Session-Layer Encryption

Matt Blaze
`mab@research.att.com`

Steven M. Bellovin
`smb@research.att.com`

AT&T Bell Laboratories

Abstract

We describe mechanisms for practical session-layer security for Internet-based terminal sessions. We discuss the tradeoffs of providing security at various layers of abstractions, from the network to the session layer. We describe two new mechanisms: our encrypting, authenticating `telnet` and our encrypted session manager (`esm`).

1 Introduction

For better or worse, many networks are protected from Bad Guys on the Internet by means of *firewalls* [CB94]. While firewalls do offer a lot of protection against certain classes of attacks, by intent they limit functionality. Paradoxically, this can actually interfere with other security mechanisms, notably those that require end-to-end encryption between machines on opposite sides of the firewall.

Firewalls can be deployed at any layer of the protocol stack. By definition, transport-level or application-level firewalls act as endpoints for the network-layer connection. What the user sees as a single connection is in fact two connections, one from the user's machine to the firewall, and a second from the firewall to the destination machine. Thus, any network-layer encryption is not end-to-end; it terminates at the firewall in either case. This may or may not be appropriate in a given security model. Apart from the difficulties this presents in authentication—the granularity of the cryptographic protection is wrong, since the firewall itself is always one of the end-points—it forces the firewall to be trusted more than might be desirable.

Even when a firewall does serve as a trusted endpoint, it may not be practical to depend upon the availability of network-layer security services. This is especially problematic in the Internet world. Few vendors today provide or support any IP-based network-layer encryption products, and it is unlikely that interoperable network-layer securitycan be relied upon as a standard feature in the immediate future.

In practice, therefore, encryption sometimes has to be performed at a layer above that intercepted by the firewall and without sophisticated, kernel-level support or infrastructure. Even here there are choices and tradeoffs. We have developed two practical tools that implement different high-level security abstractions: ESM (Encrypted Session Manager) and an encrypting version of `telnet`. Each has its own advantages and disadvantages, though they also have a lot in common.

2 Encrypted, Authenticated `Telnet`

2.1 Protocol Requirements

Recent incidents and research results [Jon95, Neu95] have us concerned that the Internet may soon become the victim of *active attacks*. More specifically, we are concerned that attackers may soon be able to hijack existing TCP connections and use them for their own nefarious purposes. Our current login sequence, which relies on a cryptographic challenge/response dialog, is not secure in the face of such attack; an enemy would wait until the login dialog was complete, and then take over control of the session. A different sort of active attack would be even more serious; if the routing tables were sufficiently disrupted that the dialog flowed through the attacker's site from the beginning, the attack would be all but undetectable. There are a number of ways in which this could be done, including bogus routing messages and subverting the Domain Name System [Bel89].

All inbound `telnet` connections to AT&T must stop at the firewall for authentication; this makes the `telnet` [PR83] protocol a natural choice for providing session security for such connections. An important goal was to base our mechanism on existing standards (e.g., keys are negotiated via the standard option mechanisms) to encourage other sites to install compatible software that remains compatible

telnet		telnetd	
			Diffie-Hellman
IAC WILL DH	$\rightarrow$		
	$\leftarrow$	IAC DO DH	
	$\leftarrow$	IAC SB DH SEND1 GEN_KEY IAC SE	
IAC SB DH IS1 $\frac{1}{2} - key$ IAC SE	$\rightarrow$		
	$\leftarrow$	IAC SB DH IS1 $\frac{1}{2} - key$ IAC SE	
IAC SB DH IS2 $\frac{1}{2} - key$ IAC SE	$\rightarrow$		
	$\leftarrow$	IAC SB DH IS2 $\frac{1}{2} - key$ IAC SE	
			Start Encryption
	$\leftarrow$	IAC DO DHCRYPT	
IAC WILL DHCRYPT	$\rightarrow$		
IAC DO DHCRYPT	$\rightarrow$		
	$\leftarrow$	IAC WILL DHCRYPT	
			Authentication
	$\leftarrow$	IAC DO DHCH	
IAC WILL DHCH	$\rightarrow$		
	$\leftarrow$	IAC SB DHCH PARM *type, length* IAC SE	
(client displays challenge; reads user response)			
IAC SB DHCH IS $response_U$ IAC SE	$\rightarrow$	*(server validates user; calculates its response)*	
	$\leftarrow$	IAC SB DHCH IS $response_S$ IAC SE	

Figure 1: The `telnet` encryption option negotiation dialog. Note the three phases: Diffie-Hellman negotiation, encryption start, and authentication.

with the existing `telnet` mechanisms.

A secondary goal of ours was to preserve our investment in hand-held authentication devices and the assorted databases and administrative procedures used to support them. While something like Kerberos-mediated `telnet encryption` [Bor93] might be a good ultimate goal, it would take a lot more effort to deploy. Also, most current implementations of Kerberos do not support one-time passwords; even with encryption, we still feel that such precautions are needed [BM91].

For key negotiation, we use Diffie-Hellman exponential key exchange [DH76]. A single large prime is used as the modulus, along with a fixed base; negotiating these parameters, though arguably more secure, would create the potential for more serious cryptanalytic attacks.

Diffie-Hellman alone would not meet our goals, since it is unauthenticated; while it would be secure against hijacked connections, there is no point to going through the deployment effort if we would need to replace it with a new version when the attackers develop the capability to divert the start of the connection. If that should happen, this simple scheme would be vulnerable to man-in-the-middle attacks. Accordingly, we use our current challenge/response devices to authenticate the key. More specifically, we use a one-way function of the exponential, which both sides know, as the challenge; the user then encrypts this challenge via the box. In the presence of a monkey-in-the-middle, the two sides would have different exponentials, and hence different challenges, so the authentication would fail. As an additional protection against active attacks, the system itself encrypts and transmits a response based on a variant of the same challenge; the user can use the same box to validate the host.

There is a subtle attack here if the exponentials are transmitted as is. Since only a small portion of the output exponential is used as a challenge (typically on the order of 20-24 bits), an attacker can do a brute-force calculation to find a user-attacker shared key that agrees in those bits with the host-attacker exponential.[1] Both sides will have the same challenge, so the authentication step will succeed. To avoid this, we use the Interlock Protocol [RS84], forcing each party to reveal evidence that it has com-

[1] Trying to calculate a more complex one-way function of the exponential doesn't help; the attacker can simply try 2^{24} random secret values until he or she finds one that results in the right output function.

mitted to its own parameters before it can learn those of the other party.

2.2 Option Negotation

Key exchange, encryption and challenge/response parameters are all negotiated and transmitted via extensions to the `telnet` options mechanism (Figure 1). Essentially, `telnet` and `telnetd` first determine that they have encryption capability (using the WILL/WONT, DO/DONT protocol), and then negotiate keys as suboptions using the SEND/IS mechanism. Once a key has been established, the telnet side presents a hash of the key as a challenge to the user, who uses the hand-held authenticator to calculate the response, which is also sent as an encapsulated suboption with SEND/IS. `telnetd` passes the locally generated challenge (which, if there is no active attack, will be the same as that calculated on the `telnet` side) and the received response and username as environment variables to the `login` program. (The authentication response and reply can also be handled as a dialog within the actual encrypted session by the `login` program itself.) The details of the option negotiation parameters will be specified in a forthcoming Internet Draft.

If the server detects an invalid authentication response $response_U$, it sends back the message

IAC SB DHCH ISNT IAC SE

instead of $response_S$. How many times incorrect replies are accepted is a local matter; it can, of course, drop the session at any time.

To allow for use of other authentication algorithms in the future, the protocol includes a message indicating which type of authentication to use, and how long the challenge and response should be. The various authentication handlers can be implemented as external programs; this allows new types to be added without modifying `telnet` or `telneted`.

To prevent an active attacker from hijacking the session in progress and forcing a return to cleartext or a change to a different key by injecting bogus DO/DONT WILL/WONT sequences, the key exchange protocol can occur at most once per session. Once encryption has commenced, `telnetd` refuses to revert to cleartext mode or change keys. In normal operation, in which the `telnet` client controls whether encryption is to be used, the exchange can occur at any time during the session, initiated either by a user keyboard escape sequence or a command line option to the `telnet` program. In "firewall" operation, however, `telnetd` needs to complete the key exchange (and calculate the challenge) before it executes the login sequence. A command-line option to `telnetd` forces the exchange at the beginning of the session and refuses to proceed if the exchange fails.

After the challenge/response dialog, the programs on either end fork and begin their normal processing. In "encrypt or die" mode, data received before the start of encryption is discarded. If it were saved, an attacker could inject evil commands in cleartext into the session before the encryption started.

In our environment, encryption of inbound `telnet` connections is not end-to-end. Incoming calls, and hence encryption, terminate at our firewall [CB94, Che90]. After the authentication is checked, the user is allowed to `rlogin` to his or her ultimate destination machine. It would be difficult to extend our current scheme in a secure fashion to provide true end-to-end encryption; the firewall *must* check authentication data, and there is no easy way to provide an out-of-band channel for the user to do a second round of authentication with the destination machine.

The dialog between the `telnetd` server and the authentication module is quite simple. The triple $\langle user, challenge_U, response_U\rangle$ is transmitted to the authenticator; it replies with either $\langle$NO$\rangle$ or $\langle$YES, $response_S\rangle$. Responses are the DES encryption of the challenge, using a shared key. In principle, the authentication server's reply should be digitally signed; in our particular environment, we rely instead on a physically secure wire between the two machines. The server's challenge is a function of the user challenge; to prevent an attacker from tricking the server into encrypting a user challenge, we use different ranges of numbers for the two values. Thus, user challenges are in the range $[0, 2^{24} - 1]$, while server challenges are in the range $[2^{24}, 2^{25} - 1]$.

3 The ESM encrypted session manager

Although the `telnet` protocol is a natural place to define network session security, it is not always possible to run `telnet` directly between arbitrary trusted endpoints. Application-level firewalls (such as our own), multi-hop login sessions and non-TCP/IP connections (like `tip`, `kermit` and `datakit`), sometimes make it necessary to consider security requirements at a higher layer than would be visible to individual network connections. `esm`, our "encrypted session manager," provides such a higher-layer security abstraction by running at the shell session level.

Essentially, `esm` exploits the BSD "pseudo-tty" mechanism to provide a layer under which every-

```
alice$ esm
ESM v0.8 - encrypted session manager
randomizing.........done
local layer ready (run 'esm -s' on remote)
alice$ rsh bob
bob$ ./esm -s
ESM v0.6 - encrypted session manager
randomizing.........done
remote server ready
Starting remote side of 1024 bit key exchange.
(press any key to abort)...
Starting local key exchange.
entering ENCRYPTED mode; type ctrl-^ to escape
Key authenticator is 0a4c3310
bob$ echo $KEYHASH
0a4c3310
bob$
  ...
              (encrypted login session between ``alice'' and ``bob'')
  ...
bob$ exit
Press <enter> to return CLEARTEXT mode:
bob$ exit
alice$
```

Figure 2: A sample ESM session.

thing between the user's local and remote login sessions is transparently encrypted and decrypted. When first invoked from an interactive shell, `esm` provides a transparent pseudo-terminal session on the local machine. When invoked in "server mode" (`esm -s`) from within an existing ESM session, however, the two ESM processes automatically encrypt all traffic passed between them. Typically, this second session is executed on a remote networked machine that was reached by using the initial session to invoke, e.g., `telnet` or `tip`, possibly across a firewall or terminal server. This is perhaps best illustrated by a simple example (Figure 2).

The local `esm` session will initially be completely transparent, passing all I/O directly from the terminal session to shell session (much like the BSD `script` program). The remote `esm -s` session initiates a Diffie-Hellman key exchange by sending an escape sequence on its standard output (which is the standard input to the local `esm` process). Once the exchange has completed and the two `esm` processes have agreed on a key, all traffic between them is encrypted with 3-key triple DES. The traffic is encoded using a simple ASCII hexadecimal representation; this reduces encrypted terminal bandwidth by a factor of just over two compared with cleartext but has the advantage of passing unmolested over virtually any transport mechanism. A session key hash, suitable for use as a challenge, is displayable on the local side and is available in the environment on the remote side. There is no other authentication or protection against an active attack.

4 Cryptographic considerations

We use triple DES [NBS77] as our bulk encryption cipher; its 168 bit effective keyspace is well above the reach of exhaustive search. We opted for triple DES because we feel that standard DES is no longer secure against exhaustive search. Even today, it appears that a $1,000,000 machine can search the entire 56 bit DES keyspace [Wie94].

Since both our ESM and `telnet` process typical user-to-host session traffic, a character-oriented cipher mode that can encrypt and decrypt each character as received is needed. Our choice is 8-bit Cipher Feedback (CFB) mode [NBS80]. CFB has the advantage of eventually "resynching" the cryptographic stream over a channel that occasionally inserts or deletes traffic. This turns out to be an important property in this application; even though `telnet` uses reliable TCP channels, its own protocol processing can drop characters under certain conditions. (The other DES stream cipher, Output Feedback keystream mode, is
unsuitable for two reasons. It is vulnerable to controlled changes by an active attacker, and it requires that sender and receiver never loose synchronization.) [PR83].

Running any encryption system above TCP has a significant drawback: an enemy can easily inject false data into the input stream. Because all error-checking and retransmission is done below the level of the encryption, packets with valid TCP checksums will be accepted, whether they will decrypt sensibly or not. This is in contrast to network-level or transport-level encryptors; bogus packets will not decrypt to be valid TCP packets, and hence will be discarded; the normal retransmission mechanisms will repair the damage.

CFB is also vulnerable to injections of previously-encrypted data. Because of the limited error propagation characteristics of CFB mode [DP89], any previously-sent input can be resent by an enemy. Worse yet, if the same key is used for input and output, an enemy can often choose the plaintext to be encrypted and reinjected. Suppose you are mailed a message containing a number of null lines followed by

```
echo + + >~/.rhosts
```

You read the message; while you are staring at it in wonder, the attacker takes the output stream and sends it back upstream. The first nine bytes reinjected will decrypt to gibberish, but that's probably harmless to the attacker; after that, the CFB decryption process will resynchronize and the remainder will be valid input to your session. The best defense is, as noted, to use different keys for different directions. (Simply picking different Initialization Vector (IV) for the two directions is not sufficient.)

For key exchange, we use 1024 bit Diffie-Hellman; this is the maximum key size supported by RSAREF and seems to provide adequate security against likely threats for at least the immediate future. We use 464 of the 1024 resulting key bits: 56×3 DES key bits plus 64 IV bits in each direction. The received parameters are checked for plausibility (e.g., that that they are non-zero); this prevents an active attacker from convincing both sides to calculate an all zero secret by zeroing each side's public parameters.

To generate the random parameters, we use the `truerand` facility (based on clock skew) from the *CryptoLib* package [Lac93]. It appears to work reasonably well on most UNIX platforms, especially when several runs are combined for each bit.

5 Encryption and the Protocol Stack

Textbooks on computer networking speak of a model protocol stack with seven layers. Textbooks on cryptography, if they address deployment at all, distinguish solely between link-level encryptors and "end-to-end" encryptors. Reality is far more complex. There are many different places where an encryption function can be placed; these don't always map neatly into the standard network layer cake.

Link-layer encryption still has most of the properties traditionally ascribed to it. It can be deployed locally to protect a particular vulnerable link; it is in general invisible to higher layers. Even so, there are problems; link-layer spoofing techniques such as proxy ARP [CMQ87] are often employed.

Network-layer encryption is more problematic. Traditionally, the network layer is the lowest end-to-end layer, and hence is a natural place for ubiquitous encryption; as we have seen, though, firewalls and protocol translators break this assumption. We are thus forced to move our encryptor to a higher layer.

Even a pure network-layer encryptor is not architecturally clean. SP3, for example [SP388], has some modes of operation that make it look much more like a link encryptor, and other modes that force recursion through the network layer. In general, an encryptor at any given level can operate at either the top or the bottom of that layer. Furthermore, there is a semantic difference between encrypting at, say, the top of the network layer versus the bottom of the transport layer.

Transport-layer encryption differs from network-layer encryption primarily in its ability to deliver a finer granularity of protection. It, too, is affected by network discontinuities. Both translators and some firewalls (i.e., the TIS Firewall Toolkit [AR94]) require the user to "redial". Ergo, transport-layer encryption cannot be end-to-end either.

Above it, matters become even blurrier, especially since the layering structure is inadequate. Where does electronic mail live? At the mail transfer level, as typified by SMTP [Pos82]? This is generally considered to be application layer. But message formatting lives [Cro82] above that, and multimedia mail above that [BF93]. Where does one encrypt mail? The usual answer is to encrypt the message itself [Lin93, Ken93, Bal93, Kal93, Zim92], though exactly how this should be done for complex mail messages isn't at all obvious [CFGM95].

By contrast, one proposal for protection of Web traffic, the Secure Socket Layer (SSL) [Hic95], encrypts the transport connection, rather than the text of the retrieved page. This does provide some added privacy protection for, say, those who are retrieving `gerbils-mmff.gif`, though often the site name itself (`rodents.com`) may be sensitive.

Our `telnet` encryptor demonstrates some of the problems. The `telnet` protocol lives on top of TCP, a reliable transport layer, but `telnet` itself can delete characters from the data stream presented to whatever lives above it. And this forced us to use CFB encryption, whereas encryption in the lower part of `telnet` could have been done via OFB mode. Furthermore, we still have the network discontinuity problem to deal with.

This is where `esm` comes in: it operates above anything else, and is set up after *all* of the connections are established. Architecturally, this may not be the best choice. But it is the *only* way we can do true end-to-end encryption in the face of a heterogeneous network.

6 Related Work

There are several other encrypting `telnet` programs available, plus an encrypted remote login package known as `deslogin`. None of these was quite suitable.

The most standardized package is a Kerberized `telnet` from MIT. But its use of Kerberos is, for us, a weakness: we would have had to deploy a full-fledged Kerberos server in order to use it. Furthermore, it uses Kerberos 4, which in our opinion suffers from two practical drawbacks: it does not support the use of hand-held authenticators, which means people must still type passwords into potentially untrustworthy machines, and it is vulnerable to outsiders requesting user tickets from the server and running a password-guessing program against these tickets [LGSN89, BM92].

A related effort is the encrypting `telnet` by Brown and Jaatun, done as a prototype of a standard `telnet` encryption option. It required use of one of the standard authentication mechanisms, such as Kerberos.

More recently, the STEL package [VTB95] has been announced. It, like ours, uses authenticated Diffie-Hellman; a variety of authentication mechanisms are supported, including some one-time password schemes. It does not appear to use standard `telnet` option negotiation; however, it can be used to replace `rshd` as well as `telnetd`.

The other secure `telnet` package is SRA, from Texas A&M University [SHS93]. It is based on Secure RPC [Sun88], and uses Diffie-Hellman key exchange to negotiate a session key. This session key is used only to transmit the user's login and password; the remainder of the session is not protected. While extending the code to do this latter is fairly straight-forward—indeed, there are at least two such implementations available for anonymous `ftp`—the scheme would still be vulnerable to active attacks, precisely the threat we wish to deflect. Furthermore, the modulus size used is too small, and has in fact been cryptanalyzed [LO91].

The `deslogin` program is similar in spirit to our `esm`, though incompatible with `telnet`. It requires its own key database, though since it uses challenge/response authentication it would not be difficult to modify it to use our current authentication server and hand-held authenticators. One very useful feature in `deslogin` is the ability to authenticate twice, once to a firewall and once to the endpoint.

7 Implementation Status

A basic UNIX implementation, based on the 4.4BSD `telnet` and `telnetd` source, is complete; it runs under most UNIX platforms. The Diffie-Hellman key exchange is performed using the RSAREF library; this simplifies the patent issues. We hope to make our code freely available, subject to the usual export control restrictions on cryptographic software.

An MS-DOS `telnet` client, probably based on the NCSA package, is in being developed by some of our colleagues. Again, we hope to release the code.

We have also completed a basic UNIX implementation of the `ESM` package. Like our `telnet`, it runs under most BSD-derived platforms and uses RSAREF for its Diffie-Hellman functions. We also expect to make this code freely available.

8 Conclusions and Future Directions

Our encrypting `telnet` is a band-aid solution. That is not necessarily bad: we need a band-aid, to cope with a threat that in our opinion is imminent. Still, a solution that was part of an integrated security architecture would be better. The Internet community is experimenting with cryptographic standards for the link layer [Mey95], network layer [Atk95], session layer, and application layer (for mail, SNMP, and many others). While all of these have their uses, it would be nice if there were an overall vision and (where feasible) a common key management structure.

Failing that, we are likely to implement the EKE or A-EKE authentication protocols [BM92, BM93]. These are password-based, but require no special hardware and are immune to password-guessing attacks.

Our current scheme has a number of limitations. The most serious is that it is not truly end-to-end. It would be nice to either re-encrypt from the firewall onward, or—better yet—to negotiate a new authenticated session between the user and the ultimate end point, so that there would be no cleartext on the firewall machine. Accomplishing either of these goals while still maintaining security and user convenience is not easy. Most likely, we will not try; rather, we will use encrypted IP tunnels between the user's machine and the firewall. Anything else, including a second layer of end-to-end network layer encryption, will be transported inside of this secure envelope.

We would also like to have an "authenticate-only" mode, for use in situations where encryption is illegal. Stream ciphers are not particularly good for such things. The best idea seems to be to send everything twice, once in cleartext and once encrypted. If the received cleartext character does not match the decrypted version, we can conclude that an enemy has tampered with the session.

References

[AR94] Frederick Avolio and Marcus Ranum. A network perimeter with secure external access. In *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, San Diego, CA, February 3, 1994.

[Atk95] R. Atkinson. Ipv6 encapsulating security payload (ESP). Internet draft; work in progress, February 16 1995.

[Bal93] D. Balenson. Privacy enhancement for internet electronic mail: Part III: algorithms, modes, and identifiers. Request for Comments (Experimental) RFC 1423, Internet Engineering Task Force, Feb 1993. (Obsoletes RFC1115).

[Bel89] Steven M. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communications Review*, 19(2):32–48, April 1989.

[BF93] N. Borenstein and N. Freed. MIME (multipurpose internet mail extensions) part one: Mechanisms for specifying and describing the format of internet message bodies. Request for Comments (Experimental) RFC 1521, Internet Engineering Task Force, Sep 1993. (Obsoletes RFC1341); (Updated by RFC1590).

[BM91] Steven M. Bellovin and Michael Merritt. Limitations of the Kerberos authentication system. In *USENIX Conference Proceedings*, pages 253–267, Dallas, TX, Winter 1991.

[BM92] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, pages 72–84, Oakland, CA, May 1992.

[BM93] Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange. In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 244–250, Fairfax, VA, November 1993.

[Bor93] D. Borman. Telnet authentication: Kerberos version 4. Request for Comments (Proposed Standard) RFC 1411, Internet Engineering Task Force, Jan 1993.

[CB94] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, Reading, MA, 1994.

[CFGM95] Steve Crocker, Ned Freed, Jim Galvin, and Sandy Murphy. MIME object security services. Internet draft; work in progress, March 1995.

[Che90] William R. Cheswick. The design of a secure internet gateway. In *Proc. Summer USENIX Conference*, Anaheim, CA, June 1990.

[CMQ87] S. Carl-Mitchell and J. Quarterman. Using ARP to implement transparent subnet gateways. Technical Report RFC 1027, Internet Engineering Task Force, October 1987.

[Cro82] D. Crocker. Standard for the format of ARPA internet text messages. Request for Comments (Standard) RFC 822, Internet Engineering Task Force, August 1982. Obsoletes RFC0733; Updated by RFC1327, RFC0987.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-11:644–654, November 1976.

[DP89] Donald W. Davies and Wyn L. Price. *Security for Computer Networks*. John Wiley & Sons, second edition, 1989.

[Hic95] Kipp E.B. Hickman. The SSL protocol. Internet draft; work in progress, April 1995.

[Jon95] Laurent Joncheray. A simple active attack against TCP. In *Proceedings of the Fifth Usenix* UNIX *Security Symposium*, Salt Lake City, UT, 1995. To appear.

[Kal93] B. Kaliski. Privacy enhancement for internet electronic mail: Part IV: key certification and related services. Request for Comments (Experimental) RFC 1424, Internet Engineering Task Force, Feb 1993.

[Ken93] S. Kent. Privacy enhancement for internet electronic mail: Part II: certificate-based key management. Request for Comments (Experimental) RFC 1422, Internet Engineering Task Force, Feb 1993. (Obsoletes RFC1114).

[Lac93] John B. Lacy. Cryptolib: Cryptography in software. In *Proceedings of the Fourth Usenix* UNIX *Security Symposium*, pages 1–17, Santa Clara, CA, October 1993.

[LGSN89] T. Mark A. Lomas, Li Gong, Jerome H. Saltzer, and Roger M. Needham. Reducing risks from poorly chosen keys. In *Proceedings of the Twelfth ACM Symposium on Operating Systems Principles*, pages 14–18. SIGOPS, December 1989.

[Lin93] J. Linn. Privacy enhancement for internet electronic mail: Part I: message encryption and authentication procedures. Request for Comments (Experimental) RFC 1421, Internet Engineering Task Force, Feb 1993. (Obsoletes RFC1113).

[LO91] Brian A. LaMacchia and Andrew M. Odlyzko. Computation of discrete logarithms in prime fields. *Designs, Codes, and Cryptography*, 1:46–62, 1991.

[Mey95] G.M. Meyer. The PPP encryption control protocol (ECP). Internet draft; work in progress, February 1995.

[NBS77] NBS. Data encryption standard, January 1977. Federal Information Processing Standards Publication 46.

[NBS80] NBS. DES modes of operation, December 1980. Federal Information Processing Standards Publication 81.

[Neu95] Michael Neuman. Monitoring and controlling suspicious activity in real-time with Watcher, 1995. Draft.

[Pos82] J. Postel. Simple mail transfer protocol. Request for Comments (Standard) RFC 821, Internet Engineering Task Force, August 1982. Obsoletes RFC0788.

[PR83] J. Postel and J. Reynolds. Telnet protocol specification. Request for Comments (Standard) RFC 854, Internet Engineering Task Force, May 1983. Obsoletes RFC0764.

[RS84] Ronald L. Rivest and Adi Shamir. How to expose an eavesdropper. *Communications of the ACM*, 27(4):393–395, 1984.

[SHS93] David R. Safford, David K. Hess, and Douglas Lee Schales. Secure RPC authentication (SRA) for TELNET and FTP. In *Proceedings of the Fourth Usenix* UNIX *Security Symposium*, pages 63–67, Santa Clara, CA, October 1993.

[SP388] SDNS secure data networking system security protocol 3 (SP3). Technical Report Revision 1.3, SDNS Protocol and Signalling Working Group, SP3 Sub-Group, July 12 1988.

[Sun88] Sun Microsystems, Inc. RPC: remote procedure call protocol specification version 2. Request for Comments (Informational) RFC 1057, Internet Engineering Task Force, June 1988. Obsoletes RFC1050.

[VTB95] David Vincenzetti, Stefano Taino, and Fabio Bolognesi. STEL: Secure TELnet. In *Proceedings of the Fifth Usenix* UNIX *Security Symposium*, Salt Lake City, UT, 1995. To appear.

[Wie94] Michael J. Wiener. Efficient DES key search. Technical Report TR-244, School of Computer Science, Carleton University, Ottawa, Canada, May 1994. Presented at the Rump Session of Crypto '93.

[Zim92] Philip Zimmerman. PGP user's guide, September 1992.