A Trusted Open Platform

Microsoft's next-generation secure computing base extends personal computers to offer mechanisms that let high-assurance software protect itself from the operating systems, device drivers, BIOS, and other software running on the same machine.

Paul England
Butler Lampson
John Manferdelli
Marcus Peinado
Bryan Willman
Microsoft Corporation

Computers are entrusted with more personal and valuable data every day, and local and remote users need mechanisms to safeguard this data against misuse. A variety of access-control mechanisms address this problem.[1] For example, most commercial systems require users to provide a password to log on. Users and administrators can configure the system to restrict access to resources, such as files containing sensitive data.

However, such mechanisms have limited effectiveness in a mass-market setting because the kernel's integrity cannot be ensured. One of the main reasons is that the commercial need for an open software and hardware architecture leads to operating systems that contain a large collection of peripheral devices and device drivers containing millions of lines of code. A single programming error or intentional back door in this large and diverse code base can give rise to an attack that renders the access-control system ineffective. Viruses and Trojan horses exploit such errors on large numbers of machines on the Internet.

Furthermore, most home and corporate desktop computers today are rather loosely administered. Even a functioning access-control system will be ineffective if it is not correctly configured.

These problems expose open-system users to concrete vulnerabilities:

- A corporate document prepared with a trustworthy program is also accessible to a virus or a game with a vulnerability or back door.
- A user's home finance transactions and data are vulnerable to Trojan horses that "snoop" actions and passwords.
- A bank cannot distinguish a legitimate transaction initiated by a person from an illegitimate or sabotaged transaction instigated by a subverted application.

One solution to these problems is to provide stricter control over platform hardware and software by using a closed system. Set-top boxes, game machines, and smart cards take this approach. If it is difficult or impossible to make a change to the operating system or run an unknown or unauthorized application, it is easier to ensure data and transaction integrity. However, closed systems are far less flexible than open systems and are unlikely to replace the personal computer.

NEXT-GENERATION SECURE COMPUTING BASE

Microsoft's next-generation secure computing base aims to provide robust access control while retaining the openness of personal computers. Unlike closed systems, an NGSCB platform can run any software, but it provides mechanisms that allow operating systems and applications to protect themselves against other software running on the same machine. For example, it can make home finance data inaccessible to programs that the user has not specifically authorized.

0018-9162/03/$17.00 © 2003 IEEE

NGSCB platforms isolate operating systems and processes and implement hardware and software security primitives.

To enable this mode of operation, NGSCB platforms implement

- isolation among operating systems and among processes. OS isolation is related to virtual machine monitors. However, some key NGSCB innovations make it more robust than traditional VMMs by enabling a small machine monitor to isolate itself and other high-assurance components from the basic input/output system (BIOS), device drivers, and bus master devices.
- hardware and software security primitives that allow software modules to keep secrets and authenticate themselves to local and remote entities. These primitives maintain the trustworthiness of OS access protections without preventing the platform from booting other operating systems.

We refer to a security regimen that allows any software to run but requires it to be identified in access-control decisions as *authenticated operation,* and we call a hardware-software platform that supports authenticated operation a *trusted open system*.

A variety of commercial requirements and security goals guided the NGSCB system design. The main commercial requirement was for an open architecture that allows arbitrary hardware peripherals to be added to the platform and arbitrary software to execute without involving a central authority. Furthermore, the system had to operate in the legacy environment of personal computers. While we introduced changes to core platform components, most of the PC architecture remained unmodified. The system had to be compatible with the majority of existing peripherals. Finally, the hardware changes had to be such that they would not have a significant impact on PC production costs.

Our main security goal was assurance. Assurance is not any particular security function. It refers to the degree of confidence the owner of a system can have in its correct behavior—especially in the presence of attacks. A further goal was to enable authenticated operation.

The hardware platforms are *not* required to provide protection against hardware tampering. Protection against tampering costs money, and it is clear that most security attacks facing users are launched by malicious software, or are remotely launched and exploit bugs in otherwise benign software. However, we anticipate platforms will be deployed that are also robust against hardware attacks, especially in high-security corporate and government settings.

AUTHENTICATED OPERATION

Traditional access-control systems protect data against unauthorized access through an authentication mechanism such as a password, biometric data, or smart card. Each access request triggers a system component, the *guard*, that is part of the trusted computing base. The guard grants or denies access, and can audit access requests according to the user, the request, and the system's access-control policy.[1]

Authenticated operation bases access-control decisions in part on the identity of the program making a request. For example, a user can restrict access to files containing financial data to only certain authorized programs.

It is straightforward to extend most existing user-based access-control systems to code-based access models.[2] For example, a resource can have an access-control list that grants access only to a list of programs rather than to users who run these programs. We expect that most systems built to support authenticated operation will base access-control decisions on both program and user resource requests.

Definition of code ID

Code-based access control requires a method of establishing a program's identity. If the operating system can guarantee file-system integrity, it can simply assume that the program "is who it says it is." However, in distributed systems or platforms that let mutually distrustful operating systems run, establishing a cryptographic identity for programs is necessary.

The simplest example is a cryptographic digest or "hash" of the program executable code. Within this model the platform or operating system makes no assumptions about the security of applications stored on disk or on the network: If the application is modified, its cryptographic hash—and hence its identity and the services to which it is entitled—will change. Similar code ID mechanisms have been used elsewhere.[3-5]

Use of code ID

Sealed storage and attestation are two mechanisms that rely on code IDs.

Sealed storage. Sealed storage is a cryptographically implemented access-control mechanism in which the *sealer* of a secret states which programs (given by their code IDs) can unseal (read) the

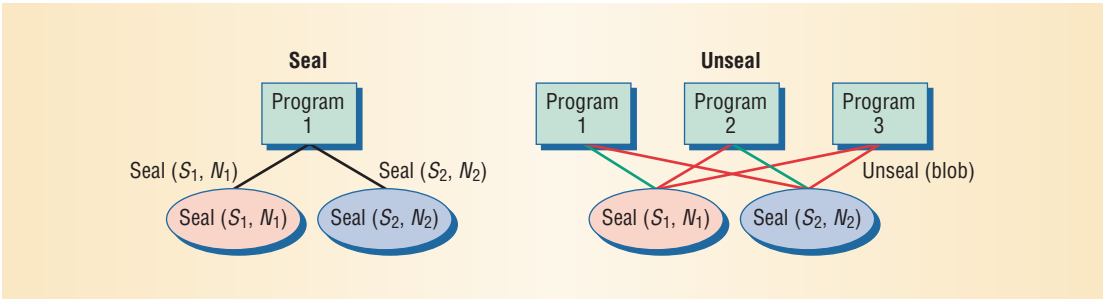

Figure 1. Seal and Unseal primitives. Seal allows a piece of software to protect a secret S and to name the programs N that can access the secret. If a program calls Unseal on previously sealed data. The secret is revealed only if the requester's identity is as specified in the sealed data block. Red lines indicate failed Unseal requests, and green lines indicate successful Unseal requests.

secret. Sealed storage provides confidentiality and integrity for persistently stored data. In principle, sealed-storage primitives can be implemented at any system layer. For example, the hardware could implement sealed storage as a service to operating systems or an operating system could implement it as a service to applications.

Figure 1 illustrates the Seal and Unseal primitives. Programs can call Seal and name their own code ID (the common case) or any other program's code ID as the entity allowed to access the data. If called by a program that has the sealed code ID, Unseal returns the sealed secret and the sealer's code ID. If the requesting program has a different code ID, Unseal returns an error.

Seal is designed as a local secret-storage mechanism: Sealed secrets are not accessible to other machines. The Seal and Unseal primitives have considerable implementation flexibility.[6] In general, the implementation layer—for example, the hardware or operating system—requires access to a cryptographic key K.

In a sample implementation, Seal creates a data structure containing the secret string provided by the program that called Seal, the code ID of the program to which the sealed data should be revealed, and the code ID of the program that called Seal so that the unsealer can identify the data source. Seal uses an authenticated encryption primitive[7] to encrypt this data structure and to protect its integrity. It can combine a cipher and a message authentication code to implement authenticated encryption.

Unseal takes the output of an earlier Seal operation. Unseal internally decrypts and verifies the integrity of its input. The integrity check fails if the input has been tampered with or if it did not originate in a Seal operation on this machine. In these cases, Unseal rejects the request.

If the integrity check succeeds, Unseal verifies that the code ID of the entity requesting Unseal is the code authorized when the data was sealed. If the code is authorized, Unseal discloses the secret string and the sealer's ID to the caller. If the code is not authorized, Unseal returns an error.

Unseal returns the caller's ID because any program can call Seal, and, in some cases, security depends on knowledge that the sealed data came from a known source. For example, in the common case of unsealed information being used as a cryptographic key, the unsealer needs to know that an adversary did not provide the key.

Attestation. Sealed storage is a restricted form of symmetric encryption that lets software programs keep long-lived secrets in persistent storage. Attestation is a variant of public-key encryption that lets programs authenticate their code ID to remote parties.[8]

A platform must have a certified public/private-key pair for attestation. Consider the signing variant that we call Quote. The Quote operation concatenates an input string from the program wishing to authenticate itself with the program's code ID, signs the resulting data structure with the platform's privacy quoting key, and returns the result to the caller. The requesting program can send this signed data structure to a remote party, typically along with platform certificates that support use of the platform-quoting key.

The recipient can verify the signature and hence the sender's code ID. If the recipient is satisfied with the sender's ID, it can engage in the requested transaction or data transfer. Of course, Quote and the rest of the transaction must form part of a cryptographic protocol that provides freshness and other guarantees.

Kernel boot and authenticated operation

Given that the identity of a piece of code is its cryptographic digest, an operating system can measure and record an application program's code ID at process creation. However, hardware must perform these functions for an operating system.

System hardware and microcode are responsible for measuring and recording a booting kernel's digest and for starting execution in an architecturally defined operating state. The platform needs a secure place to store the cryptographic keys necessary to implement sealed storage and attestation. We anticipate that a common implementation will be a cheap cryptographic processor, which we call a *security coprocessor*. The SCP is attached to or is part of the platform chipset.

The hardware provides authenticated operation services to the kernels that it hosts—for example, allowing a kernel to keep secrets and authenticate itself. In turn, the kernel will use these secrets to pro-

To accommodate conflicting requirements, machine partitioning lets two or more operating systems run on the same hardware.

vide similar services to the applications that it hosts.

IMPLEMENTING AUTHENTICATED OPERATION

Most existing access-control models provide specific solutions to specific usability and manageability problems. Authenticated operation brings with it both familiar and unfamiliar problems.

Upgrade

If a kernel is identified by its digest, then a single bit change results in an unrelated code ID, and any secrets sealed to the original kernel—or to applications that it hosted—are no longer accessible. To enable upgrade and other forms of data sharing, a kernel must provide controlled disclosure of secrets to other kernels.

A kernel can implement any form of upgrade policy. For example, the administrator could type in the upgraded kernel's code ID, and the running kernel could then reseal secrets to the new code ID.

Another example uses public-key cryptography to group operating systems based on cryptographic certification. Suppose kernel n has a public key embedded in it. Kernel $n + 1$ is accompanied by a statement naming the digest of kernel $n + 1$ with a signature verifiable against the public key embedded in kernel n.

If kernel n receives an upgrade request that includes a properly formed signed upgrade statement, the kernel will reseal all secrets—or a root secret that protects all other secrets—to the kernel digest named in the upgrade statement. When the user boots the new kernel, it will be able to access all old secrets.

Applications can handle their own upgrade using similar techniques, or the kernel can provide services to assist application upgrade.

Generic program code identity

Programs often base their execution behavior on external input. An interpreter is an extreme example of this. The interpreter's code identity is not particularly meaningful; in this case, code identity is best defined as "interpreter A, running script B." To reflect this, we can define a generic program's code ID as the hash of the concatenation of the program and its input data.

Other examples include a program that allows itself to be debugged if so instructed by data passed in by this means and a generic secure "chat" program that talks to a chat server identified by a public key in the input data. In these examples, a debuggable program will have a different code ID from that of a nondebuggable program, and a chat program used for corporate chat can be distinguished from the same program used for personal purposes.

Backup and migration

Platform sealed storage also provides strong data binding to the machine, but users commonly share data between machines.

Rather than burden the hardware with inflexible mechanisms for data sharing and migration, OS kernels and applications can implement their own policies for exporting data. Implementing a family of secure and flexible sharing mechanisms using the authenticated operation primitives is straightforward.

NGSCB SYSTEM OVERVIEW

NGSCB implements authenticated operation in the context of a complete system.

Machine partitioning

One approach to satisfying security requirements is to add security features, such as sealed storage and attestation, to legacy software systems. We believe, however, that this strategy is unlikely to succeed.

Mainstream mass-market operating systems are huge, containing tens of millions of lines of code, and they are optimized for functionality and performance. The diverse and ever-growing collection of PC peripherals and the corresponding device drivers are also often large and optimized for functionality and performance. The resulting collection of code that must be trusted to maintain security is large, decentralized, heterogeneous, and frequently changing—seemingly at odds with the basic design principles for secure systems.

Conceptually, machine partitioning accommodates these conflicting requirements by letting two or more operating systems run side by side on the same hardware, separated by a machine monitor.[9] One of these could be a traditional mass-market operating system in charge of managing most devices and running arbitrary legacy applications. One or more other systems could be dedicated to providing high-assurance execution environments. Possible implementations of the latter include dedicated high-assurance operating systems, regular operating systems that are effectively sandboxed by the machine monitor, or stand-alone applications.

Figure 2 shows an NGSCB configuration using the dedicated high-assurance operating system approach. The left half of the figure represents soft-

ware running on today's mass-market computers—an operating system, device drivers, and applications. The right half of the figure represents a small high-assurance OS kernel, which we call a *nexus*, and applications or *agents* running on it. Both systems coexist on a single computer. A machine monitor isolates the two systems to prevent them from interfering with each other.

The NGSCB hardware platform will let any software boot and run. However, the authenticated operation primitives enable each operating system to execute free of subversion or surveillance risk from other operating systems. Similarly, successful operating systems and nexuses can host any application but will provide authenticated operation primitives that protect hosted applications from other applications or operating systems.

Trusted paths

Many platforms will likely also offer a limited form of secure local user input and output. Such facilities will allow limited screen output mediated by the machine monitor without requiring the monitor to contain a complete graphics driver. Similarly, we expect secure keyboard and mouse input.

Initialization

NGSCB platforms allow a lightweight boot of a machine monitor from within an already running operating system. This decouples the nexus boot process from the platform boot process, which often involves BIOS and optional ROM firmware from many sources.

The security coprocessor required for authenticated operation is involved in the boot process so that it can measure or be reliably informed of the running monitor's identity. It also implements the sealed-storage and attestation primitives in internal firmware. Upcoming versions of the Trusted Computing Platform Alliance's Trusted Platform Module[10] can serve as a security coprocessor.

The coprocessor provides other services related and unrelated to authenticated operation. For example, it provides a random number generator and one or more hardware-based monotonic counters.

PRIVACY

Authenticated operation provides building blocks for much stronger protection of personal and private data on personal computers.

For example, if a stock trading program uses sealed storage to store the authentication token that authorizes stock trades, viruses and other unknown applications cannot access the data and use it or send it to a remote recipient. Similarly, even if the machine security is ill-configured—for example, the file system is improperly shared—the sealed data is meaningless if it is copied to a remote machine. Finally, a local application infected with a Trojan horse will have a different code ID and will not be able to unseal and hence misuse the private data.

In short, it is very difficult for unknown or untrustworthy code to misuse personal data protected by these mechanisms.

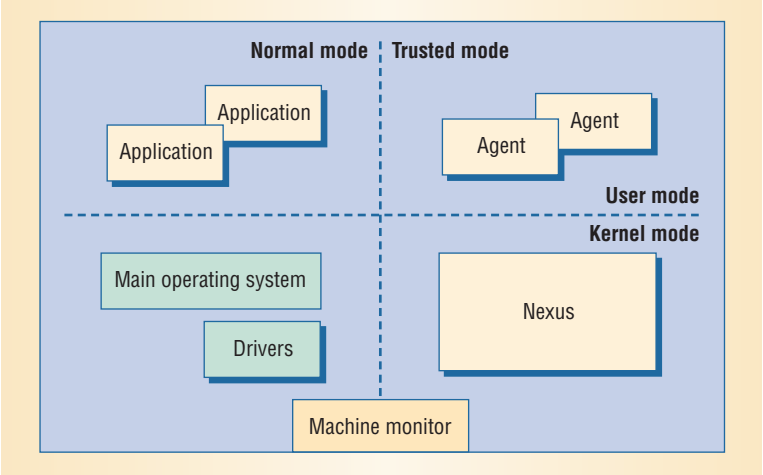

Figure 2. NGSCB system overview. The monitor partitions a machine between (left) an unmodified legacy operating system and (right) a smaller security-critical system manager called a nexus. The nexus hosts applications called agents that run in an isolated address space and have access to authenticated operation primitives. The nexus and its hosted agents may also have access to secure user input and output.

Randomizing operations

Unique cryptographic keys are a prerequisite to implementing authenticated operation. If platform-sealing keys are not unique, other machines can read the sealed data; if platform-quoting keys are not unique, others can impersonate the machine in network interactions.

However, if platform keys are improperly used or authenticated operation primitives are poorly implemented, the keys themselves present a different kind of privacy hazard—for example, servers could track the machine's activity as the user surfs the Web.

To ameliorate such concerns, the NGSCB system provides several software and hardware mechanisms. For example, sealed storage is designed to reveal no platform-identifying information: Seal, at both the hardware and software layers, adds randomness to each sealed data object so that malicious software calling Seal repeatedly on the same data is returned a different value each time.

In addition, all new hardware and software services in the NGSCB system are opt-in and under the user's control. Hence, users can choose not to use any of the authenticated operation security services, or they can choose to use the sealed storage primitives, but not to use the attestation functions.

Identity service providers

The platform attestation operations themselves do not reveal platform information. However, in

To ensure document integrity, an application can use digital signature technologies to reveal tampering with a signed document.

conjunction with the platform public key, the Quote operation is designed to provide strong platform authentication. Hence, operating systems must carefully control disclosure of the platform public key and use of the attestation functions.

At the hardware layer, the platform public-key and attestation function are access-controlled: The user must specifically authorize software before it can perform operations that reveal platform information. At the software layer, the nexus lets users restrict the parties to which it will reveal attestation information, or even whether the information is available at all.

Given the clear benefits of attestation, and the hazards of unrestricted platform ID disclosures, Microsoft encourages the formation of third-party *identity service providers* to act as trusted intermediaries between service providers and their customers. Users can configure their platforms to permit attestation to one or a few identity service providers. They can then request secondary attestation tokens, which vouch for a trusted platform without revealing its identity. Users can obtain an unlimited number of these pseudonyms for everyday Web transactions.

Microsoft also encourages using zero knowledge mechanisms that will allow the SCP to authenticate running software without revealing machine identifying data.

APPLICATIONS

The NGSCB system provides a strong security foundation for a broad set of applications. Several examples illustrate the benefits of authenticated operation.

Soft smart cards

Smart cards typically implement cryptographic protocols and keep private keys secure. To protect against loss or theft, the cards often require a host device to provide a short password or personal identification number (PIN) before they will operate.

An NGSCB-hosted smart-card application could use attestation to obtain authorization keys and sealed storage to keep keys secure. The smart-card application could then implement the cryptographic protocol as macrocode.

An NGSCB-hosted application could also use secure I/O to safeguard the PIN and give users reliable descriptions of the transaction they are authorizing. Existing smart-card deployments typically do not provide this protection.

Network logon

Users often log on to networks and services using an account name and password. Clients and servers use cryptographic protocols to prevent password snooping and man-in-the-middle attacks on the network, but the password is often vulnerable to client-side Trojan horses and key sniffers.

Using a soft smart card could strengthen network logon by preventing untrustworthy code from obtaining a user's authentication key. If the smart-card application uses secure input to authorize use of the keys, it is also more difficult for a local virus to automate logon against a user's wishes.

If the service wants to retain the account/password authentication on the server, the server could use an attested logon application that obtains passwords from users with secure input to strengthen network logon. Attestation allows servers to reject logon attempts from unknown software that has somehow obtained logon credentials.

Transaction authorization

Web services typically authenticate users by account name and password. Once a user has logged in, the service presumes that the software providing the password and requesting subsequent transactions is acting in the authenticated user's best interests. With the advent of widespread viruses and Trojan horses, however, this assumption may not always be correct.

To strengthen transaction authorization, a transaction-authorization application could use NGSCB's secure output to indicate to the user exactly the transaction being requested. For example, a vendor sends the user an HTML page description of the transaction, which an agent renders using secure output.

The agent could also use secure input to obtain local authorization of the displayed transaction. The application could use attestation to convey the transaction request and vouch for the trustworthiness of the client issuing the request.

Rights-managed data

The world is moving away from centrally located and administered data repositories and toward a heterogeneous collection of locally administered peers. In this model, users and servers have little assurance that documents, e-mail, or media content sent to others will remain secure.

Attestation allows peers to authenticate the platform and software to which they are revealing data. If senders can authenticate the receiver, they can attach rights restrictions to their data with a higher

degree of assurance that the recipient will honor the rights.

Document signing

E-mail and other electronic documents are replacing paper, but the ease of modifying electronic documents places them at a disadvantage to physical media.

If an application must ensure document integrity, it can use digital signature technologies to reveal any tampering that occurs after a document is signed. However, the open nature of client computers makes it easy for malicious software to compromise signing keys or tamper with the document being signed.

To help solve this problem, a system can use sealed storage to secure signing keys, use secure output to reliably display the document that the user is signing, and use secure input to authorize the document signature. Applications might also use attestation to obtain or certify the signing keys.

THREAT MODELS

The techniques we have described for improving the robustness of computers—especially with respect to data confidentiality and integrity—have strengths and limitations in the context of broadly deployed distributed systems.

In principle, individual machines should be unconditionally robust against software attack. More precisely, on computers with uncompromised hardware and correctly configured software, no action performed by external software should violate the access-control policy. Such protection would preserve data confidentiality even in the presence of Internet viruses, Trojan horses, worms, and so on.

In practice, achieving unconditional robustness against software attacks will depend critically on the ability to build hardware and software that are free from security-relevant bugs. NGSCB implements measures to reduce the number of hazards in a commercially viable system. For example, architectural decisions allow us to exclude the main operating system, most device drivers, and the BIOS from the trusted computing base.

On the other hand, there exists a whole range of physical attacks on the hardware that can succeed, given sufficient sophistication and computing power from the adversary. Hence, we expect that some fraction of machines will be compromised.

In the worst case, the adversary can gain knowledge of all data a compromised machine was guarding (sealed storage) as well as the attestation private key. The latter allows the adversary to impersonate a legitimate system in communications with remote computers. If the compromised information is used broadly (for example, the attestation private key is posted on the Internet), it is possible to detect the compromise and revoke the attestation key. Otherwise, detecting the break requires some out-of-band mechanism, such as hardware inspections on a corporate network by a security officer.

NGSCB holds the promise of protecting the confidentiality and integrity of data stored on computers that are under the physical control of the data's owner. In practical terms, consumers will enjoy increased PC security and privacy protections for activities such as online banking, online shopping, and storing arbitrary personal information.

Similar benefits apply in corporate settings. In addition, well-managed corporate networks are distributed systems that are under the corporation's physical control. In such settings, data can be distributed to a controlled set of corporate machines with reasonable assurance that data integrity and confidentiality will not be compromised.

NGSCB system applications that distribute confidential data to machines in potentially hostile physical environments must take into account the potential compromise of some machines and the exposure of confidential data. The consequences for the application depend on many application-specific factors. For example, in the case of broadly distributed, copy-protected entertainment content, the distributor must assume that some fraction of the recipients operate compromised machines and can therefore circumvent the copy protection system and redistribute the data. The commercial impact of this depends among other things on the redistribution channel's efficiency.[11]

Unconditional robustness against software attacks will depend on hardware and software that are free from security-relevant bugs.

The complexity, heterogeneity, and rate of change of the code base in today's open systems are contrary to the basic tenets of secure system design. The NGSCB system aims to provide security and openness while meeting the demands of commercially successful mass-market operating systems.

Microsoft is working with a broad coalition of hardware partners to enable NGSCB. Core hardware components—such as CPUs, chipsets, transaction process monitors, and video and input

support—are in different stages of development. Microsoft's Trusted Platforms team is developing the corresponding software components.

References

1. B. Lampson, "Protection," *Proc. 5th Princeton Symp. Information Sciences and Systems*, Princeton Univ., Mar. 1971; reprinted in *ACM Operating Systems Review*, Jan. 1974, pp. 18-24.
2. J. McLean, "Security Models," *Encyclopedia of Software Engineering*, 3rd ed., J. Marciniak, ed., Wiley Press, 1994.
3. E. Meijer and J. Gough, "Technical Overview of the Common Language Runtime," tech. report, Microsoft, 2001; http://research.microsoft.com/~emeijer/Papers/CLR.pdf.
4. P. Johns, "Signing and Marking ActiveX Controls," *Developer Network News*, 15 Oct. 1996; available at msdn.microsoft.com/.
5. D.S. Wallach et al., "Extensible Security Architectures for Java," tech. report 546-97, Dept. of Computer Science, Princeton Univ., Apr. 1997.
6. P. England and M. Peinado, "Authenticated Operation of Open Computing Devices," *Proc. 7th Australasian Conf. Information Security and Privacy* (ACISP), Springer-Verlag, 2002, pp. 346-361.
7. M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm," *Advances in Cryptology—Asiacrypt 00*, Springer-Verlag, 2000, pp. 531-545.
8. B. Lampson et al., "Authentication in Distributed Systems: Theory and Practice," *ACM Trans. Computer Systems*, Nov. 1992, pp. 265-310.
9. R. Goldberg, "Survey of Virtual Machine Research," *Computer*, June 1974, pp. 34-45.
10. Trusted Computing Platform Alliance, *TCPA Main Specification Version 1.1*, 2001.
11. P. Biddle et al., "The Darknet and the Future of Content Protection," to be published in *Proc. 2002 ACM Workshop on Digital Rights Management*, Springer-Verlag, 2003.

Paul England *is a software architect in Microsoft's Security Business Unit. He received a PhD in physics from Imperial College, London. Contact him at pengland@microsoft.com.*

Butler Lampson *is a Distinguished Engineer at Microsoft Research, where he works on systems architecture, security, and advanced user interfaces. He received the ACM's Turing Award in 1992 and the IEEE's von Neumann Medal in 2001. Contact him at blampson@microsoft.com.*

John Manferdelli *is the general manager of the Windows Trusted Platform and Infrastructure product unit at Microsoft. He received a PhD in mathematics from the University of California, Berkeley. Contact him at jmanfer@microsoft.com.*

Marcus Peinado *is an architect in Microsoft's Trusted Platform Technologies group. He received a PhD in computer science from Boston University. Contact him at marcuspe@microsoft.com.*

Bryan Willman *is a software architect in the Windows Kernel group at Microsoft. Contact him at bryanwi@microsoft.com.*