Network Working Group
Internet Draft

Steven M. Bellovin
AT&T Labs Research
Charlie Kaufman
IBM
Jeffrey I. Schiller
MIT

Expiration Date: July 2003

January 2003

Security Mechanisms for the Internet

draft-iab-secmech-02.txt

1. Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

2. Abstract

Security must be built into Internet Protocols for those protocols to offer their services securely. Many security problems can be traced to improper implementations. However, even a proper implementation will have security problems if the fundamental protocol is itself exploitable. Exactly how security should be implemented in a protocol will vary, because of the structure of the protocol itself. However, there are many protocols for which standard Internet security mechanisms, already developed, may be applicable. The precise one that is appropriate in any given situation can vary. We review a number of different choices, explaining the properties of each.

3. Introduction

Internet Security compromises can be divided into several classes, ranging from Denial of Service to Host Compromise. Denial of Service attacks are beyond the scope of this document except to the extent that they are made more difficult by good security practices. Host Compromise (most commonly caused by undetected Buffer Overflows) represent flaws in individual implementations rather than flaws in protocols. Nevertheless, carefully designed protocols can make such flaws less likely to occur and harder to exploit.

However there are security compromises that are facilitated by the very protocols that are in use on the Internet. If a security problem is inherent in a protocol, no manner of implementation will be able to prevent the problem.

It is therefore vitally important that protocols developed for the Internet provide this fundamental security.

Exactly how a protocol should be secured depends on the protocol itself as well as the security needs of the protocol. However we have developed a number of standard security mechanisms in the IETF. In many cases appropriate application of these mechanisms can provide the necessary security for a protocol.

A number of possible mechanisms can be used to provide security on the Internet. Which one should be selected depends on many different factors. We attempt here to provide guidance, spelling out the factors and the currently-standardized (or about-to-be-standardized) solutions, as discussed at the IAB Security Architecture Workshop [RFC2316].

Security, however, is an art, not a science. Attempting to follow a

recipe blindly can lead to disaster. As always, good taste in protocol design should be exercised.

Finally, security mechanisms are not magic pixie dust that can be sprinkled over completed protocols. It is rare that security can be bolted on later. Good designs--that is, secure, clean, and efficient designs--occur when the security mechanisms are crafted along with the protocol. No conceivable exercise in cryptography can secure a protocol with flawed semantic assumptions.

4. Decision Factors

4.1. Threat Model

The most important factor in choosing a security mechanism is the threat model. That is, who may be expected to attack what resource, using what sorts of mechanisms? A low-value target, such as a Web site that offers public information only, may not merit much protection. Conversely, a resource that if compromised could expose significant parts of the Internet infrastructure--say, a major backbone router or high-level Domain Name Server--should be protected by very strong mechanisms. The value of a target to an attacker depends on the purpose of the attack. If the purpose is to access sensitive information, all systems that handle this information or mediate access to it are valuable. If the purpose is to wreak havoc, systmes on which large parts of the Internet depend are exceedingly valuable. Even if only public information is posted on a web site, changing its contents can cause embarassment to its owner and could result in substantial damage. It is difficult when designing a protocol to predict what uses that protocol will someday have.

All Internet connected systems require a minimum amount of protection. Starting in 2000 and continuing to the present, we have witnessed the advent of a new type of Internet security attack: an Internet "worm" program that seeks out and automatically attacks systems that are vulnerable to compromise via a number of attacks built into the worm program itself. These worm programs can compromise literally thousands of systems within a very short period of time.

As of the writing of this document, all of these worms have taken advantage of programming errors in the implementation of otherwise reasonably secure protocols. However, it is not hard to envision an attack that targets a fundamental security flaw in a widely deployed protocol. It is therefore imperative that we strive to minimize such flaws in the protocols we design.

[footnote: The first Internet Worm was the "Morris" worm of 1988. However it was not followed up with similar programs for over 12 years!]

The value of a target to an attacker may depend on where it is located. A network monitoring station that is physically on a backbone cable is a major target, since it could easily be turned into an eavesdropping station. The same machine, if located on a stub net and used for word processing, would be of much less use to a sophisticated attacker, and hence would be at significantly less risk.

One must also consider what sorts of attacks may be expected. At a minimum, eavesdropping must be seen as a serious threat; there have been very many such incidents since at least 1993. Often, active attacks--that is, attacks that involve insertion or deletion of packets by the attacker--are a risk as well. It is worth noting that such attacks can be launched with off-the-shelf tools, and have in fact been observed "in the wild". Of particular interest is a form of attack called "session hijacking", where someone on a link between the two communicating parties wait for authentication to complete and then impersonate one of the parties and continue the connection with the other.

One of the most important tools available to us for securing protocols is cryptography. Cryptography permits us to apply various kinds of protection to data as it traverses the network, without having to depend on any particular security properties of the network itself. This is important because the Internet, by its distributed management and control, cannot be considered a trustworthy media in and of itself. Its security derives from the mechanisms that we build into the protocols themselves, independent of the underlying media or network operators.

Finally, of course, there is the cost to the defender of using cryptography. This cost is dropping rapidly; Moore's Law, plus the easy availability of cryptographic components and toolkits, makes it relatively easy to use strong protective techniques. Although there are exceptions--public key operations are still expensive, perhaps prohibitively so if the cost of each public-key operation is spread over too few transactions--careful engineering design can generally let us spread this cost over many transactions.

In general, the default today should be to use the strongest cryptography available in any protocol. Strong cryptography often costs no more, and sometimes less, then weaker cryptography. The actual performance cost of an algorithm is often unrelated to the security it provides.

4.2. A word about Mandatory Mechanisms

We have evolved in the IETF the notion of "mandatory to implement" mechanisms. This philosophy evolves from our primary desire to ensure interoperability between different implementations of a protocol. If a protocol offers many options for how to perform a particular task, but fails to provide for at last one that all must implement, it may be possible that multiple, non-interoperable implementations may result. This is the consequence of the selection of non-overlapping mechanisms being deployed in the different implementations.

Although a given protocol may make use of only one or a few security mechanisms, these mechanisms themselves often can make use of several cryptographic systems. The various cryptographic systems vary in strength and performance. However, in many protocols we need to specify a "mandatory to implement" to ensure that any two implementations will eventually be able to negotiate a common cryptographic system between them.

There are some protocols that were originally designed to be run in a very limited domain. It is often argued that the domain of implementation for a particular protocol is sufficiently well defined and secure that the protocol itself need not provide any security mechanisms.

History has shown this argument to be wrong. Inevitably, successful protocols - even if developed for limited use - wind up used in a broader environment, where the initial security assumptions do not hold.

To solve this problem, the IETF requires that *ALL* protocols provide appropriate security mechanisms, even when their domain of application is at first believed to be very limited.

It is important to understand that mandatory mechanisms are mandatory to *implement*. It is not necessarily mandatory that end-users actually use these mechanisms. If an end-user knows that they are deploying a protocol over a "secure" network, then they may choose to disable security mechanisms that they believe are adding insufficient value as compared to their performance cost. (We are generally skeptical of the wisdom of disabling strong security even then, but that is beyond the scope of this document.)

Insisting that certain mechanisms are mandatory to implement means that those end-users who need the protocol provided by the security mechanism have it available when needed.

4.3. Granularity of Protection

Some security mechanisms can protect an entire network. While this economizes on hardware, it can leave the interior of such networks open to attacks from the inside. Other mechanisms can provide protection down to the individual user of a timeshared machine, though perhaps at risk of user impersonation if the machine has been compromised.

When assessing the desired granularity of protection, protocol designers should take into account likely usage patterns, implementation layers (see below), and deployability. If a protocol is likely to be used only from within a secure cluster of machines (say, a Network Operations Center), subnet granularity may be appropriate. By contrast, a security mechanism peculiar to a single application is best embedded in that application, rather than inside TCP; otherwise, deployment will be very difficult.

4.4. Implementation Layer

Security mechanisms can be located at any layer. In general, putting a mechanism at a lower layer protects a wider variety of higher-layer protocols, but may not be able to protect them as well. A link-layer encryptor can protect not just IP, but even ARP packets. However, its reach is just that one link. Conversely, a signed email message is protected even if sent through many store-and-forward mail gateways, can identify the actual sender, and the signature can be verified long after the message is delivered. However, only that one type of message is protected. Messages of similar formats, such as some Netnews postings, are not protected unless the mechanism is specifically adapted and then implemented in the news-handling programs.

5. Standard Security Mechanisms

5.1. One-Time Passwords

One-time password schemes, such as that described in [RFC2289], are very much stronger than conventional passwords. The host need not store a copy of the user's password, nor is it ever transmitted over the network. However, there are some risks. Since the transmitted string is derived from a user-typed password, guessing attacks may still be feasible. (Indeed, a program to launch just this attack is readily available.) Furthermore, the user's ability to login necessarily expires after a predetermined number of uses. While in

many cases this is a feature, an implementation most likely needs to provide a way to reinitialize the authentication database, without requiring that the new password be sent in the clear across the network.

There are commercial hardware authentication tokens. Apart from the session hijacking issue, support for such tokens (especially challenge/response tokens, where the server sends a different random number for each authentication attempt) may require extra protocol messages.

5.2. HMAC

HMAC [RFC2104] is the preferred shared-secret authentication technique. If both sides know the same secret key, HMAC can be used to authenticate any arbitrary message. This includes random challenges, which means that HMAC can be adapted to prevent replays of old sessions.

An unfortunate disadvantage of using HMAC for connection authentication is that the secret must be known in the clear by both parties, making this undesireable when keys are long-lived.

When suitable, HMAC should be used in preference to older techniques, notably keyed hash functions. Simple keyed hashes based on MD5 [RFC1321], such as that used in the BGP session security mechanism [RFC2385], are especially to be avoided in new protocols, given the hints of weakness in MD5.

HMAC can be implemented using any secure hash function, including MD5 and SHA-1 [RFC3174]. SHA-1 is preferable for new protocols because it is more frequently used for this purpose and may be more secure.

It is important to understand that an HMAC-based mechanism needs to be employed on every protocol data unit (aka packet). It is a mistake to use an HMAC-based system to authenticate the beginning of a TCP session and then send all remaining data without any protection.

Attack programs exist that permit a TCP session to be stolen. An attacker merely needs to use such a tool to steal a session after the HMAC step is performed.

5.3. IPsec

IPsec [RFC2401],[RFC2402],[RFC2406],[RFC2407],[RFC2411] is the generic IP-layer encryption and authentication protocol. As such, it protects all upper layers, including both TCP and UDP. Its normal granularity of protection is host-to-host, host-to-gateway, and gateway-to-gateway. The specification does permit user-granularity protection, but this is comparatively rare. As such, IPsec is currently inappropriate when host-granularity is too coarse.

Because IPsec is installed at the IP layer, it is rather intrusive. Implementing it generally requires either new hardware or a new protocol stack. This makes it a poor choice for individual applications, at least until IPsec is more widely deployed. Most modern operating systems have IPsec available; most routers do not, at least for the control path.

The key management for IPsec can use either certificates or shared secrets. For all the obvious reasons, certificates are preferred; however, they may present more of a headache for the system manager.

There is strong potential for conflict between IPsec and NAT [Hain99]. NAT does not easily co-exist with any protocol containing embedded IP address; with IPsec, every packet, for every protocol, contains such addresses, if only in the headers. The conflict can sometimes be avoided by using tunnel mode, but that is not always an appropriate choice for other reasons.

Most current IPsec usage is for virtual private nets. Assuming that the other constraints are met, IPsec is the security protocol of choice for VPN-like situations.

5.4. TLS

TLS [RFC2246] provides an encrypted, authenticated channel that runs on top of TCP. While TLS was originally designed for use by Web browsers, it is by no means restricted to such. In general, though, each application that wishes to use TLS will need to be converted individually.

Generally, the server side is always authenticated by a certificate. Clients may possess certificates, too, providing mutual authentication, though this is rarely deployed. It's an unfortunate reality that even server side authentication it not as secure in practice as the cryptography would imply because most implementations allow users to ignore authentication failures (by clicking OK to a warning) and most users routinely do so [Bell98]. Designers should

thus be wary of demanding plaintext passwords, even over TLS-protected connections. (This requirement can be relaxed if it is likely that implementations will be able to verify the authenticity and authorization of the server's certificate.)

Although application modification is generally required to make use of TLS, there exist toolkits, both free and commercial, that provide implementations. These are designed to be incorporated into the application's code.

5.5. SASL

SASL [RFC2222] is a framework for negotiating an authentication and encryption mechanism to be used over a TCP stream. As such, its security properties are those of the negotiated mechanism. Specifically, unless the negotiated mechanism authenticates all of the subsequent messages or underlying protection protocol such as TLS is used, TCP connections are vulnerable to session stealing.

If you need to use TLS (or IPSec) under SASL, why bother with SASL in the first place? Why not simply use the authentication facilities of TLS and be done with it?

The answer here is subtle. TLS makes extensive use of certificates for authentication. As commonly deployed, only servers have certificates, whereas clients go unauthenticated (at least by the TLS processing itself).

SASL permits the use of more traditional client authentication technologies, such as passwords (one-time or otherwise). A powerful combination is TLS for underlying protection and authentication of the server, and a SASL-based system for authenticating clients.

5.6. GSS-API

GSS-API [RFC2744] provides a framework for applications to use when they require authentication, integrity, and/or confidentiality. Unlike SASL, GSS-API can be used easily with UDP-based applications. It provides for the creation of opaque authentication tokens (aka chunks of memory) which may be embedded in a protocol's data units. Note that the security of GSS-API-protected protocols depends on the underlying security mechanism; this must be evaluated independently. Similar considerations apply to interoperability, of course.

5.7. DNSSEC

DNSSEC [RFC2535] digitally signs DNS records. It is an essential tool for protecting against DNS cache contamination attacks [Bell95]; these in turn can be used to defeat name-based authentication and to redirect traffic to or past an attacker. The latter makes DNSSEC an essential component of some other security mechanisms, notably IPsec.

Although not widely deployed on the Internet at the time of the writing of this document, it offers the potential to provide a secure mechanism for mapping domain names to IP protocol addresses. It may also be used to securely associate other information with a DNS name. This information may be as simple as a service that is supported on a given node, or a key to be used with IPsec for negotiating a secure session. (Note that the concept of storing application keys in the DNS is still controversial.)

5.8. Security/Multipart

Security/Multiparts [RFC1847] are the preferred mechanism for protecting email. More precisely, it is the MIME framework within which encryption and/or digital signatures are embedded. Both S/MIME and OpenPGP (see below) use Security/Multipart for their encoding. Conforming mail readers can easily recognize and process the cryptographic portions of the mail.

Security/Multiparts represents one form of "object security", where the object of interest to the end user is protected, independent of transport mechanism, intermediate storage, etc. Currently, there is no general form of object protection available in the Internet.

5.9. Digital Signatures

One of the strongest forms of challenge/response authentication is based on digital signatures. Using public key cryptography is preferable to schemes based on secret key ciphers because no server needs a copy of the client's secret. Rather, the client has a private key; servers have the corresponding public key.

Using digital signatures properly is tricky. A client should never sign the exact challenge sent to it, since there are several subtle number-theoretic attacks that can be launched in such situations.

The Digital Signature Standard [DSS] and RSA [RSA] are both good choices; each has its advantages. Signing with DSA requires the use of good random numbers [RFC1750]. If the enemy can recover the

random number used for any given signature, or if you use the same random number for two different documents, your private key can be recovered. DSS has much better performance than RSA for generating new private keys, and somewhat better performance generating signatures, while RSA has much better performance for verifying signatures.

5.10. OpenPGP and S/MIME

Digital signatures can be used to build "object security" applications which can be used to protect data in store and forward protocols such as electronic mail.

At this writing, two different secure mail protocols, OpenPGP [OpenPGP] and S/MIME [S/MIME], have been proposed to replace PEM [PEM]. It is not clear which, if either, will succeed. While specified for use with secure mail, both can be adapted to protect data carried by other protocols. Both use certificates to identify users; both can provide secrecy and authentication of mail messages; however, the certificate formats are very different. Historically, the difference between PGP-based mail and S/MIME-based mail has been the style of certificate chaining. In S/MIME, users possess X.509 certificates; the certification graph is a tree with a very small number of roots. By contrast, PGP uses the so-called "web of trust", where any user can sign anyone else's certificate. This certification graph is really an arbitrary graph or set of graphs.

With any certificate scheme, trust depends on two primary characteristics. First, it must start from a known-reliable source--either an X.509 root, or someone highly trusted by the verifier, often him or herself. Second, the chain of signatures must be reliable. That is, each node in the certification graph is crucial; if it is dishonest or has been compromised, any certificates it has vouched for cannot be trusted. All other factors being equal (and they rarely are), shorter chains are preferable.

Some of the differences reflect a tension between two philosophical positions represented by these technologies. Others resulted from having separate design teams.

S/MIME is designed to be "fool proof." That is, very little end-user configuration is required. Specifically, end-users do not need to be aware of trust relationships, etc. The idea is that if an S/MIME client says, "This signature is valid", the user should be able to "trust" that statement at face value without needing to understand the underlying implications.

To achieve this, S/MIME is typically based on a limited number of "root" Certifying Authorities (CAs). The goal is to build a global trusted certificate infrastructure.

The down side to this approach is that it requires a deployed public key infrastructure before it will work. Two end-users may not be able to simply obtain S/MIME-capable software and begin communicating securely. This is not a limitation of the protocol, but a typical configuration restriction for commonly available software. One or both of them may need to obtain a certificate from a mutually trusted CA; furthermore, that CA must already be trusted by their mail handling software. This process may involve cost and legal obligations. This ultimately results in the technology being harder to deploy, particularly in an environment where end-users do not necessarily appreciate the value received for the hassle incurred.

The PGP "web of trust" approach has the advantage that two end-users can just obtain PGP software and immediately begin to communicate securely. No infrastructure is required and no fees and legal agreements need to be signed to proceed. As such PGP appeals to people who need to establish ad-hoc security associations.

The down side to PGP is that it requires end-users to have an understanding of the underlying security technology in order to make effective use of it. Specifically it is fairly easy to fool a naive users to accept a "signed" message that is in fact a forgery.

To date PGP has found great acceptance between security-aware individuals who have a need for secure e-mail in an environment devoid of the necessary global infrastructure.

By contrast, S/MIME works well in a corporate setting where a secure internal CA system can be deployed. It does not require a lot of end-user security knowledge. S/MIME can be used between institutions by carefully setting up cross certification, but this is harder to do than it seems.

As of this writing a global certificate infrastructure continues to elude us. Questions about a suitable business model, as well as privacy considerations, may prevent one from ever emerging.

5.11. Firewalls and Topology

Firewalls are a topological defense mechanism. That is, they rely on a well-defined boundary between the good "inside" and the bad "outside" of some domain, with the firewall mediating the passage of information. While firewalls can be very valuable if employed

properly, there are limits to their ability to protect a network.

The first limitation, of course, is that firewalls cannot protect against inside attacks. While the actual incidence rate of such attacks is not known (and is probably unknowable), there is no doubt that it is substantial, and arguably constitutes a majority of security problems. More generally, given that firewalls require a well-delimited boundary, to the extent that such a boundary does not exist, firewalls do not help. Any external connections, whether they are protocols that are deliberately passed through the firewall, links that are tunneled through, unprotected wireless LANs, or direct external connections from nominally-inside hosts, weaken the protection. Firewalls tend to become less effective over time as users tunnel protocols through them and may have inadequate security on the tunnel endpoints. If the tunnels are encrypted, there is no way for the firewall to censor them. An oft-cited advantage of firewalls is that they hide the existence of internal hosts from outside eyes. Given the amount of leakage, however, the likelihood of successfully hiding machines is rather low.

In a more subtle vein, firewalls hurt the end-to-end model of the Internet and its protocols. Indeed, not all protocols can be passed safely or easily through firewalls. Sites that rely on firewalls for security may find themselves cut off from new and useful aspects of the Internet.

Firewalls work best when they are used as one element of a total security structure. For example, a strict firewall may be used to separate an exposed Web server from a back-end database, with the only opening the communication channel between the two. Similarly, a firewall that permitted only encrypted tunnel traffic could be used to secure a piece of a VPN. On the other hand, in that case the other end of the VPN would need to be equally secured.

5.12. Kerberos

Kerberos [RFC1510] provides a mechanism for two entities to authenticate each other and exchange keying material. On the client side, an application obtains a Kerberos "ticket" and "authenticator." These items, which should be considered opaque data, are then communicated from client to server. The server can then verify their authenticity. Both sides may then ask the Kerberos software to provide them with a session key which can be used to protect or encrypt data.

Kerberos may be used by itself in a protocol. However, it is also available as a mechanism under SASL and GSSAPI.

5.13. SSH

SSH provides a secure connection between client and server. It operates very much like TLS; however, it is optimized as a protocol for remote connections on terminal-like devices. One of its more innovative features is its support for "tunneling" other protocols over the SSH-protected TCP connection. This feature has permitted knowledgeable security people to perform such actions as reading and sending e-mail or news via insecure servers over an insecure network. It is not a substitute for a true VPN, but it can often be used in place of one.

6. Insecurity Mechanisms

Some common security mechanisms are part of the problem rather than part of the solution.

6.1. Plaintext Passwords

Plaintext passwords are the most common security mechanism in use today. Unfortunately, they are also the weakest. When not protected by an encryption layer, they are completely unacceptable. Even when used with encryption, plaintext passwords are quite weak, since they must be transmitted to the remote system. If that system has been compromised or if the encryption layer does not include effective authentication of the server to the client, an enemy can collect the passwords and possibly use them against other targets.

Another weakness arises because of common implementation techniques. It is considered good form [MT79] for the host to store a one-way hash of the users' passwords, rather than their plaintext form. However, that may preclude migrating to stronger authentication mechanisms, such as HMAC-based challenge/response.

The strongest attack against passwords, other than eavesdropping, is password-guessing. With a suitable program and dictionary (and these are widely available), 20-30% of passwords can be guessed in most environments [Klein90].

6.2. Address-Based Authentication

Another common security mechanism is address-based authentication. At best, it can work in highly constrained environments. If your environment consists of a small number of machines, all tightly administered, secure systems run by trusted users, and if the network

is guarded by a router that blocks source-routing and prevents spoofing of your source addresses, and you know there are no wireless bridges, and if you restrict address-based authentication to machines on that network, you are probably safe. But these conditions are rarely met.

Among the threats are ARP-spoofing, abuse of local proxies, renumbering, routing table corruption or attacks, DHCP, IP address spoofing (a particular risk for UDP-based protocols), sequence number guessing, and source-routed packets. All of these can be quite potent.

6.3. Name-Based Authentication

Name-based authentication has all of the problems of address-based authentication and adds new ones: attacks on the DNS [Bell95] and lack of a one to one mapping between addresses and names. At a minimum, a process that retrieves a host name from the DNS should retrieve the corresponding address records and cross-check. Techniques such as DNS cache contamination can often negate such checks.

DNSSEC provides protection against this sort of attack. However, it does nothing to enhance the reliability of the underlying address. Further, the technique generates a lot of false alarms. These lookups do not provide reliable information to a machine, though they might be a useful debugging tool for humans and could be useful in logs when trying to reconstruct how and attack took place.

7. Security Considerations

No security mechanisms are perfect. If nothing else, any network-based security mechanism can be thwarted by compromise of the endpoints. That said, each of the mechanisms described here has its own limitations. Any decision to adopt a given mechanism should weigh all of the possible failure modes. These in turn should be weighed against the risks to the endpoint of a security failure.

8. Acknowledgements

Brian Carpenter, Tony Hain, and Marcus Leech made a number of useful suggestions. Much of the substance comes from the participants in the IAB Security Architecture Workshop.

9. References

[RFC2316] "Report of the IAB Security Architecture Workshop". S. Bellovin. April 1998.

[RFC2289] "A One-Time Password System". N. Haller, C. Metz, P. Nesser, M. Straw. February 1998.

[RFC2104] "HMAC: Keyed-Hashing for Message Authentication". H. Krawczyk, M. Bellare, R. Canetti. February 1997.

[RFC1321] "The MD5 Message-Digest Algorithm". R. Rivest. April 1992.

[RFC2246] "The TLS Protocol Version 1.0. T. Dierks, C. Allen. January 1999."

[RFC2385] "Protection of BGP Sessions via the TCP MD5 Signature Option". A. Hefferman. August 1998.

[RFC2401] "Security Architecture for the Internet Protocol". S. Kent, R. Atkinson. November 1998.

[RFC2402] "IP Authentication Header. S. Kent, R. Atkinson. November 1998."

[RFC2406] "IP Encapsulating Security Payload (ESP). S. Kent, R. Atkinson. November 1998."

[RFC2407] "The Internet IP Security Domain of Interpretation for ISAKMP. D. Piper. November 1998."

[RFC2411] "IP Security Document Roadmap". R. Thayer, N. Doraswamy, R. Glenn. November 1998.

[RFC2744] "Generic Security Service API Version 2 : C-bindings. J. Wray. January 2000."

[RFC3174] "US Secure Hash Algorithm 1 (SHA1)". D. Eastlake, 3rd, and P. Jones. September 2001.

[Hain99] "Architectural Implications of NAT". T. Hain. April 1999. Work in progress.

[Bell95] "Using the Domain Name System for System Break-Ins". Proc. Fifth Usenix Security Conference, 1995.

[Bell98] "Cryptography and the Internet", S.M. Bellovin, in Proceedings of CRYPTO '98, August 1998.

[RFC2222] "Simple Authentication and Security Layer (SASL)". J. Myers. October 1997.

[RFC2535] "Domain Name System Security Extensions". D. Eastlake. March 1999.

[RFC1847] "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted". J. Galvin, S. Murphy, S. Crocker & N. Freed. October 1995.

[DSS] "Digital Signature Standard". NIST. May 1994. FIPS 186.

[RSA] "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". R. Rivest, A. Shamir, L.M. Adleman, Communications of the ACM, February 1978.

[RFC1750] "Randomness Recommendations for Security". D. Eastlake, 3rd, S. Crocker & J. Schiller. December 1994.

[MT79] "UNIX Password Security", R.H. Morris and K. Thompson, Communications of the ACM. November 1979.

[Klein90] "'Foiling the Cracker': A Survey of, and Implications to, Password Security". D. Klein. Usenix UNIX Security Workshop, August 1990.

10. Author Information

Steven M. Bellovin
AT&T Labs Research
Shannon Laboratory
180 Park Avenue
Florham Park, NJ 07974
USA
Phone: +1 973-360-8656
email: smb@research.att.com

Charlie Kaufman
IBM - WTF5
4 Technology Park Drive
Westford, MA 01886
USA
Phone: +1 978-399-6383
email: ckaufman@us.ibm.com

Jeffrey I. Schiller
Massachusetts Institute of Technology
Room W92-190
77 Massachusetts Avenue
Cambridge, MA 02139-4307
USA
Phone: +1 617-253-8400
email: jis@mit.edu