

# Key Challenges in DRM: An Industry Perspective

Brian A. LaMacchia\*

Microsoft Corporation  
One Microsoft Way  
Redmond, WA 98052-6399 USA  
bal@microsoft.com

**Abstract.** The desires for robust digital rights management (DRM) systems are not new to the commercial world. Indeed, industrial research, development and deployment of systems with DRM aspects (most notably crude copy-control schemes) have a long history. Yet to date the industry has not seen much commercial success from shipping these systems on top of platforms that support general-purpose computing. There are many factors contributing to this lack of acceptance of current DRM systems, but I see three specific areas of work that are key adoption blockers today and ripe for further academic and commercial research. The lack of a general-purpose rights expression/authorization language, robust trust management engines and attestable trusted computing bases (TCBs) all hamper industrial development and deployment of DRM systems for digital content. In this paper I briefly describe each of these challenges, provide examples of how the industry is approaching each problem, and discuss how the solutions to each one of them are dependent on the others.

## 1 Introduction

When we think about digital rights management (DRM) systems, we tend to focus on the content that is to be managed by the system and the infrastructure needed to protect the content while it is in the system. Questions about content protections quickly turn into questions about content encryption, which generally yields questions about management of various types of cryptographic keys. This is comforting for us because the cryptography of DRM is familiar, and as a community we have developed (and continue to improve) a vast body of knowledge in cryptography and related spaces. If the design and construction of DRM systems was solely a matter of choosing the right cryptographic techniques to apply we would be in great shape: simply choose the algorithms we want to use for encryption, key management, secret sharing, traitor tracing, etc., and we could go build the systems.

---

\* The views expressed in this paper are those of its author and are not necessarily those of Microsoft Corporation.

Of course, DRM systems are not solely a collection and application of cryptographic techniques to content. In addition to the managed content there are also policies describing the access rules for that content, and a DRM system must manage these policies in addition to the content controlled by the policies. Unlike content management, policy management is not just a matter of encrypting some bits and distributing the decryption keys in the proper manner. A DRM system must perform all of the policy-related tasks necessary to “project policy, with confidence that the policy will be respected, from the content owner to the remote environment where the content will be used.” Thus, policy management also includes tasks such as authoring, distributing, and evaluating policy expressions, for we must be able to create and reason about policy statements first before we can address the problem of projecting those statements into remote execution environments.

Collectively we have given significantly more attention in our research and development to content management rather than policy management, yet the key technical challenges we face today relate to the latter. In this paper I outline three of these challenges: authoring policy expressions, evaluating policy expressions, and projecting policy expressions with confidence into remote environments. Work is required in all three areas in order to make interoperable DRM systems built on top of general-purpose computing platforms viable.

## 2 Authoring Policy Expressions

A DRM policy management system has two core components: a language for expressing policy statements and an evaluator that can make decisions on the basis of such expressions. Both components are critical to acceptance of the DRM system. If the language is not sufficiently expressive to allow users (content owners, distributors and consumers) to write the types of policies they wish, then they will not be willing to use the system and the system will not attract content. Similarly, if the language is not easy enough for users to reason about and clearly communicate managed content policies (either directly or with support from appropriate tools) then user acceptance of the system will be low. Interoperability of statements written in the language is also a requirement as the policy evaluator must consider statements from many different sources (policy specifications, authentication credentials, authorization credentials) when making content access decisions.

When taken together, these requirements indicate that the success of DRM systems will depend in part on a “general-purpose” rights expression language (REL)—an extensible syntax and semantics for expressing grants of authorizations.<sup>1</sup> An REL is a type of policy authorization language where the focus of the language is on expressing and transferring rights (capabilities) from one party

---

<sup>1</sup> The need for industry-standard authorization languages is much broader than just the DRM space; as we continue to build larger and larger distributed systems we need a *lingua franca* for communicating authorizations among all networked nodes. The need is especially apparent in the “web services” model of distributed programming

to another in an interoperable format. Issuance rights (the right to issue grants of other rights) and delegation rights (the right to delegate a grant to another party) are core concepts in an REL. A “general-purpose” REL must also allow flexibility and extensibility in the types of rights and resources that it references. There are a number of efforts within various standards bodies working on general authorization languages [1, 7–9], but the most advanced work to date on a rights expression language is that based on the XML Rights Management Language (XrML 2.X) [5], including the REL and RDD groups within MPEG-21 [11] and the RLTC group within OASIS [14].

XrML 2.X is a direct descendant of Stefik’s Digital Property Rights Language (DPRL) [15]. The first version of DPRL (version 1.0) focused on specifying machine-enforceable rights; a subsequent update (version 2.0) enabled the specification of more complex rights (potentially including fees, terms and conditions) for digital works. In 2000 the data model in DPRL was converted to XML and the resulting language, together with some additional language extensions, was named XrML 1.0. Version 2.0 of XrML, released by ContentGuard in November 2001, restructured the syntax and added a number of features that significantly increased the expressive power of the language.

XrML 2.X was designed to make it easy to create policy statements—called *licenses*—that represent arbitrary authorization grants from one party to another. A single authorization statement in XrML is always of the form, “*Issuer* authorizes *principal* to exercise a *right* with respect to a *resource* subject to (zero or more) *conditions*.” Multiple authorizations from the same issuer may be grouped together into a single license. For example, “John says ‘Bill has the right to print the book’” is an example of the form of authorization statements expressible in XrML—John is the issuer, Bill is the principal, and printing (with respect to the book resource) is the right being granted. Grants may be chained together either through direct trust of the issuer or transitively through licenses that grant rights to issue other licenses. As an example of the latter situation consider the following two licenses:

1. Alice says, “Bob has the right to issue a license to anyone to print the book.”
2. Bob says, “Carol has the right to print the book.”

If only Alice is implicitly trusted by the DRM system, then license #2 alone is insufficient to prove that Carol has the right to print the book. However, if both licenses #1 and #2 are present then the XrML evaluator can determine that Alice granted Bob the right to issue licenses such as license #2, and therefore the presence of both licenses proves Carol has the right to print the book.

Two other features of XrML 2.X are particularly important in the DRM space. First, XrML 2.X licenses may include patterns, variables and quantifiers anywhere within a grant. This allows us to write licenses such as “Alice says, ‘Anyone who can read the book has the right to print the book,’” where one or more clauses of the licenses are instantiated at evaluation time. Without such

---

as it is expected that any networked node can dynamically discover, learn how to communicate with and access any available service (with proper authorization).

expressions it is difficult to write licenses that refer to groups of principals or resources (especially resources that may not yet exist at the time the license is granted). Second, XrML 2.X licenses may also contain *prerequisite rights* that condition the grant contained within the license. If the grant in a license contains one or more prerequisite rights, then the grant is valid only in the presence of other licenses that prove that the prerequisites are also satisfied. For example, suppose Alice issues a license that says, “Bob has the right to read the book if Bob is a member of the book club.” In this license the phrase “if Bob is a member of the book club” is a prerequisite right; the license is only valid in the presence of other licenses that prove Bob’s membership in the club.

The prerequisite rights feature implies that the “compliance checking” algorithm for XrML 2.X is more complicated than simple “chain walking,” such as that used by X.509/PKIX certificates [13]. When evaluating an X.509/PKIX certificate, the job of the compliance checker is to construct a valid chain of related certificates (a path through the certificate graph) from a trusted root certificate to the end-entity certificate that needs to be validated. In contrast, evaluating an XrML 2.X license may require constructing a valid directed acyclic graph (DAG) that proves multiple conditions in parallel in order to prove the validity of a single license. Each additional prerequisite right in an encountered license creates an additional branch to be satisfied. (In the degenerate case where no prerequisite rights are present the DAG collapses down to a single chain of licenses.) Thus, the complexity of a general-purpose rights expression language like XrML 2.X necessitates a similarly advanced license compliance checker (policy evaluator).

Perhaps the most challenging issue yet to resolve in the field of policy expression languages is the tension that arises naturally when attempting to represent liability-based systems such as copyright law through explicit expressions of rights or permissions. Policy evaluators want expressions and credentials that can be evaluated and determined to be true facts or false statements. Evaluating laws, however, often calls for a fact-finder to balance competing interests and make judgment calls.<sup>2</sup> It is possible that short-term progress can be made by establishing safe harbors for system behaviors that approximate the balance of interests [6], but long-term solutions remain hidden.

### 3 Evaluating Policy Expressions

When we think about policy management systems we tend to focus first on the types of statements we want to make and how parties will author them. Once a set of semantics for policy statements has been agreed to our attention turns to

---

<sup>2</sup> The canonical example of the potential fuzziness of copyright law in the U.S. is the process of determining whether a particular use of a copyrighted work is fair or infringing. The fact-finder is directed (17 USC 107) to consider and weigh at least four different factors in each case. Under the U.S. copyright regime it is impossible to know whether a particular use is a fair use without an inquiry and determination by a court of law.

designing algorithm to evaluate such statements. The policy evaluator has to be able to reason correctly about all the types of policy statements and credentials it may encounter when making a trust decision, thus the design of the evaluator is going to be influenced by the design of the language. This is especially true in the DRM case, as the DRM policy evaluator may need to inspect and verify many different credentials in order to make its decision.<sup>3</sup> A DRM policy evaluator has to decide for each requested access whether the policy (or policies) relevant to the request allows it to occur, given the credentials. This formulation of DRM policy evaluation is the “compliance checking” decision problem in trust management [3], and since the policies and credentials governing access to a resource managed by the DRM system may be arbitrarily complex, it is clear that our DRM policy evaluator is just an instance of a robust, general-purpose trust management engine.

Starting with the development of PolicyMaker [3] we have seen a succession of active research [2, 4] and commercial deployment [10] of general-purpose trust management engines. The attractiveness of this approach has grown with the increased complexity of distributed systems as well as the types of resources that need to be protected. In the .NET Framework’s Common Language Runtime, for example, the trust management engine at the core of the policy system is responsible for dynamically associating authorizations with every piece of executable code loaded into a process. Content distribution adds another dimension (or two) to the problem, because the set of resources to be protected is in fact the entire set of content potentially available to the client over the network, and the types of activities authorized with respect to any particular piece of content may be arbitrarily precise. That is, the set of objects to be managed by a DRM system is unbounded (all potentially-available content), and even if the set of subjects granted access to that content is limited to a small number of users, the number of credentials granting access rights to those users is also likely to be unbounded. Paradoxically, the fact that these sets are unbounded makes general-purpose trust management engines *more* attractive as policy evaluators as it is the programmatic nature of a trust management engine that allows it to efficiently deal with instances drawn from arbitrarily large sets of subjects, objects and credentials.

While trust management algorithms may work well in policy environments with potentially unbounded input sets, human reasoning suffers. One of the primary lessons learned from real-world implementations of trust management engines is that they are often too general (and thus too complex) for most users to be able to reason about effectively. DRM systems will need complex policy evaluators in order to perform the reasoning and decision-making tasks we desire, but system builders will also need to make engineering tradeoffs to simplify the model wherever possible for users. By way of example, the .NET Framework initially exposes a “simplified” policy model and administration tools

---

<sup>3</sup> As DRM systems provide conditional access to content they manage, at a minimum access decisions are always based on at least two sets of statements: content-specific policies and credentials that relate to the party requesting access to the content.

that were designed to be much easier to understand than a general-purpose trust management engine yet still meet the needs of most customer scenarios. For users and administrators who require more functionality than that exposed in the simplified model it is possible to “dive beneath the surface” and access the full power of the underlying trust management engine. Such tradeoffs and multi-tiered policy authoring and administration systems will almost certainly be necessary if generalized DRM systems are to be successful.

The need for good user interfaces for describing and configuring trust management policies is an open work area for DRM system policy evaluators. One of the major challenges of any security system is creating a management interface that is comprehensible to its users. Lack of an easy-to-understand interface can significantly slow or completely inhibit acceptance of a new technology.<sup>4</sup> The move from trust management models based solely on programming languages [2–4] to hybrid models that include more familiar management structures [10] is indicative of the types of improvements that need to be made in tools for creating and inspecting content policies and credentials. Such improvements will be especially important given the complexity of general-purpose rights expressions languages.

## 4 Projecting Policy Expressions with Confidence into Remote Environments

In addition to authoring and evaluating policy expressions, a DRM system operating across multiple nodes in a network must be able to accomplish a third important task: projecting policy to remote nodes with confidence that the policy will be respected. Fear about platform behavior is anathema to the distribution of information, and such fear is rampant today across all segments of potential DRM users. Owners of digital content will not distribute their works to platforms they consider “hostile” (or potentially so) and the same is true of individual users requested to reveal private information to remote systems. Every content owner needs some way to be convinced that the remote system receiving his or her valuable information will behave as the owner expects, which ultimately means that the remote system will implement the policy that the content owner has defined and associated with his content.

In computer systems security research, we routinely design security protocols that are grounded in trusted computing bases (TCBs). Policy authors must implicitly trust TCBs to operate correctly and behave in accordance with their design parameters, for any TCB is ultimately capable of violating the policy it is supposed to enforce. For local policy enforcement we care about the local TCB that is interpreting the policy, but in DRM systems is it not sufficient for the content owner to trust the TCB for the system on which he creates

---

<sup>4</sup> As an example, consider the poor performance to date of authentication protocols based on X.509/PKIX client certificates. Cumbersome user enrollment protocols and procedures, coupled with inadequate user interfaces for communicating key- and certificate-related information to users, has blocked significant use of the technology.

(authors) the policy for his content.<sup>5</sup> Additionally, the content owner must also have confidence that remote nodes receiving the policy will behave as the author expects and faithfully implement the defined policies. That is, in DRM systems we need the ability to be able to prove to the local node (and, ultimately, the content owner) that a remote node is operating with and relying on a TCB with known properties. (Depending on the scenario we may also need to prove to the remote node that the local node is running a TCB with certain properties.) Only after all parties to the transaction are convinced that the relevant network nodes are operating with behaviors that are defined, understood and acceptable can the content transaction occur.

In order to prove their existence and operation to a remote entity, DRM system node TCBs need an additional property: *attestability*. An *attestable TCB* is a TCB that is able to convince a remote party that it is running and behaving according to some specification.<sup>6</sup> Specifically, an attestable TCB must have four key properties: it must be open, auditable, comprehensible and provable to a remote party. The need for openness is obvious: parties depending on the behavior of a TCB must be able to inspect the construction of the TCB in order to convince themselves that the TCB implements its specification correctly. Once the TCB is running, it must be possible to audit its behavior and check for deviations from the specification, thus the need for the second property. Comprehensibility is an aspect of openness but deserves to be called out explicitly: it is important that the operation of the TCB not only be observable (open) but also understandable by those observing it.<sup>7</sup> (There is an obvious tension here between the need to make the policy evaluator more complex—to handle the various types of authorizations and resources—and the need to make it “attestable.”) Finally, the TCB must support one or more mechanisms to prove to remote parties that it is operating.

There are two separate industry initiatives underway currently that are attempting to build attestable TCBs on top of personal computer hardware and software: the Trusted Computing Platform Alliance (TCPA) [12, 16, 17] and Mi-

---

<sup>5</sup> Without loss of generality we assume here that the content owner is also the author of the content access policy in a DRM system. Of course the content owner may have delegated that authority to another party (e.g. the operator of a digital library).

<sup>6</sup> It is important to note that a TCB is attestable relative to some threat model that is part of the behavior specification. For example, an attestable TCB that depends on a hardware-based cryptographic key for the attestability property only has that property so long as the hardware containing the key material has not been compromised. There are thus many flavors of attestability depending on the mechanism used to make and convey the proof of correct operation.

<sup>7</sup> One fact implied by the need for comprehensibility is that it is not sufficient to simply publish the source code for a software-based TCB. Additionally, the algorithms used in the source code must be understandable to all observers. This means, for example, that we may need to use simpler, less-efficient algorithms inside a TCB than we would otherwise. The most likely place such a need will arise initially is in number-theoretic algorithms (e.g. bignum multiplication).

icrosoft’s “Palladium” initiative.<sup>8</sup> TCGA is specifying changes to the PC hardware platform to allow the platform to make attestations about the entire software stack running on the computer (starting with the BIOS boot block). In contrast, the goal of “Palladium” is to create a separate, parallel execution environment inside the computer that is rigidly controlled by the user, and make attestations only about code loaded and executing in that parallel environment.

Both TCGA and “Palladium” leverage hardware-based public-key cryptography to generate attestations about software. A hardware component<sup>9</sup> is added to the PC motherboard that can perform RSA digital signatures and compute SHA1 cryptographic hash values (along with other cryptographic operations). The component is similar to a smartcard core—it includes a small amount of physical storage and one or more RSA key pairs. The hardware component computes the SHA1 hash value of the software stack of interest.<sup>10</sup> A digital signature over the hash value is then created by an RSA key that was certified by some third party as being associated with the cryptographic hardware component. The digital signature together with whatever certifications the third party provided for the signing key forms the attestation. Whether the attestation will convince a remote party is thus an instance of the typical scenario for using some form of public key infrastructure: the remote party must be convinced to his satisfaction that the RSA signing key belongs to a TCGA- (respectively, “Palladium”-) enabled platform. Assuming that the certifications are sufficient to accomplish this, the remote party can further deduce that a particular set of software is executing on the machine.

The attestations produced in TCGA- or “Palladium”-based systems are necessarily conditioned on maintaining the integrity of the cryptographic hardware. If the hardware is compromised then it is no longer possible to prove that the TCB is actually running, since the hash value computed by the hardware could be incorrect or the private key could be extracted from the chip and made to sign non-corresponding hash values. When evaluating an attestation produced on one of these systems the remote/relying party must evaluate the risk that the generating node’s hardware has been compromised. Hardware significantly improves the security of the attesting keys over what is possible to do with software alone and addresses the vast majority of scenarios that require shared attestations, but relying parties must still understand the risk profiles and weigh them against their particular scenario.

We began this section by describing policy projection in DRM systems; it should now be clear that attestable TCBs are one mechanism for doing so. A content owner distributing his content can ask a remote node to prove it is run-

---

<sup>8</sup> Microsoft is also a member of the TCGA.

<sup>9</sup> In TCGA the hardware component is called the “Trusted Platform Module” (TPM). In “Palladium” it is called the “security support component” (SSC).

<sup>10</sup> In TCGA everything from the boot block forward can be hashed; in “Palladium” the software of interest is a security kernel that runs in a special CPU mode. How the two types of systems guarantee that the software of interest is hashed is beyond the scope of this paper.

ning a TCB that can understand the owner’s policy (and will enforce it) before sending content to the node. In fact, once we have an attestable TCB, the TCB itself can make attestations recursively about the code running on top of it, so a stack of related attestations could be required. Ultimately, though, everything depends on the attestable TCB and the fact that all parties understand the TCB and believe that it operates properly.

The attempts to build attestable TCBs described above depend on hardware cryptographic components for some core key storage and cryptographic computation. Assuming that the hardware components have not been tampered with it is possible to use the hardware to attest to the operation of a software component; the software component can then recursively attest to the operation of other pieces of software “higher up the stack.” Software components within the TCB can be inspected and audited for proper operation, but the same is not easily true for hardware components. We need to figure out how to build the hardware components of an attestable TCB so that they too are open, auditable and comprehensible to all parties. Today we implicitly trust the manufacturer of the hardware that the parts behave as specified,<sup>11</sup> and it is unclear what we can do to improve the accessibility of any hardware component.

## 5 Summary

In this paper I have summarized three policy-related technical challenges and some approaches to solving them currently being pursued by DRM system builders. From a policy perspective, the ultimate goal of a distributed DRM system is for content authors to be able to project policies governing their content into remote environments with confidence that those policies will be respected by the remote nodes. The first policy-related challenge is to define an interoperable rights expression language that is sufficiently expressive that it can encode the types of content policies desired while still being comprehensible to content users. Finding the right balance between usability and complexity when designing DRM policy evaluators is the second challenge and in many ways progress in this area is co-dependent on advancements in policy languages. Finally, once we have a widely-accepted, interoperable rights expression language and policy evaluators that can interpret policies and credentials authored in the language, we will also need to create attestable TCBs to serve as the foundations of DRM system nodes. Content owners, distributors and consumers must mutually have confidence that all nodes participating in a DRM system will behave as expected in order for content and their corresponding policies to be introduced into and flow through the system.

---

<sup>11</sup> To be precise, while it is possible today to inspect a hardware component and deduce its operation, the process can be quite time-consuming and may require specialized instruments. (Not many people have the skills and equipment to “shave the top off a chip” and probe its inner workings to confirm behavior.)

## References

1. J. Ayars, *XMCL—the eXtensible Media Commerce Language*, draft as of June 2001. Available at <http://xmcl.org/specification.html>.
2. M. Blaze, J. Feigenbaum, and A. D. Keromytis, “KeyNote: Trust Management for Public Key Infrastructures,” *Proceedings of the Sixth International Workshop on Security Protocols*, B. Christianson, B. Crispo, W. Harbinson and M. Roe, eds., Lecture Notes in Computer Science **1550**, Springer-Verlag, NY (1999), 59–63.
3. M. Blaze, J. Feigenbaum, and J. Lacy, “Decentralized Trust Management,” *Proceedings 1996 IEEE Symposium on Security and Privacy*, 164–173, May 1996.
4. Y.-H. Chu, J. Feigenbaum, B. LaMacchia, P. Resnick and M. Strauss, “REFEREE: Trust Management for Web Applications,” *Proceedings of the Sixth International World Wide Web Conference*, Santa Clara, CA, April 1997. Reprinted in *Computer Networks and ISDN Systems* 29 (1997), 953-964.
5. “eXtensible Rights Markup Language (XrML) 2.1,” submission by ContentGuard to the OASIS Rights Language Technical Committee, May 2002. Available at <http://www.oasis-open.org/committees/rights/documents/xrml200205.zip>.
6. B. Fox and B. LaMacchia, “A Safe Harbor for Designers of DRM Systems,” *Communications of the ACM*, to appear.
7. S. Godik and T. Moses, eds., “OASIS eXtensible Access Control Markup Language (XACML),” OASIS eXtensible Access Control Markup Language Technical Committee, Working Draft, September 2002.
8. P. Hallam-Baker and E. Maler, eds., “Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML),” OASIS XML-Based Security Services Technical Committee, May 2002.
9. R. Iannella, ed., “Open Digital Rights Language (ODRL), Version 1.1.” Available at <http://odrl.net/1.1/ODRL-11.pdf>.
10. B. LaMacchia, S. Lange, M. Lyons, R. Martin and K. Price, *.NET Framework Security*, Addison-Wesley, April 2002.
11. Moving Picture Experts Group (MPEG), ISO/IEC JTC1/SC29/WG11. Working documents available at [http://mpeg.telecomitalia.com/working\\_documents.htm](http://mpeg.telecomitalia.com/working_documents.htm).
12. S. Pearson, ed., *Trusted Computing Platforms, TCPA Technology in Context*, Prentice Hall PTR, July 2002.
13. PKIX Working Group, Internet Engineering Task Force. RFC 3280, “Internet X.509 Public Key Infrastructure: Certificate and Certificate Revocation List (CRL) Profile,” R. Housley, W. Ford, W. Polk, D. Solo, eds., April 2002. Available at <http://www.ietf.org/rfc/rfc3280.txt>.
14. H. Reddy, Chairperson, OASIS Rights Language Technical Committee. Charter and documents available at <http://www.oasis-open.org/committees/rights/>.
15. M. Stefik, *The Digital Property Rights Language, Manual and Tutorial, Version 1.02*, September 18th, 1996. Xerox Palo Alto Research Center, Palo Alto, CA.
16. Trusted Computing Platform Alliance, *TCPA Main Specification, Version 1.1b*. Available at <http://www.trustedcomputing.org/docs/main%20v1.1b.pdf>.
17. Trusted Computing Platform Alliance, *TCPA PC Specific Implementation Specification, Version 1.00*. Available at [http://www.trustedcomputing.org/docs/TCPA\\_PCSpecificSpecification.v100.pdf](http://www.trustedcomputing.org/docs/TCPA_PCSpecificSpecification.v100.pdf).