The Internet Protocol Journal

March 2002 Volume 5, Number 1

A Quarterly Technical Publication for Internet and Intranet Professionals

In This Issue

You can download IPJ back issues and find subscription information at: **www.cisco.com/ipj**

From The Editor

Major Internet events such as the IETF meetings, the Regional Internet Registry meetings, APRICOT, SIGCOMM, and NetWorld+Interop to name a few, all provide Internet access for attendees. Commonly referred to as the "Terminal Room," these facilities have evolved into complex high-speed networks with redundant paths, IPv6 routing, multicast, and more. In the last five years or so, these networks have also been providing wireless access using various flavors of the IEEE 802.11 standard. As I write this, I am sitting in the lobby of the Minneapolis Hilton Hotel, where the 53rd IETF meeting is being held. The lobby area and two floors of meeting rooms have IEEE 802.11 coverage, and a directional high-gain antenna provides access in the pub across the street. Wireless Internet computing is a reality, at least when you have a large gathering of engineers such as an IETF meeting. In our first article, Edgar Danielyan takes a closer look at this technology, its applications and evolution.

More and more software is being distributed via the Internet rather than through the use of conventional media such as CD ROMs or floppy disks. Downloading software via the Internet is very convenient, especially if you have reasonably high bandwidth. However, with this convenience comes a certain risk that you may be receiving a modified copy of the software, perhaps one that contains a virus. Code signing is a method wherein software is cryptographically signed and later verified. Eric Fleischman explains the details of code signing.

I should have known better than to announce the imminent availability of our online subscription system in the previous issue. We are working on it, but it isn't ready yet, so please continue to send your subscription requests and updates to: **ipj@cisco.com**

—Ole J. Jacobsen, Editor and Publisher
ole@cisco.com

IEEE 802.11

by Edgar Danielyan

Introduced in 1997, the IEEE Standard 802.11 for wireless local-area networks has seen modifications and improvements in the past years and is promising a brighter wireless future, so yearned for by many of us. However, during its lifetime, the standard also has had a few setbacks, which are reminders that nothing is perfect in this world, much less in networking. This article provides a brief but comprehensive introduction to IEEE 802.11 wireless networking, its present and future, and highlights some of its security, performance, and safety aspects.

IEEE 802.11

The initial IEEE Standard 802.11 was published by the *Institute of Electrical and Electronics Engineers* (IEEE) in 1997. That standard is known as IEEE 802.11-1997 and is now updated by the current standard, IEEE 802.11-1999. The current standard has also been accepted as an American national standard by the *American National Standards Institute* (ANSI) and has been adopted by the *International Organization for Standardization* (ISO) as ISO/IEC 8802-11:1999. The completion of IEEE 802.11 in 1997 set in motion the development of standards-based wireless LAN networking. The 1997 standard specified a bandwidth of 2 Mbps, with fallback to 1 Mbps in hostile (noisy) environments with *Direct Sequence Spread Spectrum* (DSSS) modulation, and bandwidth of 1 Mbps with *Frequency Hopping Spread Spectrum* (FHSS) modulation, with possible 2-Mbps operation in friendly (noiseless) environments. Both methods operate in the unlicensed 2.4-GHz band. What is less known about IEEE 802.11 is that it also defines a baseband infrared medium, in addition to the DSSS and FHSS radio specifications, although its usefulness seems somewhat limited. There are also several task groups inside the 802.11 working group itself that work on substandards of 802.11:

- 802.11D: Additional Regulatory Domains
- 802.11E: Quality of Service (QoS)
- 802.11F: Inter-Access Point Protocol (IAPP)
- 802.11G: Higher data rates at 2.4 GHz
- 802.11H: Dynamic Channel Selection and Transmission Power Control
- 802.11i: Authentication and Security

The IEEE 802 group has an official Web site at **`www.ieee802.org`**, and IEEE 802.11 has an official Web site at **`www.ieee802.org/11/`**.

DSSS

Direct Sequence Spread Spectrum (DSSS) is one of the modulation techniques provided for by the IEEE 802.11 and the one chosen by the 802.11 Working Group for the widely used IEEE 802.11b devices. DSSS modulation is governed in the United States by FCC Regulation 15.247 and in Europe by ETSI Regulations 300-328. DSSS in IEEE 802.11 uses *Differential Binary Phase Shift Keying* (DBPSK) for 1 Mbps, and *Differential Quadrature Phase Shift Keying* (DQPSK) for 2 Mbps. The *Higher-Rate DSSS* (DSSS/HR) defined in IEEE 802.11b uses *Complementary Code Keying* (CCK) as its modulation scheme and provides 5.5- and 11-Mbps data rates. Because of their compatibility, all three modulation schemes can coexist using the rate-switching procedures defined in the IEEE 802.11. The *Orthogonal Frequency Division Multiplexing* (OFDM) used by the IEEE 802.11a is regulated in the United States by Title 47 Section 15.407 of the U.S. *Code of Federal Regulation* (CFR). IEEE 802.11a uses a system of 52 subcarriers modulated by BPSK or QPSK and 16-quadrature amplitude modulation. It also uses *forward error correction* (FEC) coding, also used by the Digital Video Broadcasting (DVB) standard with coding rates of 1/2, 2/3, and 3/4.

FHSS

Although specified by the original IEEE 802.11, *Frequency Hopping Spread Spectrum* (FHSS) modulation is not favored by vendors and, it seems, the 802.11 working group itself. DSSS has won the battle—very few vendors support 802.11/FHSS, and further developments with 802.11 use DSSS. Some have expressed ideas that frequency hopping in FHSS may contribute to the security of 802.11, but these are invalid expectations—the hopping codes used by FHSS are specified by the standard and are available to anyone, thus making the expectation of security through FHSS unreasonable.

Two supplements to the IEEE 802.11-1999, known as IEEE 802.11a and IEEE 802.11b, brought considerable changes and improvements to the IEEE 802.11-1999 standard.

IEEE 802.11a

IEEE 802.11a specifies a high-speed physical layer operating in the 5-GHz unlicensed band utilizing a complex coding technique known as OFDM. The data rates specified by IEEE 802.11a are 6, 9, 12, 18, 24, 36, 48, and 54 Mbps, with support for 6, 12, and 24 Mbps as a mandatory requirement. IEEE 802.11a is seen by some in the industry as the future of IEEE 802.11. Some products already implement the IEEE 802.11a, such as the chip from Atheros (**www.atheros.com**) and a PCMCIA/CardBus adapter from Card Access Inc (**www.cardaccess-inc.com**) based on it. However, 802.11a is not without disadvantages. The increased bandwidth of IEEE 802.11a results in a shorter operation range.

Additionally, because of the protocol overhead and interference/error correction, the real bandwidth may be considerably less than the nominal. New surveys and installation will also be required in many cases; the underlying infrastructure will also be more expensive because of the shorter operation range (about 1/3 of 802.11b) and higher density of *base stations* (also known as *access points*).

IEEE 802.11b

Probably the most widely implemented and used wireless LAN technology today, IEEE 802.11b specifies 5.5- and 11-Mbps data rates (in addition to the already specified 1 and 2 Mbps), but operates in the original 2.4-GHz band also using DSSS modulation. Most currently selling IEEE 802.11 products implement IEEE 802.1b. IEEE 802.11b-compliant devices can operate at 1, 2, 5.5, and 11 Mbps.

It is important to note that both incarnations of IEEE 802.11 use the same *Media Access Control* (MAC) protocol, *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA); therefore, these modifications affect only the physical layer (PHY layer in IEEE parlance) of the standard. The 1/2- and 5.5/11-Mbps DSSS (IEEE 802.11b) networks can coexist, enabling a painless transition to IEEE 802.11b (High Rate) at 11 Mbps. Eleven to fourteen radio channels are available for use with IEEE 802.11b in the 2.4-GHz band, depending on the local legal and administrative restrictions.

Distance, Power, and Speed Issues

It is obvious that all three of these parameters of wireless systems are interconnected. However, as with other radio-based technologies, the external conditions (such as the line of sight in case of outdoor use) greatly affect the operation of IEEE 802.11 devices.

Antennae

Antennae used with IEEE 802.11b devices may be grouped into two categories: *omnidirectional* and *point-to-point*. Obviously, omnidirectional antennae are the easiest to use, because they do not require positioning. Omnidirectional antennae are used in most base stations, as well as in most access cards. However, because of their nature, omnidirectional antennae do not work well over longer distances, unless used with external amplifiers; and these are not always legal or appropriate to use. Directional, or point-to-point antennae, on the other hand, require careful positioning and are used outdoors. Although the typical range for an omnidirectional antenna system is 150 ft (45m), configurations with high-gain directional antennae can work on distances up to 25 miles (about 40 km). In localities where amplifiers are allowed, the maximum distance may be considerably increased and is limited only by the line of sight.

Among other factors affecting the operational range of IEEE 802.11b devices are the base-station placement (when used in the infrastructure mode) and radio interference. As mentioned earlier, IEEE 802.11b devices will auto-configure for the highest possible speed and fall back to lower speeds when circumstances so require.

Performance Issues

Aside from obvious factors that affect performance (such as antennae, distance, radio interference) there are numerous other, more subtle issues. In the infrastructure mode, when all devices have to register with the base station(s), the load on the base station(s) increases with the number of clients and may reach a point when the performance reaches unacceptable lows. For example, Apple's AirPort Base Station (Version 2) can support up to 50 simultaneous clients. However, the actual performance of the whole system also depends on the kind of traffic. In particular, isochronous traffic (time-sensitive traffic, such as some types of video, audio, and telemetry), as well as multicast traffic, are particularly taxing for IEEE 802.11 networks and are better kept off the wireless LAN. However, several groups are currently working on extensions to 802.11 to provide for such kinds of traffic in a future version of the standard.

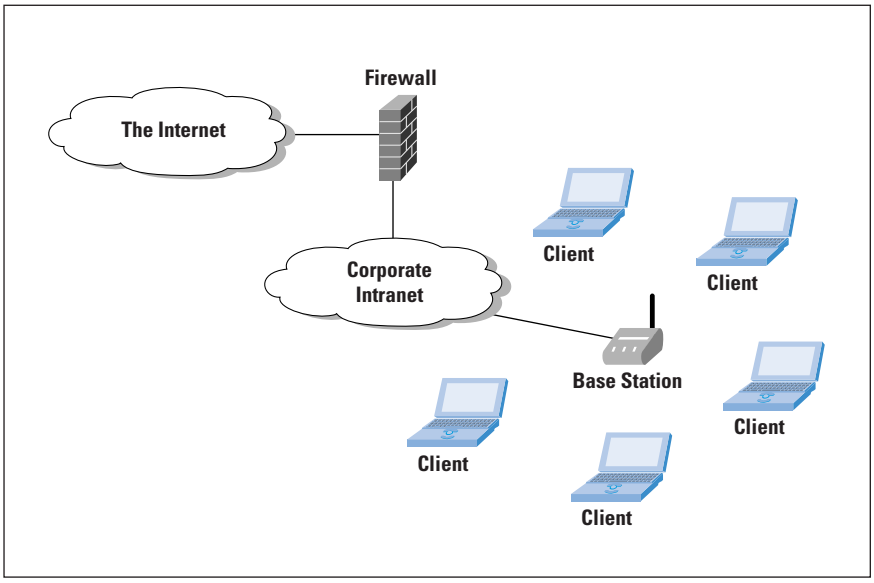


Figure 1: Typical IEEE 802.11 Configuration in Infrastructure Mode

IEEE 802.11 Base Stations and Clients

All IEEE 802.11 devices can be grouped into one of two groups: base stations or clients. Base stations can function as clients; however, not all clients can function as base stations. The reason for this is that base stations are required to provide certain network services to clients (association, distribution, integration, reassociation, and so on) that not all client hardware, firmware, or software can or intended to provide.

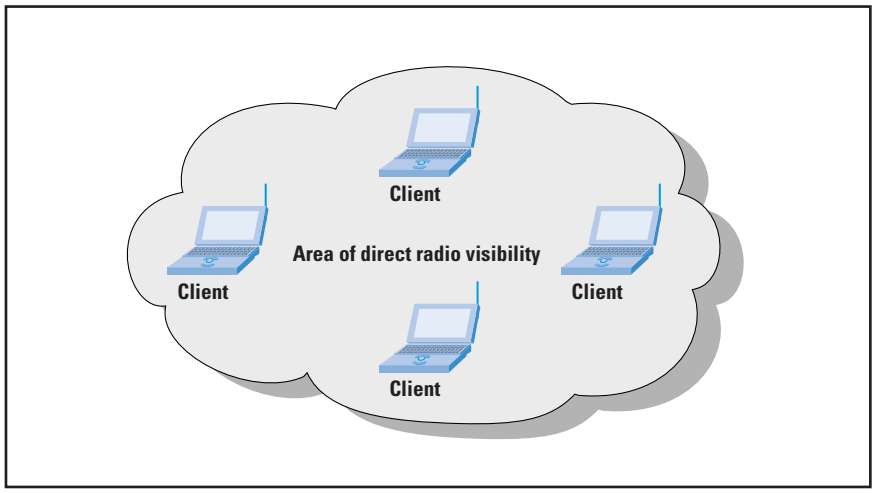
These considerations apply when the infrastructure mode of IEEE 802.11 is deployed. In *ad hoc* networks, where there are no base stations, all clients communicate directly with each other, reminiscent of a traditional shared Ethernet network, with all nodes sharing equal rights and responsibilities. As noted earlier, 11 to 14 radio channels are available, but separate networks may coexist on the same frequency (using different network IDs (*Service Set Identifiers* [SSIDs]), albeit with performance penalties.

The workings of 802.11 devices also differ in the infrastructure and ad hoc modes. In the infrastructure mode (Figure 1), clients associate (and optionally authenticate) themselves with a base station, and the presence of the base station is necessary for the operation of the network.

Complex 802.11 networks may be built using the infrastructure mode, with numerous base stations providing coverage over relatively large physical areas, and clients may roam within this roaming domain, which theoretically may extend from a single building to the entire campus or town. The *Spanning-Tree Protocol* (STP) is usually used in these cases to provide loop-free bridging in this wireless LAN.

In the ad hoc mode (Figure 2), base stations are not used and are not necessary, because all nodes of the wireless LAN have direct reachability (that is, they "see" each other). This mode is usually used in circumstances where all devices are in close proximity to each other (such as a floor or office) and when omnidirectional antennae are used.

Figure 2: IEEE 802.11 ad hoc Network



IEEE 802.11 Roaming and Mobility

IEEE 802.11 provides for roaming and mobility of 802.11 client devices and allows clients to roam among multiple 802.11 base stations that may be operating on the same or different frequencies (channels). This is achieved through the use of *beacon frames*, which are used to synchronize 802.11 devices and, in the infrastructure mode, to associate with a base station.

There are two ways to scan for existing 802.11 networks: active and passive scanning. In active scanning mode, the 802.11 device sends out "probe" frames, soliciting "I am here" responses from existing 802.11 devices. In the passive mode, the devices just listen for beacon frames, which are periodically transmitted by the active devices. In addition, the IEEE 802.11 Task Group F is working on the IAPP, which is to provide better and interoperable mobility and roaming mechanisms.

Security of IEEE 802.11

Up to this point IEEE 802.11 could be considered an absolute success; however, security of IEEE 802.11 is not quite on par with other aspects of the standard. Although an entire chapter (Chapter 8) of the standard is dedicated to authentication and privacy, it is now the common consensus that designers of IEEE 802.11 did not excel in this area. Two reports widely covered in the media, "Your 802.11 Wireless Network Has No Clothes"[7], and "Intercepting Mobile Communications: The Insecurity of 802.11"[6], shed light on the apparent shortcomings of the standard, or to be more exact, on its "vulnerability by design." They demonstrated that although the designers were well aware of the need to plan for authentication and privacy, the actual implementation was not an excellent one. The WEP algorithm, used to provide authentication and privacy in 802.11 wireless networks, is the problem.

WEP

Before discussing the security weaknesses discovered in IEEE 802.11, we quote the aim of the *Wired Equivalent Privacy* (WEP) algorithm as specified in the IEEE 802.11 standard document:

> "Eavesdropping is a familiar problem to users of other types of wireless technology. IEEE 802.11 specifies a wired LAN equivalent data confidentiality algorithm. Wired equivalent privacy is defined as protecting authorized users of a wireless LAN from casual eavesdropping. This service is intended to provide functionality for the wireless LAN equivalent to that provided by the physical security attributes inherent to a wired medium."

As you see, the aim of WEP is to provide a level of privacy equivalent to that of a wired LAN. The wording of standard is very important here: the developers of the standard did not intend to provide a level of security superior to or higher than that of a regular wired LAN, such as Ethernet. The very name of the algorithm, "Wireless Equivalent Privacy," signifies the actual intention of the developers. However, as the practice has shown, the level of security roughly equivalent to the level of security provided by wired LANs is not sufficient—and it is the assumption that "it is OK if wireless LANs are as secure as wired LANs" that is wrong. Other problems, such as the choice of *Cyclic Redundancy Check 32* (CRC-32) instead of *Message Digest Algorithm 5* (MD5) or some other secure hash algorithm, just worsen the problem.

How WEP Works

Let's now look at the workings of WEP. WEP uses a secret key shared between 802.11 nodes to encrypt 802.11 frames (Layer 2). It also uses a checksum (CRC-32) to provide data integrity. The checksum itself is also encrypted using the shared secret key. The decryption is the reverse of the encryption process: the frame is decrypted using the key and the CRC-32 checksum is computed and checked. The cipher used in WEP is RC4, a stream cipher designed by Ron Rivest, and believed to be cryptographically strong. The key is 40 or more bits long (up to 128 bits in some implementations). However, the *Initialization Vector* that is used during the encryption process is only 24 bits long. It is difficult to understand why the designers chose such a small number—more about this later. WEP does not provide any key management—the standard itself does not specify how the shared secret key should be managed and distributed. This leaves one of the most vulnerable parts of any cryptographic system—*key distribution*—open for misuse.

The Borisov Goldberg Wagner Attacks (February 2001)

In their paper entitled "Intercepting Mobile Communications: The Insecurity of 802.11," Nikita Borisov, Ian Goldberg, and David Wagner describe the vulnerabilities present in WEP and attacks against it. In the introduction to their paper, they state:

> "Unfortunately, WEP falls short of accomplishing its security goals. Despite employing the well-known and believed-secure RC4 cipher, WEP contains several major security flaws. The flaws give rise to a number of attacks, both passive and active, that allow eavesdropping on, and tampering with, wireless transmissions."

They go on to say that WEP fails to achieve all three of its security goals, namely confidentiality, access control, and data integrity.

As has been noted earlier, WEP uses the RC4 stream cipher with a 24-bit Initialization Vector for encryption. Borisov, Goldberg, and Wagner show that the poor design of WEP makes the system vulnerable in many areas, and one of the weakest parts of WEP is the 24-bit Initialization Vector, which may result in keystream reuse. Keystream reuse in turn permits successful cryptanalysis attacks against the ciphertext. However, what is surprising is that:

> "The WEP protocol contains vulnerabilities despite the designers' apparent knowledge of the dangers of keystream reuse attacks."

Another not less important but equally poorly designed aspect of WEP is the use of CRC-32. It is known that CRCs are not cryptographically strong and are not intended to be used in place of message digest or hash functions such as MD5 or the *Secure Hash Algorithm* (SHA). Because of the nature of CRC, it fails to provide the required integrity protection.

Some in the industry suggest that MD5 or SHA would introduce performance penalties if used—and indeed they would—one cannot disagree. But let's not forget that CRC-32 was intended as a security measure—which it isn't—yes, it is fast, but it is also insecure. Presumably, a slower but really secure solution is better than an inadequate though fast solution.

The Arbaugh Shankar Wau Attack (April 2001)

In the paper "Your 802.11 Wireless Network Has No Clothes,"[7] authors present their research of the authentication flaws in the IEEE 802.11 and demonstrate a simple eavesdropping attack against IEEE 802.11 authentication. This work is partially based on the knowledge obtained by Borisov, Goldberg, and Wagner in the paper described previously. The attack described in this work is possible even with WEP enabled; however, in that case it will also require application of attack(s) against WEP presented by Borisov et al. The authors also note that a good key management architecture would increase the security of the system; however, in their opinion only a comprehensive redesign of the standard would provide a good long-term solution to these issues.

The Fluhrer Mantin Shamir Attack (August 2001)

Scott Fluhrer, Itsik Mantin, and Adi Shamir describe a passive ciphertext-only attack against the key scheduling algorithm of RC4 as used in WEP[11]. They identify a large number of weak keys, in which knowledge of a small number of key bits suffices to determine many state and output bits with nonnegligible probability. They also show that the first byte generated by the RC4 leaks information about individual key bytes. This paper in particular shows how to reconstruct the secret key in WEP by analyzing enough WEP-encrypted packets. The authors have not tried to do this in practice—others did that.

The Stubblefield Ioannidis Rubin Implementation of Fluhrer Mantin Shamir Attack (August 2001)

In an AT&T Laboratories report published on August 21, 2001[14], Adam Stubblefield, John Ioannidis, and Aviel Rubin describe a real-world successful implementation of the Fluhrer Mantin Shamir attack using a $100 Linksys card on a Linux machine. They report that it took less than a week from ordering the card to recovering the WEP key on a production network. This practical work has shown that no expensive hardware or software is necessary in order to break WEP. They summarize that it is the poor implementation of reasonable secure technologies (such as RC4) that is responsible for WEP weaknesses.

WECA's Response

The *Wireless Ethernet Compatibility Alliance* (WECA) is the organization responsible for certifying compliance with the IEEE 802.11 standards. It also awards the WiFi (*Wireless Fidelity*) industry mark to the products that have passed IEEE 802.11 compliance testing.

In response to the Berkeley paper, WECA has published an official statement, clarifying its understanding of the situation. The main line of this statement is that poor security is better than no security, as well as that WEP was not intended to be a panacea for all security needs. The statement correctly notes that the biggest security threat is the failure to use available protection methods, including WEP.

IEEE 802.11 Chair's Response

In response to the research made at UC Berkeley and the University of Maryland, the Chair of the IEEE 802.11 Working Group, Stuart Kerry, has published a Chair's response intended to clarify some of the issues around the security of IEEE 802.11. He denied allegations made in the media that the security weaknesses of WEP are due to the closed standardization process. In fact, because WEP is a part of IEEE 802.11, it was developed through an open process, like other IEEE standards. The IEEE 802.11 Working Group itself is open to all interested parties to participate. He also rejects the viewpoint that frequency-hopping wireless networks would be less vulnerable to security attacks. It is evident that this is not true because both hopping codes and timing are unencrypted and are available to the attacker. Reminding us that the goal of WEP was to provide a level of security comparable to wired LANs, he states that the IEEE 802.11 Working Group is currently working on improvements to WEP to incorporate better security into the next version of the standard.

IEEE 802.1X

Security in 802.11 networks can be broken down into three components: authentication framework, authentication algorithm/protocol, and encryption. IEEE 802.1X is trying to address the authentication framework part of the puzzle. Although still in development, 802.1X provides a scalable, centralized framework for authentication. 802.1X may deploy a variety of authentication protocols (currently Cisco's *Lightweight Extensible Authentication Protocol* [LEAP] and Microsoft's *Extensible Authentication Protocol – Transport Layer Security* [EAP-TLS] are available), and it works with both wired and wireless LANs. The widely used *Remote Access Dial-In User Service* (RADIUS) protocol is also used in the 802.1X framework. 802.1X/LEAP is available with the Cisco Aironet 350 Series of wireless LAN devices; EAP-TLS is supported in Windows XP. Although it is still a draft, 802.1X may one day become the solution to the authentication issues of 802.11.

IEEE 802.11i

Task Group I of the IEEE Working Group 802.11 is currently defining MAC enhancements to provide enhanced security for 802.11. This is a work in progress, and no IEEE 802.11i draft exists at the time of writing.

Cisco's Solution

Cisco Systems has responded to both papers on the security of the WEP[10]. Cisco agrees that the WEP has serious shortcomings, and states that its Aironet series of wireless networking products offers many solutions to these problems: dynamic WEP keys, secure key derivation, and mutual authentication using LEAP[13]. However, Cisco agrees that improvements are needed in the standard itself.

RC4 Fast Packet Keying for WEP

In a Document Nr 550r2, "Temporal Key Hash," submitted by Russ Housley of RSA Security and Doug Whiting of Hifn to the IEEE 802.11 Working Group, they describe a solution to the WEP problem that uses a hashing technique that rapidly generates a unique RC4 key for each packet of data sent over the wireless network. This technique addresses the performance aspect of the security solution as well—the hash algorithm used in *Fast Packet Keying* (FPK) is much faster than traditional hash algorithms such as MD5 and SHA1 because of the special caching approach. The IEEE 802.11 Working Group has decided to include this technique in the IEEE 802.11i as an informative document. In most cases, FPK may be implemented as a firmware upgrade for the existing hardware. It is possible that when released, IEEE 802.11i may use FPK as the solution—but this decision is yet to be made. No definite plans are announced at the time of writing. For more information, see: **`http://www.rsasecurity.com/rsalabs/technotes/wep-fix.html`**.

Health and IEEE 802.11

Concerns about safety and health effects of various wireless solutions such as mobile phones and wireless network devices periodically surface in the media. In particular, the question of whether mobile phones are linked to brain cancer and other diseases is still open. However, in response to these concerns regarding wireless networking equipment health effects, Cisco Systems has published a white paper entitled "Cisco Systems Spread Spectrum Radios and RF Safety," which explains why these devices do not present a threat to human health when correctly used. The bottom line is that devices certified as compliant with U.S. Federal Communications Commission or Industry Canada's regulations are safe to use because of their low emitted power.

Practical Uses

Many companies, such as MobileStar, Wayport, Surf&Sip, and Airwave, have begun providing IEEE 802.11b Internet access at numerous locations throughout the United States. Several international airports also provide 802.11b service free of charge to travelers. No doubt more such services will continue to appear all over the world, maybe making a dream—Internet anywhere—a reality.

Summary

IEEE Standard 802.11 brought the long-awaited standardization to wireless LAN networking. Unfortunately, it also brought various security problems. Despite that, IEEE 802.11 is widely used, and with the coming of IEEE 802.11a, it can only gain in popularity. What now remains to be done is more effective and truly secure privacy and authentication for 802.11 wireless networks.

The IEEE 802.11 Working Group is actively working to improve what has been done to date. The most improvements are obviously needed in the area of security, where Working Groups 802.1X and 802.11i are working to define better security mechanisms. In particular, 802.11 WG is working on a new release of 802.11, which will include improvements over 802.11-1999. In the meantime, consider your wireless LAN as an external, insecure network—just like the Internet—and employ additional security measures, such as Virtual Private Networks, Transport Layer Security, SSH, and IP Security Architecture—in addition to WEP.

References

[1] IEEE Standard 802-1990: "IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture," ISBN 1-55937-052-1.

[2] IEEE Standard 802.11-1999: "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications."

[3] IEEE Standard 802.11a-1999: "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications (5 GHz)."

[4] IEEE Standard 802.11b-1999: "Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specificiations (2.4 GHz)."

[5] "IEEE 802.11b Wireless Equivalent Privacy (WEP) Security," February 19, 2001, Wireless Ethernet Compatibility Alliance (WECA).

[6] Nikita Borisov, Ian Goldberg, and David Wagner, "Intercepting Mobile Communications: The Insecurity of 802.11."
http://www.isaac.cs.berkeley.edu/isaac/wep-draft.pdf

[7] William A. Arbaugh, Narendar Shankar, and Y.C. Justin Wan, "Your 802.11 Wireless Network Has No Clothes,"
http://www.cs.umd.edu/~waa/wireless.pdf

[8] William A. Arbaugh, "An Inductive Chosen Plaintext Attack Against WEP/WEP2," IEEE Document 802.11-01/230.

[9] J. R. Walker, "Unsafe at Any Key Size; An Analysis of the WEP Encapsulation," IEEE Document 802.11-00/362.

[10] "Cisco Comments on Recent WLAN Security Paper from University of Maryland," Cisco Systems, Product Bulletin 1327.

[11] Fluhrer S., Mantin L., and Shamir A., "Weaknesses in the Key Scheduling Algorithm of RC4," Eighth Annual Workshop on Selected Areas in Cryptography, August 2001.

[12] Stuart J. Kerry et al, "Response from the IEEE 802.11 Chair on WEP Security," IEEE 802.11 Working Group.
http://www.ieee802.org/11/

[13] "Cisco Aironet Security Solution Provides Dynamic WEP to Address Researchers' Concerns," Cisco Systems, Product Bulletin 1281.

[14] Adam Stubblefield, John Ioannidis, and Aviel Rubin, "Using the Fluhrer, Mantin, and Shamir Attack to Break WEP, Revision 2," AT&T Laboratories Technical Report TD-4ZCPZZ, August 21, 2001.

EDGAR DANIELYAN is a Cisco Certified Network, Design, and Security Professional, as well as member of IEEE, ACM, USENIX, SAGE, and the IEEE Computer Society. Currently self-employed, he consults and writes on internetworking, UNIX, and security. His book, *Solaris 8 Security*, was published by New Riders Publishing in October 2001. The author is not affiliated with any of the organizations (except the IEEE) mentioned in this article. E-mail: **edd@danielyan.com**

Code Signing

by Eric Fleischman, The Boeing Company

Code signing is a common mechanism that authors of executable code use to assert their authorship of that code and to provide integrity assurance to the users of the code that an unauthorized third party has not subsequently modified the code in any way. Code signing is widely used to protect software that is distributed over the Internet. It is also widely used for mobile code security, being a core element of the mobile code security systems of both Microsoft's ActiveX and JavaSoft's Java applet systems. Despite this widespread use, common misunderstandings have arisen concerning the actual security benefits provided by code signing. This article addresses this issue. It explains how code signing works, including its dependence upon underlying *Public Key Infrastructure* (PKI) technologies.

Motivation for Code Signing

Code signing, which is also known as *object signing* in certain programming environments, is a subset of electronic document signing. In many ways code signing is a simplification of the more generic technology in that generally only a single signature is permitted and that signature pertains to the entire file. That is, code signing usually does not support multiple signatures, encryption of (data) content, dynamic data placement, or sectional signing, which are commonly available in many document-signing systems. As a result, code signing provides only authenticity and integrity for *electronic executable files*—it does not provide privacy, authentication, or authorization, which are supported by several electronic document-signing approaches.

A signature provides authenticity by assuring users as to where the code came from—who really signed it. If the certificate originated from a trusted third-party *Certificate Authority* (CA), then the certificate embedded in the digital signature as part of the code-signing process provides the assurance that the CA has certified that the code signer is who he or she claims to be. Integrity occurs by using a signed hash function as evidence that the resulting code has not been tampered with since it was signed.

In the pre-Internet era, software was distributed in a packaged manner via branding or trusted sales outlets. It frequently came in a shrink-wrapped form directly from the vendor or a trusted distributor. In the Internet era, software is often distributed via the Web, by e-mail, or by file transfer. Code signing provides users with a similar level of assurance as to software authenticity in this comparatively anonymous—and comparatively insecure—new distribution paradigm as was previously offered by packaged software in the pre-Internet era.

In all cases, what is assured is the authorship of the software, including the verification that third parties have not subsequently modified the code. In no case does the user receive any assurance that the code itself is safe to run or actually does what it claims. Thus, the actual value of code signing remains a function of the reliability and integrity of its author. Code signing, therefore, is solely a mechanism for software creators to assert their authorship of the product and validate that it has not been modified. In no case does it provide the end user with any claim as to the quality, intent, or safety of the code.

How Code Signing Works

Code signing appends a digital signature to the executable code itself. This digital signature provides enough information to authenticate the signer as well as to ensure that the code has not been subsequently modified.

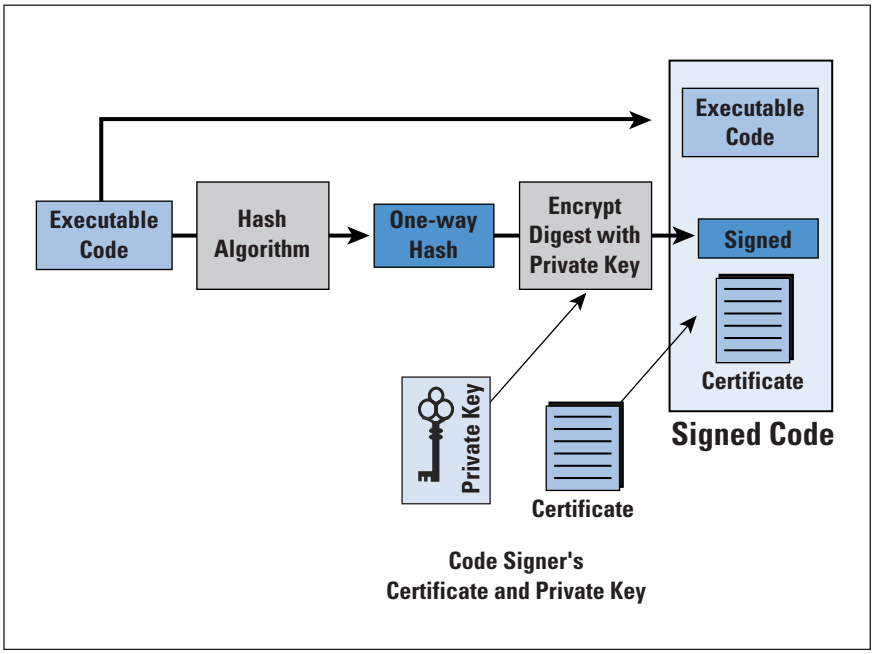
Code signing is an application within a PKI system. A PKI is a distributed infrastructure that supports the distribution and management of public keys and digital certificates. A digital certificate is a signed assertion (via a digital signature) by a trusted third party, known as the Certificate Authority (CA), which correlates a public key to some other piece of information, such as the name of the legitimate holder of the private key associated with that public key. The binding of this information then is used to establish the identity of that individual. All system participants can verify the name-key binding coupling of any presented certificate by merely applying the public key of the CA to verify the CA digital signature. This verification process occurs without involving the CA.

A *public key* refers to the fact that the cryptographic underpinnings of PKI systems rely upon asymmetric ciphers that use two related but different keys, a public key, which is generally known, and a *private key,* which should be known only by the legitimate holder of the public key. This approach is known as *public-key cryptography* and directly contrasts to symmetric ciphers, which contrastingly require the two entities to share an identical secret key in order to encrypt or decrypt information.

The certificates used to sign code can be obtained in two ways: They are either created by the code signers themselves by using one of the code-signing toolkits or obtained from a CA. The signed code itself reveals the certificate origin, clearly indicating which alternative was used. The preference of code-signing systems (and of the users of signed code) is that the certificates come from a CA, and CAs, to earn the fee they charge for issuing certificates, are expected to perform "due diligence" to establish and verify the identity of the individual or institution identified by the certificate. As such, the CA stands behind (validates) the digital certificate, certifying that it was indeed issued only to the individual (or group) identified by the certificate and that the identity of

that individual (or group) has been verified as stated. The CA then digitally signs the certificate in order to formally bind this verified identity with a given private and public key pair, which is logically contained within the certificate itself. This key pair will subsequently be used in the code-signing process. Self-created certificates, by contrast, are unconstrained as to the identities they may impersonate.

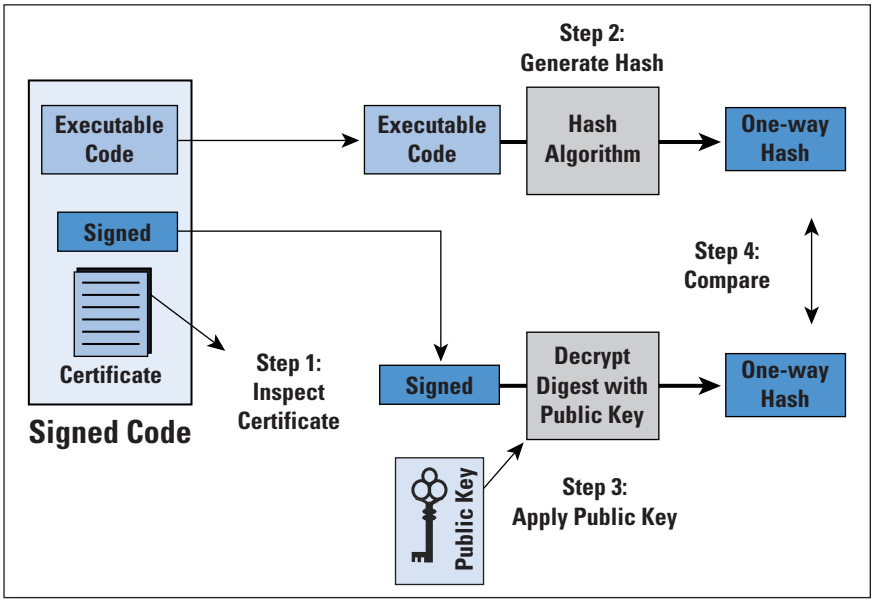
Figure 1: Code-Signing Process



Code signing itself is accomplished as follows: Developers use a *hash function* on their code to compute a *digest,* which is also known as a *one-way hash*. The hash function securely compresses code of arbitrary length into a fixed-length digest result. The most common hash function algorithms used in code signing are the *Secure Hash Algorithm* (SHA), *Message Digest Algorithm 4* (MD4), or MD5. The resulting length of the digest is a function of the hash function algorithm, but a common digest length is 128 bits. The digest is then encrypted using the developer's private key, which is part of the developer's certificate. A package containing the encrypted digest and the developer's Digital Certificate is encapsulated into a special structure called the *signature block*. The signature block is then appended to the executable code to form the signed code.

In a Java context, the signed Java byte code is called a JAR file. First introduced in the *Java Developer's Kit* (JDK) version 1.1, this capability was greatly expanded with Java 2.

Figure 2: Code Verification Process



At some subsequent time, this signed code will be presented to a recipient, usually through the agency of a code-signing verification tool on the recipient's computer. This tool will inspect the signature block to verify the authenticity and integrity of the received code. This inspection is done in the following manner, as shown in Figure 2:

1. The certificate is inspected from the signature block to verify that it is recognizable to the code-signing verification system as a correctly formatted certificate.
2. If it is, the certificate identifies the hash function algorithm that was used to create the signed digest within the received signature block. With this information, the same hash algorithm code that was used to create the original digest is then applied to the received executable code, creating a digest value, which then is temporarily stored. If it is not a correctly formatted certificate, then the code-signing verification process fails.
3. The signed digest value is then taken from the signature block and decrypted with the code signer's public key, revealing the digest value, which was originally computed by the code signer. Failure to successfully decrypt this signed digest value indicates that the code signer's private key was not used to create the received signature. If this is the case, then that signature is a fraud and the code-signing verification process fails.
4. The recomputed digest of Step 2 is then compared to the received digest that was decrypted in Step 3. If these two values are not identical, then the code has subsequently been modified in some way and the code-signing verification process fails. If any such anomaly occurs, then the verification system alerts the recipient concerning the nature of the failure, indicating that the resulting code is suspect and should not be trusted. However, if the digests are identical, then the identity of the code signer is established.

5. If establishment occurs, then the code signer's certificate is copied from the signature block and presented to the recipient. The recipient then has the option to indicate whether or not he or she trusts the code signer. If so, then the code is executed. If not, then it is not executed.

Types of Code Signing

Code signing is a mechanism to sign executable content. The term executable content refers to presenting executable programs in a manner so that they could be run locally—regardless of whether the executable file originated locally or remotely. Code signing is commonly used to identify authorship within several distinct usage scenarios:

- Applications can be code signed to identify their ownership within comparatively anonymous software distribution mechanisms using the Web, the *File Transfer Protocol* (FTP), or e-mail. This type of code signing establishes the origin for downloadable JAR, tar, zip, or CAB file software distributions, for example.
- Code signing can provide Web users more control over mobile code that is available to their Web browsers. Mobile code is code that travels a network in its lifetime in order to execute on a destination machine. The term is usually associated today with active Web content that executes on the client's machine via technologies such as Java, JavaScript, VBScript, ActiveX, and MS Word macros.
- Device drivers can be code signed to inform an operating system of the authorship of that driver. For example, the device drivers for Windows 98, Windows ME, and Windows 2000 operating systems should preferentially be certified by Microsoft's device driver certification laboratory[25]. The entity signs the device driver executable in order to certify that the device driver in question has indeed been successfully demonstrated by a Microsoft certification laboratory to correctly run on that operating system.
- A recent news report[20] has stated that Microsoft will be using code signing as a security mechanism within its forthcoming Windows XP operating system. The article stated: "Microsoft is to incorporate a 'signed application' system in Whistler [that is, Windows XP], the intention being to furnish users with a super-secure mode of operation that just plain stops [unsigned] code executing on the machine."

Code Signing Does Not Provide Total Security

A fundamental problem with code signing is that it cannot provide any guarantee about the good intentions of the signer or the quality, intent, operations, or safety of the code. The VeriSign and Thawte CAs, for example, combat this limitation somewhat for executables signed by certificates they issue by requiring the entities receiving their certificates to sign a "software publisher's pledge" not to sign a piece of malicious software. If they subsequently learn of violations of this agreement, they ask the owner to correct the problem.

If the owner refuses, then they cancel the owner's digital certificate and potentially bring a lawsuit against the offender. The code-signing literature has documented that the latter has occurred at least once[21].

Another problem is that the digital signing by even a reputable entity can be forged if the private key of the signer becomes known. This forging can occur when the criminally minded exploit any of numerous potential vulnerabilities, including hacking into the key store on the signer's machine, carelessness on the part of the signer exposing this information, or an error in a CA PKI key distribution system.

Perhaps the best summary of these issues is provided by Schneier, who wrote:

> "Code signing, as it is currently done, sucks. There are all sorts of problems. First, users have no idea how to decide if a particular signer is trusted or not. Second, just because a component is signed doesn't mean that it is safe. Third, just because two components are individually signed does not mean that using them together is safe; lots of accidental harmful interactions can be exploited. Fourth, "safe" is not an all-or-nothing thing; there are degrees of safety. And fifth, the fact that the evidence of attack (the signature on the code) is stored on the computer under attack is mostly useless: The attacker could delete or modify the signature during the attack, or simply reformat the drive where the signature is stored." (Quoted from page 163 of [17]).

Mobile Code Security

Mobile code security is a two-edged sword: it seeks to protect computer systems receiving potentially hostile mobile code and it also seeks to protect mobile code from potentially hostile users of those computer systems.

Code signing has emerged as a major adjunct to mobile code security. Because mobile code probably represents the dominant use of code signing that occurs today, this section examines how code signing assists mobile code security.

There is substantial and growing literature on mobile code security (for example, see [3] through [16]). The literature identifies four distinct approaches to mobile code security, together with a few hybrids that merge two or more methods. Each of the four approaches has an inherent trust model that identifies the assumptions upon which the approach is based. Rubin and Geer[4] list these four approaches as being:

- The *sandbox approach,* which restricts mobile code to a small set of safe operations. This is the historic approach used by Java applets. In the approach, each Java interpreter implementation attempts to adhere to a security policy, which explicitly describes the restrictions that should be placed on remote applets. "Assuming that the policy

itself is not flawed or inconsistent, then any application that truly implements the policy is said to be secure. ... The biggest problem with the Java sandbox is that any error in any security component can lead to a violation of the security policy. ... Two types of applets cause most of the problems. Attack applets try to exploit software bugs in the client's virtual machine; they have been shown to successfully break the type safety of JDK 1.0 and to cause buffer overflows in HotJava. These are the most dangerous. Malicious applets are designed to monopolize resources, and cause inconvenience rather than actual loss."[4] The trust model assumed by the sandbox approach is that the sandbox is trustworthy in its design and implementation but that mobile code is universally untrustworthy.

- In code signing, the client manages a list of entities that it trusts. When a mobile code executable is received, the client verifies that it was signed by an entity on this list. If so, then it is run; otherwise it does not run. This approach is most commonly associated with Microsoft's ActiveX technology. "Unfortunately, there is a class of attacks that render ActiveX useless. If an intruder can change the policy on a user's machine, usually stored in a user file, the intruder can then enable the acceptance of all ActiveX content. In fact, a legitimate ActiveX program can easily open the door for future illegitimate traffic, because once such a program is run, it has complete access to all of the user's files. Such attacks have been demonstrated in practice."[4] The trust model for this approach assumes that it is possible to distinguish untrustworthy authors from trustworthy ones and that the code from trustworthy authors is dependable.
- The *firewalling approach* involves selectively choosing whether or not to run a program at the very point where it enters the client domain. "Research shows that it may not always be easy to block unwanted applets while allowing other applets ... to run. The firewalling approach assumes that applets can somehow be identified. ... This approach is fundamentally limited, however, by the halting problem, which states that there is no general-purpose algorithm that can determine the behavior of an arbitrary program."[4]

  A related and more viable alternative is the playground architecture that has been used to separate Java classes that prescribe graphics actions from all other actions. The former are loaded on the client, whereas the latter are loaded on a "sacrificial" playground machine for execution and then reporting of the results to the browser. Because this approach requires byte-code modification, it cannot be used in conjunction with the usual approach to code signing.
- The *Proof-Carrying Code* (PCC) technique is a theoretical approach that statistically checks code to ensure that it does not violate safety policies. "PCC is an active area of research so its trust model may change. At present, the design and implementation of the verifier are considered trustworthy but mobile code is universally untrustworthy."[4]

The most common hybrid approach occurs for Java's JDK 1.1 and Java 2. Each combines the sandbox approach, which was the security mechanism for JDK 1.0, with code signing. This hybrid originated from the realization that the inherent restrictions of the sandbox model kept applications from doing "interesting and useful things." Therefore, a mechanism for running applications outside of the sandbox, code sharing, was devised to supplement the sandbox-based original. Specifically, in JDK 1.1 a signed applet enjoys unlimited access to system resources, just like local applications do, provided that the corresponding public key is trusted in the executing environment. This system evolved within Java 2 to optionally provide a consistent and flexible policy for applets and applications, determined by the policies established within a protection domain.

The literature is unanimous that the net result of this hybrid version "introduces the same security problems [as those] inherent in the ActiveX code-signing approach."[4] For this reason, Bernard Cole[11] has stated "neither [the sandbox nor the code signing] model is appropriate to the new environment of small information appliances, connected embedded devices, numerous web-enabled wireless phones and set-top boxes."[11] Indeed, several articles (for example, perhaps the best collection is contained in[13]) contained worrying descriptions of how to compromise specific sandbox and code-signing products.

The literature (see [3] through [16]) is also clear that despite the demonstrable weaknesses of both the sandbox and code-signing approaches as mechanisms for securing mobile code, they are the best practical alternatives available today. In the meantime, researchers are currently exploring enhanced mobile code security by making hybrids containing three—or all four—of the above mechanisms.

Researchers have also begun to investigate alternative techniques. For example, Zhao[16] reports that "Additional innovative authentication functions are needed for mobile code. One approach is to apply digital fingerprinting to authenticate mobile code. Analogous to 'biometric authentication' for access control, a digital fingerprint of mobile code is a unique authentication code that is an integral and intrinsic part of the thing being authenticated. It is placed into the mobile code during its development by using digital watermarking techniques."

Major Code-Signing Systems

Code-signing systems are often functions of specific applications. For example, Thawte[22] is a CA that provides the following certificate types:

- The *Apple Developer Certificate* is used by Apple MacOS-based application developers to sign software for electronic distribution.
- The *JavaSoft Developer Certificate* can be used with JavaSoft's JDK 1.3 and later to sign Web applets.
- A *Marimba Channel Signing Certificate* is used to sign Castanet channels on the Marimba platform.

- A *Microsoft Authenticode Certificate* is used with the Microsoft InetSDK developer tools to sign Web applets (for instance, ActiveX controls) as well as `.CAB`, `.OCX`, `.CLASS`, `.EXE`, `.STL`, and `.DLL` files, and other potentially harmful active content on Microsoft OS platforms. These Authenticode certificates work only with Microsoft IE 4.0 and later browsers.
- *VBA Developer Certificates* are identical to the Microsoft Authenticode certificates. They are used by developers to sign macros in Office 2000 and other VBA 6.0 environments.
- *Netscape Code-Signing Certificate*s are used to sign Java applets, browser plug-ins, and other active content on the Netscape Communicator platform.

Despite this diversity, the clearly dominant code-signing systems today come from Microsoft, Netscape, and JavaSoft. Although these three systems generally adhere to the same set of standards, their approaches are highly diverse from each other. Each has its own certificate type. Each system approaches code signing with different orientations, goals, and expectations.

Interoperability Problems

Although all code signing uses similar technology, interoperability problems currently impact code signing. These problems may originate from interoperability problems within the underlying PKI infrastructure, from certificate differences, or from different (vendor) approaches to code signing itself.

PKI Infrastructure Interoperability

The PKI Forum has identified ten impediments to the widespread adoption of PKI[23], the most significant being the "lack of interoperability" between PKI products. Because of this, the technical working group of the PKI Forum is currently concentrating on addressing PKI interoperability problems: "The Technical Working Group continues its focus on multi-vendor interoperability projects. Over the last six months, it has sponsored monthly interoperability "bake-offs" based on the *Certificate Management Protocol* (CMP) standard, with participation from a growing number of vendors. In addition, two workshops have been held to date on application-level interoperability through the use of digital certificates, with remote testing ongoing. Looking forward, the Technical Working group plans to initiate two new interoperability projects in the areas of Smart Card/Token Portability and CA interoperability, and it will be defining a large-scale, multi-vendor interoperability project for public demonstration in the first quarter of 2001."[24]

Certificate Interoperability

Numerous potential interoperability issues stem from the certificates themselves because certain certificates are themselves tied to specific types of applications.

However, not every certificate is a code-signing certificate. Rather, code-signing certificates are special certificates whose associated private keys are used to create digital signatures. In addition, the id-kp-codesigning value within the extended key usage field of the certificate itself (see Section 4.2.1.13 of RFC 2459) needs to be set to indicate that the certificate can be used for code signing.

In any case, code-signing certificates must be packaged in the appropriate format [*Public Key Cryptographic Standards* (PKCS)]), and the various code-signing approaches (for example, Microsoft, Netscape, JavaSoft) expect both the signing certificates and the code that is to be signed to conform to different file format requirements.

These differences between code-signing systems introduce opportunities for incompatibility, even if each approach otherwise rigorously adheres to the same basic certificate standards.

Not all certificates can be used to support all potential certificate uses, even if they originate from the same CA. For example, the Java Developer Certificates are not interoperable (exchangeable) with any other certificates at this time. Fortunately, it is possible to buy certificates that can be used for many (but not all) potential uses. For example, a single certificate can support Microsoft Authenticode, Microsoft Office 2000/VBA Macro Signing, Netscape Object Signing, Apple Code Signing, and Marimba Channel Signing.

Code Signing System Interoperability

Probably the least understood of the potential interoperability problems are due to different vendor approaches to code signing itself. Perhaps McGraw and Felten have provided the best insight to code-signing system interoperability within Appendix A of their book *Securing Java*[15]. Unfortunately, those insights were in regard to an earlier version of Java, which has evolved considerably since then.

Certificate Issues

Each of the three major code-signing systems (Microsoft, Netscape, JavaSoft) has its own certificates. Each provides its own certificate stores to house certificates within its system.

Each of the three systems supports mechanisms by which certificates may be exported from a given user's certificate store and imported into a different user's certificate store on the same or on a different machine. The Microsoft and Netscape systems also have provisions for importing certificates between code-signing systems.

Certificates are usually exported between PKI systems or certificate stores in the PKCS-12 format (`.p12` files if Netscape or `.pfx` files if Microsoft Authenticode), which contains both certificate and key pair information within the same file. Certificates can also be exported in the PKCS-7 format (for example, `.cer` or `.spc` files).

The latter approach lacks information to permit the certificate to be used for code signing by the importing system unless the missing elements can be retrieved via other mechanisms.

Code-Signing Certificates

The Netscape certificate utility (that is, *signtool –L*) indicates which of the certificates located within a certificate store can be used for code signing. By contrast, all certificates (except for those explicitly prohibited from doing code signing according to the provisions of RFC 2459 Section 4.2.1.13) within a Microsoft certificate store can be used for code signing within the Microsoft system. This means that a certificate that is unable to be used for code signing in a Netscape system can be imported into the Microsoft system and be successfully used for code signing there.

This difference stems from RFC 2459 Section 4.2.1.13, which deals with the extended key usage field. The relevant text of the standard is as follows:

> "If the extension is flagged critical, then the certificate MUST be used only for one of the purposes indicated. If the extension is flagged noncritical, then it indicates the intended purpose or purposes of the key, and may be used in the correct key/certificate of an entity that has multiple keys/certificates. It is an advisory field and does not imply that usage of the key is restricted by the certification authority to the purpose indicated. Certificate using applications may nevertheless require that a particular purpose be indicated in order for the certificate to be acceptable to that application."

What has occurred is that Netscape has implemented its system such that certificates can be used only for the purposes specified in the extended usage field. Netscape does this for both critical and noncritical markings. Microsoft, by contrast, provides that restriction solely to certificates that have been marked "critical," permitting certificates without a critical marking to be used for any activity possible. Both approaches are legal, and both fully conform to the standard.

Code Signing from an End User's Perspective

The results obtained when you try to execute signed code is a function of your underlying operating system, the browser you are using, and whether or not the executable is a Java applet. This should not be surprising, because similar differences also occur with unsigned code. For example, a Microsoft executable file will execute on a Microsoft Windows operating system but is unlikely to execute on operating systems that do not recognize that format. Similarly, a Java applet cannot be directly invoked on a Windows operating system, because that operating system does not recognize the `.jar` file extension. However, it will cleanly execute when accessed off of a Web page, regardless of the underlying operating system.

References

[1] "A Closer Look at the E-signatures Law," by Linda Rosencrance, *Computer World,* October 5, 2000.

[2] "Standards Issue Mars E-signature," by Jaikumar Vijayan and Kathleen Ohlson, *Computer World,* July 10, 2000.

[3] "Mobile Code and Security," by Gary McGraw and Edward Felten, *IEEE Internet Computing,* Volume 2, Number 6, November/December 1998.

[4] "Mobile Code Security," by Aviel Rubin and Daniel Geer, *IEEE Internet Computing,* Volume 2, Number 6, November/December 1998.

[5] "Securing Systems Against External Programs," by Brant Hashii, Manoj Lal, Raju Pandey, and Steven Samorodin, *IEEE Internet Computing,* Volume 2, Number 6, November/December 1998.

[6] "Secure Web Scripting," by Vinod Anupam and Alain Mayer, *IEEE Internet Computing,* Volume 2, Number 6. November/December 1998.

[7] "Secure Java Class Loading" by Li Gong, *IEEE Internet Computing,* Volume 2, Number 6, November/December 1998.

[8] "Mobile Code Security: Taking the Trojans out of the Trojan Horse," by Alan Muller, University of Cape Town. April 5, 2000.
http://www.cs.uct.ac.za/courses/CS400W/NIS/papers00/amuller/essay1.htm

[9] "Understanding the keys to Java Security—The Sandbox and Authentication" by Gary McGraw and Edward Felten, *JavaWorld Magazine,* May 1997.

[10] "Repair Program or Trojan Construction Kit?" by Greg Guerin, September 7, 1999.
http://www.amug.org/~glguerin/opinion/crypto-repair-kit.html

[11] "Security, Reliability Twin Concerns in Net Era," by Bernard Cole, *Electrical Engineering Times,* July 24, 2000.

[12] "Java Security: From HotJava to Netscape and Beyond," by Drew Dean, Edward Felten, and Dan Wallach, Proceedings of 1996 IEEE Symposium on Security and Privacy, May 1996.

[13] "Formal Aspects of Mobile Code Security," by Richard Drews Dean, PhD thesis, Princeton University, January 1999.
http://www.cs.princeton.edu/sip/pub/ddean-dissertation.php3

[14] "A Flexible Security Model for Using Internet Content," by Nayeem Islam, Rangachari Anad, Trent Jaeger, and Josyula Rao, IBM Thomas J Watson Research Center, June 28, 1997.
http://www.ibm.com/java/education/flexsecurity/

[15] *Securing Java—Getting Down to Business with Mobile Code,* by Gary McGraw and Edward Felten, ISBN 0-471-31952-X, John Wiley & Sons, 1999.

[16] "Mobile Code: Emerging Cyberthreats and Protection Techniques," by Dr. Jian Zhao, Proceedings of the Workshop on Emerging Threats Assessment—Biological Terrorism, July 7–9, 2000, Dartmouth College, Hanover, NH.

[17] *Secrets and Lies—Digital Security in a Networked World,* by Bruce Schneier, ISBN 0-471-25311-1, John Wiley and Sons, 2000.

[18] Telephone conversation between Bob Moskowitz and Eric Fleischman on September 26, 2000.

[19] E-mail correspondence between Joseph M. Reagle, Jr., of the W3C and Eric Fleischman on December 6, 2000.

[20] **http://www.theregister.co.uk/content/4/14592.html**

[21] **http://www.halcyon.com/mclain/ActiveX/Exploder/FAQ.htm**

[22] **https://www.thawte.com/cgi/server/step1.exe?zone=devel**

[23] **http://pkiforum.org/About/Overview/sld037.htm**

[24] **http://www.pkiforum.org/News/2000/PKI-Forum-third-meeting-20000919.htm**

[25] **http://www.microsoft.com/hwtest/Signatures/**

[A longer version of this article can be obtained from the author.]

ERIC FLEISCHMAN has university degrees from Wheaton College (Illinois), the University of Texas at Arlington, and the University of California at Santa Cruz. He currently works in data communications security. He is employed as an Associate Technical Fellow by The Boeing Company. Eric was formerly employed by the Microsoft Corporation, AT&T Bell Laboratories, Digital Research, and Victor Technologies. He can be contacted at **Eric.Fleischman@boeing.com**

Book Review

Internet Performance Survival Guide

Internet Performance Survival Guide: QoS Strategies for Multiservice Networks, by Geoff Huston, ISBN 0-471-37808-9, John Wiley & Sons, 2000.

Many readers of IPJ are familiar with the name Geoff Huston. He contributes articles frequently. I find his style to be very lucid and his writings to be very well structured and organized.

I have need at my job to begin implementation of *Quality of Service* (QoS) strategies to deal with an ever-increasing demand for *Virtual Private Network* (VPN) tunnels over shared media. So, when I came across the title of this book and saw who wrote it, I jumped at the opportunity to review it for IPJ.

Organization

This book is organized more like a textbook than a reference manual. If you are looking for a quick and dirty guide that simply lists all the tricks of the trade and gives examples of how to implement them on specific equipment, then this book is not for you. If, however, you are looking for a well-written text that will help you to understand the issues, the practices that address them, and the theory that underlies these practices, then this is an excellent book.

The book begins with a chapter that explains in detail the problems that administrators and engineers on heterogeneous, multiprotocol networks face today. There is a quick historical survey of the evolution of networking and how that has shaped the nature of the problem. In a very topical fashion, this introduction covers the basic techniques that can be used to implement QoS, but also explains the complexity involved with these techniques, their limitations, and why they are not widely deployed yet. The book continues from there, starting with a low-level view of the building blocks of the network and gradually building to higher- and higher-level topics.

The second chapter begins with some details about the performance features built into the Internet Protocol, and in particular IPv6. This chapter continues into TCP and covers all the well-known performance features that are built into it, and then moves on to routing, switching, and *Multiprotocol Label Switching,* or MPLS. MPLS is a unified approach to switching across large networks, and it has particular applications to QoS. This topic is one of the main reasons I sought for this book, and I am glad it was covered in such detail. The second chapter ends with a survey of the various transmission systems that are available today, and discusses in detail the performance characteristics and problems that are peculiar to each.

The third chapter is a well-organized exposition of the various types of performance-tuning techniques that are available. The author keeps the discussion at a reasonably abstract level, yet is not afraid to discuss the details of the application of these techniques to the specifics of the network when such details are important. In particular, the use of QoS techniques in conjunction with the *Open Shortest Path First* (OSPF) routing protocol is discussed.

The fourth chapter combines the building blocks of Chapter 2 and the techniques of Chapter 3 into an architectural view that spans the network. The author discusses the metrics that can be used to analyze network performance, the protocols that can be used to implement service strategies, the tradeoffs that are inherent in the problem, and the policy choices that need to be made in order to come up with a clear design. In particular, the Integrated Service and Differentiated Service models are discussed separately, and then the author shows how these can be combined into an end-to-end network design. As with Chapter 3, the author explains important specific cases such as the use of the *Resource Reservation Protocol* (RSVP) with ATM.

The fifth chapter moves on to explain how the architectures that have been described can be used to attack the various kinds of problems that exist on real networks. The emphasis is clearly on the end user of the system and how to measure the levels of service being provided and to bring into play the techniques already discussed to assure a consistent level of service. The organization of this chapter seemed less clear than that of the previous chapters, but that is perhaps due more to the nature of the complexity of the problems being discussed than to the author's limitations or inattention.

The sixth chapter provides little new material, per se, and is more of a perspective on the material already provided. However, it contributes highly to the content of the book in two important ways. First, it provides more of a top-down view of QoS to complement the material in the preceding four chapters, which present a mostly bottom-up view. Secondly, it acts as a natural bookend for the first chapter. The first chapter raises the issues and poses the questions. The middle of the book examines the protocols, techniques, and architectures in detail. The last chapter then attempts to answer the questions that were initially raised.

The author does an excellent job of presenting material that is complex, vast, and is still in the process of evolving in the field. He is very diligent about managing the level of detail, and is careful to first cover the material topically before diving into the details. The examples are appropriate and have been carefully chosen.

One of the features of the material that is most appreciated is the practical perspective that the author brings to his work. The theory never gets out of hand, and is always balanced by a real-life approach to problems that, unfortunately, can never be completely solved. And, the author's observations always seem in tune with the experiences of the reader.

The material is well organized, and readers will appreciate the effort expended on the textual conventions that help to organize and structure the material. The diagrams that accompany the text are clear and well-placed, and they contribute to the reader's comprehension.

A glossary in the back helps a reader who has not thoroughly read the preceding sections of the book. The index is also well done, and the reference material is copious and pertinent.

Recommended

Overall, I would recommend this book to any professional who manages large, integrated networks, particularly those professionals who work for Internet Service Providers in an engineering capacity. I think this reflects the particular interests of the author, but that is as it should be.

—David P. Feldman, Tudor Investment Corporation
David.Feldman@Tudor.com

Would You Like to Review a Book for IPJ?

We receive numerous books on computer networking from all the major publishers. If you've got a specific book you are interested in reviewing, please contact us and we will make sure a copy is mailed to you. The book is yours to keep if you send us a review. We accept reviews of new titles, as well as some of the "networking classics." Contact us at **ipj@cisco.com** for more information.

Call for Papers

The Internet Protocol Journal (IPJ) is published quarterly by Cisco Systems. The journal is not intended to promote any specific products or services, but rather is intended to serve as an informational and educational resource for engineering professionals involved in the design, development, and operation of public and private internets and intranets. The journal carries tutorial articles ("What is…?"), as well as implementation/operation articles ("How to…"). It provides readers with technology and standardization updates for all levels of the protocol stack and serves as a forum for discussion of all aspects of internetworking.

Topics include, but are not limited to:

- Access and infrastructure technologies such as: ISDN, Gigabit Ethernet, SONET, ATM, xDSL, cable, fiber optics, satellite, wireless, and dial systems
- Transport and interconnection functions such as: switching, routing, tunneling, protocol transition, multicast, and performance
- Network management, administration, and security issues, including: authentication, privacy, encryption, monitoring, firewalls, trouble-shooting, and mapping
- Value-added systems and services such as: Virtual Private Networks, resource location, caching, client/server systems, distributed systems, network computing, and Quality of Service
- Application and end-user issues such as: e-mail, Web authoring, server technologies and systems, electronic commerce, and application management
- Legal, policy, and regulatory topics such as: copyright, content control, content liability, settlement charges, "modem tax," and trademark disputes in the context of internetworking

In addition to feature-length articles, IPJ will contain standardization updates, overviews of leading and bleeding-edge technologies, book reviews, announcements, opinion columns, and letters to the Editor.

Cisco will pay a stipend of US$1000 for published, feature-length articles. Author guidelines are available from Ole Jacobsen, the Editor and Publisher of IPJ, reachable via e-mail at `ole@cisco.com`

ICANN Considers Structural Reform

Stuart Lynn, President and CEO of *The Internet Corporation for Assigned Names and Numbers* (ICANN) recently proposed a sweeping series of structural reforms designed to lead ICANN towards attainment of its core mission. "The current structure of ICANN was widely recognized as an experiment when created three years ago," noted Board Chairman Vint Cerf. "The rapid expansion of and increasing global dependence on the Internet have made it clear that a new structure is essential if ICANN is to fulfill its mission."

ICANN was formed three years ago as an entirely private global organization designed to assume responsibility for the DNS root from the U.S. government and to coordinate technical policy for the Internet's naming and address allocation systems. In the new proposals, the basic mission remains intact, but the means of achieving that mission changes. "What has become clear to me and others is that a purely private organization will not work," said Lynn. "The Internet has become too important to national economic and social progress. Governments, as the representatives of their populations, must participate more directly in ICANN's debates and policymaking functions. We must find the right form of global public-private partnership—one that combines the agility and strength of a private organization with the authority of governments to represent the public interest."

Noting that current organizational inertia and obsession with process over substance has impeded agility, Lynn laid out a roadmap designed to instill confidence in key stakeholders and to ensure that ICANN can be more effective. This roadmap entails restructuring the Board of Directors into a Board of Trustees composed in part of trustees nominated by those governments who participate in the ICANN process; in part by the chairs of proposed new "policy councils" that would replace the existing supporting organizations and that would provide expert advice; and in part by trustees proposed by a broadly-based nominating committee and appointed by the Board itself. The roadmap is designed to bring all critical stakeholders to the table, something that has been difficult to achieve with the present structure and has slowed ICANN's progress and its ability to fulfill its responsibilities. It is also designed to establish a broad-based funding mechanism sufficient to support the critical mission of ICANN.

"We need to build a stronger organization, supported by our key stakeholders, led by the best team that can be assembled, and properly funded," Lynn said. "We must be structured to function effectively in this fast-paced global Internet environment." "A key requirement is to keep the best of the present ICANN," added Cerf, "in ensuring transparency, openness, and participation, while creating an ICANN that can act responsibly and quickly. That will mean rejecting practices that have emphasized process over achievement. Above all, ICANN must be—and be seen to be—effective and supportive of technical innovation and of a reliable Internet."

A paper written by Lynn that explains the reasons for change and the roadmap for reform is posted on the ICANN web site:
http://www.icann.org/general/lynn-reform-proposal-24feb02.htm

The Internet Protocol Journal

Ole J. Jacobsen, Editor and Publisher

The Internet Protocol Journal is published quarterly by the Chief Technology Office, Cisco Systems, Inc.
www.cisco.com
Tel: +1 408 526-4000
E-mail: ipj@cisco.com

The Internet Protocol Journal, Cisco Systems
170 West Tasman Drive, M/S SJ-10/5
San Jose, CA 95134-1706
USA

ADDRESS SERVICE REQUESTED

PRSRT STD
U.S. Postage
PAID
Cisco Systems, Inc.