A Logical Account of NGSCB

Martín Abadi[1] and Ted Wobber[2]

[1]University of California at Santa Cruz
[2]Microsoft Research, Silicon Valley

Abstract. As its name indicates, NGSCB aims to be the "Next-Generation Secure Computing Base". As envisioned in the context of Trusted Computing initiatives, NGSCB provides protection against software attacks. This paper describes NGSCB using a logic for authentication and access control. Its goal is to document and explain the principals and primary APIs employed in NGSCB.

1. Introduction

NGSCB ("Next-Generation Secure Computing Base", formerly known as "Palladium") integrates hardware and software components that aim to help in protecting data and processes against software attacks [8,9,14,15]. The hardware includes a cryptographic co-processor that contains keys and offers basic cryptographic services. The software includes new, trusted operating system components.

While the architecture and the implementation of NGSCB continue to evolve, quite a few of its features have been discussed in public. We believe that it is worthwhile to elucidate them further. Many of these features seem likely to remain important as NGSCB matures, and also appear in other projects and research efforts in the area of Trusted Computing [1,10,12,13,16].

In this paper, we present an attempt to understand the fundamentals of NGSCB in terms of a logic for authentication and access control. This formalism had its origins in the context of the Taos operating system and of the Digital Distributed System Security Architecture [11,12,18]. In this application to NGSCB, we use the logic for describing relationships between principals while abstracting away most of the details of the underlying cryptographic protocols. Although it may be feasible and perhaps attractive, we do not relate the logic to concrete implementations, nor base new implementations on the logic. Our goal is to document the components and primary APIs employed in NGSCB, and to provide concise and principled explanations for them.

At present, one may view all work on NGSCB as "work in progress". This paper is no exception. Because the specifics of NGSCB remain subject to change, we are less concerned with giving a detailed and up-to-the-minute account than with providing a consistent explanation of important concepts and techniques.

The next section reviews the logic. Section 3 reviews the basics of NGSCB. Sections 4, 5, and 6 describe principals, derived authorities, and the certification infra-

structure (which is external to NGSCB but necessary for its applications). Section 7 deals with the main system services in logical terms. Section 8 briefly addresses privacy. Section 9 discusses an example. Section 10 concludes.

2. A Brief Logic Review

The logic enables us to describe a system in terms of principals and their statements. The logical formula *P says s* means that principal *P* makes or supports statement *s*. The principal may for example be a user, a piece of hardware, the combination of some hardware and some software, or a cryptographic key.

We also allow compound principals, particularly of the form $P \mid C$. The meaning of $P \mid C$ is "*P* quoting *C*"; we have that $P \mid C$ says *s* when *P* says that *C* says *s*. For instance, *P* may represent a piece of hardware, and *C* a piece of code or a user.

In addition, the logical formula $P \Rightarrow Q$ means that *P* speaks for *Q*, so if *P says s* then *Q says s*, for every *s*. We generally assume the *hand-off axiom* which says that if *Q says* $P \Rightarrow Q$ then indeed $P \Rightarrow Q$. We use the "speaks for" relation for many purposes. For instance, we often write that a key *K* speaks for a principal *P* when *K* is *P*'s public signature key. Typically only *P* knows the corresponding signing key (*K*'s inverse) and can produce signatures that can be checked with *K*. We may also write that a principal speaks for any group of which it is a member; thus, when we write that *P* speaks for a group we typically mean that *P* is or speaks for some member of the group, not necessarily all members of the group. (It would be easy to extend the logic with a membership relation, and to replace "speaks for" with membership in these uses; whether this extension is worth the trouble remains open to debate.) We may represent an access control list (ACL) as the group of the principals authorized by the list. If G_X is the ACL for accessing an object *X*, then *P* speaks for G_X when *P* is authorized to access *X*.

Although the logic does not lead to correctness proofs of the kind expected in high-assurance systems, this logic and its relatives have been useful in several ways in the past. Much as in this paper, the logic has served for describing and documenting the workings of a system, what the system achieves, and on what assumptions it relies, after the fact or in the course of development. In this respect, formal notations do not accomplish anything beyond the reach of careful, precise prose, but they are helpful. The logic has also served in validating particular techniques for authorization, reducing them to logical reasoning, and also as a basis for new techniques; research on stack inspection and proof-carrying authorization exemplify this line of work [3,6,17,18]. Finally, the logic has served as a foundation for languages for writing general security policies [7].

We refer to previous papers for more detailed descriptions of the logic and its applications.

3. Assumptions on NGSCB

We assume that at the root of any NGSCB node there exists a hardware-based security facility that implements cryptosystems, random number generation, and key storage. We use the term *Security Support Component* (*SSC*) to describe this facility since it has been previously used in related literature. The Trusted Platform Module (TPM) from the Trusted Computing Group [16] may be the main current example of an SSC.

We further assume that the hardware has the capability to load an operating system that can be reliably identified by taking a hash (or *code-id*) of the initial operating system image and data. This operating system, so loaded, will reside in a protected area of memory that cannot be accessed by untrusted code that the operating system might load. Therefore, the hardware has reason to believe that the statements made by the securely loaded operating system can be attributed to the principal identified by the code-id. In turn, the operating system can load a child process and attribute statements made by that child process to the principal identified by the hash of its code and data. Following one existing nomenclature for NGSCB, we call the securely loaded operating system the *Nexus*, and we call any child process that it loads a *Nexus Computing Agent* (*NCA*). The Nexus may be implemented, for example, by combining a virtual-machine hypervisor with a trusted guest operating system [10,15]. For simplicity, we focus on situations with one distinguished Nexus and one distinguished NCA; of course, other software may be running on the hardware at the same time.

Finally, we assume trusted input and output paths for communication with users. In particular, the hardware may guarantee that only a particular Nexus receives input from the keyboard and can send output to a display. The Nexus may in turn provide a similar guarantee to a particular NCA.

These assumptions are consistent with previous, public descriptions of NGSCB, such as the ones found in papers and on the Web. Those descriptions, like most informal descriptions, are however incomplete and imprecise in some respects. One of the goals of the present logical account is to complement those descriptions, with additional details (some of them validated in private conversations with the NGSCB team, and some of them conjectured rather than based on an official NGSCB design), and with a partial rationale for the workings of NGSCB.

4. Principals

Next we enumerate principals relevant for NGSCB security.

The following principals are particular to each NGSCB node. Each node will have different instances of them.

K_0	the permanent public key of the SSC
K_T	a per-boot public key of the SSC

S_0	the master symmetric key of the SSC
S_T	a per-boot symmetric key derived from S_0

The inverse of the key K_0 and the symmetric key S_0 never leave the SSC hardware; the inverse of the key K_T and the symmetric key S_T may or may not leave the hardware, as discussed below. We rely on asymmetric cryptography (public-key operations with K_0, K_T, and their inverses) primarily for digital signatures, rather than for public-key encryption. When encryption is needed, we indicate it explicitly. The symmetric key S_0 may be replaced with a pair of keys for asymmetric encryption, with only minor changes.

The following principals represent software images. There can, of course, be many different images in which we might be interested. For simplicity of exposition, we will be concerned with only two:

C_{NEX}	the code-id of a particular Nexus
C_{NCA}	the code-id of a particular NCA

The following principals complete the cast; they provide the context outside an NGSCB system:

M	a manufacturer of SSC hardware
V	a vendor or author of Nexus software
A	a vendor or author of NCA software
K_M	M's public signature key
K_V	V's public signature key
K_A	A's public signature key
G_M	a group of public signature keys for SSCs produced by M
G_V	a group of code-ids for Nexus software images produced by V
G_A	a group of code-ids for NCA software images produced by A
CA	a trusted certification authority

5. Derived Authorities

It would be possible for an SSC to make statements only with its permanent public key K_0. However, it is desirable to sign as few certificates as possible with this key. Therefore, we assume that, at boot time, the SSC generates a temporary key pair, consisting of the public key K_T and its inverse. Then K_0 transfers all of its authority to K_T. This hand-off of authority is captured in the following statement:

$$K_0 \text{ says } K_T \Rightarrow K_0$$

The certificate described here, and all subsequent certificates mentioned in this paper, should be considered valid only for a limited period of time. The logic does not directly model time, so we do not represent time formally; one could probably add it with a modest effort.

In this formulation, we assume that the SSC holds the temporary secret key (the inverse of K_T) in hardware and uses the key for signing statements on behalf of other principals on the local machine. This key could instead reside in the Nexus and be accessed through a similar interface. In this case, the key would no longer be protected within the SSC, so the two arrangements entail different security properties.

Because of the secure loading steps described above, a successfully loaded Nexus will have the authority of the compound principal $K_T \mid C_{NEX}$. An SSC can run any Nexus software, but the rights of a specific Nexus instance are exactly those of the SSC parameterized by the code-id of the Nexus. Similarly, an NCA loaded on top of a Nexus would speak as $K_T \mid C_{NEX} \mid C_{NCA}$.

6. Certification Infrastructure

In order to deduce anything useful about statements made by the software running on an NGSCB node, we must have trust assumptions. We hypothesize the presence of a certification authority *CA* that makes statements that are globally trusted. In particular, we trust *CA* to specify the set of acceptable NGSCB nodes and the set of trusted Nexus and NCA software images; we express this trust as follows:

$$CA \Rightarrow G_M$$
$$CA \Rightarrow G_V$$
$$CA \Rightarrow G_A$$

We simplify a bit here: in practice, *CA* will almost always be implemented by a hierarchy of certification authorities and there will be multiple subgroups of G_M, G_V, and G_A, according to the intended applications and trust relationships.

Next, we must give some key (or set of keys) the authority to certify membership in the groups G_M, G_V, and G_A. We represent such statements in the following certificates:

$$CA \text{ says } K_M \Rightarrow G_M$$
$$CA \text{ says } K_V \Rightarrow G_V$$
$$CA \text{ says } K_A \Rightarrow G_A$$

Finally, we use the signing keys that correspond to K_M, K_V, and K_A for making membership certificates for the specific hardware/software stack that we intend to construct:

$$K_M \text{ says } K_0 \Rightarrow G_M$$
$$K_V \text{ says } C_{NEX} \Rightarrow G_V$$
$$K_A \text{ says } C_{NCA} \Rightarrow G_A$$

Combining the certificates and trust assumptions, we can derive:

$$K_0 \Rightarrow G_M$$
$$C_{NEX} \Rightarrow G_V$$
$$C_{NCA} \Rightarrow G_A$$

Note that if K_0 is a member of G_M (so $K_0 \Rightarrow G_M$ in our model) then K_0 can also define new group members of G_M. In particular, $K_0 \Rightarrow G_M$ and K_0 *says* $K_T \Rightarrow K_0$ imply that $K_T \Rightarrow G_M$. Using a primitive membership relation rather than "speaks for" would remove this possibility.

7. Programmatic Interface

The programmatic interface of NGSCB supports the sealing of information and hardware-based attestation. Next we explain those functions in terms of the logic and of the definitions of the previous sections.

7.1. Sealing

***Seal*(*X*,*C*).** The *Seal* function stores the data X in such a way that it can be retrieved later by the same SSC, and only by that SSC. Furthermore, a code-id C is bound into the result so that the SSC can restrict subsequent access to that code-id. For example, an NCA might seal data under its own code-id for later retrieval, or the Nexus might seal data under the code-id of a subsequent Nexus version as part of a version migration strategy. The sealed data might be private user information, and the goal of the sealing might be to protect this information from viruses on the same machine.

Sealing amounts to setting up an access control rule, which we model with the following statement:

$$S_0 \textit{ says } C \Rightarrow G_X$$

This means that the hardware asserts that C is a member of the group G_X of principals that can access an object with the data X. Nothing here precludes the possibility that other objects also contain the data X and that code other than C may access those objects.

In practice, the SSC does not store the data, but rather encrypts it with its private key and returns the sealed data item. Here we can use the symmetric key S_0 instead of K_0 since the statement is always evaluated in the context of the local machine. So this kind of sealing can be accomplished through authenticated encryption using symmetric ciphers and message authentication codes (MACs).

In order to limit the exposure of S_0, the use of a per-boot symmetric secret, S_T, might be desirable. (Indeed, the TPM design goes further in permitting chains of intermediate keys.) Suppose that we generate a per-boot nonce, N, and derive S_T as a function of S_0 and N for example by setting S_T = HMAC(S_0,N). This definition implies that N must be stored in plaintext with the sealed content in order to allow the recovery of S_T. In the logic, the access control rule for the sealed object can be expressed with the following statements:

$$S_0 \textit{ says } S_T \Rightarrow S_0$$
$$S_T \textit{ says } C \Rightarrow G_X$$

The *Seal* function might be offered to NCAs by the Nexus, or NCAs might be given direct access to the SSC's *Seal* function. In the former case, a symmetric key held by the Nexus would be used for sealing instead of S_T. This design has the advantage of allowing more straightforward migration to different hardware, but the disadvantage of exposing temporary keys within the memory system.

***Unseal*(*X,C*).** The *Unseal* function retrieves data held under seal. Conceptually, access to the sealed data is granted on the basis of the result of evaluating the corresponding ACL using the code-id of the caller. When *Seal* relies on encryption, the SSC implements *Unseal* by decrypting data that it previously sealed under its own secret.

***PKSeal*(*X,K*).** The *PKSeal* function is similar to *Seal* except that a target key *K* is used instead of a code-id. In this case the unsealer is not assumed to be the same SSC as the sealer. Therefore, *PKSeal* can be used to seal data for retrieval on another machine.

We can describe *PKSeal* by an access control rule, much as we did for *Seal*:

$$K_T \text{ says } K \Rightarrow G_X$$

If *PKSeal* is implemented by placing *X* on a storage server operated by a trusted third party, then the certificate K_T *says* $K \Rightarrow G_X$ can be directly useful as input to the reference monitor on that server: when *K* requests access to *X*, the reference monitor grants it.

Alternatively, as its name suggests, the implementation *PKSeal* can perform public-key encryption on *X*. For this purpose, the public key *K* should be an encryption key (and not just a key for checking signatures).

***PKUnseal*(*X,K*).** Much like *Unseal*, *PKUnseal* implements the corresponding access control rule. When *PKUnseal* relies on encryption, the implementation of *PKUnseal* relies on decryption. In this case, whoever holds the inverse of the public key used for sealing will have de facto access to *X*.

7.2. Attestation

***Quote*(*ST*).** The *Quote* function allows the SSC to attest to statements made by principals under its control. For example, the SSC may attest that a particular, trusted application (not a virus) is making a request to write a file.

Suppose that an NCA wishes to utter the statement *ST* and have the SSC attest to this statement over a network. As described in Section 5, the NCA speaks with the authority:

$$K_T \mid C_{NEX} \mid C_{NCA}$$

Since only cryptographic keys can securely make statements over an otherwise unprotected network, the SSC encodes the uttered statement as:

$$K_T \text{ says } (C_{NEX} \mid C_{NCA} \text{ says } ST)$$

According to the definition of quoting in the logic, this formula can be written more straightforwardly:

$$K_T \mid C_{NEX} \mid C_{NCA} \text{ says } ST$$

In particular, *Quote* can be used to attest that a key speaks for a principal. For example, suppose that an NCA wishes to indicate that a key K is authorized to make statements on its behalf. In this case, the quoted statement ST is:

$$K \Rightarrow K_T \mid C_{NEX} \mid C_{NCA}$$

and the certificate that the SSC would form in order to attest to this hand-off of authority would be:

$$K_T \text{ says } (C_{NEX} \mid C_{NCA} \text{ says } (K \Rightarrow K_T \mid C_{NEX} \mid C_{NCA}))$$

which reduces to:

$$K_T \mid C_{NEX} \mid C_{NCA} \text{ says } K \Rightarrow K_T \mid C_{NEX} \mid C_{NCA}$$

Here the key K may be a public key, but it may also be a symmetric key that underlies an authenticated communication channel from the NCA.

This kind of quoting might be used by the Nexus directly as well. In this case, the SSC would produce:

$$K_T \text{ says } (C_{NEX} \text{ says } K \Rightarrow K_T \mid C_{NEX})$$

which implies:

$$K_T \mid C_{NEX} \text{ says } K \Rightarrow K_T \mid C_{NEX}$$

***Verify*(*ST*).** The *Verify* function must decode a statement generated by *Quote* and check the consistency of its cryptographic evidence. Furthermore, the results of *Verify* should enable reasoning about the principal that made the statement in question.

Often, the receiver of a statement interprets the statement in a different trust environment (and in a different machine) than the sender. The receiver must come to conclusions based on its own trust assumptions. For example, if the receiver sees:

$$K_T \text{ says } (C_{NEX} \mid C_{NCA} \text{ says } ST)$$

and believes:

$$K_0 \text{ says } K_T \Rightarrow K_0$$

then the receiver can conclude:

$$K_0 \mid C_{NEX} \mid C_{NCA} \text{ says } ST$$

Suppose that the receiver has the certificates and trust assumptions introduced in Section 6. Then the receiver can deduce:

$$G_M \mid G_V \mid G_A \text{ says } ST$$

The receiver may trust $G_M \mid G_V \mid G_A$ on *ST*. For example, when *ST* represents a request for access to an object *X*, the ACL for *X* may include $G_M \mid G_V \mid G_A$, granting access to a member of G_M quoting a member of G_V quoting a member of G_A. (In other words, the receiver may have that $G_M \mid G_V \mid G_A \Rightarrow G_X$.) Thus, the receiver can reason about the principal that said *ST* and use the identity of that principal as the basis for access control decisions.

8. Privacy

It may be undesirable for all certification chains associated with the statements from a given SSC to be rooted at a single key K_0. If this were the case, then an observer might be able to track the activity of specific machines and use this information to compromise user privacy. In light of such concerns, several privacy-enhancing mechanisms have been developed.

Upon boot, the Nexus might communicate with an anonymization service that would be trusted to issue semi-permanent key pairs to trusted system components, and also trusted to respect privacy. In this case, the Nexus carries out a secure transaction with the anonymization service, using the authority $K_T \mid C_{NEX}$. After establishing that $K_T \mid C_{NEX} \Rightarrow G_M \mid G_V$, the service returns a collection of public keys K_i, their inverses, and certificates of the following form:

$$K_{ANON}\ says\ K_i \Rightarrow G_M \mid G_V$$
$$CA\ says\ K_{ANON} \Rightarrow G_M \mid G_V$$

where K_{ANON} is the public key of the anonymization service. The inverse of any key K_i can be used by the Nexus to sign subsequent statements. Since neither K_i nor K_{ANON} are linked to any single user, SSC, or Nexus, this indirection provides some anonymity for the holder of the inverse of K_i.

Variants of this scheme can provide keys K_i that speak for G_M or for $G_M \mid G_V \mid G_A$ (rather than for $G_M \mid G_V$). The inverse of a key that speaks for G_M should not be under the control of the Nexus; therefore, the anonymization service should return the key sealed in such a way that only the SSC itself can access it.

More sophisticated schemes rely on group cryptography [2,4,5]. Using group cryptography, an SSC may issue signatures that cannot be distinguished from signatures generated by some set of other SSCs. Instead of K_0, each SSC has a share of the group key $K_{NODE\text{-}GROUP}$. The manufacturer *M* makes $K_M\ says\ K_{NODE\text{-}GROUP} \Rightarrow G_M$. Then the SSC can produce a certificate for a temporary key K_T:

$$K_{NODE\text{-}GROUP}\ says\ K_T \Rightarrow K_{NODE\text{-}GROUP}$$

Although this certificate does not identify a particular SSC, shares of the group key can be revoked; the specifics of revocation vary across schemes.

9. An Example

In this section we exercise the logic on a practical example of an application of NGSCB, due to Butler Lampson and Paul Leach. In this application, an NCA presents an image (perhaps of a sales order or an online-banking transfer) to the user of a machine, and attests to the fact that the user clicked "OK" to accept the consequences implied by the image. For this application, we have to assume that there is a trusted path between the NCA and the keyboard and display in front of the user. We also assume that an untrusted banking or purchasing application is running (perhaps in a web browser) on the user's machine outside the context of NGSCB. This untrusted application uses the trusted NCA to carry out security-critical aspects of online transactions directly with the merchant or bank.

There are many different protocols that could support this sort of scenario. We will assume a simple model in which the NCA establishes an authenticated channel to a bank, and uses that channel to assert that a specific user has confirmed the contents of a specific image. For these purposes, we assume that the user can present a password known to the bank.

Although the Nexus and NCA might depend on an untrusted operating system to communicate with the network, the cryptography used to establish and maintain a secure channel can be implemented within a trusted NCA. Let us say that the NCA establishes an SSL channel to the bank and authenticates the bank using a certificate chain in the usual style. The NCA can also authenticate itself to the bank in that SSL exchange. In the logic, this authentication can be written as:

$$K_T \mid C_{NEX} \mid C_{NCA} \text{ says } Channel \Rightarrow K_T \mid C_{NEX} \mid C_{NCA}$$

where

$$K_0 \text{ says } K_T \Rightarrow K_0$$

Much as in Section 7.2, the bank can now deduce that *Channel* speaks for a trusted NGSCB node running a trusted NCA. Using *Channel* or by other means, the bank can transmit an appropriate image for the user to accept. The image is received by the NCA and shown on the user's display via the trusted path to the hardware. Perhaps the target window is distinguished with a specialized border that indicates secured images of this form. If the image is acceptable to the user, then the user is asked to provide a password and click "OK". Gathering this input on the trusted path from the keyboard, the NCA can now make the following statement on the trusted channel to the bank:

$$Password \text{ says } OK\text{-}Image$$

Here, the image might be represented by its hash. If all is in order, the bank can deduce:

$$Channel \mid User \text{ says } OK\text{-}Image$$

then

$$K_0 \mid C_{NEX} \mid C_{NCA} \mid User \text{ says } OK\text{-}Image$$

and hence

$$G_M \mid G_V \mid G_A \mid \textit{User says OK-Image}$$

Now the bank may conclude that the user (or at least an NCA holding the user's password) authorized the consequences represented by *Image*. Furthermore, the bank may also conclude that the transaction took place over a channel from a trusted NGSCB node and a trusted NCA.

The bank may impose restrictions on the set of machines and software that can serve as origin of the channel. These restrictions may thwart certain types of attacks. For example, even if the password is compromised, it cannot be used directly by just any application.

10. Conclusion

This paper describes NGSCB in terms of a logic for authentication and access control. Its goal is to document and explain NGSCB's principals and primary APIs. It aims to complement previous descriptions of NGSCB, with additional design elements, and with a (partial) formal rationale for the workings of NGSCB.

As discussed in the introduction, NGSCB is still "work in progress". We will not venture predictions on its future evolution or applications. It is possible that some or all of the features of NGSCB described in this paper will change. That represents a risk, but an inevitable one whenever one applies formal techniques in the course of the development process. We believe that, in any case, those features and their principles will be valuable beyond the context of NGSCB.

Acknowledgements

We wish to thank Butler Lampson, John Manferdelli, Fred Schneider, and Jeannette Wing for discussions on this work and encouragement, and Marcus Peinado for information on NGSCB. Martín Abadi's work was done at Microsoft Research, Silicon Valley.

References

1. Abadi, M.: Trusted computing, trusted third parties, and verified communications. To appear in *Proceedings of the 19th IFIP International Security Conference (SEC 2004),* Kluwer, 2004.
2. Ateniese, G., Camenisch, J., Joye, M., and Tsudik, G.: A practical and provably secure coalition-resistant group signature scheme. In *Proceedings of Crypto 2000*, pages 255–270, Springer-Verlag, 2000.

3. Appel, A., and Felten, E.: Proof-carrying authentication. In *Proceedings of the 5th ACM Conference on Computer and Communications Security*, pages 52–62, 1999.
4. Boneh, D., Boyen, X., and Shacham, H.: Short group signatures. To appear in *Proceedings of Crypto 2004*, Springer-Verlag, 2004.
5. Brickell, E.: An efficient protocol for anonymously providing assurance of the container of a private key. Submitted to the Trusted Computing Group, 2003.
6. Bauer, L., Schneider, M., and Felten, E.: A general and flexible access control system for the Web. In *Proceedings of the 11th USENIX Security Symposium 2002*, pages 93–108, 2002.
7. DeTreville, J.: Binder, a logic-based security language. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 105–113, 2002.
8. England, P., Lampson, B., Manferdelli, J., Peinado, M., and Willman, B.: A trusted open platform. *IEEE Computer*, 36(7):55–62, 2003.
9. England, P., and Peinado, M.: Authenticated operation of open computing devices. In *Proceedings of the 7th Australasian Conference on Information Security and Privacy,* pages 346–361, Springer-Verlag, 2002.
10. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., and Boneh, D.: Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the 19th Symposium on Operating System Principles (SOSP 2003)*, pages 193–206, 2003.
11. Gasser, M., Goldstein, A., Kaufman, C., Lampson, B.: The Digital distributed system security architecture. In *Proceedings of 12th National Computer Security Conference*, pages 305–319, NIST/NCSC, 1989.
12. Lampson, B., Abadi, B., Burrows, M., and Wobber, E.: Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4):265–310, 1992.
13. Lie, D., Thekkath, C., Mitchell, M., Lincoln, P., Boneh, D., Mitchell, J., and Horowitz, D.: Architectural support for copy and tamper resistant software. In *Ninth International ACM Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-IX)*, pages 168–177, 2000.
14. Microsoft Corporation: Next-generation secure computing base. Archive Product Information, http://www.microsoft.com/resources/ngscb/archive.mspx.
15. Peinado, M., Chen, Y., England, P., and Manferdelli, J.: NGSCB: A trusted open system. To appear in *Proceedings of the 9th Australasian Conference on Information Security and Privacy (ACISP 2004)*, Springer-Verlag, 2004.
16. Trusted Computing Group: Home page, http://www.trustedcomputinggroup.org.
17. Wallach, D., Appel, A., and Felten, E.: SAFKASI: a security mechanism for language-based systems. *ACM Transactions on Software Engineering and Methodology*, 9(4):341–378, 2000.
18. Wobber, E., Abadi, M., Burrows, M., and Lampson, B.: Authentication in the Taos operating system. *ACM Transactions on Computer Systems*, 12(1):3–32, 1994.