S3L – An API Supporting Secure Communication In Distributed Applications

José C. Rufino Amaro and Francisco Soares Moura
Department of Computer Science and Communications - ESTG, Polytechnic Institute of Bragança, Portugal
Department of Computer Science–Distributed Systems Group, University of Minho, Portugal
Tel.: +351-73-3303074, +351-53-604467; e-mail: rufino@ipb.pt, fsm@di.uminho.pt

Summary
The *Stateless Secure Socket Layer* (S3L) is, fundamentally, an API that supplies Security services (Privacy, Authentication, Integrity and Sequencing) immediately above the Berkeley *sockets* programming interface, preserving great syntactic and semantic likeness with this programming model. S3L allows for the migration, with a minimum of effort, of Berkeley *sockets* based applications, towards a more secure mode of operation, in which Security information (*e.g.*, keys, algorithms, sequence numbers, authentication codes, etc.) circulates in an almost *stateless* way.

1. INTRODUCTION

Relying on cryptographic core services supplied by the *Secure Development Environment* (SecuDE) [Sch95], S3L is conceptually based on the (but not compatible nor interoperable with the) Sun's *Simple Key Management for Internet Protocols* (SKIP) [AMP95a], which supplies Security at the Network Layer. Though, the S3L transport (and it extends) the SKIP concepts to the Application Layer, in the neighborhood of the *sockets* programming interface, being distinguished of another kindred approaches (like, SSL [FKK96], for instance) in the following fundamental aspects:

- by its *stateless* character;
- by explicitly supporting secure communication in *IP Multicast* groups.

The S3L *stateless* character is derived from its auxiliary protocol *Secure Key Exchange* (Skeyx), responsible for the establishment of the Diffie-Hellman key agreement [DH76] between the parties involved in a S3L communication. Remember that a Diffie-Hellman key agreement between two entities produces, in each one of them, a common secret key (the agreed key), being enough that each entity combines its private key with the other's public key and admitting that the key pairs (public and private) have been generated, in both entities, with base in common public parameters. The advantage of such an agreement is clear: later on, the agreed key can be used to secure the message exchange between both entities, without needing to previously establish a dedicated secure session (and so, perhaps the term *sessionless* would translate, more precisely, this kind of approach), being inclusively possible the unilateral choice, by each entity, of specific security items for each message (as, for instance, keys, algorithms, sequence numbers, authentication codes, etc.) that can travel side by side with the user data, once its protection is guaranteed, in first instance, by the agreed key.

The functionalities offered by S3L translate themselves in the encapsulating of the user data in a special packet format, followed by the invocation of reading and writing primitives on Berkeley *sockets*, allowing for the secure communication over *connectionless* protocols (IP/UDP) - including the *IP Multicast* variant - and *connection-oriented* ones (TCP), without the need for the previous establishment of specific secure sessions for each transaction. It is also viable the assembling of S3L messages without necessarily delivering them directly to the network. Thus, it is possible the production of secure information containers, in order to store them or to send them via alternative transmission means.

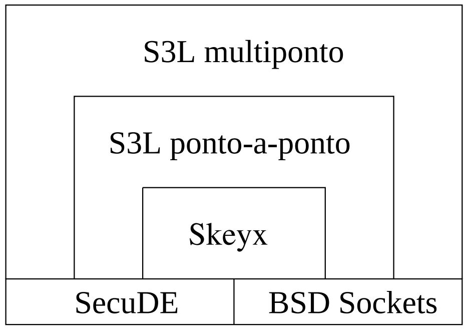

Figure 1: The S3L architecture

In architectural terms, the relationship among the fundamental S3L components is translated by Figure 1. In this article, we describe, in a synthetic way, some of the most important issues of those components.

2. KEY MANAGEMENT

In S3L, security ultimately relies in the Diffie-Hellman key agreement. Therefore, in order to avoid that the transmission of all and any S3L message is preceded by the exchange of Diffie-Hellman public keys (or by their research and retrieval from a distributed repository, such as the X.500 Directory), the Skeyx protocol is executed, at least once, between the two communicating entities. Skeyx is executed transparently, in an automatic way, during the first S3L communication between two entities. The agreed key is preserved, by each entity, in an proper *cache*, in order to be eventually reused in a future S3L communication, without the need of repeating the agreement process. In that *cache*, other elements resulting from this previous negotiation are also preserved, namely the kind of symmetric cryptographic algorithms supported by the remote entity.

The Diffie-Hellman public keys exchanged by Skeyx don't need to be certified, and so Skeyx is basically a *challenge-response* protocol, where each player has to prove that possesses the private key corresponding to the public key that sends to the other. During this process, possible *man-in-the-middle* attacks are prevented through the use of digital signatures based on the use of RSA certified public keys. However, Skeyx was conceived in a way that one entity doesn't need the previous knowledge of the other's RSA public key, being the only restriction to this scheme that the certification chain of both keys has, at least, one common point.

Replay-Attacks are prevented by marking each protocol message with a timestamp. Each party maintains a record, by origin, of the last received valid timestamp. These timestamps are used not only as sequence numbers, but also as binding elements for the several messages comprising a Skeyx transaction.

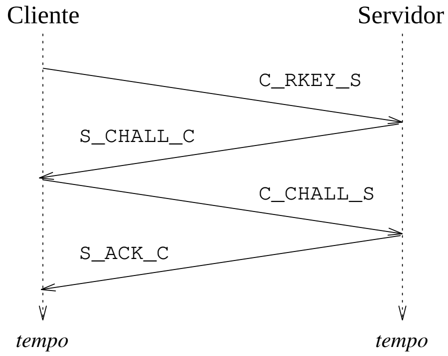

Figure 2: Skeyx model of operation

Figure 2 offers a representation for the Skeyx operational model. Briefly[1], a Client begins the Skeyx protocol by sending a *Client_RequestKey_to_Server* (C_RKEY_S) message to a certain Server. Through that message, the Client's Diffie-Hellman public key and the list of the supported symmetrical algorithms are supplied to the Server. Implicitly, the Client is requesting the same type of information from the Server, which is then returned in a *Server_Challenge_Client* (S_CHALL_C) message, jointly with a challenge (a random number, enciphered with a symmetric key derived from the provisional[2] Diffie-Hellman agreed key), so that the Client proves the ownership of the Diffie-Hellman private key corresponding to the public key sent to the Server. The Client also derives a symmetric key based on a locally generated provisional Diffie-Hellman agreed key and then tries to decipher, with it, the challenge received, which then returns to the Server in a *Client_Challenge_Server* (C_CHALL_S) message. Through this message, the Client not only answers to the Server's challenge but also makes its own challenge: another random number enciphered with the same Diffie-Hellman agreed derived key. After the Server has verified the Client's answer correctness to its challenge, the Server answers to the Client's challenge by replying with a *Server_Acknowledge_Client* (S_ACK_S) message. This message will allow the Client to verify the Server's answer correctness to its challenge.

3. SECURE UNICAST COMMUNICATION

Although placed at a different architectural level, S3L shares, just as it was already assumed, the *stateless* philosophy of SKIP. This similarity reflects itself in a similar mechanism of operation, whose unicast variant is described next.

Admitting that both communicating entities *i* and *j* accomplished a Diffie-Hellman key agreement (appealing, in the present context, to the Skeyx protocol), then both share a common secret key. Each message exchanged between the two entities will be secured upon a symmetrical key Kp, randomly generated (and, eventually, specific for each message), with a variable lifetime and belonging itself to the secured message. By its turn, Kp be will be secured by the symmetric key Kij, derived from the long term Diffie-Hellman secret key, agreed between *i* and *j*. Limiting the use of Kij to the protection of Kp (instead of the whole message), diminishes the exposure of Kij to attacks based on the traffic flow analysis. In a complementary way, making Kij to change in time will make Kp even more difficult to be compromised. Thus, instead of Kij, its n'th version, Kijn, is used.

```
 0               1               2           . . . 16
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Ver  |  Rsvd |            Dest ID (16 bytes)                 |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 17              18              19          . . . 32
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Source ID (16 bytes)                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 33              34              35              36
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Source IP Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 37              38        . . . 41          . . . 45
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Delta (32 bits)        |           n (64 bits)         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 49              50              51              52
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Kij Alg    |   Crypt Alg   |    MAC Alg    |   Comp Alg    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 53              54              55          . . . 92
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Kijn(Kp) - Kp encrypted in Kijn (40 bytes)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 93              94              95          . . . 108
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         MAC - Message Authentication Code (128 bits)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 109             110             111             112
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   sizeof of data (32 bits)                    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 113             114             115             ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         data (? bits)                       ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Structure of a S3L unicast message

Figure 3 shows the structure of a S3L unicast message. The message destiny and the origin fields (Dest ID and Source ID, respectively) consist of a MD5 *hash* of the distinguished name that identifies the subject of the X.509 certificates of the RSA public keys of the communicating entities. Kij Alg identifies the symmetrical algorithm used to encipher Kp with Kijn. Crypt Alg identifies the algorithm used to encipher the user data (data field) with Kp. MAC Alg identifies the algorithm used to encipher a MD5 *hash* of the whole message with Kp[3], which results in a *Message Authentication Code* (MAC field) whose verification in the destiny assures Authentication and Integrity. Comp Alg identifies the Compression algorithm to apply in the user data.

The Delta and n fields sustain, together, a protection mechanism against *replay-attacks*. This mechanism avoids preserving the last valid timestamp received, for each possible origin, and thus it is in full agreement with the S3L *stateliness*. The n field contains the elapsed time from a SKIP standard pre-defined instant (Jan 1, 1995 00:00:00 in *Universal Time Coordinated* (UTC)) to the instant the message is assembled by the originator, being the resolution of these values of the order of the microsecond. The Delta field is the maximum deviation admitted between the instant the message is assembled (n, in this new SKIP-alike time scale) and the instant the message is received at its final destiny (with this arriving instant also normalized to the SKIP-alike

[1] A detailed description of Skeyx is given in [Ama98].

[2] To the Server, the agreed key is still provisional because, when the Server generated it, he still was not sure that the Client's Diffie-Hellman public key he used really belonged to the Client; this will only be confirmed later when checking the Client's answer to the Server's challenge.

[3] In fact, as a precaution measure, Kp is not directly used to encipher the data and the MD5 *hash* of the message. Instead, other keys are used, specifically E_Kp and A_Kp, both derived from Kp, accordingly to the method described in [AMP95a].

UTC based time scale). [Ama98] gives a set of rules so that the originator can set `Delta` based on estimates for the propagation times, transmission times and clocks skew.

From the programmer's point of view, S3L appears as a group of C coded data structures and functions, allowing to use the Berkeley *sockets* abstraction, but with new *Qualities of Service* (QoS), namely: Privacy, Authentication, Integrity, Sequencing (against *replay-attacks*) and Compression, in several combinations. The relevant data structures summarize, in practice, to a context (of the type `S3LCtx`), that keeps the essential information concerning the involved entities as well as the QoS on which relies the security of the S3L transaction. The "S3L primitives" insulate applications from the necessary operations in order to apply the chosen QoS and, thanks to the synthatic and semantic resemblance with the "Berkeley primitives", migration of Berkeley based existing applications towards the "S3L model " is reasonably expedite. For instance, the "S3L primitive" equivalent to `write(int fd, char *buf, size_t count)` is **`S3Lwrite`**`(int fd, char *buf, size_t count,` **`S3LCtx *ctx`**`).` S3L offers other primitives, corresponding to `send`, `sendto`, `writev`, `sendmsg`, `read`, `recv`, `recvfrom`, `readv` and `recvmsg`.

4. SECURE MULTICAST COMMUNICATION

Using *IP Multicast* [Dee89] as a group communication protocol in a Security sensitive environment cannot be done without additional services. On one side, joining to a certain *IP Multicast* group does not involve selective barriers, that is, a system having network interfaces with multicast capabilities can join to (almost) any group, being enough the execution of a minimum group of operations on Berkeley *sockets*. On the other side, joining is only necessary in case one intends to receive the specific traffic of a group. Thus, the generation of traffic towards a group can be made by entities that do not belong to the group. This latter fact offers good opportunities for *denial-of-service* attacks.

An obvious solution for the permeability problem of the *IP Multicast* groups is the constitution of safe subgroups from the more general groups. Thus, the traffic of a secure subgroup will be subject to special security measures (in terms of Privacy, Authentication, etc.) that will insulate it from the elements of the complementary subgroup, as well as from traffic eventually generated by non members of the same *IP Multicast* group.

Traditionally, key distribution in a multicast context is based on a Key Distribution Center (KDC) that supplies each element of the group with the *group-key*, based on which each element is capable of having access to the secured traffic of the group. However, [AP95] identifies some fundamental problems with this approach, mainly:

1. the change of the *group-key* does not scale well with the number of elements of the group;
2. the reusing of the same key by all the elements of the group excludes the possibility to use certain *stream ciphers*[4] (*e.g.*, RC4).

These problems suggest the need of a key management policy that combines, in an independent way, for each element of the group, factors such as the lifetime and the use rate of the key. Precisely, the S3L multicast extensions offer a solution (at a certain extent inspired by SKIP) for these issues, through the mechanism we will describe next.

S3L multicast is primarily based on the role of "group owners". A group owner controls the admission of new members and sets the key management policies and the QoS for the whole group. The security of the joining process relies on the exchange of S3L unicast messages between the member candidate and the group owner. The identification of a safe subgroup of an *IP Multicast* group reuses the same class D IP address of that group. This obviously limits the number of secure subgroups of an *IP Multicast* group to exactly one. Joining will only be successful if the X.500 distinguished name of the candidate can be found in the group Access Control List (ACL) maintained by the group owner.

In the join acknowledge message, the group owner returns, to the new member, the information needed to operate as a member of the group. In the case of SKIP, a *group-key*, named *Group Interchange Key* (`GIK`), is returned. This key is randomly generated in the group owner and is similar to the unicast `Kij` key, that is, it is used to protect the `Kp` key (in this case through an algorithm defined by the group owner). The `GIK` key has a limited lifetime and so an expiry time is given to the new member, after which a rejoin becomes necessary. In the case of the S3L, a join acknowledge typically carries more than one `GIK`-alike key, each one impaired with a timestamp of the instant the group owner generated the key. A member commits itself to use the most recent `GIK` of that set. The remaining `GIK`s are considered obsolete, but they can still be useful in the reading of messages protected with base in those keys and that, due to delivery delays, will be received during the validity period of a new `GIK`. The size of the `GIK` set should be controlled carefully in order to prevent abusive reuse of obsolete `GIK`s.

Unlike SKIP, in which the (only) `GIK` is always *transient*, S3L also allows for a `GIK` to be *permanent.* In S3L this flexibility applies also to the lifetime of the group, who can be defined, when launching its group owner, as *permanent* or *transient.* In this latter case, rejoins to acquire a new `GIK` will only take place if the member concludes that the group owner is still alive.

With relationship to the `Kp` keys, its use is also ruled by group owner centrally established parameters. Those parameters concern not only to the Encryption, Authentication and Integrity algorithms, as well as they define the periodicity with which `Kp` changes. This can be a function of a certain time interval, of the amount of protected traffic or of both.

Notice that S3L opts for the use of "remaining lifetimes" and not SKIP-alike "end instants" (although these could be "normalized" to an UTC based time scale). That is, S3L proposes a *group-key* management mechanism independent of clock synchronization between the group owner and the group members. This is a mechanism tolerant to the (controlled) operation with obsolete `GIK`s and in which rejoins take place *on-demand* (instead of *by-anticipation*, as would occur in the synchronous model of SKIP)[5].

[4] [Sch96], pages 197-198, describes this problem.

[5] [Ama98] describes this mechanism with more detail.

```
0               1               2         . . . 4
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Ver  |  Rsvd |         GIK_i birthdate (32 bits)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 5               6        . . . 9         . . . 16
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Delta (32 bits)        |           n (64 bits)          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 17              18              19       . . . 56
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       GIK_i_n(Kp) - Kp encrypted in GIK_i_n (40 bytes)         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 57              58              59       . . . 72
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        MAC - Message Authentication Code (128 bits)            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 73              74              75              76
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          size of data (32 bits)                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
 77              78              79              80
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              data (? bits)                  ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Structure of a S3L multicast message

Figure 4 presents the structure of a S3L multicast message, which is considerably simplified relatively to the unicast kind, once most of the necessary security items are centrally defined and become known, after the join, by all the members. The `GIK_i birthdate` field allows to identify different generations of `GIKs.` The `GIK_i_n(Kp)` field contains a symmetrical key, `Kp`, enciphered with the `n`'th version of the `GIK` generated, by the group owner, in the instant `GIK_i birthdate`. The remaining fields carry out similar functions to the homonymous fields of the unicast messages. Note, though, that the identification of the communicating entities is based on the unicast IP address of the originator member and the multicast IP address of the destiny group.

Just as in the unicast variant, S3L multicast also offers "primitives" that hide many details from the programmer, namely the circumstances in which (re)joining takes place and the way how the writing and reading of messages in the S3L multicast format is performed. In the Berkeley *sockets* programming model, the primitives `sendto` and `recvfrom` carry out the writing and reading on *IP Multicast sockets*. Correspondingly, S3L offers **`S3Lsendto (..., void *ctx`**`)` and **`S3Lrecvfrom (..., void *ctx`**`).` Internally, these functions transform the generic context `ctx` into a unicast context (`S3LCtx`) or a multicast one (`S3LMctx`), as necessary. A multicast context of type `S3LMctx` gathers all the important information so that an entity can operate as a member of a S3L group.

5. PERFORMANCE EVALUATION

The performance evaluation made on S3L proves that, in spite of its dependence in software based cryptography, the benefits obtained with the Security extra QoS surpass the inevitable introduced overload.

As expected, the tests stress out that the use of Compression contributes, decisively, to minimize the impact of that overload.

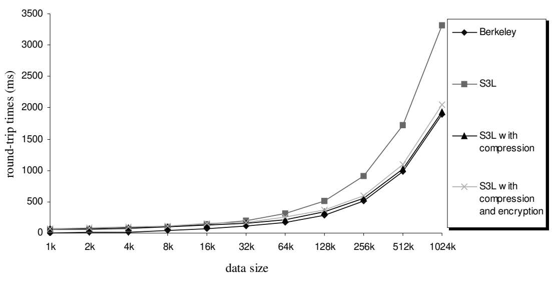

Figure 5: Performance of the primitive (S3L)write/read on TCP

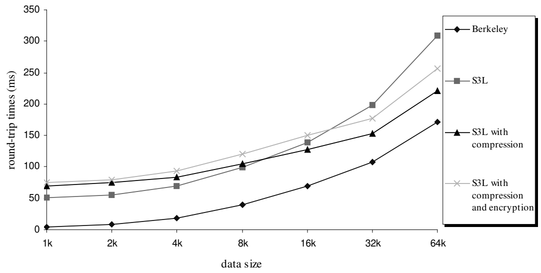

Figure 6: Zoom of Figure 5 for the interval [1K,64K]

5. CONCLUSIONS

The development of the S3L is considered opportune in a context where, in spite of the already many approaches to the secure unicast communication problem, they are little those that seat in a *stateless* operation mechanism and that, additionally, ease out the secure operation of *IP Multicast* groups. Proposing some interesting solutions in issues such as the Key Management and the Message Sequencing, S3L transports, with some success, fundamental SKIP concepts, to an API whose use is relatively expedite and does not place a significant burden in the performance of the communication process.

6. REFERENCES

[Ama98] J.C. Rufino Amaro. *An Approach to Secure Communication in Distributed Applications.* MsC Thesis in Computer Science, University of Minho. March, 1998.

[AMP95a] A. Aziz, T. Markson & H. Prafullchandra. *Internet Draft–Simple Key Management for Internet Protocols (SKIP).* December, 1995.

[AMP95b] A. Aziz, T. Markson & H. Prafullchandra. *Internet Draft–SKIP extensions for IP Multicast.* December, 1995.

[AP95] A. Aziz & M. Patterson. *Design and Implementation of SKIP.* TR ICG-95-004, Sun Microsystems. June, 1995.

[Dee89] Steve Deering. *RFC 1112–Host Extensions for IP Multicasting.* August, 1989.

[DH76] W. Diffie & M.E. Hellman. *New Directions in Cryptography.* IEEE Transactions on Information Theory. November, 1976.

[FKK96] A.O. Freier, P. Karlton & P.C. Kocher. *Internet Draft–The SSL Protocol (Version 3.0).* March, 1996.

[Sch96] Bruce Schneier. *Applied Cryptography.* John Willey & Sons, 2nd edition. 1996.

[Sch95] W. Schneider. *SecuDE Overview (version 4.4).* GMD, Institut für TeleKooperations Technik, Darmstadt, Germany. March, 1995.