro *system as well as discuss its security properties.*

Keywords: Internet Services, Security, Scalable Uploads

1 Introduction

Hot spots are a major obstacle to achieving scalability in the Internet. At the application layer, hot spots are usually caused by either (a) high demand for some data or (b) high demand for a certain service. This *high demand* for data or services, is typically the result of a *real-life event* involving availability of new data or approaching deadlines; therefore, relief of these hot spots may improve quality of life.

In this paper, we focus on secure *upload* applications that have the following characteristics. The real-life event which causes the hot spots imposes a *hard deadline* on the data transfer service. Each transfer corresponds to a relatively large file size, e.g., as in IRS tax form submissions or conference paper submissions[1]. And, no interactivity is required for uploading data, i.e., applications such as e-commerce are excluded.

For this class of applications, we observe that the existence of hot spots in uploads is largely due to approaching deadlines. The hot spot is exacerbated by the *long transfer times.* This is depicted in Figure 1. We also observe that what is actually required is an assurance that specific data was submitted before a specific time, and that the transfer of the data needs to be done in a timely fashion, but does *not* have to occur by that deadline (since the data is not consumed by the server right away).

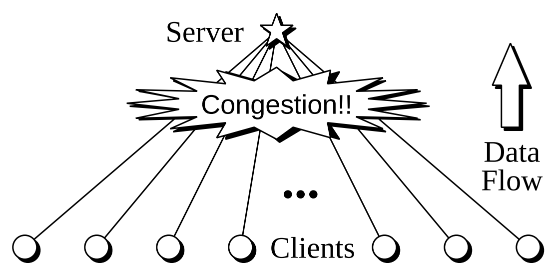

Figure 1: Depiction of a *single* upload event.

In a recent paper [1], we proposed (at a very high level) to break the original deadline-driven upload problem into: (a) a real-time *timestamp* subproblem, where we ensure that the data is timestamped and that the data cannot be subsequently tampered with; (b) a low latency *commit* subproblem, where the data goes "somewhere" and the user is ensured that the data is safely and securely "on its way" to the server; and (c) a timely *data transfer* subproblem, which can be carefully planned (and coordinated with other uploads) and must go to the

[1]A basic US IRS tax form 1040 is on the order of 100 KBytes [3], business tax forms are much larger, and a typical paper submitted to a conference is on the order of 1 to 2 Mbytes.

original destination. This means that we have taken a traditionally *synchronized client-push* solution and replaced it with a *non-synchronized* solution that uses some combination of *client-push* and *server-pull* approaches. Thus, we eliminate the hot spots by spreading most of the demand on the server over time; this is accomplished by making the actual data transfers "independent" of the deadline. In order to construct a general architecture for implementing upload applications, we introduced *Bistro*, an application-layer platform which implements a set of *primitive services*, such as timestamping, security, commit, and data transfer. This platform allows us to build scalable wide-area *upload* applications.

Our intent for deploying the *Bistro* platform is *not* to rely on adding resources (such as hosts) to the Internet. Rather, we envision that people will want to install *Bistro* services on their hosts on the Internet and contribute their resources to the overall *Bistro* infrastructure because it will improve their performance as well. The existing *bistro*s will discover the new installations and integrate them into the *Bistro* infrastructure. Since anyone will be allowed to install a *Bistro* server, security becomes a great concern since sensitive information, such as income tax forms, will be uploaded through the *Bistro* system.

In this paper, we describe our basic framework and present a security protocol for the *Bistro* system and discuss its security properties. An important consideration in designing this security protocol is not only the strength of its security characteristics but also the performance and scalability characteristics of the resulting system. Hence, the main contribution of this work is a *security protocol* which provides the needed integrity, privacy, authentication, and non-repudiation for our upload framework over *untrusted intermediaries* which are an integral part of our *Bistro* framework. An important characteristic of this protocol is that it does this under the constraints of *performance* and *scalability* requirements.

1.1 Basic Framework

As stated in [1], our approach is to break the upload problem into the following subproblems: (a) timestamp, (b) commit, and (c) transfer, and then design and develop the *Bistro* architecture which implements solutions to these subproblems. Given this breakup into subproblems, the original data transfer is now done using two data transfers (1) from a client to one or possibly more hosts on the Internet, which we will refer to as *bistro*s, and then (2) from one or more *bistro*s to the server. This flow of data is illustrated in Figure 2. The timestamp has to be

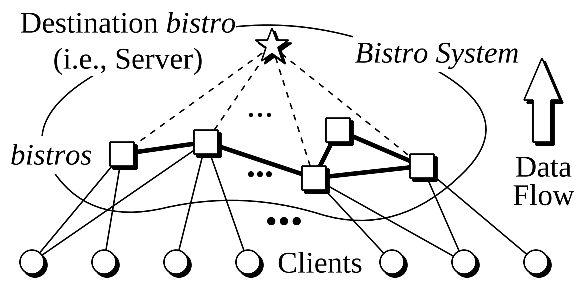

Figure 2: A *single* upload with *Bistro*.

produced before the deadline; the commit has to be performed with low latency, and the data transfer from a *bistro* to the server has to be done in a timely manner. The exact constraints on all these operations are again a function of the requirements of the particular upload application.

Note that Figure 1 depicts the current state of the upload problem which is implemented using a collection of one-to-one independent communications. Although Figures 1 and 2 only depict a single upload application, it is understood that the *bistro*s may be shared by many simultaneous upload activities/applications, each with different deadlines, characteristics, and requirements. Coordination of *multiple* simultaneous upload applications is an open research problem. The ability to share an infrastructure, such as an infrastructure of proxies or *bistro*s, between a variety of wide-area applications has clear advantages which have been outlined in [1].

In [2], we showed the performance advantages of the *Bistro* framework. We believe that the *Bistro* framework is a very flexible solution that takes advantage of available resources in the system and the network to the best extent possible.

2 A Secure and Scalable Architecture

As we noted in [1], adding intermediaries (i.e., *bistro*s) in the data transfer has obvious security implications: clearly, *bistro*s should not be allowed to corrupt the data in transit in any way or even read it. In general, the set of security properties desirable for an upload service are: (1) *Integrity*: The data cannot be changed in transit by any principal. (2) *Privacy*: For some transfers, it may be necessary to ensure that the data is encrypted and cannot be interpreted by intermediaries on the transfer path. (3) *Authentication and non-repudiation*: Since the destination now receives data from nodes that are *not* the source of the data, it may be necessary to authenticate the source of the data. The mechanisms employed to authenticate the data should also be able to discriminate "replays" by malicious *bistro*s and provide non-repudiatable transfer.

All of these properties are, in fact, desirable for many data transfer applications and many cryptographic techniques have been developed for implementing these properties [4, 5, 6, 7]. Usually, these techniques assume a powerful adversary capable of intercepting and changing messages in transit. This model is immediately applicable to the *Bistro* framework, with malicious *bistro*s being the adversaries. Thus, we can use existing cryptographic techniques (e.g., timestamping and bit-commitment [6]) to implement all these security properties for *Bistro* transfers. Below, we detail our security protocol for *Bistro*.

2.1 Setup

Here we outline the security protocol for operating a set of independent *bistro*s to handle a set of independent upload meta-events. We define an *upload meta-event* (or simply a *meta-event*) to be a *Bistro* counter-part of a real-life event[2]. Our solution is similar to the timestamped certified mail strategy of a postal service. In the public postal service, a post office is a trusted entity. The recipient trusts that a timestamp put on by a post office is accurate and that the content of the letter is unaltered after the timestamp has been issued. The main difficulty we need to overcome is the problem that if anyone can setup a post office, how can a "trusted" timestamp still be produced for purposes of certified mail? This is analogous to our situation since anyone can setup a *bistro*. A simple solution is to require the users to obtain a timestamp from the server which is hosting the specific meta-event (which we call the *destination bistro*).

For simplicity, we assume that all *bistro*s have identical software installed, which is referred to as *server software*. We also assume that a *submission* software package, which is referred to as *client software*, is available to all users who wish to submit data through the *Bistro* system. It is our intent to make all *Bistro* software publically available in both source and binary form.

Every *bistro* is identified by a (host,port) pair. The server software may include code for more than one public key crypto system. For every public crypto system, a *bistro* generates a pair of public and private keys for a set of possible key lengths. All the public keys are published and available through an HTTP connection. These keys will be used below to authenticate the *bistro*s. For *bistro* X, we use $K_{X,\mathrm{pub}}$ and $K_{X,\mathrm{priv}}$ to denote a pair of public and private keys, respectively (the corresponding crypto system and key length are left out for clarity).

2.2 Meta-event Creation

The procedure for someone (e.g., IRS, or a program chair for a paper submission application) to declare a meta-event, such as income tax forms submission or conference paper submission, is for him/her to visit a *bistro*, with a web browser (or custom software) and *create* a meta-event. Let us denote this meta-event by γ. This person will be known as the *Meta-event Owner* for meta-event γ, and the *bistro* to which he/she connects will be known as the *Meta-event Destination bistro* (or simply, the *destination bistro* – this can be just the original server with *Bistro* software installed) for meta-event γ. Please see Section 2.6 regarding the discussion on how to choose a destination *bistro*. The following steps occur during meta-event creation:

(1) For the γ meta-event, the meta-event owner provides the following information to the *destination bistro* in a secure way (such as

[2] The term *event* is often used in the context of a distribute system as an atomic entity. Here we use the term in a broader context.

through HTTPS, Diffe-Hellman, etc.): (a) a *deadline* for the meta-event; (b) the desired *security properties* for the upload (such as which public/private key crypto system to use and key lengths); (c) a *user-ID* and a *password* for future administrative access to the destination *bistro* for this meta-event; and (d) other administrative information about the meta-event owner, such as e-mail address. This information is stored in the private database of the *destination bistro*.

(2) The *destination bistro* creates a *Meta-event Identifier* (or simply, EID_γ). This EID_γ encodes the following information: (a) the *destination bistro*'s host name and port number; (b) a unique meta-event number (which can be generated sequentially on this *destination bistro*); and (c) the security properties of the upload. Please note that below we will simply use *EID* to denote EID_γ for clarity.

(3) The *destination bistro* generates a public/private key pair K^γ_{pub} and K^γ_{priv} according to the *EID*. Please note that we will also drop the γ superscripts for clarity.

(4) The *destination bistro* records *EID* in its private database and publishes *EID* and K_{pub}.

(5) At this point, the meta-event owner can publicize *EID* and K_{pub} in the announcement of the real-life upload event that corresponds to the meta-event γ (e.g., in a call for papers).

2.3 Client Submission

A client would use the *Bistro* submission software package provided as part of the *Bistro* system to submit data T for a meta-event identified by *EID*. The following steps are used to accomplish the initial exchange between the client software and the *destination bistro*:

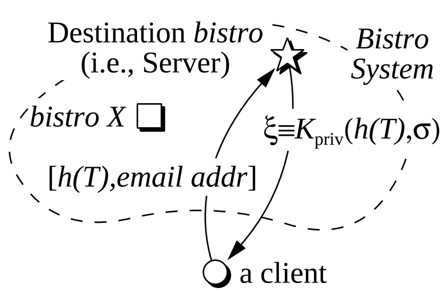

Figure 3: Obtaining a timestamp.

Step 1. This is the *timestamp* step and is depicted in Figure 3.

(1a) The client-side submission software prompts the user for the file name that identifies T and the user's email address.

(1b) The client-side submission software computes a message digest based on the method specified in the *EID*, e.g., using SHA-1, for T and produces $h(T)$.

(1c) It then sends $h(T)$ and the user's email address to the *destination bistro*. This is depicted in Figure 3.

(1d) The *destination bistro* generates a timestamp σ.

(1e) The *destination bistro* concatenates $h(T)$ with σ and encrypts it with the private key of the meta-event, K_{priv}, and sends $\xi \equiv K_{\text{priv}}(h(T), \sigma)$, a digitally signed *upload ticket*, back to the client-side submission software. This is also depicted in Figure 3. At the same time, *destination bistro* records $h(T)$, user's email address, σ, and the *upload ticket* in its private database. (It may also suggests a list of candidate *bistro*s for the *commit* step below. However, the problem of *bistro* selection is beyond of the scope of this paper.)

(1f) Upon receiving the *upload ticket*, the user can authenticate the *destination bistro* since the *upload ticket* is digitally signed by the *destination bistro*. The user can use the publically available K_{pub} to decrypt the *upload ticket* to make sure that the message digest has not been corrupted and that the timestamp was given before the deadline (based on the server's clock).

Step 2. This is the *commit* step and is depicted in Figure 4.

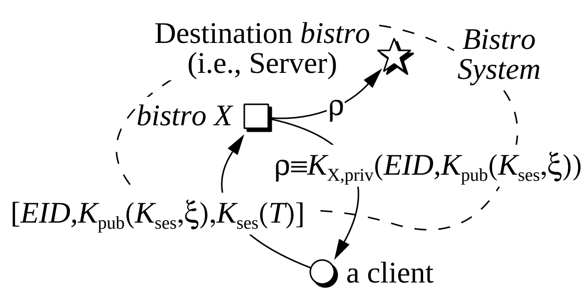

Figure 4: Commit.

(2a) The client-side submission software generates a session key, $K_{\rm ses}$, according to *EID* and encrypts T with $K_{\rm ses}$ using a symmetric-key cryptographic system standard specified in the *EID*, e.g., using triple-DES. The encrypted text is denoted by $K_{\rm ses}(T)$. It also concatenates $K_{\rm ses}$ with ξ and encrypts it with the public key of the meta-event, $K_{\rm pub}$, and sends [*EID*, $K_{\rm pub}(K_{\rm ses}, \xi)$, $K_{\rm ses}(T)$] to *any bistro* (let us call this *bistro* X).

Note that $K_{\rm pub}(K_{\rm ses})$ can be sent to the *destination bistro* in step (1c) above if $K_{\rm ses}$ is generated earlier. We choose to delay the sending of the session key for the following reasons. (1) We would like to minimize the amount of data transmitted in step (1c) above. Since the message length of $K_{\rm pub}(K_{\rm ses})$ is relatively small, this choice is not significant. (2) For fault tolerance reasons, we may want to send multiple copies of the encrypted text to multiple *bistro*s. In this case, we can generate different session keys to generate different copies of the encrypted text to reduce the chance that the session key is compromised.

Note that, the identity of *bistro* X can be hard-coded by the client-side submission software or be specified using a configuration file or be specified through appropriate setting of environment variables. It can also be chosen from the list of candidates from step (1e) above or determined using any of the policies mentioned in [2].

(2c) Finally, *bistro* X issues a *receipt*, $\rho \equiv K_{X,\rm priv}(EID, K_{\rm pub}(K_{\rm ses}, \xi))$, to the client-side submission software and stores ρ with the submission, i.e., with $K_{\rm ses}(T)$, in its temporary storage. (The choice of $K_{X,\rm priv}$ depends on the crypto system and key lengths specified in *EID*.) It also sends ρ to the *destination bistro* notifying the *destination bistro* that it has a submission for *EID*. The *destination bistro* records X and ρ in its private database.

2.4 Data Transfer

The following steps are used to transfer the submission to the *destination bistro*:

(1) After the deadline corresponding to *EID* has passed, the *destination bistro* reviews all records stored in its private database and starts retrieving submissions that correspond to *EID* from all the *bistro*s that have sent receipts for this meta-event. The retrieval can be done sequentially if the *destination bistro* has limited capacity (in terms of one or more of its resources), or the *destination bistro* can retrieve multiple submissions in parallel, based on its capacity considerations.

(2) After a submission is successfully retrieved from a *bistro* (by the *destination bistro*), that *bistro* can delete the submission from its temporary storage.

2.5 Retrieval by Meta-event Owner

The following steps are used by a meta-event owner to retrieve the submission from the *destination bistro*:

(1) Using the same method as in step (1) in Section 2.2, the meta-event owner connects to the *destination bistro* and presents the *EID*. The *destination bistro* prompts the meta-event owner for a user-ID and a password.

(2) If the user-ID and password match, the *destination bistro* uses the event private key, $K_{\rm priv}$, to decrypt $K_{\rm pub}(K_{\rm ses}, \xi)$ to obtain the session key $K_{\rm ses}$ and the *upload ticket* ξ. Then it can use the session key to decrypt the submission and obtain T. It can easily verify that the *upload ticket* is genuine by recomputing $h(T)$ and ξ.

(3) The text is sent to the meta-event owner via the secure connection.

(4) A user is notified by the *destination bistro* through email whether or not his/her submission was successfully received. In the case of a successful upload, the user can delete his/her copy of the submission. In the case that the submission was lost or corrupted, the user can simply use the client software to generate a new session key, $K^*_{\rm ses}$, and send [*EID*, $K_{\rm pub}(K^*_{\rm ses}, \xi)$, $K^*_{\rm ses}(T)$] directly to the *destination bistro*.

2.6 Discussion

In this section, we focus on the security properties of *Bistro*'s security protocol presented above. However, we first note that the security considerations of our protocol are also constrained by the desired *scalability* and *performance* characteristics which motivated our work on wide-area uploads in the first place. Hence, we first briefly mention these scalability and performance characteristics, as achieved by our system in conjunction with the security protocol.

Specifically, we believe that our solution is scalable for the following reasons. During the client submission process, only the obtaining of the *upload ticket* ξ step must occur before the deadline, which greatly reduces the network traffic to the *destination bistro* around the time of the deadline. The traffic due to the submission of the "real" data, i.e., from other *bistro*s, can be spread over time after the deadline and controlled by the *destination bistro*. This makes the submission of the "real" data "independent" of the deadline. We note that the service provided by the *Bistro* system is a *secure* upload service. Currently, secure upload is achieved through the use of HTTPS, which has comparable encryption overhead as *Bistro*. This also results in a greatly improved response time experienced by the user, due to the parallelism provided by the *bistro*s and the potential closer proximity of the *bistro*s. These performance issues are studied quantitatively in [2]. We now proceed to the discussion of security properties.

Who Should Trust Whom? Note that the meta-event owner can choose any *bistro* as the *destination bistro*. However, the meta-event owner needs to trust the *destination bistro* to give out timestamps and decrypt data. Since anyone is allowed to install a *bistro* (and possibly modify the *bistro* software), the meta-event owner should only choose a *bistro* that he or she trusts! This means that if there is a *bistro* within his/her administrative domain, he/she will most likely choose that *bistro*. If none exist, the *bistro* software can be easily downloaded and installed or he/she can choose a reputable *bistro* (such as the one in the administrative domain of someone he/she trusts). Note that, this chosen *destination bistro* needs to be trusted for the meta-event in question *only*, i.e., it can be considered an untrusted intermediary for all other meta-events which do not choose it as the *destination bistro*. Further note that only encrypted data passes through untrusted *bistro*s.

Why a Pair of Public/Private Keys per Meta-event? In step (3) of Section 2.2, we use a pair of public/private keys on a *per meta-event basis*. An alternative would be to use the same pair of public/private keys for all meta-events on the *destination bistro* with the same security requirements, as specified in the *EID*. However, the security properties of some public key crypto systems can benefit significantly from generation of a new pair of public/private keys for each meta-event [6].

Why not Just Use the Event Owner's Public/Private Keys? In step (3) of Section 2.2, we use a pair of public/private keys on a *per meta-event basis* generated by the trusted *destination bistro*. An alternative would be to use a pair of public/private keys belonging to the meta-event owner, if he/she wishes to use this alternative. There are several problems with this alternative some of which are as follows. The crypto system used by the meta-event owner may not be supported by the *Bistro* system. Furthermore, step (1e) of Section 2.3 requires the private key corresponding to the meta-event. Therefore, if the meta-event owner's keys are to be used, either the meta-event owner must expose his/her private key to the *destination bistro*, or special software needs to be provided for the meta-event owner to generate the *upload ticket* needed in step (1e) of Section 2.3. In addition, if the meta-event owner's system is behind a firewall further complications arise. Moreover, the meta-event owner's system will have to remain connected during this entire process.

Can a Destination bistro Be Mirrored? If the meta-event owner expects the uploads to swamp the *destination bistro* during the *timestamping* process, he/she can designate multiple trusted *destination bistro*s for the meta-event. This can be accomplished easily with mirroring. For instance, the meta-event owner can choose an additional *bistro* by connecting to a trusted *bistro* (as in step (1) in Section 2.2) and sending a mirroring request for *EID*. This *bistro* can then prompt the user for a user-ID and a password and send them to the original *destination bistro*. The original *destination bistro* can grant the mirroring request, if the user-ID and password match. Mirroring can also reduce the vulnerability of the *Bistro* system under denial-of-service type attacks.

Can One Upload to the Destination bistro Directly? If the *destination bistro* is not highly loaded, upon receiving $h(T)$, it can suggest itself as a candidate for the commit step. The *destination bistro* can use a simple algorithm (e.g., based on the time to deadline and a count of the number of "self-uploads" already granted) to determine if it should allow this option.

Where Are the Malicious bistros? If *bistro*

X refuses (or is unable, e.g., due to failure) to send the receipt to the *destination bistro* in step (2b) of Section 2.3, the *destination bistro* will not know where to obtain the user's submission, other than the user him/herself. However, since it had the corresponding user's email address it can notify this user that a resubmission is needed. The client-side submission software can easily assist the user in this resubmission since it has the appropriate *upload ticket* which it received from the *destination bistro* earlier. Hence, the client-side submission software can contact the *destination bistro* and present the *receipt*, generated by the possibly *malicious bistro* X, as well as the *upload ticket*, generated by the *destination bistro*, and request a re-submission directly to the *destination bistro*. Since the *destination bistro* has the message digest of the text, the client cannot change the text without this being detected. The *destination bistro* can also collect statistics about this *bistro* X (using the receipts) and eventually declare it as either malicious or unreliable, if it is deemed so.

How to Distribute Event Public Keys? As in any public key crypto system, there is the problem of key distribution. We choose to simply publish the *EID* and K_{pub} over the web. This is vulnerable to the usual attacks, such as man-in-the-middle, and so on.

What About bistro Break-ins? If a *destination bistro* is compromised, all events hosted on this *bistro* are compromised. It is up to the individual administrator to protect his/her *bistro*s.

Are Clients Authenticated? We do not authenticate the client as part of the *Bistro* framework. This problem is similar to certified mail in a postal service – one never knows who the physical sender is. In a tax form submission, the government agency relies on the signature of the tax-payer on the tax form for authentication. In the same vein, we do not authenticate the client, but assume that the end-user can provide the needed authentication, for instance, in the submitted document. For example, a digital signature can be embedded in the submitted text to authenticate the user. Such needs are application dependent.

3 Conclusions

In this paper we presented a security protocol for our *Bistro* architecture, which is designed for scalable wide-area upload applications, as well as discussed the corresponding security properties of this protocol. An important consideration in designing this security protocol was not only the strength of its security characteristics but also the *performance* and *scalability* characteristics of the resulting system.

Our long term goal is to accomplish scalability in Internet-based upload applications through the use of the *Bistro* framework over a wide range of applications and problem sizes. We believe that the *Bistro* platform is extensible to other Internet-based applications which have a many-to-one data transfer component, such as digital government, Internet-based storage, e-commerce, and many more.

References

[1] S. Bhattacharjee, W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. Bistro: a platform for building scalable wide-area upload applications. *ACM SIGMETRICS Performance Evaluation Review (also presented at the Workshop on Performance and Architecture of Web Servers (PAWS) in June 2000)*, 28(2):29–35, September 2000.

[2] W. C. Cheng, C.-F. Chou, L. Golubchik, and S. Khuller. A performance study of bistro, a scalable wide-area upload architecture. *Submitted for publication.*

[3] IRS. *Fill-in Forms.* http://www.irs.ustreas.gov/prod/forms_pubs/fillin.html, 2001.

[4] RSA Laboratories. *Public Key Cryptography Standard #1: RSA Encryption Standard Version 1.5.* RSA Data Security, Inc., Redwood City, CA, USA, November 1993.

[5] L. L. Peterson and B. S. Davie. *Computer Networks - a Systems Approach.* Morgan Kaufmann, 2000, 2nd Edition.

[6] B. Schneier. *Applied Cryptography, Second Edition.* Wiley, 1996.

[7] United States Department of Commerce. *Data Encryption Standard*, Jan. 1988.