Key Exchange in IPSec:
Analysis of IKE

RADIA PERLMAN
Sun Microsystems Laboratories
CHARLIE KAUFMAN
Iris Associates

The IPSec protocol is a recently proposed standard of the IETF for securing real-time communications on the Internet. The authors explain how its key exchange mechanism (IKE) works and suggest improvements.

A lot has been written recently about the IPSec standard.[1,2] It is criticized for being overly complex, partially because of allowing too many ways of accomplishing essentially the same thing. Most of the debate focuses on why there are so many options, for example, authentication header (AH)[3] and encapsulation security payload (ESP)[4] data packet encodings, and tunnel versus transport modes. In this article we focus on the Internet Key Exchange (IKE)[5] mechanism, which has not been as thoroughly studied beyond its cryptographic properties.[6] We focus on the properties of the standard as it stands today, and give suggestions for improvement. In order to be comprehensible, we do not describe all the fields in all the messages, but rather simplify it down to the essence needed for our analysis.

BACKGROUND

IPSec is the Internet Engineering Task Force (IETF) proposed standard for "layer 3 real-time communication security." In a real-time security system an *initiator*, say Alice, initiates communication with a *responder* system, say Bob. They authenticate to each other by proving knowledge of some secret, and then establish a secret key for the protection (integrity and/or privacy) of the remainder of the session. We use the term "real-time" to distinguish it from a system such as secure e-mail, in which Alice can create an encrypted, signed message for Bob without interacting with Bob.

IPSec can be thought of as a protocol that operates "on top" of IP (layer 3) but "below" layer 4 (TCP or UDP or any other potential layer 4 protocol), meaning that it encrypts each data packet independently of all others and if packets are lost or delayed, layer 4 (the software that requests retransmission of lost packets) sees only validated information. There are other real-time communication protocols that operate in other layers. For example, secure sockets layer (SSL)[7] operates above layer 4 so an entire stream is encrypted rather than individual packets, and it is the job of TCP to break the stream into individual packets and to make sure that they all arrive, in order, before presenting them to SSL at the other end. An argument for operating above

 1089-7801/00/$10.00 ©2000 IEEE

layer 4 is that security can be deployed without changes to the operating system (typically all the layers up to and including layer 4 are implemented in the kernel). A problem with operating above TCP is that TCP will, by definition, not be participating in the cryptography and will have no way of knowing whether a particular packet is cryptographically valid. If an active attacker injects a TCP packet with a valid sequence number, the TCP implementation at the receiver will acknowledge the packet, and pass it up to SSL. SSL will detect the data as invalid, but it's now too late, since there is no way for SSL to inform TCP that the data that TCP had accepted was invalid. If the cryptographically valid packet arrives with a sequence number that TCP has already acknowledged, TCP will discard it as a duplicate and SSL will never receive it. As a result, a single "rogue" packet can cause a connection to break.

In contrast, IPSec is not vulnerable to this form of active attack, since IPSec will be able to recognize and discard invalid packets and TCP will only receive valid data. Although IPSec is a technically superior solution, SSL was widely deployed much earlier, partly because it did not depend on kernel changes, and because IPSec emerged from the standards process so late. As a result, IPSec may wind up not being deployed in many situations where it would be appropriate, because SSL will be "good enough."

The two main pieces of IPSec are the data packet encodings (AH and ESP) and the key exchange portion (IKE). IKE uses some key associated with the parties, such as a preshared secret key or individuals' public keys to authenticate and establish a session key. After the exchange, the remainder of the session is cryptographically protected with the session key.

The design of IPSec took place within the IETF's IPSec working group over a long period of time with many contributors. Some of its features are best understood in the context of how the design evolved. It began as two relatively simple proposals called Photuris[8] and SKIP.[9] When the proponents of the competing protocols failed to reach agreement, ISAKMP,[10] a general-purpose syntax in which many different protocols could be encoded, was agreed to in order to make progress. At around the same time, a third more complex protocol attempting to combine the advantages of the two protocols was proposed called Oakley.[11] Finally, a document originally titled "ISAKMP/Oakley Resolution Document" evolved into what is now known as IKE.[5]

There has been significant debate about the necessity for AH, which provides only integrity protection, since ESP can provide integrity protection or encryption or both. The integrity protection provided by AH extends to portions of the IP header, whereas ESP's integrity protection is only of the payload. The opponents of AH argue that it is unnecessary to protect the IP header, and if it were necessary, could be provided by tunnel mode. The debate has been heated and the issues are complex.[1] Although we believe that AH is unnecessary, we will focus on the IKE portion of IPSec rather than rehashing those issues here.

TRANSPARENT TO APPLICATIONS?

Because IPSec is implemented at layer 3, it is possible to deploy it within an operating system without changing the applications. However, the security gained by implementing IPSec without changing the applications is not as strong as many people assume.

Implementing IPSec without changing the application has the same effect as putting firewalls between the two systems and implementing IPSec between the firewalls. It accomplishes the following:

- It causes the traffic on the path between the two firewalls to be encrypted, hiding it from eavesdroppers.
- As with firewalls, IPSec can access a policy database that specifies which IP addresses are allowed to talk to which other IP addresses.
- Some applications do authentication based on IP addresses, and the IP address from which information is received is passed up to the application. With IPSec, this form of authentication can become much more secure because one of the types of endpoint identifiers IPSec can authenticate is an IP address, making it impossible for a node that does not know the key associated with that IP address to impersonate the source.

What IPSec (with the current API and an unmodified application) does *not* accomplish for the application is authentication of anything other than IP addresses. Most principals would have some identity such as a name, and be allowed to access the network from a variety of IP addresses. In these cases, the most likely scenario for IPSec operation is that IPSec would do its highly secure and expensive authentication, establishing a security association to some name, but would have no way of telling the application who is on the other side. The

application would have to depend on existing mechanisms, most likely a name and password, to determine who it is talking to. IPSec is still of value in this scenario, since the name and password will be encrypted when transmitted.

An unmodified application would also not be able to detect whether the connection was protected by IPSec or not, and so would be subject to a "configuration attack" where IPSec is turned off and the application continues to run unprotected.

To take full advantage of IPSec, applications will have to change. The API has to change in order to pass identities other than IP addresses, and the applications have to change to make use of this information.

IKE PHASES

The IKE exchange consists of two phases. The phase 1 exchange assumes each of the two parties involved in the exchange has an identity (name) by which the other side knows them, and associated with that identity is some sort of secret that can be verified by the other side. This secret might be a preshared secret key or the private portion of a key pair. Phase 1 does mutual authentication based on that secret, and establishes a session key used to protect the remainder of the session.

The phase 1 exchange happens once (and before any phase 2 exchanges), and allows subsequent setup of multiple phase 2 connections between the same pair of nodes. The phase 2 exchange relies on the session key established in phase 1 to do mutual authentication and establish a phase 2 session key used to protect all the data in the phase 2 security association.

It would certainly be simpler and cheaper to just set up a security association in a single exchange, and do away with the phases. The argument on the other side is that the phase 1 exchange is expensive, but the phase 2 exchanges can then be less expensive because they can utilize the session key created out of the phase 1 exchange. So if you anticipate many such exchanges, you can save processing by breaking setup into an expensive phase 1 and many cheap phase 2s. This reasoning only makes sense if there will be multiple phase 2 setups inside the same phase 1 exchange. Why would there be multiple exchanges between the same pair of nodes? Here are the traditional arguments in favor of having two phases:

- It is a good idea to change keys periodically. You can do key rollover of a phase 2 connection by doing another phase 2 connection setup, which would be cheaper than restarting the phase 1 connection setup.
- You can set up multiple connections with different security properties, such as integrity-only, encryption with a short (insecure, snooper-friendly) key, or encryption with a strong key.
- You can set up multiple connections between two nodes because the connections are application-to-application, and you'd like each application to use its own key, perhaps so that the IPSec layer can give the key to the application. A vulnerability is described in Bellovin[12] if the same key is used for multiple connections, assuming these connections are not utilizing integrity protection (that is, they are only using encryption).

We argue against each of these points:

- If you want perfect forward secrecy when you do a key rollover, then the phase 2 exchange is not significantly cheaper than doing another phase 1 exchange. If you are simply rekeying, either to limit the amount of data encrypted with a single key, or to prevent replay after the sequence number wraps around, then a protocol designed specifically for rekeying would be simpler and less expensive than the IKE phase 2 exchange.
- It would be logical to use the strongest protection needed by any of the traffic for all the traffic rather than having separate security associations in order to give weaker protection to some traffic. There might be some legal or performance reasons to want to use different protection for different forms of traffic, but we claim that this will be a relatively rare case that we need not optimize. A cleaner method of doing this would be to have completely different security associations rather than multiple security associations loosely linked together with the same phase 1 security association.
- Wanting to have each application have a separate key also seems likely to be a rare case, and in that case, setting up a totally unrelated security association for each application would suffice. In some cases, different applications (or different users) use different identities to authenticate. In that case they would need to have separate Phase 1 security associations anyway.
- The vulnerability addressed in Bellovin[12] is really an argument for never using encryption without integrity protection, rather than an argument for using different keys for different applications.

In this article we concentrate on phase 1 of IKE. Aside from arguably being unnecessary, we did not

find any problems with the security or functionality of IKE's phase 2.

OVERVIEW OF IKE'S PHASE 1

There are *eight* variants of phase 1 of IKE. This is because there are three types of keys (preshared, public encryption keys, and public signature keys), and additionally there are two versions of protocols based on public encryption keys, one of which is intended to replace the other, but the first must still be documented for backward compatibility. Thus there are four "types" of keys (preshared secret key, old-style public encryption key, new-style public encryption key, and public signature key). And for each type of key there are two types of phase 1 exchange: "main" and "aggressive." The main mode has six messages and is intended to provide additional functionality, such as hiding endpoint identifiers, whereas the aggressive mode has only three messages. The variants have surprisingly different characteristics.

In main mode there are three pairs of messages. In the first pair Alice sends a "cookie" (see following section) and requested cryptographic algorithms, and Bob responds with his cookie value and the cryptographic algorithms he will agree to. The second pair of messages consists of a Diffie-Hellman exchange. In the third pair of messages, which are encrypted with the Diffie-Hellman value agreed upon in the second pair of messages, each side reveals its identity and proves it knows the relevant secret (for example, private key or pre-shared secret key). In aggressive mode there are only three messages. The first two messages consist of a Diffie-Hellman exchange to establish a session key, and in the second and third messages each side proves they know both the Diffie-Hellman value and their secret.

COOKIES

Stateless cookies were originally proposed in Photuris.[8] The motivation is as follows: Bob has finite memory and computation capacity. An attacker could mount a "denial of service" attack against Bob by sending fake connection requests from random IP addresses and using up Bob's memory for connections in progress and computation power for cryptography. While the attacker could never get Bob to do anything wrong, she could prevent Bob from servicing requests from legitimate users. To prevent this, Bob will not keep any state nor do any significant computation unless the connect request is accompanied by a number, known as a "cookie," that consists of some function of the IP address from which the connection is made and a secret known only to Bob. In order to connect to Bob, Alice makes an initial request for a cookie. After responding with the cookie, Bob does not need to remember anything about the connect request. Then, when Alice contacts Bob again with a valid cookie, Bob will keep the state and do the processing necessary to authenticate Alice and create a security association. By only accepting connection requests with valid cookies, Bob can be assured that they come from nodes that can receive data at the address from which they claim to be sending.

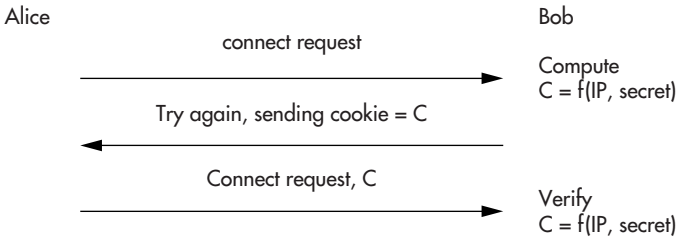

Figure 1. Oakley stateless cookies.

If Bob had enough memory to hold the maximum number of connect requests that could possibly arrive within the time window before he could time them out and delete the state for the uncompleted connection, stateless cookies would not be necessary. And although stateless cookies are probably necessary, one could argue that they're not sufficient, since they don't protect against an attacker, Trudy, launching packets from IP addresses at which she *can* receive responses. But cookies almost certainly make it easier to trace where the attacks are coming from, since Trudy's attack will only work if she can receive at the IP address she is claiming to be sending from. Even if it can't be traced down to the exact node so a person or faulty node could be blamed and disconnected from the net, Bob or a firewall can be configured to drop packets from the entire set of IP addresses Trudy can receive.

Oakley[11] allowed the cookies to be optional. If Bob was not being attacked and therefore had sufficient resources, he could accept connection requests without cookies. A round-trip delay and two messages could be saved. In Photuris the cookie (and the extra two messages) was always required. The idea behind the Oakley stateless cookies is shown in Figure 1.

Surprisingly, although IKE was designed years after Photuris, and it has fields in the messages named "cookies," none of the IKE variants allows Bob to be stateless. This was pointed out in Simpson.[2] In the "main mode" variants, the cookie protects Bob from being forced to do a significant

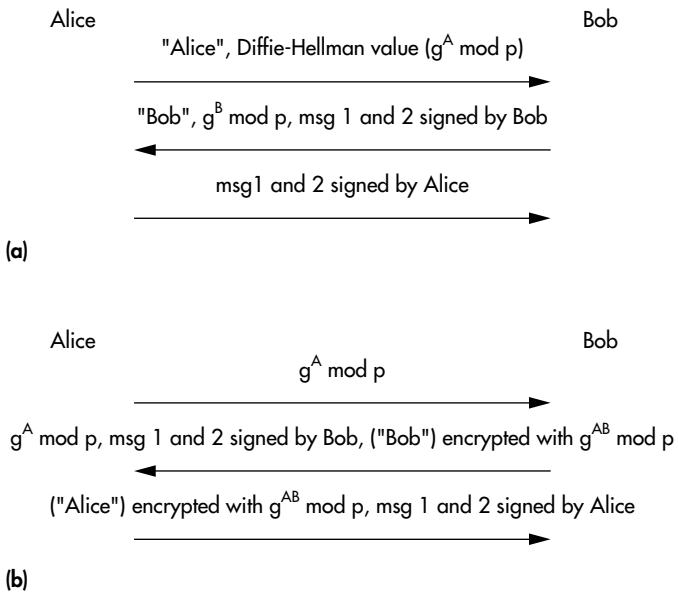

Figure 2. Signature key variant in aggressive mode: (a) current exchange and (b) modified exchange.

amount of computation. However, IKE requires Bob to keep state from the first message, before he knows whether the other side would be able to return a cookie. It would be straightforward to add two messages to IKE to allow for a stateless cookie. In fact, we claim stateless cookies could have been implemented in all of the variants of IKE main mode without additional messages by repeating in message 3 the information in message 1.

HIDING ENDPOINT IDENTIFIERS

One of the design goals of main mode was to be able to hide the endpoint identifiers. The goal would be to prevent an eavesdropper from learning who was communicating (though the source and destination IP addresses would always provide a hint). With some key types it is difficult to design a protocol to prevent an active attacker from learning the identity of one end or the other. In this case we'd argue it would be better for the protocol to hide the initiator's identity rather than the responder's (because the responder is likely to be at a fixed IP address so that it can be easily found while the initiator may roam and arrive from a different IP address each day). Keeping that in mind, we'll summarize how well the variants do at hiding endpoint identifiers.

In all of the aggressive mode variants, both endpoint identities are exposed, as would be expected. Surprisingly, however, we noticed that the signature key variant of aggressive mode could have easily been modified, with no technical disadvantages, to hide both endpoint identifiers from an eavesdropper, and the initiator's identity even from an active attacker! The relevant portion of that protocol is shown in Figure 2a.

The endpoint identifiers could have been hidden by removing them from messages 1 and 2 and including them, encrypted with the Diffie-Hellman shared value g^{AB} mod p, in messages 2 (Bob's identifier) and 3 (Alice's identifier). The modified exchange would look like Figure 2b.

In the next sections we discuss how the main mode protocols hide endpoint identifiers.

Public Signature Keys

In the public signature key main mode, Bob's identity is hidden even from an active attacker, but Alice's identity is exposed to an active attacker impersonating Bob's address to Alice. The relevant part of the protocol is shown in Figure 3.

An active attacker impersonating Bob's address to Alice will negotiate a Diffie-Hellman key with Alice and discover her identity in message 5. The active attacker will not be able to complete the protocol since it will not be able to generate Bob's signature in message 6.

The protocol could be modified to hide Alice's identity instead of Bob's from an active attacker. This would be done by moving the information from message 6 into message 4. This even completes the protocol in one fewer message. And as we said earlier, it is probably in practice more important to hide Alice's identity than Bob's.

Public Encryption Keys

In this variant both sides' identities are protected even against an active attacker. Although the protocol is much more complex, the main idea is that the identities (as well as the Diffie-Hellman values in the Diffie-Hellman exchange) are transmitted encrypted with the other side's public key, so they will be hidden from anyone who doesn't know the other side's public key.

Pre-Shared Key

In this variant, both endpoints' identities are revealed, even to an eavesdropper! The relevant part of the protocol is shown in Figure 4.

Since the endpoint identifiers are exchanged encrypted, it would seem as though both endpoint identifiers would be hidden. However, Bob has no idea who he is talking to after message 4, and the

key with which messages 5 and 6 are encrypted is a function of the pre-shared key between Alice and Bob. So Bob can't decrypt message 5, which reveals Alice's identity, unless he already knows, based on messages 1-4, who he is talking to!

The IKE spec recognizes this property of the protocol, and specifies that in this mode the endpoint identifiers have to be the IP addresses. In which case, there's no reason to include them in messages 5 and 6 since Bob (and an eavesdropper) already knows them!

Main mode with pre-shared keys, happens to be the only required protocol. One of the reasons you'd want to use IPSec is in the scenario in which Alice, an employee traveling with her laptop, connects into the corporate network from across the Internet. IPSec with pre-shared keys would seem a logical choice for implementing this scenario. However, the protocol as designed is completely useless for this scenario since by definition Alice's IP address will be unpredictable if she's attaching to the Internet from different locations.

It would be easy to fix the protocol. The fix is to encrypt messages 5 and 6 with a key that is a function of the shared Diffie-Hellman value, and not also a function of the preshared key. In this way an active attacker who is acting as a "man in the middle" in the Diffie-Hellman exchange would be able to discover the endpoint identifiers, but an eavesdropper would not. And more importantly than whether the endpoint identifiers are hidden, it allows use of true endpoint identifiers, such as the employee's name, rather than IP addresses. This change would make this mode useful in the scenario (travelling employee) in which it would be most valuable.

NEGOTIATING SECURITY PARAMETERS

IKE allows the two sides to negotiate which encryption, hash, integrity protection, and Diffie-Hellman parameters they will use. Alice makes a proposal of a set of algorithms and Bob chooses. Bob does not get to choose one from column A, one from column B, one from column C, and one from column D, so to speak. Instead Alice transmits a set of complete proposals. While this is more powerful in the sense that it can express the case where Alice can only support certain combinations of algorithms, it greatly expands the encoding in the common case where Alice is capable of using the algorithms in any combination. For instance, if Alice can support three of each type of algorithm, and would be happy with any combination, she'd have to specify 81 (3^4) sets of choices to Bob in order to tell Bob all the combinations she can support! Each choice takes 20 bytes to specify: 4 bytes for a header and 4 bytes each for encryption, hash, authentication, and Diffie-Hellman.

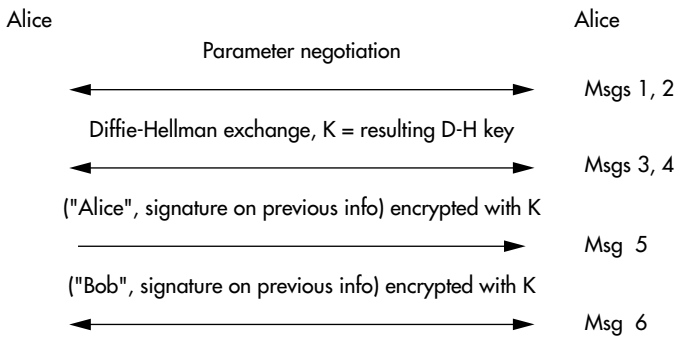

Figure 3. Public signature key main mode.

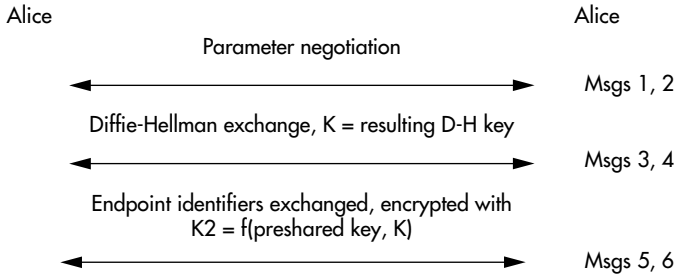

Figure 4. Preshared key main mode.

CONCLUSION

The main points covered in the article are:

- By operating below layer 4, IPSec avoids the problem of an active attacker fatally disrupting a session by injecting a single rogue packet. Solutions such as SSL, which operate above TCP, are vulnerable to this threat.
- Although IPSec can be deployed without changes to applications, the power of IPSec cannot be exploited until the API is changed to inform applications of the endpoint identifier, and applications are modified to use the information in the modified API.
- IKE is far too complex, and the specifications are so difficult to understand that it has not gotten a thorough review; many of the properties we point out were not known.
- One major simplification of IKE would be to remove the second phase.
- IKE does not allow stateless cookies. We pro-

pose modifying IKE by repeating in message 3 the information in message 1, allowing stateless cookies without adding messages.

- The encoding should be changed to allow negotiating sets of independent choices of cryptographic parameters, to avoid exponential explosion.
- In some modes it is only possible to hide one endpoint's identity. It is better to hide the initiator's identity. In some modes IKE only hides the responder's identity.
- The only mandated IKE key type, pre-shared secret keys, forces the endpoint identifiers to be the IP addresses in the packet. This makes this mode useless for the case where a traveling employee wishes to log into the corporate network using his laptop. We described how to modify IKE to allow much more useful endpoint identifiers, at no cost in security or performance. ■

REFERENCES

1. N. Ferguson and B. Schneier, "A Cryptographic Evaluation of IPSec," available online at http://www.counterpane.com/ipsec.html, Apr. 1999.
2. W.A. Simpson, "IKE/ISAKMP Considered Dangerous," Internet Draft (draft-simpson-danger-isakmp-00.txt), Apr. 1999 (expired Oct.1999).
3. S. Kent and R. Atkinson, "IP Authentication Header," Internet Engineering Task Force RFC 2402, Nov. 1998; available at http://www.ietf.org/rfc/rfc2402.txt.
4. S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)," IETF RFC 2406, Nov. 1998; available at http://www.ietf.org/rfc/rfc2406.txt.
5. D. Harkins and D. Carrel, "The Internet Key Exchange Protocol (IKE)," IETF RFC 2409, Nov. 1998; available at http://www.ietf.org/rfc/rfc2409.txt.
6. C. Meadows, "Analysis of the Internet Key Exchange Protocol Using the NRL Protocol Analyzer," *IEEE Symp. Security and Privacy*, IEEE Computer Soc. Press, Los Alamitos, Calif., 1999.
7. A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 Protocol," Netscape Communications Corp., Nov. 1996; available at http://home.netscape.com/eng/ssl3/.
8. P. Karn, "The Photuris Key Management Protocol," Internet Draft (draft-karn-photuris-00.txt), Dec. 1994 (expired June 1995).
9. G. Caronni et al., "SKIP—Securing the Internet," *Proc. Fifth Workshop on Enabling Technologies* (WET-ICE), 1996; available online at http://www.skip-vpn.org/wet-ice.html.
10. D. Maughan et al., "Internet Security Association and Key Management Protocol (ISAKMP)," IETF RFC 2408, Nov. 1998; available at http://www.ietf.org/rfc/rfc2408.txt.
11. H. Orman, "The OAKLEY Key Determination Protocol," IETF RFC 2412, Nov. 1998; available at http://www.ietf.org/rfc/rfc2412.txt.
12. S. Bellovin, "Problem Areas for the IP Security Protocols," *Proc. Sixth Usenix Security Symp.*, Usenix Assoc., Berkeley, Calif., 1996; available at http://www.usenix.org/publications/library/proceedings/sec96/bellovin.html.

Radia Perlman is a distinguished engineer at Sun Microsystems. Her research interests involve network protocols. She is known for her contributions to bridging (spanning-tree algorithm) and routing (link-state routing) as well as to security (sabotage-proof networks). She wrote *Interconnections: Bridges, Routers, Switches, and Internetworking Protocols*, is coauthor of *Network Security: Private Communication in a Public World*, and holds about 50 patents in the fields of routing and security. Perlman has a PhD in computer science and degrees in mathematics from Massachusetts Institute of Technology as well as an honorary doctorate of technology from KTH.

Charlie Kaufman is security architect for Lotus Notes and Domino at Iris Associates. He was a member of the National Research Council expert panel on Information Systems Trustworthiness that produced the report "Trust in Cyberspace." He participates in several IETF standards efforts and is chair of the Web Transaction Security working group. He is coauthor of *Network Security: Private Communication in a Public World*.

Readers can contact Perlman at radia.perlman@sun.com and Kaufman at ckaufman@iris.com.