Strict Access Control in a Key-Management Server

Christian Cachin Anil Kurmus Marko Vukolić
IBM Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{cca,kur,mvu}@zurich.ibm.com

3 June 2009

1 Introduction

Key management is concerned with operations to manage the lifecycle of cryptographic keys, for creating, storing, distributing, deploying, and deleting keys. An important aspect is to manage the attributes of keys that govern their usage and their relation to other keys. Multiple efforts are currently underway to build and standardize key-management systems accessible over open networks: the W3C XML Key Management Specification (XKMS) [18], the IEEE P1619.3 Key Management Project [12], the OASIS Key Management Interoperability Protocol (KMIP) standardization effort [14], and the Sun Crypto Key Management System [16] are some of the most prominent ones. Cover [9] gives an up-to-date summary of the current developments.

Many proprietary key-management systems are on the market, including HP StorageWorks Secure Key Manager, IBM Distributed Key Management System (DKMS), IBM Tivoli Key Lifecycle Manager (TKLM), NetApp Lifetime Key Management, and Thales/nCipher keyAuthority. The need for enterprise-wide key management systems has been recognized widely [4], and NIST, an agency of the US Government, has issued a general recommendation for key management [3].

Such a key-management server is generally accessed by multiple clients, who perform operations on keys and other cryptographic objects maintained by the server. The objects may include *symmetric keys*, *public keys*, *private keys*, *certificates*, and more; they typically have a range of attributes describing their lifecycle and their usage in cryptographic operations. Operations allow to create, import, read, search, update, and delete keys by the server, and generally focus on attribute handling rather than on cryptographic functions. A comprehensive key-management server will also support some small set of cryptographic operations, including to create a key, to issue a certificate, to *derive* a new key (a deterministic operation that creates a symmetric key from an existing one), and to *wrap* or *unwrap* a key with another key (wrapping means to encrypt a target key with another key for export and transfer to another system). These features can be found in many of the above-mentioned protocols and systems.

In this work, we report on a design for controlling access to operations and to keys in a key-server prototype, which we are currently developing. The key server is able to distinguish between different *users*, which are the principals that invoke operations, and to securely authenticate them.

Because the key-management server provides the above-mentioned cryptographic functions, it represents a *cryptographic security API* accessible over a network. Security APIs stand at the boundary between untrusted code and trusted modules capable of maintaining internal state. Cryptographic security APIs are typically provided by cryptographic tokens [1], hardware-security modules (HSM) like IBM's 4764 cryptoprocessor that supports the IBM CCA interface [11, 13] and generic PKCS #11-compliant [15] modules, smartcards, or the Trusted Platform Module [17]. This work extends the study of cryptographic security APIs to protocols over open networks.

2 Access Control

We distinguish between *basic* and *strict* access control in the key server. In *basic* mode, access-control decisions for a key are taken directly from an *access-control list (ACL)* associated with it. But because the operations of our key server allow users to create complex relationships between keys, primarily through key derivation and key wrapping, basic access control may have security problems. For example, if there exists a particular key that some user is not allowed to read, but the user may wrap that key under another key of its choice and export the wrapped representation, the user may nevertheless obtain the bits of the first key. Another example is a key that was derived from a parent key by the server; when a user reads the parent key, the user implicitly also obtains the cryptographic material of the derived key.

In general, a cryptographic interface that manages keys and allows to create such dependencies among keys poses the problem that access to one key may give a user access to many another keys. This issue has been identified in the APIs of several cryptographic modules [2, 6, 8, 10] and may lead to serious security breaches when an organization does not fully understand all implications of an API.

In *strict* mode, therefore, access-control decisions by the key server take the semantics of the key-management API into account and implement a cryptographically sound access-control policy on all symmetric keys and private keys. The above issues with basic access control are eliminated with strict access control. Our strict access-control policy builds on the work of Cachin and Chandran [7], which describes a secure cryptographic token interface, introduces a cryptographically strong security policy, and shows how to implement it. A strict access-control decision may not only depend on the ACL of the corresponding key, but takes also into account the ACLs of related keys and the history of past operations executed on them. It prevents any unauthorized disclosure of a symmetric key or a private key.

Every key maintained by the server has several *attributes* that govern if an access is permitted. The basic policy is determined by an *access-control list (ACL)* attribute. It can be modified by clients and contains a list of user/privilege-pairs. A boolean attribute *strict* determines if the key underlies only the basic or the strict access-control policy.

Every key maintained by the key-management server in strict mode benefits from an explicitly stated security policy that respects cryptographic side-effects of the server's operations. In particular, it guarantees that a user may only retrieve the information she is authorized to, i.e., that she cannot abuse the API to violate the access control policy. Thus, it avoids the problems mentioned above and similar problems existing in other APIs [2, 6, 8, 10], which arise from interdependencies among the keys.

In a forthcoming paper [5], we report on the challenges with designing and on the lessons learned from implementing strict access control in the prototype key-management server.

References

[1] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov, "Cryptographic processors — a survey," *Proceedings of the IEEE*, vol. 94, pp. 357–369, Feb. 2006.

[2] R. J. Anderson, "Why cryptosystems fail," in *Proc. 1st ACM Conference on Computer and Communications Security (CCS)*, pp. 215–227, 1993.

[3] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid, "Recommendation for key management," NIST special publication 800-57, National Institute of Standards and Technology (NIST), 2007. Available from `http://csrc.nist.gov/publications/PubsSPs.html`.

[4] BITS Security Working Group, "Enterprise key management." Whitepaper, BITS Financial Services Roundtable, available from `http://www.bits.org/downloads/Publications%20Page/BITSEnterpriseKeyManagementMay2008.pdf`, May 2008.

[5] M. Björkqvist, C. Cachin, R. Haas, X.-Y. Hu, A. Kurmus, R. Pawlitzek, and M. Vukolić, "Design and implementation of a key-lifecycle management system," Research Report RZ 3739, IBM Research, June 2009.

[6] M. Bond, "Attacks on cryptoprocessor transaction sets," in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, vol. 2162 of *Lecture Notes in Computer Science*, pp. 220–234, 2001.

[7] C. Cachin and N. Chandran, "A secure cryptographic token interface," in *Proc. Computer Security Foundations Symposium (CSF-22)*, IEEE, July 2009. To appear.

[8] J. Clulow, "On the security of PKCS#11," in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, vol. 2779 of *Lecture Notes in Computer Science*, pp. 411–425, 2003.

[9] "Cover pages: Cryptographic key management." `http://xml.coverpages.org/keyManagement.html`, Apr. 2009.

[10] S. Delaune, S. Kremer, and G. Steel, "Formal analysis of PKCS#11," in *Proc. 21st IEEE Computer Security Foundations Symposium (CSF)*, 2008.

[11] J. G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, and S. Weingart, "Building the IBM 4758 secure coprocessor," *IEEE Computer*, vol. 34, pp. 57–66, Oct. 2001.

[12] IEEE Security in Storage Working Group (SISWG), "P1619.3/D6 draft standard for key management infrastructure for cryptographic protection of stored data." Available from `https://siswg.net/index.php`, 2009.

[13] International Business Machines Corp., *CCA Basic Services Reference and Guide for the IBM 4758 PCI and IBM 4764 PCI-X Cryptographic Coprocessors*, 19th ed., Sept. 2008. Available from `http://www-03.ibm.com/security/cryptocards/pcicc/library.shtml`.

[14] OASIS Key Management Interoperability Protocol Technical Committee, "Key Management Interoperability Protocol," Apr. 2009. Editor's draft 0.98; available from `http://www.oasis-open.org/committees/documents.php?wg_abbrev=kmip`.

[15] RSA Laboratories, "PKCS #11 v2.20: Cryptographic Token Interface Standard." Available from `http://www.rsa.com/rsalabs/`, 2004.

[16] Sun Microsystems, "Sun Crypto Key Management System (KMS)." `http://opensolaris.org/os/project/kmsagenttoolkit/`, 2009.

[17] Trusted Computing Group, "Trusted platform module specifications." Available from `http://www.trustedcomputinggroup.org`, 2008.

[18] World Wide Web Consortium, XML Key Management Working Group, "XML Key Management Specification (XKMS 2.0)." Available from `http://www.w3.org/2001/XKMS/`, 2005.